

Apache Doris 2.1 Documentation (English)

Archived documentation

2026-11-24 · 1221 documents

Source: e0fed2a66fcd

Contents

1. Introduction to Apache Doris
2. Quick Start | GettingStarted
3. Alternative to ClickHouse
4. Alternative to Elasticsearch
5. Apache Doris vs Trino / Presto
6. Environment Checking
7. Cluster Planning
8. OS Checking
9. Deploy Integrated Storage Compute Cluster Manually
10. Deploy Doris Operator
11. Config Doris to Deploy
12. Deploy Doris Cluster
13. Access Cluster
14. Cluster Operation
15. Deploying on AWS
16. Connecting by MySQL Protocol
17. Connecting by Arrow Flight SQL Protocol
18. Overview | Table Design
19. Table Model Overview
20. Duplicate Key Model
21. Unique Key Model
22. Aggregate Model
23. Usage Notes
24. Data Distribution Concept
25. Manual partitioning
26. Dynamic Partitioning
27. Auto Partition
28. Data Bucketing
29. Common Issues
30. Basic concepts
31. Data Types
32. Column Compression
33. Index Overview
34. Sort Key and Prefix Index
35. Inverted Index
36. BloomFilter Index
37. N-Gram BloomFilter Index
38. Schema Change

39. Auto-Increment Column
40. Tiered Storage Overview
41. Tiered Storage of SSD and HDD
42. Remote Storage
43. Hybrid Row-Columnar Storage
44. Best Practices | Table Design
45. Loading Overview
46. local file
47. Kafka
48. Flink
49. HDFS | Data Source
50. Amazon S3
51. Google Cloud Storage | Data Source
52. Azure Storage
53. Alibaba Cloud OSS
54. Huawei Cloud OBS
55. Tencent Cloud COS
56. MinIO | Data Source
57. S3 Compatible Storage
58. Snowflake
59. BigQuery
60. Redshift
61. Migrating Data from Other OLAP
62. Migrating Data from Other OLTP
63. Stream Load
64. Broker Load
65. Routine Load
66. Insert Into Select
67. Insert Into Values
68. MySQL Load
69. Log Storage and Analysis | Doris Docs
70. CSV
71. JSON | File Format
72. Parquet | File Format
73. ORC | File Format
74. ARRAY | Complex Types
75. MAP | Complex Types
76. STRUCT | Complex Types
77. JSON | Complex Types
78. Bitmap
79. HLL
80. Variant

81. Handling Data Issues
82. Transforming Data During Load
83. Load High Availability
84. High Concurrency LOAD Optimization(Group Commit)
85. Load Best Practices
86. Data Update Overview
87. Updating Data with UPDATE Command
88. Updating Data on Unique Key Model
89. Updating Data on Aggregate Key Model
90. Concurrency Control for Updates in the Primary Key Model
91. Delete Overview
92. Deleting Data with DELETE Command
93. Batch Deletion Based on Load
94. Deleting Data with TRUNCATE Command
95. Replacing Atomic Table
96. Temporary Partition
97. Transaction
98. Export Overview
99. Export Manual | Export
100. SELECT INTO OUTFILE
101. Using MySQL Dump
102. Best Practices | Export
103. MySQL Compatibility
104. Join
105. Subquery
106. Multi-Dimensional Analytics
107. Window Function
108. Common Table Expression
109. Alias Function
110. Java UDF, UDAF, UDWF, UDTF
111. Complex Type
112. Column to Row (Lateral View)
113. Tuning Overview
114. Diagnostic Tools
115. Analysis Tools
116. Tuning Process
117. Materialized View Overview
118. Sync-Materialized View
119. Overview of Asynchronous Materialized Views
120. Creating, Querying, and Maintaining Asynchronous Materialized Views
121. Best Practices | Async Materialized View
122. Use Advice

123. FAQ
124. SQL Cache
125. High-Concurrency Point Query Optimization
126. BITMAP Precise Deduplication
127. HLL Approximate Deduplication
128. Colocation Join
129. Overview of Hints
130. Leading Hint
131. Distribute Hint
132. Optimizing Table Schema Design
133. Optimizing Table Index Design
134. Optimizing Table Scanning
135. Transparent Rewriting with Sync-Materialized View
136. Transparent Rewriting by Async-Materialized View
137. Optimizing Join with Colocate Group
138. Adjusting Join Shuffle Mode with Hint
139. Control CBO Rules With Hint
140. Reordering Join With Leading Hint
141. Accelerating Queries with SQL Cache
142. DML Tuning Plan
143. Adjustment of RuntimeFilter Wait Time
144. Data Skew Handling
145. Parallelism Tuning
146. Common Tuning Parameters
147. Query Optimizers
148. Parallel Execution
149. Runtime Filter
150. TOPN Query Optimization
151. Statistics | Optimization Technology Principle
152. Lakehouse Overview
153. Data Catalog Overview
154. Hive Catalog
155. Iceberg Catalog
156. Hudi Catalog
157. Paimon Catalog
158. MaxCompute Catalog
159. Delta Lake Catalog
160. BigQuery Catalog
161. Kudu Catalog
162. Elasticsearch Catalog
163. LakeSoul Catalog
164. JDBC Catalog

165. MySQL JDBC Catalog
166. PostgreSQL JDBC Catalog
167. Oracle JDBC Catalog
168. SQL Server JDBC Catalog
169. IBM Db2 JDBC Catalog
170. Clickhouse JDBC Catalog
171. SAP HANA JDBC Catalog
172. Oceanbase JDBC Catalog
173. Analyze Files on S3/HDFS
174. Hive Metastore
175. AWS Glue
176. Google Dataproc
177. Aliyun DLF
178. Iceberg Rest Catalog API
179. File System
180. HDFS | Storages
181. S3 | Storages
182. Azure Blob
183. Google Cloud Storage | Storages
184. Aliyun OSS
185. Tencent COS
186. Huawei OBS
187. Baidu BOS
188. MinIO | Storages
189. Parquet | File Formats
190. ORC | File Formats
191. Text/CSV/JSON
192. Data Cache
193. Metadata Cache
194. Elastic Compute Node
195. Statistics | Lakehouse
196. Data Lake Query Optimization
197. Using Doris and Hudi
198. Using Doris and Paimon
199. Using Doris and Iceberg
200. Using Doris and LakeSoul
201. Integration with Glue + AWS S3 Tables
202. Integration with Aliyun DLF Rest Catalog
203. From MaxCompute to Doris
204. Generating TPC-H on Hive/Iceberg
205. Generating TPC-DS on Hive/Iceberg
206. Overview | Observability

207. Log
208. Trace
209. Logstash
210. Filebeat
211. OpenTelemetry
212. FluentBit
213. LoongCollector (iLogtail) Doris Flusher
214. Langfuse on Doris
215. Integrating Vector with Doris
216. Security Overview
217. Authentication and Authorization
218. Built-in Authentication
219. Federated Authentication
220. Built-in Authorization
221. Ranger Authorization
222. Data Access Control
223. Audit Log
224. MySQL Client Certificate
225. FE SSL Certificate
226. Encryption and Masking Function
227. Star Schema Benchmark
228. TPC-H Benchmark
229. TPC-DS Benchmark
230. Upgrading Cluster
231. Elastic Scaling
232. Load Balancing
233. Time Zone
234. FQDN
235. Overview | Workload Management
236. Resource Group
237. Workload Group
238. Workload Analysis Diagnosis
239. Concurrency Control and Queuing
240. Query Circuit Breaker
241. Kill Query
242. Job Scheduler | Workload Management
243. Disaster Recovery Management Overview
244. Backup and Restore Overview
245. Backup
246. Restore
247. Overview | CCR
248. Quick Start | CCR

249. Operation Manual
250. Feature Details
251. Configuration Instructions
252. Performance
253. Recover from Recycle Bin
254. FE Log Management
255. BE Log Management
256. Monitor Metrics
257. Monitoring and alarming
258. Disk Capacity Management
259. Data Replica Management
260. Automated Service Startup
261. Config Dir
262. FE Configuration | Config
263. BE Configuration
264. User Property
265. Overview | System Tables
266. active_queries
267. backend_active_tasks
268. catalog_meta_cache_statistics
269. character_sets
270. collations
271. column_privileges
272. Column Statistics | Information Schema
273. columns
274. engines
275. events
276. file_cache_statistics
277. files
278. global_variables
279. key_column_usage
280. metadata_name_ids
281. parameters
282. partitions
283. processlist
284. profiling
285. referential_constraints
286. routines
287. rowsets
288. schema_privileges
289. schemata
290. session_variables

291. statistics
292. table_constraints
293. table_options
294. table_privileges
295. table_properties
296. tables
297. triggers
298. user_privileges
299. views
300. workload_group_privileges
301. workload_group_resource_usage
302. workload_groups
303. workload_policy
304. procs_priv
305. user
306. audit_log
307. Column Statistics | Internal Schema
308. partition_statistics
309. Overview | Memory Management
310. Memory Issue FAQ
311. Jemalloc Memory Analysis
312. Global Memory Analysis
313. Cache Memory Analysis
314. Metadata Memory Analysis
315. Query Memory Analysis
316. Load Memory Analysis
317. Query error Process Memory Not Enough
318. Query error Memory Tracker Limit Exceeded
319. OOM Killer Crash Analysis
320. Memory Log Analysis
321. Heap Profile Memory Analysis
322. Memory Tracker
323. Memory Control Strategy
324. Compaction
325. Metadata Operations and Maintenance
326. FE Lock Management
327. Tablet Local Debug
328. Tablet metadata management tool
329. Repairing Data
330. Overview | Open Api
331. Config Action
332. HA Action

333. Hardware Info Action
334. Help Action
335. Log Action
336. Login Action
337. Logout Action
338. Query Profile Action | Fe Http
339. Session Action
340. System Action
341. Colocate Meta Action
342. Meta Action
343. Cluster Action
344. Node Action
345. Query Profile Action | Fe Http
346. Backends Action
347. Bootstrap Action
348. Cancel Load Action
349. Check Decommission Action
350. Check Storage Type Action
351. Connection Action
352. Extra Basepath Action
353. Fe Version Info Action
354. Get DDL Statement Action
355. Get Load Info Action
356. Get Load State
357. Get FE log file
358. Get Small File Action
359. Get WAL size
360. Health Action
361. Meta Info Action | Fe Http
362. Meta Replay State Action
363. Metrics Action
364. Profile Action
365. Query Schema Action
366. Query Stats Action
367. Row Count Action
368. Set Config Action
369. Show Data Action
370. Show Meta Info Action
371. Show Proc Action
372. Show Runtime Info Action
373. Show Table Data Action
374. Statement Execution Action

375. Table Query Plan Action
376. Table Row Count Action
377. Table Schema Action
378. Upload Action
379. Import Action
380. Meta Info Action | Fe Http
381. Debug Point
382. Statistic Action
383. Check Stub Cache
384. Reset Stub Cache
385. View Compaction Status
386. Manually Trigger Compaction
387. View Tablet Compaction Score
388. View Meta
389. Make Snapshot
390. Check All Tablet Segment Lost
391. Config of BE
392. Metrics
393. View Tablet Distribution
394. Migration Tablet
395. View Tablet Info
396. Checksum
397. Download Log about Load Error
398. Pad Rowset
399. Be Version Info
400. Check Alive
401. Reload Tablet
402. Restore Tablet
403. Modify BE VLOG
404. Spark Doris Connector
405. Flink Doris Connector
406. Doris Kafka Connector
407. Doris Kubernetes Operator
408. Recommendations on Alibaba Cloud
409. Recommendations on AWS
410. Doris Streamloader
411. Apache Superset
412. FineBI
413. Metabase
414. Power BI
415. Tableau
416. QuickSight

417. Quick BI
418. Smartbi
419. CloudDM
420. DBeaver
421. DataGrip
422. BladePipe
423. DataX Doriswriter
424. DBT Doris Adapter
425. Seatunnel Doris Sink
426. Kettle Doris Plugin
427. Kyuubi
428. AutoMQ Load
429. Hive Bitmap UDF
430. Install Error
431. Data Operation Error
432. SQL Error
433. Data Lakehouse FAQ
434. BI FAQ
435. Data Integrity FAQ
436. Load FAQ
437. Overview | Sql Data Types
438. BOOLEAN
439. TINYINT
440. SMALLINT
441. INT
442. BIGINT
443. LARGEINT
444. FLOAT
445. DOUBLE
446. DECIMAL
447. DATE | Date Time
448. TIME
449. DATETIME
450. CHAR | String Type
451. VARCHAR
452. STRING
453. ARRAY | Semi Structured
454. MAP | Semi Structured
455. STRUCT | Semi Structured
456. JSON | Semi Structured
457. VARIANT
458. HLL (HyperLogLog)

459. BITMAP
460. QUANTILE_STATE
461. AGG_STATE
462. IPV4
463. IPV6
464. Numeric Type Literal
465. String Type Literal
466. Date Type Literal
467. NULL
468. Object Identifier
469. Reserved Keywords
470. Variables
471. Comments
472. Arithmetic Operators
473. Logic Operators
474. Comparison Operators
475. Boolean Testing Operators
476. Pattern Matching Operators
477. Full Text Search Operators
478. In Operators
479. Exists Operators
480. Bitwise Operators
481. Assignment Operators
482. Operator Precedence
483. ABS
484. ASIN
485. ACOS
486. ATAN
487. ATAN2
488. BIN
489. SIN
490. CBRT
491. COS
492. COSH
493. CONV
494. DEGREES
495. E
496. PI
497. EXP
498. LOG
499. LOG2
500. LOG10

501. LN
502. CEIL
503. FLOOR
504. FMOD
505. MONEY_FORMAT
506. TAN
507. TANH
508. PMOD
509. ROUND
510. ROUND_BANKERS
511. TRUNCATE
512. SQRT
513. POW
514. RADIANS
515. SIGN
516. POSITIVE
517. NEGATIVE
518. RANDOM
519. MOD
520. NORMAL_CDF
521. UUID_NUMERIC
522. WIDTH_BUCKET
523. APPEND_TRAILING_CHAR_IF_ABSENT
524. ASCII
525. AUTO_PARTITION_NAME
526. CHAR | String Functions
527. CHAR_LENGTH
528. CONCAT
529. CONCAT_WS
530. COUNT_SUBSTRINGS
531. CUT_TO_FIRST_SIGNIFICANT_SUBDOMAIN
532. DIGITAL_MASKING
533. DOMAIN
534. DOMAIN_WITHOUT_WWW
535. ELT
536. ENDS_WITH
537. EXTRACT_URL_PARAMETER
538. FIND_IN_SET
539. FIRST_SIGNIFICANT_SUBDOMAIN
540. FROM_BASE64
541. HEX
542. INITCAP

543. INSTR
544. INT_TO_UUID
545. LCASE/LOWER
546. LENGTH
547. LOCATE
548. LPAD
549. LTRIM
550. LTRIM_IN
551. MASK
552. MASK_FIRST_N
553. MASK_LAST_N
554. MULTI_MATCH_ANY
555. MULTI_SEARCH_ALL_POSITIONS
556. NGRAM_SEARCH
557. PARSE_URL
558. PROTOCOL
559. QUOTE
560. RANDOM_BYTES
561. REGEXP
562. REGEXP_EXTRACT
563. REGEXP_EXTRACT_ALL
564. REGEXP_EXTRACT_OR_NULL
565. REGEXP_REPLACE
566. REGEXP_REPLACE_ONE
567. REPEAT
568. REPLACE
569. REPLACE_EMPTY
570. REVERSE
571. RPAD
572. RTRIM
573. RTRIM_IN
574. SPACE
575. SPLIT_BY_STRING
576. SPLIT_PART
577. STARTS_WITH
578. STRCMP
579. STRLEFT
580. STRRIGHT
581. SUB_REPLACE
582. SUBSTRING
583. SUBSTRING_INDEX
584. TO_BASE64

585. TOKENIZE
586. TOP_LEVEL_DOMAIN
587. TRANSLATE
588. TRIM
589. TRIM_IN
590. UCASE/UPPER
591. UNHEX
592. URL_DECODE
593. URL_ENCODE
594. UUID
595. CONVERT_TZ
596. CURDATE,CURRENT_DATE
597. CURTIME,CURRENT_TIME
598. DATE | Date Time Functions
599. DAYS_ADD
600. DATE_CEIL
601. DATE_FLOOR
602. DATE_FORMAT
603. DAYS_SUB
604. DATE_TRUNC
605. DATEDIFF
606. DAY
607. DAY_CEIL
608. DAY_FLOOR
609. DAYNAME
610. DAYOFWEEK
611. DAYOFYEAR
612. EXTRACT
613. FROM_DAYS
614. FROM_ISO8601_DATE | Date Time Functions
615. FROM_MICROSECOND
616. FROM_MILLISECOND
617. FROM_SECOND
618. FROM_UNIXTIME
619. HOUR
620. HOUR_CEIL
621. HOUR_FLOOR
622. HOURS_ADD
623. HOURS_DIFF
624. HOURS_SUB
625. LAST_DAY
626. MAKEDATE

627. MICROSECOND
628. MICROSECOND_TIMESTAMP
629. MICROSECONDS_ADD
630. MICROSECONDS_DIFF
631. MICROSECONDS_SUB
632. MILLISECOND_TIMESTAMP
633. MILLISECONDS_ADD
634. MILLISECONDS_DIFF
635. MILLISECONDS_SUB
636. MINUTE
637. MINUTE_CEIL
638. MINUTE_FLOOR
639. MINUTES_ADD
640. MINUTES_DIFF
641. MINUTES_SUB
642. MONTH
643. MONTH_CEIL
644. MONTH_FLOOR
645. MONTHNAME
646. MONTHS_ADD
647. MONTHS_DIFF
648. MONTHS_SUB
649. NOW
650. QUARTER
651. SEC_TO_TIME
652. SECOND
653. SECOND_CEIL
654. SECOND_FLOOR
655. SECONDS_ADD
656. SECONDS_DIFF
657. SECONDS_SUB
658. STR_TO_DATE
659. TIME_TO_SEC
660. TIMEDIFF
661. TIMESTAMP
662. TIMESTAMPADD
663. TIMESTAMPDIFF
664. TO_DATE
665. TO_DAYS
666. TO_ISO8601 | Date Time Functions
667. TO_MONDAY
668. UNIX_TIMESTAMP

669. UTC_TIMESTAMP
670. WEEK
671. WEEK_CEIL | Date Time Functions
672. WEEK_FLOOR | Date Time Functions
673. WEEKDAY
674. WEEKOFYEAR
675. WEEKS_ADD
676. WEEKS_DIFF
677. WEEKS_SUB
678. YEAR
679. YEAR_CEIL
680. YEAR_FLOOR
681. YEARWEEK
682. YEARS_ADD
683. YEARS_DIFF
684. YEARS_SUB
685. ST_ANGLE
686. ST_ANGLE_SPHERE
687. ST_AREA_SQUARE_KM
688. ST_AREA_SQUARE_METERS
689. ST_ASBINARY
690. ST_ASTEXT
691. ST_AZIMUTH
692. ST_CIRCLE
693. ST_CONTAINS
694. ST_DISJOINT
695. ST_DISTANCE_SPHERE
696. ST_GEOMETRYFROMTEXT
697. ST_GEOMETRYFROMWKB
698. ST_INTERSECTS
699. ST_LINEFROMTEXT
700. ST_POINT
701. ST_POLYGON
702. ST_TOUCHES
703. ST_X
704. ST_Y
705. AES_DECRYPT
706. AES_ENCRYPT
707. CRC32
708. MD5
709. MD5SUM
710. MURMUR_HASH3_32

711. MURMUR_HASH3_64
712. SHA1
713. SHA2
714. SM3
715. SM3SUM
716. SM4_DECRYPT
717. SM4_ENCRYPT
718. XXHASH_32
719. XXHASH_64
720. COSINE_DISTANCE
721. INNER_PRODUCT
722. L1_DISTANCE
723. L2_DISTANCE
724. BIT_LENGTH
725. BITAND
726. BIT_COUNT
727. BITNOT
728. BITOR
729. BIT_SHIFT_LEFT
730. BIT_SHIFT_RIGHT
731. ARRAY | Array Functions
732. ARRAY_APPLY
733. ARRAY_AVG
734. ARRAY_COMPACT
735. ARRAY_CONCAT
736. ARRAY_CONTAINS
737. ARRAY_COUNT
738. ARRAY_CUM_SUM
739. ARRAY_DIFFERENCE
740. ARRAY_DISTINCT
741. ARRAY_ENUMERATE
742. ARRAY_ENUMERATE_UNIQ
743. ARRAY_EXCEPT
744. ARRAY_EXISTS
745. ARRAY_FILTER
746. ARRAY_FIRST
747. ARRAY_FIRST_INDEX
748. ARRAY_INTERSECT
749. ARRAY_JOIN
750. ARRAY_LAST
751. ARRAY_LAST_INDEX
752. ARRAY_MAP

753. ARRAY_MAX
754. ARRAY_MIN
755. ARRAY_POPBACK
756. ARRAY_POPFRONT
757. ARRAY_POSITION
758. ARRAY_PRODUCT
759. ARRAY_PUSHBACK
760. ARRAY_PUSHFRONT
761. ARRAY_RANGE
762. ARRAY_REMOVE
763. ARRAY_REPEAT
764. ARRAY_REVERSE_SORT
765. ARRAY_SHUFFLE
766. ARRAY_SIZE
767. ARRAY_SLICE
768. ARRAY_SORT
769. ARRAY_SORTBY
770. ARRAY_SUM
771. ARRAY_UNION
772. ARRAY_WITH_CONSTANT
773. ARRAY_ZIP
774. ARRAYS_OVERLAP
775. COUNTEQUAL
776. MAP | Map Functions
777. MAP_CONTAINS_KEY
778. MAP_CONTAINS_VALUE
779. MAP_KEYS
780. MAP_SIZE
781. MAP_VALUES
782. NAMED_STRUCT
783. STRUCT | Struct Functions
784. STRUCT_ELEMENT
785. GET_JSON_BIGINT
786. GET_JSON_DOUBLE
787. GET_JSON_INT
788. GET_JSON_STRING
789. JSON_ARRAY
790. JSON_CONTAINS
791. JSON_EXISTS_PATH
792. JSON_EXTRACT
793. JSON_EXTRACT_BIGINT
794. JSON_EXTRACT_BOOL

795. JSON_EXTRACT_DOUBLE
796. JSON_EXTRACT_INT
797. JSON_EXTRACT_ISNULL
798. JSON_EXTRACT_LARGEINT
799. JSON_EXTRACT_STRING
800. JSON_INSERT
801. JSON_LENGTH
802. JSON_OBJECT
803. JSON_PARSE
804. JSON_PARSE_ERROR_TO_INVALID
805. JSON_PARSE_ERROR_TO_NULL
806. JSON_PARSE_ERROR_TO_VALUE
807. JSON_PARSE_NOTNULL
808. JSON_PARSE_NOTNULL_ERROR_TO_INVALID
809. JSON_PARSE_NOTNULL_ERROR_TO_VALUE
810. JSON_PARSE_NULLABLE
811. JSON_PARSE_NULLABLE_ERROR_TO_INVALID
812. JSON_PARSE_NULLABLE_ERROR_TO_NULL
813. JSON_PARSE_NULLABLE_ERROR_TO_VALUE
814. JSON_QUOTE
815. JSON_REPLACE
816. JSON_SEARCH
817. JSON_SET
818. JSON_TYPE
819. JSON_UNQUOTE
820. JSON_VALID
821. IPV4_CIDR_TO_RANGE
822. IPV4_NUM_TO_STRING
823. IPV4_STRING_TO_NUM
824. IPV4_STRING_TO_NUM_OR_DEFAULT
825. IPV4_STRING_TO_NUM_OR_NULL
826. IPV6_CIDR_TO_RANGE
827. IPV6_NUM_TO_STRING
828. IPV6_STRING_TO_NUM
829. IPV6_STRING_TO_NUM_OR_DEFAULT
830. IPV6_STRING_TO_NUM_OR_NULL
831. IS_IP_ADDRESS_IN_RANGE
832. IS_IPV4_COMPAT
833. IS_IPV4_MAPPED
834. IS_IPV4_STRING
835. IS_IPV6_STRING
836. TO_IPV4

837. TO_IPV4_OR_DEFAULT
838. TO_IPV4_OR_NULL
839. TO_IPV6
840. TO_IPV6_OR_DEFAULT
841. TO_IPV6_OR_NULL
842. BITMAP_AND
843. BITMAP_AND_COUNT
844. BITMAP_AND_NOT,BITMAP_ANDNOT
845. BITMAP_AND_NOT_COUNT,BITMAP_ANDNOT_COUNT
846. BITMAP_CONTAINS
847. BITMAP_COUNT
848. BITMAP_EMPTY
849. BITMAP_FROM_ARRAY
850. BITMAP_FROM_BASE64
851. BITMAP_FROM_STRING
852. BITMAP_HAS_ALL
853. BITMAP_HAS_ANY
854. BITMAP_HASH
855. BITMAP_HASH64
856. BITMAP_MAX
857. BITMAP_MIN
858. BITMAP_NOT
859. BITMAP_OR
860. BITMAP_OR_COUNT
861. BITMAP_REMOVE
862. BITMAP_SUBSET_IN_RANGE
863. BITMAP_SUBSET_LIMIT
864. BITMAP_TO_ARRAY
865. BITMAP_TO_BASE64
866. BITMAP_TO_STRING
867. BITMAP_XOR
868. BITMAP_XOR_COUNT
869. SUB_BITMAP
870. TO_BITMAP
871. HLL_CARDINALITY
872. HLL_EMPTY
873. HLL_FROM_BASE64
874. HLL_HASH
875. HLL_TO_BASE64
876. CONNECTION_ID
877. CURRENT_CATALOG
878. CURRENT_USER

879. DATABASE
880. LAST_QUERY_ID
881. SESSION_USER
882. USER
883. VERSION
884. CONVERT_TO
885. ESQUERY
886. FIELD
887. G
888. QUANTILE_PERCENT
889. QUANTILE_STATE_EMPTY
890. TO_QUANTILE_STATE
891. COALESCE
892. GREATEST
893. IF
894. IFNULL
895. LEAST
896. NOT_NULL_OR_EMPTY
897. NULL_OR_EMPTY
898. NULLIF
899. ANY_VALUE
900. APPROX_COUNT_DISTINCT
901. ARRAY_AGG
902. AVG
903. AVG_WEIGHTED
904. BITMAP_AGG
905. BITMAP_INTERSECT
906. BITMAP_UNION
907. BITMAP_UNION_COUNT
908. BITMAP-UNION-INT
909. COLLECT_LIST
910. COLLECT_SET
911. CORR
912. COUNT
913. COUNT_BY_ENUM
914. COVAR,COVAR_POP
915. COVAR_SAMP
916. GROUP_ARRAY_INTERSECT
917. GROUP_BIT_AND
918. GROUP_BIT_OR
919. GROUP_BIT_XOR
920. GROUP_BITMAP_XOR

921. GROUP_CONCAT
922. GROUPING
923. GROUPING_ID
924. HISTOGRAM
925. HLL_RAW_AGG
926. HLL_UNION_AGG
927. INTERSECT_COUNT
928. MAP_AGG
929. MAX
930. MAX_BY
931. MIN
932. MIN_BY
933. PERCENTILE
934. PERCENTILE_APPROX
935. PERCENTILE_ARRAY
936. QUANTILE_UNION
937. RETENTION
938. SEQUENCE_COUNT
939. SEQUENCE_MATCH
940. STDDEV,STDDEV_POP
941. STDDEV_SAMP
942. SUM
943. SUM0
944. TOPN
945. TOPN_ARRAY
946. TOPN_WEIGHTED
947. VAR_SAMP,VARIANCE_SAMP
948. VARIANCE,VAR_POP,VARIANCE_POP
949. WINDOW_FUNNEL
950. FOREACH
951. MERGE
952. STATE
953. UNION
954. CUME_DIST
955. DENSE_RANK
956. FIRST_VALUE
957. LAG
958. LAST_VALUE
959. LEAD
960. NTILE
961. OVERVIEW
962. PERCENT_RANK

963. RANK
964. ROW_NUMBER
965. EXPLODE
966. EXPLODE_BITMAP
967. EXPLODE_JSON_ARRAY_DOUBLE
968. EXPLODE_JSON_ARRAY_INT
969. EXPLODE_JSON_ARRAY_JSON
970. EXPLODE_JSON_ARRAY_STRING
971. EXPLODE_JSON_OBJECT
972. EXPLODE_MAP
973. EXPLODE_NUMBERS
974. EXPLODE_SPLIT
975. BACKENDS
976. CATALOGS
977. FRONTENDS
978. FRONTENDS_DISKS
979. HDFS | Table Valued Functions
980. ICEBERG_META
981. JOBS
982. LOCAL
983. MV_INFOS
984. NUMBERS
985. PARTITION_VALUES
986. PARTITIONS
987. QUERY
988. S3 | Table Valued Functions
989. TASKS
990. SELECT
991. EXPLAIN
992. INSERT
993. INSERT OVERWRITE
994. SHOW LAST INSERT
995. UPDATE
996. DELETE
997. SHOW DELETE
998. BROKER LOAD
999. MYSQL LOAD
1000. SHOW LOAD
1001. SHOW STREAM LOAD
1002. SHOW CREATE LOAD
1003. CANCEL LOAD
1004. SHOW LOAD WARNINGS

1005. CREATE ROUTINE LOAD
1006. ALTER ROUTINE LOAD
1007. PAUSE ROUTINE LOAD
1008. RESUME ROUTINE LOAD
1009. STOP ROUTINE LOAD
1010. SHOW ROUTINE LOAD
1011. SHOW ROUTINE LOAD TASK
1012. SHOW CREATE ROUTINE LOAD
1013. CREATE SYNC JOB
1014. PAUSE SYNC JOB
1015. RESUME SYNC JOB
1016. STOP SYNC JOB
1017. SHOW SYNC JOB
1018. SYNC
1019. EXPORT | Load And Export
1020. CANCEL EXPORT
1021. SHOW EXPORT
1022. OUTFILE
1023. CLEAN LABEL
1024. CREATE REPOSITORY
1025. DROP REPOSITORY
1026. SHOW CREATE REPOSITORY
1027. SHOW REPOSITORIES
1028. BACKUP
1029. CANCEL BACKUP
1030. RESTORE
1031. SHOW RESTORE
1032. CANCEL RESTORE
1033. SHOW SNAPSHOT
1034. CREATE USER
1035. ALTER USER
1036. DROP USER
1037. SET PASSWORD
1038. CREATE ROLE
1039. ALTER ROLE
1040. DROP ROLE
1041. SHOW ROLES
1042. GRANT TO
1043. REVOKE FROM
1044. SHOW PRIVILEGES
1045. SHOW GRANTS
1046. REFRESH LDAP

1047. SET LDAP_ADMIN_PASSWORD
1048. SET PROPERTY
1049. SHOW PROPERTY
1050. SWITCH CATALOG
1051. USE DATABASE
1052. SET VARIABLE
1053. UNSET VARIABLE
1054. SHOW VARIABLES
1055. SHOW PROCESSLIST
1056. KILL QUERY
1057. CLEAN ALL PROFILE
1058. SHOW QUERY STATS
1059. CLEAN ALL QUERY STATS
1060. KILL CONNECTION
1061. BEGIN
1062. COMMIT
1063. ROLLBACK
1064. SHOW TRANSACTION
1065. CREATE CATALOG
1066. ALTER CATALOG
1067. DROP CATALOG
1068. SHOW CATALOG
1069. SHOW CATALOGS
1070. SHOW CREATE CATALOG
1071. REFRESH
1072. CREATE DATABASE
1073. ALTER DATABASE
1074. DROP DATABASE
1075. SHOW CREATE DATABASE
1076. SHOW DATABASES
1077. SHOW-DATABASE-ID
1078. CREATE TABLE
1079. DESC TABLE
1080. ALTER TABLE COLUMN
1081. ALTER TABLE PARTITION
1082. ALTER TABLE ROLLUP
1083. ALTER TABLE RENAME
1084. ALTER TABLE REPLACE
1085. ALTER TABLE PROPERTY
1086. ALTER TABLE COMMENT
1087. CANCEL ALTER TABLE
1088. SHOW ALTER

1089. TRUNCATE TABLE
1090. DROP TABLE
1091. SHOW CREATE TABLE
1092. SHOW TABLES
1093. SHOW TABLE ID
1094. SHOW TABLE STATUS
1095. SHOW CONVERT LIGHT SCHEMA CHANGE PROCESS
1096. SHOW PARTITION
1097. SHOW PARTITION ID
1098. SHOW PARTITIONS
1099. SHOW DYNAMIC PARTITION
1100. SHOW COLUMNS
1101. ALTER COLOCATE GROUP
1102. CREATE INDEX
1103. DROP INDEX
1104. SHOW INDEX
1105. BUILD INDEX
1106. CANCEL BUILD INDEX
1107. SHOW BUILD INDEX
1108. CREATE VIEW
1109. ALTER VIEW
1110. DROP VIEW
1111. SHOW CREATE VIEW
1112. SHOW VIEW
1113. CREATE SYNC MATERIALIZED VIEW
1114. DROP MATERIALIZED VIEW
1115. SHOW ALTER TABLE MATERIALIZED VIEW
1116. SHOW CREATE SYNC MATERIALIZED VIEW
1117. ALTER ASYNC MATERIALIZED VIEW
1118. CANCEL MATERIALIZED VIEW TASK
1119. CREATE ASYNC MATERIALIZED VIEW
1120. PAUSE MATERIALIZED VIEW
1121. DROP ASYNC MATERIALIZED VIEW
1122. REFRESH MATERIALIZED VIEW
1123. RESUME MATERIALIZED VIEW
1124. SHOW CREATE ASYNC MATERIALIZED VIEW
1125. REBALANCE DISK
1126. CANCEL REBALANCE DISK
1127. SHOW DATA
1128. SHOW DATA SKEW
1129. COMPACT TABLE
1130. REPAIR TABLE

1131. CANCEL REPAIR TABLE
1132. SET TABLE STATUS
1133. SET TABLE PARTITION VERSION
1134. DIAGNOSE TABLET
1135. ADMIN COPY TABLET
1136. ADMIN CHECK TABLET
1137. SHOW TABLET
1138. SHOW TABLETS BELONG
1139. SHOW TABLET STORAGE FORMAT
1140. SHOW TABLET DIAGNOSIS
1141. ADMIN SET REPLICA STATUS
1142. ADMIN SET REPLICA VERSION
1143. SHOW REPLICA STATUS
1144. SHOW REPLICA DISTRIBUTION
1145. ADMIN CLEAN TRASH
1146. SHOW TRASH
1147. SHOW CATALOG RECYCLE BIN
1148. RECOVER
1149. CREATE FUNCTION
1150. DROP FUNCTION
1151. SHOW CREATE FUNCTION
1152. DESC FUNCTION
1153. SHOW FUNCTIONS
1154. ANALYZE
1155. ALTER STATS
1156. DROP STATS
1157. SHOW TABLE STATS
1158. SHOW COLUMN STATS
1159. DROP ANALYZE JOB
1160. KILL ANALYZE JOB
1161. SHOW ANALYZE
1162. ADD FOLLOWER
1163. DROP FOLLOWER
1164. ADD OBSERVER
1165. DROP OBSERVER
1166. SET FRONTEND CONFIG
1167. SHOW FRONTEND CONFIG
1168. MODIFY FRONTEND HOSTNAME
1169. SHOW FRONTENDS
1170. SHOW FRONTENDS DISKS
1171. ADD BACKEND
1172. DROP BACKEND

1173. MODIFY BACKEND
1174. SHOW BACKENDS
1175. DECOMMISSION BACKEND
1176. CANCEL DECOMMISSION BACKEND
1177. ADD BROKER
1178. DROP BROKER
1179. SHOW BROKER
1180. CREATE RESOURCE
1181. ALTER RESOURCE
1182. DROP RESOURCE
1183. SHOW RESOURCES
1184. CREATE WORKLOAD GROUP
1185. ALTER WORKLOAD GROUP
1186. DROP WORKLOAD GROUP
1187. SHOW WORKLOAD GROUPS
1188. CREATE WORKLOAD POLICY
1189. ALTER WORKLOAD POLICY
1190. DROP WORKLOAD POLICY
1191. CREATE STORAGE POLICY
1192. ALTER STORAGE POLICY
1193. DROP STORAGE POLICY
1194. SHOW STORAGE POLICY
1195. CREATE FILE
1196. DROP FILE
1197. SHOW FILE
1198. CREATE ENCRYPTKEY
1199. DROP ENCRYPTKEY
1200. SHOW ENCRYPTKEY
1201. CREATE SQL_BLOCK_RULE
1202. ALTER SQL_BLOCK_RULE
1203. DROP SQL_BLOCK_RULE
1204. SHOW SQL_BLOCK_RULE
1205. CREATE ROW POLICY
1206. DROP ROW POLICY
1207. SHOW ROW POLICY
1208. CREATE JOB
1209. PAUSE JOB
1210. DROP JOB
1211. RESUME JOB
1212. CANCEL TASK
1213. INSTALL PLUGIN
1214. UNINSTALL PLUGIN

1215. SHOW PLUGINS
1216. SHOW COLLATION
1217. SHOW CHARSET
1218. SHOW DATA TYPES
1219. SHOW TYPECAST
1220. SHOW PROC
1221. LOCALTIME,LOCALTIMESTAMP

Introduction to Apache Doris

What’s Apache Doris

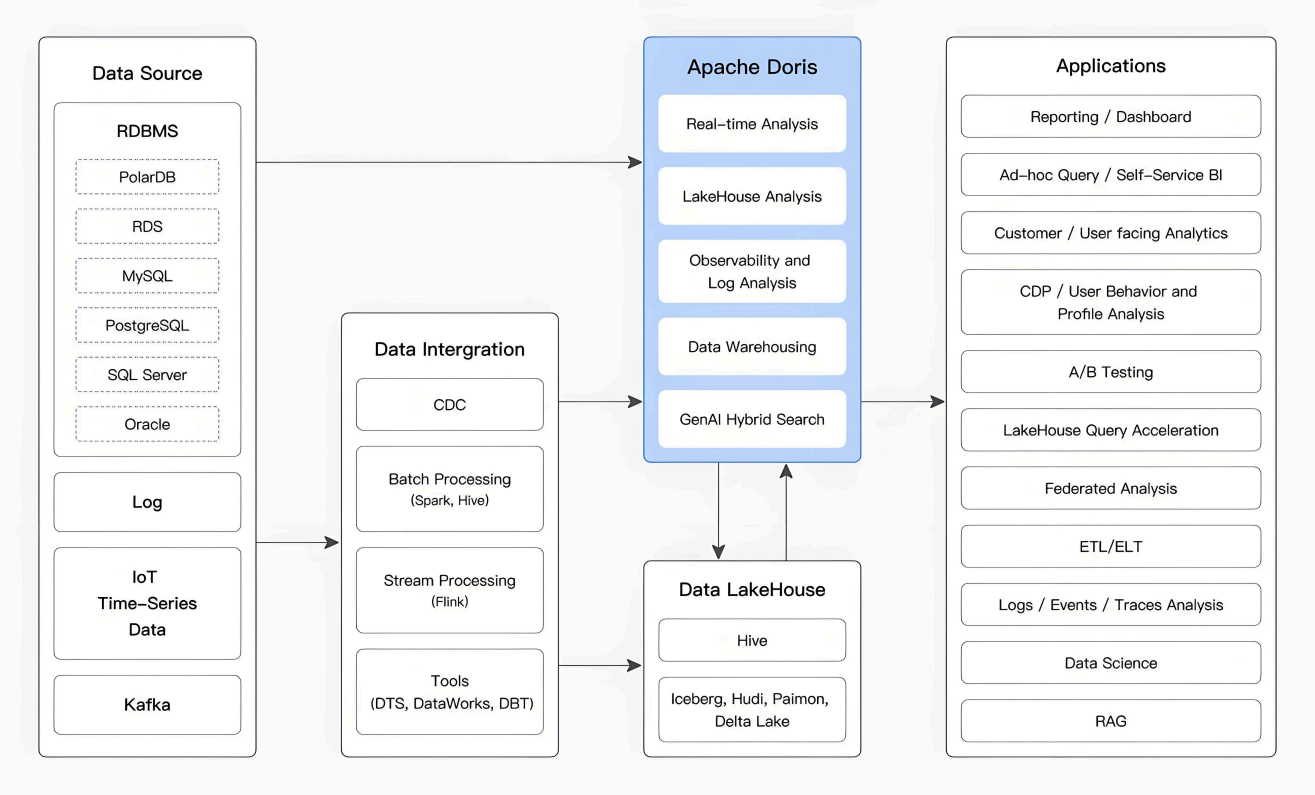
Apache Doris is an MPP-based real-time data warehouse known for its high query speed. For queries on large datasets, it returns results in sub-seconds. It supports both high-concurrency point queries and high-throughput complex analysis. It can be used for report analysis, ad-hoc queries, unified data warehouse, and data lake query acceleration. Based on Apache Doris, users can build applications for user behavior analysis, A/B testing platform, log analysis, user profile analysis, and e-commerce order analysis.

Apache Doris, formerly known as Palo, was initially created to support Baidu’s ad reporting business. It was officially open-sourced in 2017 and donated by Baidu to the Apache Software Foundation in July 2018, where it was operated by members of the incubator project management committee under the guidance of Apache mentors. In June 2022, Apache Doris graduated from the Apache incubator as a Top-Level Project. By 2024, the Apache Doris community has gathered more than 600 contributors from hundreds of companies in different industries, with over 120 monthly active contributors.

Apache Doris has a wide user base. It has been used in production environments of over 5000 companies worldwide, including giants such as TikTok, Baidu, Tencent, and NetEase. It is also widely used across industries from finance, retailing, and telecommunications to energy, manufacturing, medical care, etc.

Usage Scenarios

As shown in the figure below, after various data integrations and processing, data sources are typically ingested into the real-time data warehouse Doris and offline lakehouses (such as Hive, Iceberg, and Hudi). These are widely used in OLAP analysis scenarios.



Apache Doris Usage Scenarios

Apache Doris is widely used in the following scenarios:

- **Real-time Data Analysis**:
  - **Real-time Reporting and Decision-making**: Doris provides real-time updated reports and dashboards for both internal and external enterprise use, supporting real-time decision-making in automated processes.
  - **Ad Hoc Analysis**: Doris offers multidimensional data analysis capabilities, enabling rapid business intelligence analysis and ad hoc queries to help users quickly uncover insights from complex data.
  - **User Profiling and Behavior Analysis**: Doris can analyze user behaviors such as participation, retention, and conversion, while also supporting scenarios like population insights and crowd selection for behavior analysis.
- **Lakehouse Analytics**:
  - **Lakehouse Query Acceleration**: Doris accelerates lakehouse data queries with its efficient query engine.
  - **Federated Analytics**: Doris supports federated queries across multiple data sources, simplifying architecture and eliminating data silos.
  - **Real-time Data Processing**: Doris combines real-time data streams and batch data processing capabilities to meet the needs of high concurrency and low-latency complex business requirements.
- **SQL-based Observability**:

- **Log and Event Analysis**: Doris enables real-time or batch analysis of logs and events in distributed systems, helping to identify issues and optimize performance.

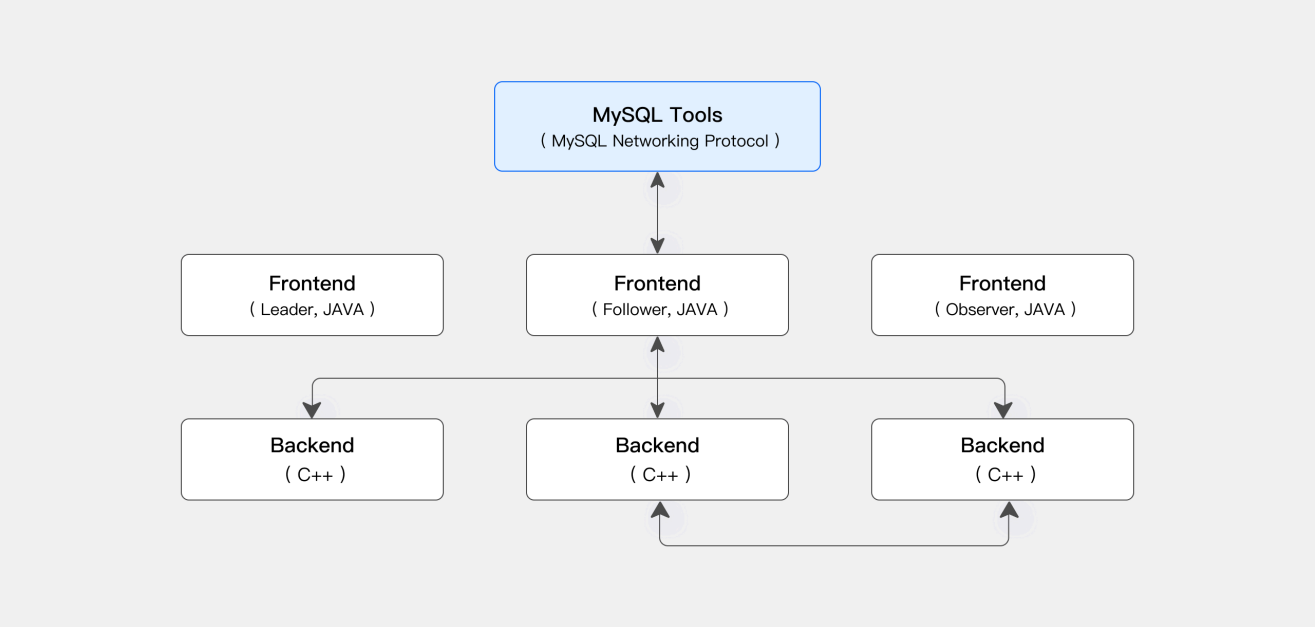
Overall Architecture

Apache Doris uses the MySQL protocol, is highly compatible with MySQL syntax, and supports standard SQL. Users can access Apache Doris through various client tools, and it seamlessly integrates with BI tools.

Storage-Compute Integrated Architecture

The storage-compute integrated architecture of Apache Doris is streamlined and easy to maintain. As shown in the figure below, it consists of only two types of processes:

- **Frontend (FE):** Primarily responsible for handling user requests, query parsing and planning, metadata management, and node management tasks.
- **Backend (BE):** Primarily responsible for data storage and query execution. Data is partitioned into shards and stored with multiple replicas across BE nodes.



MPP architecture

In a production environment, multiple FE nodes can be deployed for disaster recovery. Each FE node maintains a full copy of the metadata. The FE nodes are divided into three roles:

Role	Function
Master	The FE Master node is responsible for metadata read and write operations. When metadata changes occur in the Master, they are synchronized to Follower or Observer nodes via the BDB JE protocol.
Follower	The Follower node is responsible for reading metadata. If the Master node fails, a Follower node can be selected as the new Master.
Observer	The Observer node is responsible for reading metadata and is mainly used to increase query concurrency. It does not participate in cluster leadership elections.

Both FE and BE processes are horizontally scalable, enabling a single cluster to support hundreds of machines and tens of petabytes of storage capacity. The FE and BE processes use a consistency protocol to ensure high availability of services and high reliability of data. The storage-compute integrated architecture is highly integrated, significantly reducing the operational complexity of distributed systems.

Core Features of Apache Doris

- **High Availability**: In Apache Doris, both metadata and data are stored with multiple replicas, synchronizing data logs via the quorum protocol. Data write is considered successful once a majority of replicas have completed the write, ensuring that the cluster remains available even if a few nodes fail. Apache Doris supports both same-city and cross-region disaster recovery, enabling dual-cluster master-slave modes. When some nodes experience failures, the cluster can automatically isolate the faulty nodes, preventing the overall cluster availability from being affected.
- **High Compatibility**: Apache Doris is highly compatible with the MySQL protocol and supports standard SQL syntax, covering most MySQL and Hive functions. This high compatibility allows users to seamlessly migrate and integrate existing applications and tools. Apache Doris supports the MySQL ecosystem, enabling users to connect Doris using MySQL Client tools for more convenient operations and maintenance. It also supports MySQL protocol compatibility for BI reporting tools and data transmission tools, ensuring efficiency and stability in data analysis and data transmission processes.
- **Real-Time Data Warehouse**: Based on Apache Doris, a real-time data warehouse service can be built. Apache Doris offers second-level data ingestion capabilities, capturing incremental changes from upstream online transactional databases into Doris within seconds. Leveraging vectorized engines, MPP architecture, and Pipeline execution engines, Doris provides sub-second data query capabilities, thereby constructing a high-performance, low-latency real-time data warehouse platform.
- **Unified Lakehouse**: Apache Doris can build a unified lakehouse architecture based on external data sources such as data lakes or relational databases. The Doris unified lakehouse solution enables seamless integration and free data flow between data lakes and data warehouses, helping users directly utilize data warehouse capabilities to solve data analysis problems in data lakes while fully leveraging data lake data management capabilities to enhance data value.
- **Flexible Modeling**: Apache Doris offers various modeling approaches, such as wide table models, pre-aggregation models, star/snowflake schemas, etc. During data import, data can be flattened into wide tables and written into Doris through compute engines like Flink or Spark, or data can be directly imported into Doris, performing data modeling operations through views, materialized views, or real-time multi-table joins.

Technical overview

Doris provides an efficient SQL interface and is fully compatible with the MySQL protocol. Its query engine is based on an MPP (Massively Parallel Processing) architecture, capable of efficiently executing complex analytical queries and achieving low-latency real-time queries. Through columnar storage technology for data encoding and compression, it significantly optimizes query performance and storage compression ratio.

Interface

Apache Doris adopts the MySQL protocol, supports standard SQL, and is highly compatible with MySQL syntax. Users can access Apache Doris through various client tools and seamlessly integrate it with BI tools, including but not limited to Smartbi, DataEase, FineBI, Tableau, Power BI, and Apache Superset. Apache Doris can work as the data source for any BI tools that support the MySQL protocol.

Storage engine

Apache Doris has a columnar storage engine, which encodes, compresses, and reads data by column. This enables a very high data compression ratio and largely reduces unnecessary data scanning, thus making more efficient use of IO and CPU resources.

Apache Doris supports various index structures to minimize data scans:

- **Sorted Compound Key Index**: Users can specify three columns at most to form a compound sort key. This can effectively prune data to better support highly concurrent reporting scenarios.
- **Min/Max Index**: This enables effective data filtering in equivalence and range queries of numeric types.
- **BloomFilter Index**: This is very effective in equivalence filtering and pruning of high-cardinality columns.
- **Inverted Index**: This enables fast searching for any field.

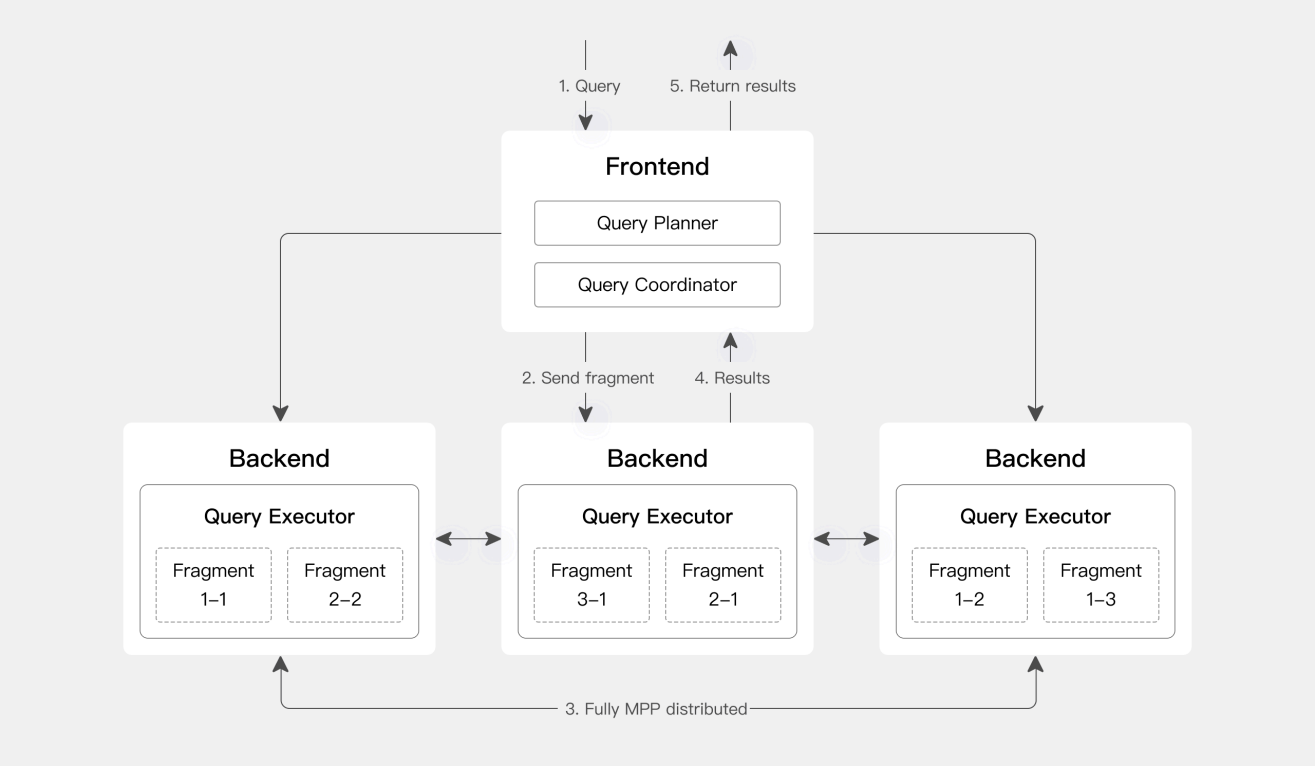
Apache Doris supports a variety of data models and has optimized them for different scenarios:

- **Detail Model (Duplicate Key Model):** A detail data model designed to meet the detailed storage requirements of fact tables.
- **Primary Key Model (Unique Key Model):** Ensures unique keys; data with the same key is overwritten, enabling row-level data updates.
- **Aggregate Model (Aggregate Key Model):** Merges value columns with the same key, significantly improving performance through pre-aggregation.

Apache Doris also supports strongly consistent single-table materialized views and asynchronously refreshed multi-table materialized views. Single-table materialized views are automatically refreshed and maintained by the system, requiring no manual intervention from users. Multi-table materialized views can be refreshed periodically using in-cluster scheduling or external scheduling tools, reducing the complexity of data modeling.

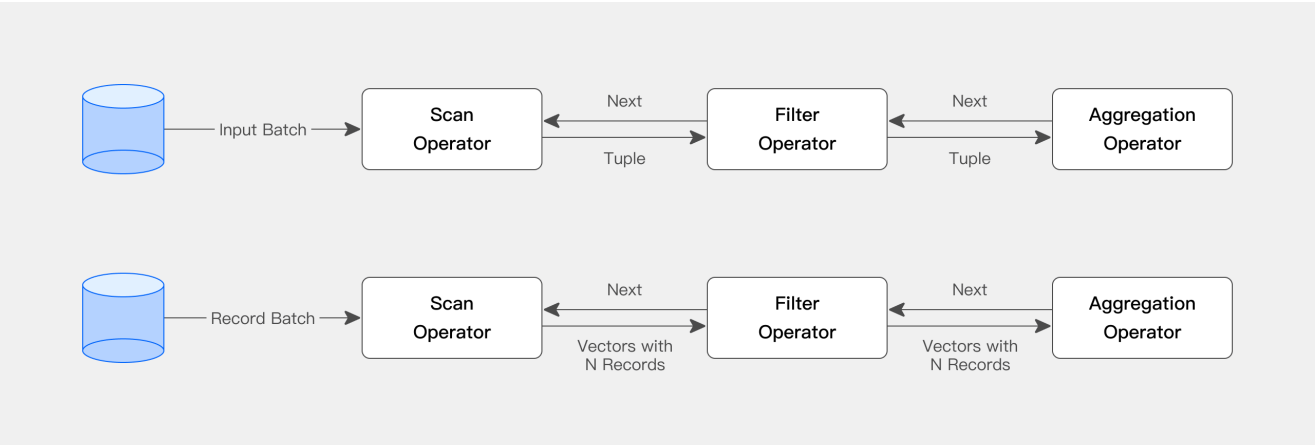
Query engine

Apache Doris has an MPP-based query engine for parallel execution between and within nodes. It supports distributed shuffle join for large tables to better handle complicated queries.



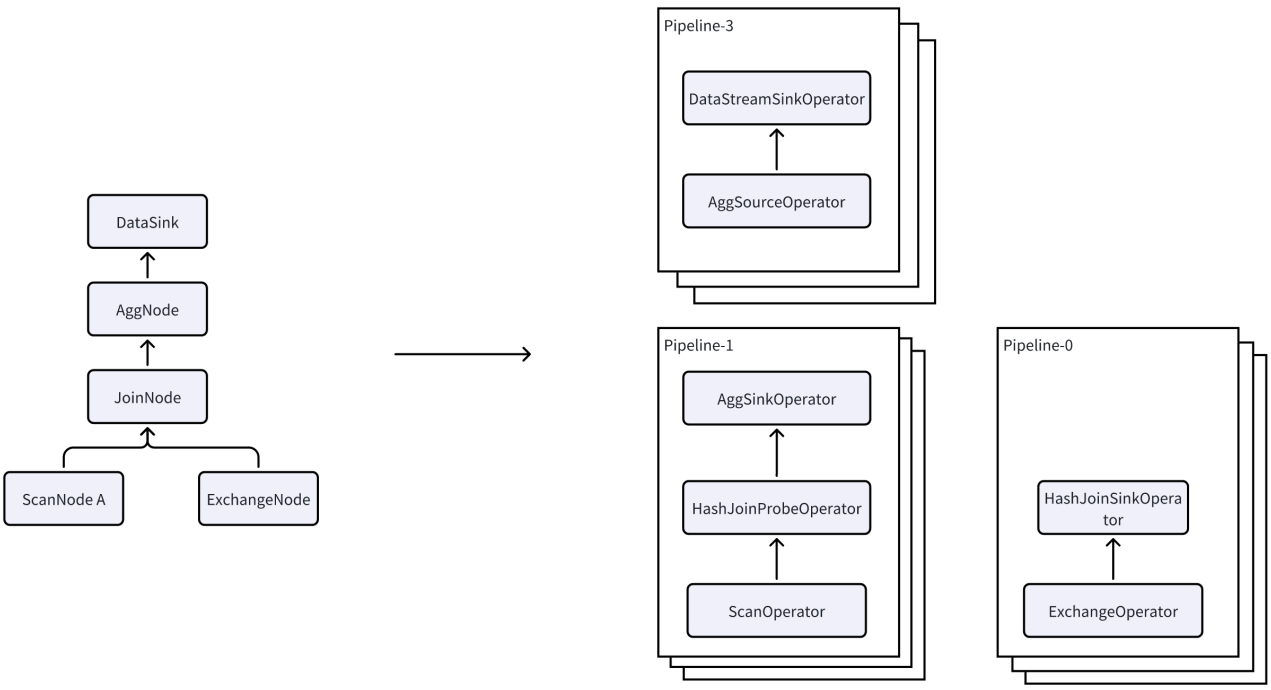
MPP-based Query engine 1

The query engine of Apache Doris is fully vectorized, with all memory structures laid out in a columnar format. This can largely reduce virtual function calls, increase cache hit rates, and make efficient use of SIMD instructions. Apache Doris delivers a 5~10 times higher performance in wide table aggregation scenarios than non-vectorized engines.



MPP-based Query engine 2

Apache Doris uses adaptive query execution technology to dynamically adjust the execution plan based on runtime statistics. For example, it can generate a runtime filter and push it to the probe side. Specifically, it pushes the filters to the lowest-level scan node on the probe side, which largely reduces the data amount to be processed and increases join performance. The runtime filter of Apache Doris supports In/Min/Max/Bloom Filter.



MPP-based pip_exec_3

Apache Doris uses a Pipeline execution engine that breaks down queries into multiple sub-tasks for parallel execution, fully leveraging multi-core CPU capabilities. It simultaneously addresses the thread explosion problem by limiting the number of query threads. The Pipeline execution engine reduces data copying and sharing, optimizes sorting and aggregation operations, thereby significantly improving query efficiency and throughput.

In terms of the optimizer, Apache Doris employs a combined optimization strategy of CBO (Cost-Based Optimizer), RBO (Rule-Based Optimizer), and HBO (History-Based Optimizer). RBO supports constant folding, subquery rewriting, predicate pushdown, and more. CBO supports join reordering and other optimizations. HBO recommends the optimal execution plan based on historical query information. These multiple optimization measures ensure that Doris can enumerate high-performance query plans across various types of queries.

Quick Start | GettingStarted

Quick Start

> **Warning:**
>
> The following rapid deployment methods are intended solely for local development and testing, and should not be used in production environments. The reasons are as follows:
>
> 1. **Data Vulnerability**: Data can be easily lost when using Docker deployment, as data is lost upon container destruction. Manual deployment of single-replica instances lacks data redundancy and backup capabilities, meaning machine failures could result in data loss.
> 2. **Single-Replica Configuration**: The table creation statements in the examples are all single-replica. In a production environment, multi-replica storage should be used to ensure data reliability.

Use Docker for Quick Deployment

Starting from Doris version 2.1.8, Docker can be used for rapid deployment.

Step 1: Download the Quick-Start script

Download the script , run the following command to grant it the corresponding execution permissions.

```
chmod 755 start-doris.sh
```

Step 2: Start the cluster

Run the script to start the cluster, using the `2.1.9` version by default

```
bash start-doris.sh
```

You can specify the startup version through the -v parameter, such as:

```
bash start-doris.sh -v 2.1.8
```

Step 3: Connect to the cluster using MySQL client and check the cluster status

```
## Check the FE status to ensure that both the Join and Alive columns are true.
mysql -uroot -P9030 -h127.0.0.1 -e 'SELECT `host`, `join`, `alive` FROM frontends()'
+-----------+------+-------+
| host      | join | alive |
+-----------+------+-------+
```

```
| 127.0.0.1 | true | true  |
+-----------+------+-------+

## Check the BE status to ensure that the Alive column is true.
mysql -uroot -P9030 -h127.0.0.1 -e 'SELECT `host`, `alive` FROM backends()'
+-----------+-------+
| host      | alive |
+-----------+-------+
| 127.0.0.1 |     1 |
+-----------+-------+
```

Local Quick Deployment

> **Environment Recommendations:**
>
> - **Operating System**: It is recommended to use AMD/ARM mainstream Linux environments such as CentOS 7.1 or Ubuntu 16.04 and above.
> - **Java Environment**: It is advised to use the Java 8 runtime environment. For non-Oracle JDK commercial license users, please use the free Oracle JDK 8u300 or later versions.
> - **User Permissions**: It is recommended to create a new Doris user on Linux and avoid using the root user for operations.

Step 1: Download the Binary Package

Download the corresponding binary installation package from the Apache Doris website here, and extract it.

Step 2: Modify the Environment Variables

1. **Modify the system's maximum open file descriptor limit**

   Use the following command to adjust the maximum file descriptor limit. After making this change, you need to restart the session to apply the configuration:

   ```
   vi /etc/security/limits.conf
   * soft nofile 1000000
   * hard nofile 1000000
   ```

2. **Modify Virtual Memory Area**

   Use the following command to permanently modify the virtual memory area to at least 2000000, and apply the change immediately:

   ```
   cat >> /etc/sysctl.conf << EOF
   vm.max_map_count = 2000000
   EOF

   ## Take effect immediately
   sysctl -p
   ```

Step 3: Install FE

1. **Configure FE**

   Modify the following contents in the FE configuration file `apache-doris/fe/conf/fe.conf`:

```
## Specify Java environment
JAVA_HOME=/home/doris/jdk

## Specify the CIDR block for FE listening IP
priority_networks=127.0.0.1/32
```

2. **Start FE**

   Run the FE process by executing the `start_fe.sh` script:

```
apache-doris/fe/bin/start_fe.sh --daemon
```

3. **Check FE Status**

   Connect to the cluster using MySQL client and check the cluster status:

```
## Check FE Status to ensure that both the Join and Alive columns are true
mysql -uroot -P9030 -h127.0.0.1 -e "show frontends;"
+-----------------------------------------+-----------+-------------+----------+-----------+---------+----------+----------
+-----------+------+-------+------------------+---------------------+----------+--------+-------------------------+-------
-----------+
| Name                                    | Host      | EditLogPort | HttpPort | QueryPort | RpcPort | Role     | IsMaster
| ClusterId | Join | Alive | ReplayedJournalId | LastHeartbeat       | IsHelper | ErrMsg | Version                 |
CurrentConnected |
+-----------------------------------------+-----------+-------------+----------+-----------+---------+----------+----------
+-----------+------+-------+------------------+---------------------+----------+--------+-------------------------+-------
-----------+
| fe_9d0169c5_b01f_478c_96ab_7c4e8602ec57 | 127.0.0.1 | 9010        | 8030     | 9030      | 9020    | FOLLOWER | true
| 656872880 | true | true  | 276               | 2024-07-28 18:07:39 | true     |        | doris-2.0.12-2971efd194 | Yes
|
+-----------------------------------------+-----------+-------------+----------+-----------+---------+----------+----------
+-----------+------+-------+------------------+---------------------+----------+--------+-------------------------+-------
-----------+
```

Step 4: Install BE

1. **Configure BE**

   Modify the following contents in the BE configuration file `apache-doris/be/conf/be.conf`:

```
## Specify Java environment
JAVA_HOME=/home/doris/jdk

## Specify the CIDR block for BE's listening IP
priority_networks=127.0.0.1/32
```

2. **Start BE**

   Start the BE process with the following command:

```
apache-doris/be/bin/start_be.sh --daemon
```

3. **Register BE Node in the Cluster**

   Connect to the cluster using MySQL client:

```
mysql -uroot -P9030 -h127.0.0.1
```

   Use the ADD BACKEND command to register the BE node:

```
ALTER SYSTEM ADD BACKEND "127.0.0.1:9050";
```

4. **Check BE Status**

   Connect to the cluster using MySQL client and check the cluster status:

```
## Check BE Status to ensure that the Alive column is true
mysql -uroot -P9030 -h127.0.0.1 -e "show backends;"
+-----------+-----------+---------------+--------+----------+----------+---------------------+---------------------+-------
+----------------------+-----------+-----------------+--------------------+---------------+---------------+---------+-----
-----------+--------------------+-------------------------+--------+-------------------------+---------------------------
-------------------------------------------------------------------------------------------------+-----------------------
--+----------+
| BackendId | Host      | HeartbeatPort | BePort | HttpPort | BrpcPort | LastStartTime       | LastHeartbeat       | Alive
| SystemDecommissioned | TabletNum | DataUsedCapacity | TrashUsedCapcacity | AvailCapacity | TotalCapacity | UsedPct |
MaxDiskUsedPct | RemoteUsedCapacity | Tag                     | ErrMsg | Version                 | Status
| HeartbeatFailureCounter | NodeRole |
+-----------+-----------+---------------+--------+----------+----------+---------------------+---------------------+-------
+----------------------+-----------+-----------------+--------------------+---------------+---------------+---------+-----
-----------+--------------------+-------------------------+--------+-------------------------+---------------------------
-------------------------------------------------------------------------------------------------+-----------------------
--+----------+
| 10156     | 127.0.0.1 | 9050          | 9060   | 8040     | 8060     | 2024-07-28 17:59:14 | 2024-07-28 18:08:24 | true
| false                | 14        | 0.000            | 0.000              | 8.342 GB      | 19.560 GB     | 57.35 % |
57.35 %        | 0.000              | {"location" : "default"} |        | doris-2.0.12-2971efd194 |
{"lastSuccessReportTabletsTime":"2024-07-28
18:08:14","lastStreamLoadTime":-1,"isQueryDisabled":false,"isLoadDisabled":false} | 0                       | mix      |
+-----------+-----------+---------------+--------+----------+----------+---------------------+---------------------+-------
+----------------------+-----------+-----------------+--------------------+---------------+---------------+---------+-----
-----------+--------------------+-------------------------+--------+-------------------------+---------------------------
-------------------------------------------------------------------------------------------------+-----------------------
--+----------+
```

Run Queries

1. **Connect to the cluster using MySQL client:**

```
mysql -uroot -P9030 -h127.0.0.1
```

2. **Create database and test table:**

```
create database demo;

use demo;
```

```
create table mytable
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.05",
    k3 CHAR(10) COMMENT "string column",
    k4 INT NOT NULL DEFAULT "1" COMMENT "int column"
)
COMMENT "my first table"
DISTRIBUTED BY HASH(k1) BUCKETS 1
PROPERTIES (
 "replication_num" = "1"
);
```

3. **Import test data:**

   Insert test data using the Insert Into statement

```
insert into mytable values
(1,0.14,'a1',20),
(2,1.04,'b2',21),
(3,3.14,'c3',22),
(4,4.35,'d4',23);
```

4. **Execute the following SQL query in the MySQL client to view the imported data:**

```
MySQL [demo]> select * from demo.mytable;
+------+------+------+------+
| k1   | k2   | k3   | k4   |
+------+------+------+------+
|    1 | 0.14 | a1   |   20 |
|    2 | 1.04 | b2   |   21 |
|    3 | 3.14 | c3   |   22 |
|    4 | 4.35 | d4   |   23 |
+------+------+------+------+
4 rows in set (0.10 sec)
```

Alternative to ClickHouse

Apache Doris and ClickHouse are both leading global real-time data warehouses, supporting columnar storage and fast queries. Doris has advantages like higher concurrency, more efficient joins, easier maintenance, and MySQL-like SQL syntax, making it simpler to use and deploy.

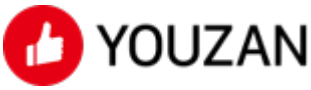
Featured Migration Cases



> "Tencent Music's data platform has migrated from ClickHouse to Apache Doris, improving data timeliness and reducing maintenance costs. Doris' flexible ingestion methods and robust consistency protocol ensure high availability and reliability."

Highlight:

- Massive boost in multi-table join performance.
- Easy scaling and maintenance.
- Efficient data processing and real-time updates.



> "Apache Doris has faster query response times than ClickHouse in the vast majority of scenarios, especially in complex join scenarios, where its performance is significantly superior to that of ClickHouse."

Highlight:

- Core business queries 2-3x.
- Complex join queries 2-10x.
- Can run all ClickHouse OOM queries.



> "By replacing ClickHouse with Doris, Kwai successfully upgraded to a lakehouse architecture, simplifying the data pipeline and eliminating the need for data import, as Doris can directly access data lake data."

Highlight:

- Directly query of data lake data.

- Improved query performance.
- Flexible data governance with materialized views.

Apache Doris vs. ClickHouse

	Apache Doris	ClickHouse
Architecture & SQL	• Based on MPP architecture • Standard SQL support, MySQL-compatible	• Uses Scatter-Gather architecture • SQL-like capabilities but with non-standard SQL
Queries	• Distributed joins • Cost-Based Optimization (CBO) • Query rewrites and multi-table materialized views • Higher concurrent performance	• Poor join implementation • Lacks a Cost-Based Optimizer (CBO) • Only supports single-table materialized views • Lower concurrency performance
Real-time Updates	• Features a strongly consistent primary key storage model, supporting synchronous data updates and deletions	• Only supports asynchronous updates, allowing old values to be read after an update.
Data API	• Offers high-throughput read APIs based on Arrow-flight, facilitating integration with other engines such as data science/AI tools	• Only inefficient data reading via JDBC API
Building Open Lakehouse	• Serves as a Lakehouse SQL engine, supporting queries on Hive, Hudi, Iceberg, and Parquet data lake formats	• Limited Lakehouse integration capabilities
Operations & Maintenance	• Supports automatic scaling in, scaling out, and replica balancing	• Requires manual rebalancing during scaling operations
Performance	• In wide table benchmarks (ClickBench), Doris ranked top 1 or top 2 in October 2022 and October 2024, outperforming ClickHouse • In large TPC-H and TPC-DS tests, Doris achieved leading performance	• In terms of ClickBench performance, ClickHouse and Doris have been taking turns leading • Experiences many OOM (Out of Memory) queries in large TPC-H and TPC-DS tests

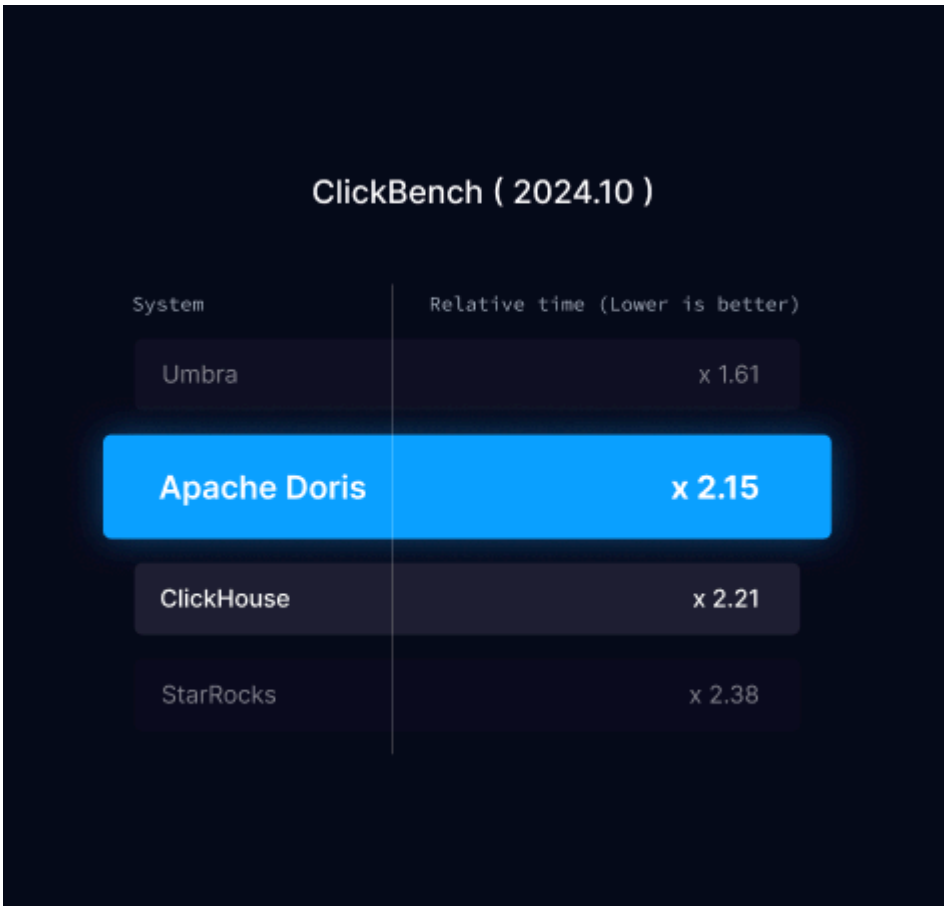
Performance Comparison

ClickBench Benchmark

ClickBench is a benchmarking tool created and maintained by the ClickHouse team to evaluate the performance of analytical databases.

It focuses on testing the performance of **large, flat tables rather than complex multi-table joins.** It uses real-world data from a major web analytics platform, covering typical scenarios such as clickstream analysis and structured logs.

The benchmark consists of a set of queries that test aggregation operations and single-table performance, without involving complex joins. This makes it especially useful for evaluating databases optimized for real-time analytics and large-scale data processing.



SSB-Flat SF100 Benchmark

SSB-Flat SF100 is a benchmark designed to test the performance of analytical databases in handling large, wide tables.

It is derived from the Star Schema Benchmark (SSB) but flattens the star schema into a single wide table to **focus on the performance of single-table queries.**

The SF100 indicates that the data scale is 100 times the base size, making it a significant test for evaluating query performance and system scalability.



TPC-H SF100 Benchmark

The TPC-H benchmark with a scale factor of 100 (SF100) is a widely used standard for evaluating database performance. It includes a set of complex SQL queries designed to simulate real-world business intelligence workloads.

The SF100 indicates that the data size is 100 times the base size, making it a large-scale test to measure query performance and system scalability.

Note: Since ClickHouse failed to execute 7 queries, the total execution time refers to the time taken by Doris to run all 22 queries, and by ClickHouse to run only 15 queries.



TPC-DS 1TB Benchmark

TPC-DS 1TB is a widely recognized benchmark for evaluating the performance of data warehouses and analytical databases. It involves a dataset of approximately 1TB in size, containing around 6.35 billion records spread across 24 tables.

The benchmark includes 99 complex queries designed to test various aspects of database performance, such as joins, aggregations, and subqueries.

The TPC-DS schema is based on a snowflake schema, representing real-world scenarios like web, catalog, and store sales. The 1TB scale is considered a moderate size for data warehouses but is still challenging due to the complexity of the queries and the large number of records

Note：TPC-DS makes heavy use of correlated subqueries which are at the time of testing (September 2024) not supported by ClickHouse. As a result, about 50% of benchmark queries will fail with errors.



More Migration Stories

- Less components, higher performance: Apache Doris instead of ClickHouse, MySQL, Presto, and HBase
- Migrating from ClickHouse to Apache Doris: What happened?
- ClickHouse & Kudu to Doris: 10X concurrency increased, 70% latency down

Alternative to Elasticsearch

Elasticsearch and Apache Doris are both popular in observability, cybersecurity, and real-time analytics. However, Elasticsearch can be costly in terms of storage and write resources. Apache Doris reduces these costs through efficient storage and high compression, and offers comprehensive analytical capabilities, such as JOIN and superior query performance.

Featured Migration Cases



> "By replacing Elasticsearch with VeloDB(Powered by Apache Doris), GuanceDB showcases a big stride in improving data processing speed and reducing costs."

Highlight:

- 70% Cost Reduction
- 2-3x Faster full-text search performance
- Variant Data type is flexible to handle semi-structured data in log tracing



> "Previously, we used multiple components for complex security analysis... Adopting Doris as a unified solution has significantly improved data writes, query performance and storage efficiency."

Highlight:

- 4x Faster write speeds
- 3x Better query performance
- 50% Storage space savings



> "Compared to the original OLAP database, query performance has improved 5-10 times, concurrency has doubled, and analysis time has dropped from 10 minutes to under 1 minute for 90% of cases, all while using just one-third of the original resources."

Highlight:

- 2x Increasing report analysis concurrency
- 65% Storage space reduction
- Simplified query with standard SQL

Apache Doris vs. Elasticsearch

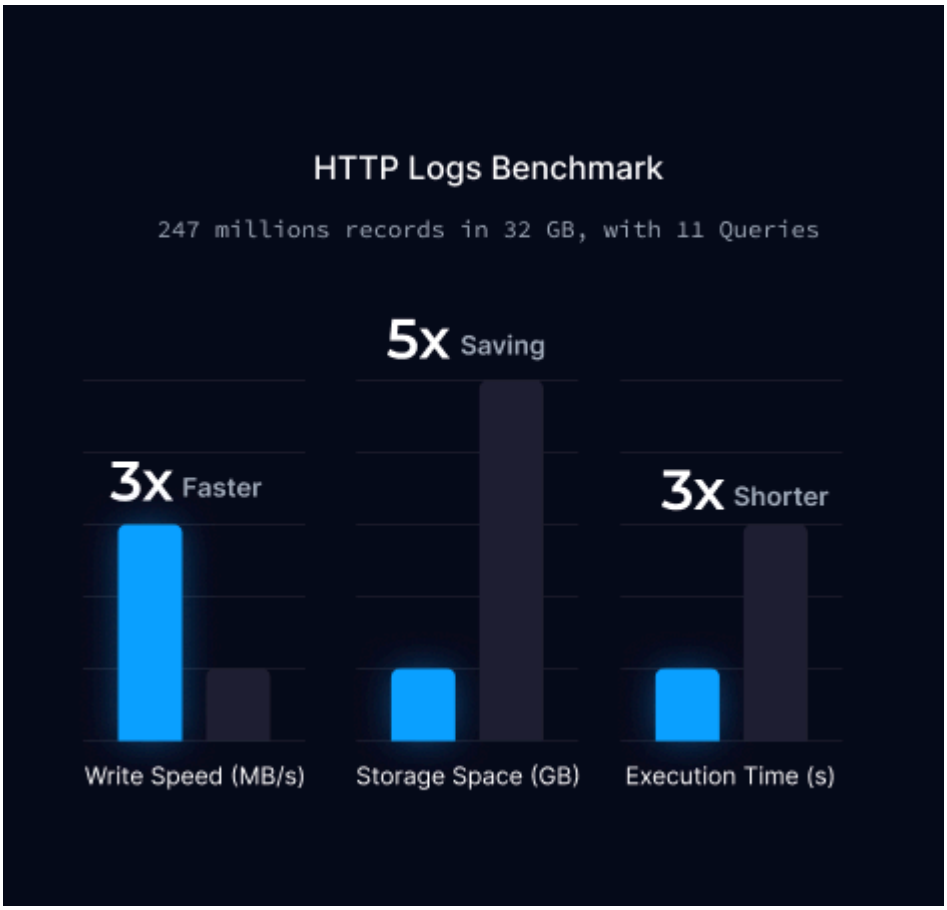
	Apache Doris	Elasticsearch
Open Source License	• Licensed under Apache License 2.0 • Stable License since governed by the Apache Software Foundation	• License changed from Apache License 2.0 to Elastic License, then to AGPL License • Changing license since governed by Elastic NV
Architecture	**Higher flexibility and elasticity:** • Strict workload isolation by workload group, powered by Linux CGroups, ideal for multi-tenancy • Compute-Storage decoupled and coupled modes	**Traditional deployment with limited elasticity:** • Soft Workload Isolation by Thread Group • Does not support decoupling compute and storage
Real-Time Data Writes	• High throughput: Indexing only on one replica • Pull-based ingestion via Kafka CDC, easier and simpler • Support Logstash and Beats output plugin	• Low throughput: Indexing for multiple data replicas • Requires additional tools like Logstash and Beats for pull-based ingestion, less convenient
Real-Time Data Storage	• Low storage consumption with compression rates up to 1:5 - 1:10 • Unique model supports both write and read optimization (MoW & MoR), retaining 90% of write speed when data is duplicated by key • Aggregation model supports strong consistency, allows aggregated data updates, and coexists with original data • Flexible Schema Change to meet dynamic business needs	• High storage consumption with a compression ratio of 1:1.5 • Unique model only supports write optimization, with write performance loss up to 3 times • The aggregation model does not allow aggregated data to be updated and does not coexist with the original data • Limited support for Schema Change
Real-Time Data Queries	• Lightning-Fast in various query workloads • Supports multi-table JOINs and optimization for complex analysis • Easy to use with standard SQL • Open MySQL ecosystem	• Good at point queries, but not suited for data analysis • No support for multi-table JOINs or complex analysis • Difficult for users due to custom DSL • Proprietary Elasticsearch ecosystem

Performance Comparison

Observability & Cyber Security

The HTTP Logs benchmark is an official Elasticsearch performance test designed for log storage and analysis. It uses a real-world HTTP log dataset to **evaluate indexing performance, storage efficiency, and query performance.**

This benchmark comprises 11 queries commonly used in log analysis scenarios, including keyword search, time range queries, aggregations, and sorting. As a result, it is highly suitable for assessing performance in observability and network security analysis contexts.



Real-Time Analytics

ClickBench is a benchmarking tool to evaluate the performance of analytical databases. It focuses on testing the performance of large, flat tables rather than complex multi-table joins. It uses real-world data from a major web analytics platform, covering typical scenarios such as clickstream analysis and structured logs.

The benchmark consists of a set of queries that test aggregation operations and single-table performance, without involving complex joins. This makes it especially useful for evaluating databases optimized for real-time analytics and large-scale data processing.

Note: These test results are archived benchmarks captured in December 2024. Current real-time comparisons are maintained at ClickBench.



More Migration Stories

- Why Apache Doris is a Better Alternative to Elasticsearch for Real-Time Analytics
- Creator of Talkie migrated from Loki and built a PB-scale logging system with Apache Doris
- How Tencent Music saved 80% in costs by migrating from Elasticsearch to Apache Doris
- Apache Doris for log and time series data analysis in NetEase, why not Elasticsearch and InfluxDB?

Apache Doris vs Trino / Presto

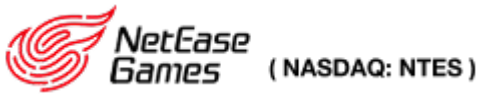
Apache Doris and Trino/Presto are both popular data lakehouse query engines, but Doris outperforms Trino/Presto in terms of performance. While Trino/Presto are primarily query engines, Doris can also function as a standalone data warehouse. This enables enterprises to unify their data warehouse and Lakehouse query engine into one with Doris, simplifying their data architecture

- **Unified**: Doris unifies data warehouse and Lakehouse query engine, simplifying the tech stack
- **10x Query Performance**: Doris native table boosts query performance by up to 10x compared to Presto/Trino
- **2-3x Faster**: Doris as a Lakehouse engine is 2-3x faster than Presto/Trino

Featured Migration Cases



> “As the world-renowed internet giant, our early data platform used Trino, Pinot, Iceberg, and Kyuubi, but faced complexity, redundancy, and poor performance. By replacing them with Apache Doris, we unified its data lakehouse and query engine, **boosting performance and reducing costs by 30%.**”



> “After switching from Presto to Doris, query performance significantly improved, **reducing query time from 20-40 seconds to 1-2 seconds.** By designing 2-3 materialized views based on common data dimensions, Doris can automatically match the optimal view for queries, further enhancing performance.”



> “Using Trino and SparkSQL, query latency was at the minute level, and performance was low. **After switching to Doris, performance improved 2 times.** Doris also unified the tech stack, simplifying the management of real-time and interactive analytics tools.”

Apache Doris vs. Trino / Presto

	Apache Doris	Trino / Presto
Architecture	• **Unified Architecture:** Combines the capabilities of a data warehouse and a Lakehouse query engine	• **Federated Querying:** Excels in querying across multiple heterogeneous data sources without data movement, but lacks built-in storage
Execution Engine	• Fully vectorized execution engine implemented in C++, for high-performance data processing	• Implemented primarily in Java, with vectorization currently in development as part of the Hummingbird project
Query Optimizer	• Advanced query optimizer with cost-based optimization for complex SQL operations like joins, aggregations, and sorting	• Supports cost-based optimization but with less advanced statistics collection and manual full collection
Caching Mechanisms	• **Metadata Caching:** In-memory metadata caching with TTL, auto-refresh, and incremental synchronization • **Data Caching:** Hot data caching on local SSDs for reduced network I/O • **Query Caching:** SQL Cache and Partition Cache for query result caching	• **Data Caching:** Relies on external caching solutions like Alluxio
Materialized Views	• **Incremental Refresh:** Supports incremental refresh and multiple update strategies • **Transparent Acceleration:** Query optimizer automatically routes queries to the most suitable materialized views	• **Manual Refresh:** Limited to manual, full refresh with less advanced features
Use Cases	• High-concurrency real-time analytics • Interactive analytics	• Only Interactive analytics

Performance Comparison

TPC-DS 1TB Benchmark

The TPC-DS 1TB Benchmark evaluates data warehouse performance using a 1TB dataset with 6.35 billion records across 24 tables. It includes 99 complex queries to test joins, aggregations, and subqueries. Based on

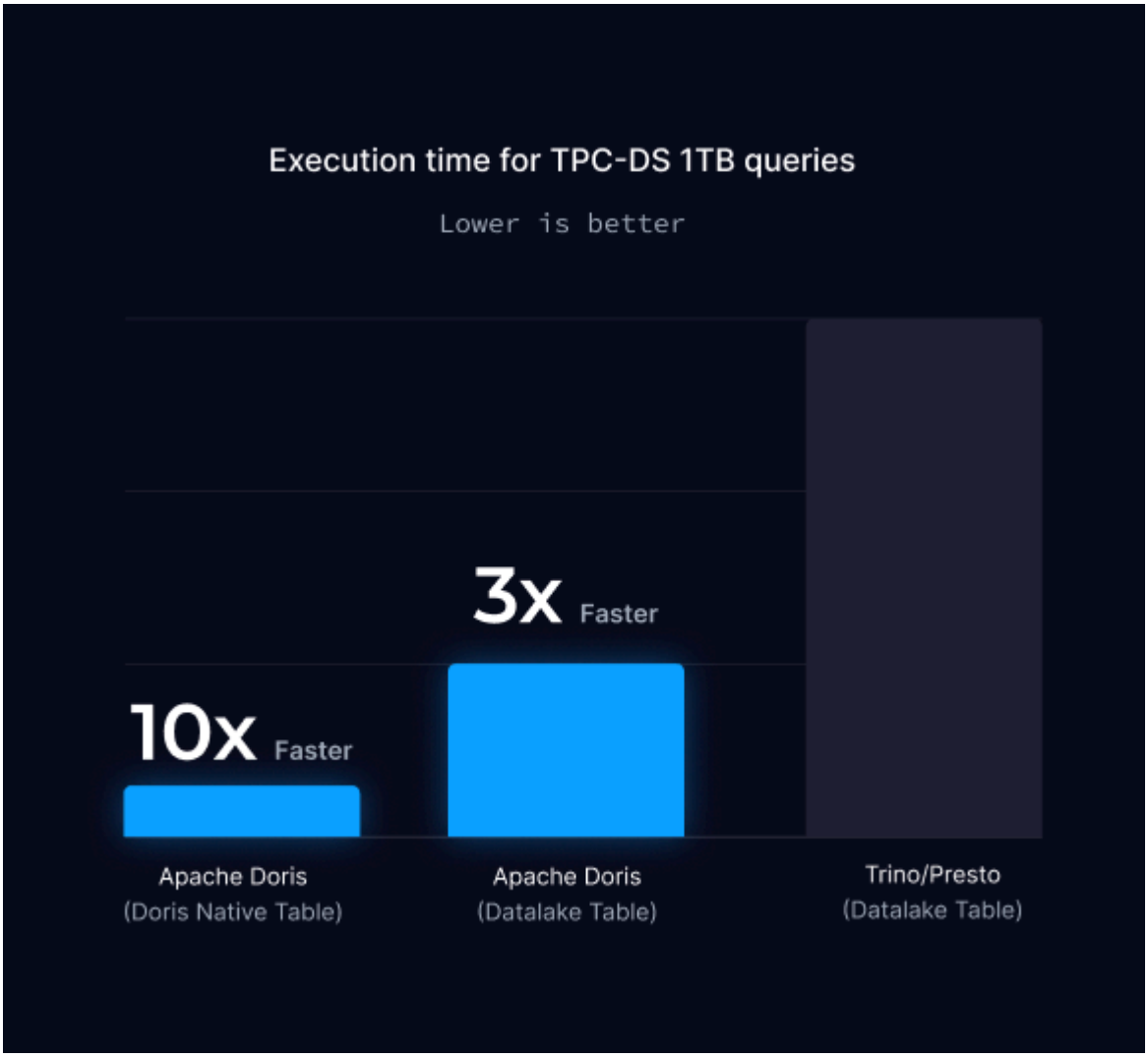
a snowflake schema, it simulates real-world sales scenarios. The 1TB scale is challenging due to query complexity.

The test environment consists of:

- 1 FE/Coordinator node and 5 BE/Worker nodes.
- Each node has 64 cores, 1.5TB of memory, and SSD storage.
- HDFS is co-located on these nodes, and Hive tables are created.

In this test, using the same dataset and equal computing service, the results shows that:

- **When data is imported into Doris' internal tables and queried using Doris, it achieves the shortest execution time.**
- **When Doris and Trino are used separately to query data directly from Hive tables, Doris demonstrates superior query acceleration performance in the data lake.**



Environment Checking

When deploying Doris, the following checks need to be performed for the hardware and software environment:

- Hardware Environment Check
- Recommended Server Configuration
- Disk Space Calculation
- Java Environment Check

Hardware Environment Check

During the hardware environment check, the following hardware conditions should be examined:

Check Item	Expected Result
CPU	Should support AVX2 instruction set.
Memory	Recommended at least 4 times the CPU size.
Storage	SSD recommended.
File System	ext4 or xfs file system.
Network Card	10 Gigabit network card.

CPU Check

When installing Doris, it is recommended to choose a machine that supports the AVX2 instruction set to leverage the vectorization capabilities of AVX2 for query acceleration.

Run the following command to check if the machine supports the AVX2 instruction set:

```
cat /proc/cpuinfo | grep avx2
```

If the machine does not support the AVX2 instruction set, you can use the no AVX2 Doris installation package for deployment.

Memory Check

Doris does not have strict memory limits. Generally, for production environments, you can choose the memory size based on the following recommendations:

Component	Recommended Memory Configuration
FE	At least 16GB recommended.

Component	Recommended Memory Configuration
BE	Memory should be at least 4 times the number of CPU cores (for example, for a 16-core machine, at least 64GB memory is recommended). Better performance can be achieved with memory 8 times the number of CPU cores.

Storage Check

Doris allows data to be stored on SSD, HDD, or object storage during deployment.

SSD is recommended for data storage in the following scenarios:

- High-concurrency point query scenarios with large-scale data
- High-frequency data update scenarios with large-scale data

File System Check

Doris recommends using EXT4 or XFS file systems:

- **EXT4**: The EXT4 file system offers good stability, performance, and lower fragmentation issues.
- **XFS**: The XFS file system performs excellently in handling large-scale data and high-concurrency write operations, making it suitable for high-throughput applications.

Network Card Check

Doris involves distributing data partitions across different instances for parallel processing, which results in some network resource overhead. To optimize Doris performance and reduce network resource overhead, it is strongly recommended to use a 10 Gigabit Ethernet (10GbE) or faster network during deployment. If multiple network cards are available, it is recommended to use link aggregation to combine multiple network cards into one virtual interface, which improves network bandwidth, redundancy, and complex balancing capabilities.

Server Configuration Recommendations

Doris can be deployed on x86-64 or ARM64 architecture server platforms.

- **Development and Testing Environments**

  In development and testing environments, FE and BE instances can be deployed in a mixed manner, following these guidelines:

  - In a validation testing environment, one FE and one BE can be deployed on a single server, but it is not recommended to deploy multiple FE and BE instances on the same machine.
  - If 3 replicas of data are required, at least 3 servers are needed, with each server deploying one BE instance.

Module	CPU	Memory	Disk	Network	Minimum Instance Count
Frontend	8 cores +	8 GB+	SSD or SATA, 10 GB+	Gigabit/Ten-Gigabit	1

Module	CPU	Memory	Disk	Network	Minimum Instance Count
Backend	8 cores +	16 GB+	SSD or SATA, 50 GB+	Gigabit/Ten-Gigabit	1

- **Production Environments**

  In a production environment, it is recommended to deploy FE and BE instances independently, following these guidelines:

  - If resources are limited and FE and BE need to be co-located on the same server, it is advised to store FE and BE data on separate hard drives.
  - BE nodes can be configured with multiple hard drives, allowing a single BE instance to utilize multiple HDD or SSD disks.

  Recommended server specifications are as follows:

Module	CPU	Memory	Disk	Network	Minimum Instance Count
Frontend	16 cores +	64 GB+	SSD or RAID card, 100 GB+	10-Gigabit	1
Backend	16 cores +	64 GB+	SSD or SATA, 100 GB+	10-Gigabit	3

Disk Space Calculation

In the Doris cluster, FE (Frontend) is mainly used for metadata storage, including metadata edit logs and images. BE (Backend) disk space is primarily used for storing data, and it needs to be calculated based on business requirements. | Component | Disk Space Description | | ——— | ——————————————————————————————————————— | | FE | It is recommended to reserve more than 100 GB of storage space, using SSD disks. | | BE | Doris uses LZ4 compression by default. The compression ratio is around 0.3 - 0.5. Disk space should be calculated as total data volume * 3 (for 3 replicas), and 40% of the space should be reserved for backend compaction and temporary data storage. |

Java Environment Check

All Doris processes depend on Java:

- **For versions before 2.1 (inclusive)**: please use Java 8, recommended version: `jdk-8u352` or later.
- **For versions from 3.0 (inclusive) onwards**: please use Java 17, recommended version: `jdk-17.0.10` or later.

Cluster Planning

Port Planning

Doris instances communicate over the network, and their proper functioning requires the following ports to be available. Administrators can adjust Doris' port configuration based on the actual environment:

Instance Name	Port Name	Default Port	Communication Direction	Description
BE	be_port	9060	FE -> BE	Thrift Server port on BE, used to receive requests from FE
BE	webserver_port	8040	BE <-> BE	HTTP Server port on BE
BE	heartbeat_service_port	9050	FE -> BE	Heartbeat service port (Thrift) on BE, used to receive heartbeats from FE
BE	brpc_port	8060	FE <-> BE, BE <-> BE	BRPC port on BE, used for communication between BEs
FE	http_port	8030	FE <-> FE, Client <-> FE	HTTP Server port on FE
FE	rpc_port	9020	BE -> FE, FE <-> FE	Thrift Server port on FE, each FE should have the same configuration
FE	query_port	9030	Client <-> FE	MySQL Server port on FE
FE	edit_log_port	9010	FE <-> FE	bdbje communication port on FE

Node Count Planning

FE Node Count

FE nodes are primarily responsible for user request handling, query parsing and planning, metadata management, and node management.

For production clusters, it is generally recommended to deploy at least 3 FE nodes to achieve a high-availability environment. FE nodes are divided into the following two roles:

- **Follower nodes**: Participate in election operations. When the Master node fails, a Follower node will be selected as the new Master.
- **Observer nodes**: Only sync metadata from the Leader node and do not participate in the election. These nodes can be used for horizontal scaling to improve the read service capacity of metadata.

In general, it is recommended to deploy at least 3 Follower nodes. In high-concurrency scenarios, increasing the number of Observer nodes can help improve the cluster's connection capacity.

BE Node Count

BE nodes are responsible for data storage and computation. In production environments, to ensure data reliability and fault tolerance, 3 copies of data are usually stored. Therefore, it is recommended to deploy at least 3 BE nodes.

BE nodes support horizontal scaling, and by increasing the number of BE nodes, the query performance and concurrent processing capabilities of the cluster can be effectively improved.

OS Checking

When deploying Doris, ensure the following operating system configurations:

- Disable the swap partition
- Disable transparent huge pages
- Ensure the system has enough virtual memory space
- Disable CPU power-saving mode
- Ensure new network connections are reset on overflow
- Ensure Doris-related ports are open or the firewall is disabled
- Ensure the system allows a sufficient number of open file descriptors
- Install and configure NTP service for clock synchronization

Disable Swap Partition

It is recommended to disable the swap partition when deploying Doris. The kernel may move memory data to the swap area when it detects memory pressure, but this can negatively impact Doris performance due to the kernel's limited understanding of application behavior.

To disable swap temporarily (swap will be re-enabled after a restart):

```
swapoff -a
```

To permanently disable swap, edit `/etc/fstab` and comment out the swap partition entry, then restart the machine:

```
# /etc/fstab
# <file system>        <dir>         <type>    <options>             <dump> <pass>
tmpfs                  /tmp          tmpfs     nodev,nosuid          0      0
/dev/sda1              /             ext4      defaults,noatime      0      1
# /dev/sda2              none          swap      defaults              0      0
/dev/sda3              /home         ext4      defaults,noatime      0      2
```

Disable Transparent Huge Pages

In high-load, low-latency scenarios, disabling Transparent Huge Pages (THP) is recommended to avoid performance degradation and memory fragmentation, ensuring stable memory usage for Doris.

Use the following commands to disable THP temporarily:

```
echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
```

To permanently disable THP, add the following commands to `/etc/rc.d/rc.local` to ensure it takes effect after a restart:

```
cat >> /etc/rc.d/rc.local << EOF
   echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
   echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
EOF
chmod +x /etc/rc.d/rc.local
```

Ensure Sufficient Virtual Memory Area

To allow Doris to handle large datasets, the system must have enough virtual memory space. Without adequate memory mapping, Doris may encounter errors like Too many open files during startup or runtime.

You can permanently modify the virtual memory area to at least 2000000 with the following command, and it will take effect immediately:

```
cat >> /etc/sysctl.conf << EOF
vm.max_map_count = 2000000
EOF

# Take effect immediately
sysctl -p
```

Disable CPU Power-Saving Mode

Disabling CPU power-saving mode ensures stable high performance during high load, preventing fluctuations and delays caused by reduced CPU frequency.

Use the following command to set the CPU governor to “performance,” disabling power-saving modes:

```
echo 'performance' | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Reset New Connections on Network Overflow

Ensure that when TCP connection buffers overflow, new connections are reset immediately. This prevents buffer blocking during high load and improves responsiveness and stability.

You can permanently configure the system to automatically reset new connections with the following commands, and it will take effect immediately:

```
cat >> /etc/sysctl.conf << EOF
net.ipv4.tcp_abort_on_overflow=1
EOF

# Take effect immediately
sysctl -p
```

Open Doris-related Ports

If Doris-related ports are blocked, you can try disabling the firewall to verify whether it is the cause. If the firewall is the issue, open the relevant ports for Doris components.

```
sudo systemctl stop firewalld.service
sudo systemctl disable firewalld.service
```

Increase System’s Open File Descriptors Limit

Since Doris manages a large number of files, you need to increase the system’s file descriptor limit.

To change the maximum number of open files, add the following to `/etc/security/limits.conf`:

```
vi /etc/security/limits.conf
* soft nofile 1000000
* hard nofile 1000000
```

Ensure NTP Service is Installed on Cluster Deployment Machines

Doris requires the metadata’s timestamp accuracy to be within 5000ms. To ensure consistent time across all nodes in the cluster and avoid metadata inconsistencies, you need to synchronize clocks across all machines using the NTP service.

Use the following commands to start and enable the NTP service:

```
sudo systemctl start ntpd.service
sudo systemctl enable ntpd.service
```

Deploy Integrated Storage Compute Cluster Manually

After completing the preliminary checks and planning, such as environment checks, cluster planning, and operating system inspections, you can begin deploying the cluster.

The integrated storage-compute architecture is shown below, and the deployment of the integrated storage-compute cluster involves four steps:

integrated-storage-compute-architecture

1. **Deploy FE Master Node**: Deploy the first FE node as the Master node;
2. **Deploy FE Cluster**: Deploy the FE cluster by adding Follower or Observer FE nodes;
3. **Deploy BE Nodes**: Register BE nodes to the FE cluster;
4. **Verify Cluster Correctness**: After deployment, connect to and verify the cluster's correctness.

Step 1: Deploy FE Master Node

1. **Create Metadata Path**

   When deploying FE, it is recommended to store metadata on a different hard drive from the BE node data storage.

   When extracting the installation package, a doris-meta directory is included by default. It is recommended to create a separate metadata directory and link it to the doris-meta directory. In production, it's highly advised to use a separate directory outside the Doris installation folder, preferably on an SSD. For testing and development environments, you can use the default configuration.

   ```
   ## Use a separate disk for FE metadata
   mkdir -p <doris_meta_created>

   ## Create FE metadata directory symlink
   ln -s <doris_meta_created> <doris_meta_original>
   ```

2. **Modify FE Configuration File**

   The FE configuration file is located in the conf directory under the FE deployment path. Before starting the FE node, modify the `conf/fe.conf` file..

   Before deploying the FE node, it is recommended to modify the following configurations:

   ```
   ## modify Java Heap
   JAVA_OPTS="-Xmx16384m -XX:+UseMembar -XX:SurvivorRatio=8 -XX:MaxTenuringThreshold=7 -XX:+PrintGCDateStamps -
   XX:+PrintGCDetails -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:+CMSClassUnloadingEnabled -XX:-CMSParallelRemarkEnabled -
   XX:CMSInitiatingOccupancyFraction=80 -XX:SoftRefLRUPolicyMSPerMB=0 -Xloggc:$DORIS_HOME/log/fe.gc.log.$DATE"

   ## modify case sensitivity
   ```

```
lower_case_table_names = 1

## modify network CIDR
priority_networks = 10.1.3.0/24

## modify Java Home
JAVA_HOME = <your-java-home-path>
```

Parameter Descriptions: For more details, refer to the FE Configuration:

Parameter	Suggestion
JAVA_OPTS	Specify the `-Xmx` parameter to adjust the Java Heap. It is recommended to set it to above 16G in production environments.
lower_case_table_names	Set case sensitivity. It is recommended to adjust it to 1, meaning case-insensitive.
priority_networks	Network CIDR is specified based on the network IP address. It can be ignored in an FQDN environment.
JAVA_HOME	It is recommended to use a JDK environment independent of the operating system for Doris.

3. **Start FE Process**

   You can start the FE process using the following command:

   ```
   bin/start_fe.sh --daemon
   ```

   The FE process will start and run in the background. By default, logs are stored in the log/ directory. If the startup fails, you can check the log/fe.log or log/fe.out files for error details.

4. **Check FE Startup Status**

   You can connect to the Doris cluster using MySQL Client. The default user is root, and the password is empty.

   ```
   mysql -uroot -P<fe_query_port> -h<fe_ip_address>
   ```

   After connecting to the Doris cluster, you can use the `show frontends` command to check the status of FE nodes. Typically, you should verify the following:

   - Alive: If true, it indicates the node is alive.
   - Join: If true, it indicates the node has joined the cluster, but it doesn't necessarily mean the node is still active in the cluster (it may have lost connection).
   - IsMaster: If true, it indicates the current node is the Master node.

Step 2: Deploy FE Cluster (Optional)

In production, it is recommended to deploy at least 3 nodes. After deploying the FE Master node, you should deploy two additional FE Follower nodes.

1. **Create Metadata Directory**

   Follow the same steps as for deploying the FE Master node to create the `doris-meta` directory.

2. **Modify FE Follower Node Configuration**

   Modify the FE configuration file for the Follower node, following the same steps as for the FE Master node. Typically, you can simply copy the configuration file from the FE Master node.

3. **Register New FE Follower Node in the Doris Cluster**

   Before starting a new FE node, you need to register the new FE node in the FE cluster.

   ```
   ## connect a alive FE node
   mysql -uroot -P<fe_query_port> -h<fe_ip_address>

   ## register a new FE follower node
   ALTER SYSTEM ADD FOLLOWER "<fe_ip_address>:<fe_edit_log_port>"
   ```

   To add an observer node, use the `ADD OBSERVER` command:

   ```
   ## register a new FE observer node
   ALTER SYSTEM ADD OBSERVER "<fe_ip_address>:<fe_edit_log_port>"
   ```

   :::caution Note

   - The number of FE Follower nodes (including Master) should be odd. It is recommended to deploy 3 nodes for high availability.
   - When FE is deployed in high availability mode (1 Master, 2 Followers), we recommend adding Observer FE nodes to extend the FE read service capacity. :::

4. **Start FE Follower Node**

   The FE Follower node can be started with the following command, which will automatically synchronize metadata.

   ```
   bin/start_fe.sh --helper <helper_fe_ip>:<fe_edit_log_port> --daemon
   ```

   Here, helper_fe_ip refers to any live node in the FE cluster. The –helper parameter is used only during the initial startup of FE to synchronize metadata; subsequent restarts do not require this parameter.

5. **Check Follower Node Status**

   The method for checking the FE Follower node status is the same as for the FE Master node status. After adding the Follower node, use the show frontendscommand to check the FE node status. Unlike the Master, theIsMaster state should be false.

Step 3: Deploy BE Node

1. **Create Data Directory**

   The BE process is responsible for data computation and storage. The data directory is by default located under `be/storage`. In a production environment, it is common to store BE data on a separate disk, placing the BE data and deployment files on different disks. BE supports distributing data across multiple disks to better utilize the I/O capabilities of multiple hard drives.

   ```
   ## Create a BE data storage directory on each data disk
   mkdir -p <be_storage_root_path>
   ```

2. **Modify BE Configuration File**

   The BE configuration file is located in the conf directory under the BE deployment path. Before starting the BE node, you need to modify the `conf/be.conf` file.

   ```
   ## modify storage path for BE node
   storage_root_path=/home/disk1/doris,medium:HDD;/home/disk2/doris,medium:SSD

   ## modify network CIDR
   priority_networks = 10.1.3.0/24

   ## modify Java Home in be/conf/be.conf
   JAVA_HOME = <your-java-home-path>
   ```

   Parameter explanations are as follows:

Parameters	Suggestions
priority_networks	Network CIDR, specified by network IP address. Can be ignored in FQDN environments.
JAVA_OPTS	Set the `-Xmx` parameter to adjust the Java heap size. It is recommended to set it to 2GB or more for production environments.
JAVA_HOME	It is recommended to use a JDK environment that is independent of the operating system for Doris.

3. **Register BE Node in Doris**

   Before starting the BE node, register it in the FE cluster:

   ```
   ## connect a alive FE node
   mysql -uroot -P<fe_query_port> -h<fe_ip_address>

   ## Register BE node
   ALTER SYSTEM ADD BACKEND "<be_ip_address>:<be_heartbeat_service_port>"
   ```

4. **Start BE Process**

   The BE process can be started with the following command:

```
bin/start_be.sh --daemon
```

The BE process starts and runs in the background. Logs are stored by default in the `log/` directory. If the startup fails, check the `log/be.log` or `log/be.out` files for error information.

5. **Check BE Startup Status**

   After connecting to the Doris cluster, use the show backends command to check the BE node status.

```
## connect a alive FE node
mysql -uroot -P<fe_query_port> -h<fe_ip_address>

## check BE node status
show backends;
```

   Typically, pay attention to the following states:

   - `Alive` being true indicates that the node is alive.
   - `TabletNum` represents the number of shards on the node. Newly added nodes will undergo data balancing, and the `TabletNum` will gradually become more evenly distributed.

Step 4: Verify Cluster Integrity

1. **Log in to the Database**

   Log in to the Doris cluster using the MySQL Client.

```
## connect a alive fe node
mysql -uroot -P<fe_query_port> -h<fe_ip_address>
```

2. **Check Doris Installation Information**

   Use `show frontends` and `show backends` to view the status of each database instance.

```
-- check fe status
show frontends \G

-- check be status
show backends \G
```

3. **Change Doris Cluster Password**

   When the Doris cluster is created, a user named `root` is automatically created, and its password is set to empty by default. For security reasons, it is recommended to set a new password for the `root` user immediately after the cluster is created.

```
-- check the current user
select user();
+-----------------------+
| user()                |
+-----------------------+
```

```
| 'root'@'192.168.88.30' |
+------------------------+

-- modify the password for current user
SET PASSWORD = PASSWORD('doris_new_passwd');
```

4. **Create a Test Table and Insert Data**

   To verify the integrity of the cluster, you can create a test table in the newly created cluster and insert some data.

```
-- create a test database
create database testdb;

-- create a test table
CREATE TABLE testdb.table_hash
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    k3 VARCHAR(10) COMMENT "string column",
    k4 INT NOT NULL DEFAULT "1" COMMENT "int column"
)
COMMENT "my first table"
DISTRIBUTED BY HASH(k1) BUCKETS 32;
```

   Doris is compatible with the MySQL protocol, and you can use the INSERT statement to insert data.

```
-- insert data
INSERT INTO testdb.table_hash VALUES
(1, 10.1, 'AAA', 10),
(2, 10.2, 'BBB', 20),
(3, 10.3, 'CCC', 30),
(4, 10.4, 'DDD', 40),
(5, 10.5, 'EEE', 50);

-- check the data
SELECT * from testdb.table_hash;
+------+-------+------+------+
| k1   | k2    | k3   | k4   |
+------+-------+------+------+
|    3 | 10.30 | CCC  |   30 |
|    4 | 10.40 | DDD  |   40 |
|    5 | 10.50 | EEE  |   50 |
|    1 | 10.10 | AAA  |   10 |
|    2 | 10.20 | BBB  |   20 |
+------+-------+------+------+
```

Deploy Doris Operator

Deploying the Doris Operator involves three steps: install the CustomResourceDefinitions, deploy the Operator service, verify the deployment status.

Step 1: Install CustomResourceDefinitions

Add the custom resource (CRD) of Doris Operator using the following command:

```
kubectl create -f https://raw.githubusercontent.com/apache/doris-operator/master/config/crd/bases/crds.yaml
```

Step 2: Install Doris Operator and RBAC rules

Install Doris Operator using the following command:

```
kubectl apply -f https://raw.githubusercontent.com/apache/doris-operator/master/config/operator/operator.yaml
```

Expected output:

```
namespace/doris created
role.rbac.authorization.k8s.io/leader-election-role created
rolebinding.rbac.authorization.k8s.io/leader-election-rolebinding created
clusterrole.rbac.authorization.k8s.io/doris-operator created
clusterrolebinding.rbac.authorization.k8s.io/doris-operator-rolebinding created
serviceaccount/doris-operator created
deployment.apps/doris-operator created
```

Step 3: Verify Doris Operator status

Check the deployment status of Doris Operator using the following command:

```
kubectl get pods -n doris
```

Expected output:

```
NAME                             READY   STATUS    RESTARTS   AGE
doris-operator-7f578c86cb-nz6jn   1/1     Running   0          19m
```

Config Doris to Deploy

Cluster planning

In the default DorisCluster resource deployment, the FE and BE images may not be the latest versions, and the default replica count for both FE and BE is set to 3. Additionally, the default resource configuration for FE is 6 CPUs and 12Gi of memory, while for BE, it is 8 CPUs and 16Gi of memory. This section describes how to modify these default configurations according to your requirements.

Image configuration

Doris Operator is decoupled from the Doris version and supports deploying Doris versions 2.0 and above.

FE image configuration

To specify the FE image version, use the following configuration:

```
spec:
  feSpec:
    image: ${image}
```

Replace ${image} with the desired image name, then update the configuration in the target DorisCluster resource. Official FE images are available at FE Image.

BE image configuration

To specify the BE image version, use the following configuration:

```
spec:
  beSpec:
    image: ${image}
```

Replace ${image} with the desired image name, then update the configuration in the target DorisCluster resource. Official BE images are available at BE Image.

Replicas configuration

FE Replicas configuration

To modify the default FE replica count of 3 to 5, use the following configuration:

```
spec:
  feSpec:
    replicas: 5
```

Update the configuration in the target DorisCluster resource.

BE replicas configuration

To modify the default FE replica count of 3 to 5, use the following configuration:

```
spec:
  beSpec:
    replicas: 5
```

Update the configuration to the DorisCluster resource that needs to be deployed.

Computing resource configuration

FE computing resource configuration

The default compute resource configuration for FE is 6 CPUs and 12Gi of memory. To modify it to 8CPUs and 16Gi, use the following configuration:

```
spec:
  feSpec:
    requests:
      cpu: 8
      memory: 16Gi
    limits:
      cpu: 8
      memory: 16Gi
```

Update the configuration in the target DorisCluster resource.

BE computing resource configuration

The default compute resource configuration for BE is 8 CPUs and 16Gi of memory. To modify it to 16 CPUs and 32Gi of memory, use the following configuration:

```
spec:
  beSpec:
    requests:
      cpu: 16
      memory: 32Gi
    limits:
      cpu: 16
      memory: 32Gi
```

Update the configuration in the target DorisCluster resource.

> **Tip**
>
> The minimum required resources for FE and BE to start are 4 CPUs and 8Gi of memory. For normal performance testing, it is recommended to configure 8 CPUs and 8Gi of memory.

Custom startup configuration

Doris uses ConfigMap to decouple configuration files from services, in Kubernetes. By default, services use the default configurations in the image as startup parameter configurations. To customize the startup parameters, create a specific ConfigMap following the instructions in the FE Configuration Document and the BE Configuration Document. Then deploy the customized ConfigMap to the namespace where the DorisCluster resource is to be deployed.

Custom FE startup configuration

Step 1: Create and deploy the FE ConfigMap

The following example defines a ConfigMap named fe-conf for use with Doris FE:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fe-conf
  labels:
    app.kubernetes.io/component: fe
data:
  fe.conf: |
    CUR_DATE=`date +%Y%m%d-%H%M%S`
    # the output dir of stderr and stdout
    LOG_DIR = ${DORIS_HOME}/log
    JAVA_OPTS="-Djavax.security.auth.useSubjectCredsOnly=false -Xss4m -Xmx8192m -XX:+UseMembar -XX:SurvivorRatio=8 -
XX:MaxTenuringThreshold=7 -XX:+PrintGCDateStamps -XX:+PrintGCDetails -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -
XX:+CMSClassUnloadingEnabled -XX:-CMSParallelRemarkEnabled -XX:CMSInitiatingOccupancyFraction=80 -XX:SoftRefLRUPolicyMSPerMB=0 -
Xloggc:$DORIS_HOME/log/fe.gc.log.$CUR_DATE"
    # For jdk 9+, this JAVA_OPTS will be used as default JVM options
    JAVA_OPTS_FOR_JDK_9="-Djavax.security.auth.useSubjectCredsOnly=false -Xss4m -Xmx8192m -XX:SurvivorRatio=8 -
XX:MaxTenuringThreshold=7 -XX:+CMSClassUnloadingEnabled -XX:-CMSParallelRemarkEnabled -XX:CMSInitiatingOccupancyFraction=80 -
XX:SoftRefLRUPolicyMSPerMB=0 -Xlog:gc*:$DORIS_HOME/log/fe.gc.log.$CUR_DATE:time"
    # INFO, WARN, ERROR, FATAL
    sys_log_level = INFO
    # NORMAL, BRIEF, ASYNC
    sys_log_mode = NORMAL
    # Default dirs to put jdbc drivers,default value is ${DORIS_HOME}/jdbc_drivers
    # jdbc_drivers_dir = ${DORIS_HOME}/jdbc_drivers
    http_port = 8030
    rpc_port = 9020
    query_port = 9030
    edit_log_port = 9010
    enable_fqdn_mode = true
```

When using the ConfigMap to mount FE startup configuration, the key corresponding to the configuration must be `fe.conf`. Write the ConfigMap to a file and deploy it to the namespace where the DorisCluster resource is deployed, using the following command:

```
kubectl -n ${namespace} apply -f ${feConfigMapFile}.yaml
```

Here, ${namespace} refers to the namespace where the DorisCluster is to be deployed, and ${feConfigMapFile} is the name of the ConfigMap file for FE.

step 2: Update the DorisCluster resource

To use the ConfigMap named `fe-conf` for mounting the startup configuration, add the following config to the FE spec of the DorisCluster resource:

```
spec:
  feSpec:
    configMapInfo:
```

```
      configMapName: fe-conf
      resolveKey: fe.conf
```

> **Tip**
>
> Please ensure that `enable_fqdn_mode=true` is included in the startup configuration.. If you want to use IP mode and K8s have the ability that the pod IP keep the same after restarted, please refer to the issue #138 to config.

Custom BE startup configuration

Step 1: Create and deploy the BE ConfigMap

The following example defines a ConfigMap named `be-conf` for use with Doris BE:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: be-conf
  labels:
    app.kubernetes.io/component: be
data:
  be.conf: |
    CUR_DATE=`date +%Y%m%d-%H%M%S`

    PPROF_TMPDIR="$DORIS_HOME/log/"

    JAVA_OPTS="-Xmx1024m -DlogPath=$DORIS_HOME/log/jni.log -Xloggc:$DORIS_HOME/log/be.gc.log.$CUR_DATE -
Djavax.security.auth.useSubjectCredsOnly=false -Dsun.java.command=DorisBE -XX:-CriticalJNINatives -DJDBC_MIN_POOL=1 -
DJDBC_MAX_POOL=100 -DJDBC_MAX_IDLE_TIME=300000 -DJDBC_MAX_WAIT_TIME=5000"

    # For jdk 9+, this JAVA_OPTS will be used as default JVM options
    JAVA_OPTS_FOR_JDK_9="-Xmx1024m -DlogPath=$DORIS_HOME/log/jni.log -Xlog:gc:$DORIS_HOME/log/be.gc.log.$CUR_DATE -
Djavax.security.auth.useSubjectCredsOnly=false -Dsun.java.command=DorisBE -XX:-CriticalJNINatives -DJDBC_MIN_POOL=1 -
DJDBC_MAX_POOL=100 -DJDBC_MAX_IDLE_TIME=300000 -DJDBC_MAX_WAIT_TIME=5000"

    # since 1.2, the JAVA_HOME need to be set to run BE process.
    # JAVA_HOME=/path/to/jdk/

    # https://doris.apache.org/community/developer-guide/debug-tool#jemalloc-heap-profile
    # https://jemalloc.net/jemalloc.3.html

JEMALLOC_CONF="percpu_arena:percpu,background_thread:true,metadata_thp:auto,muzzy_decay_ms:15000,dirty_decay_ms:15000,oversize_thres
hold:0,lg_tcache_max:20,prof:false,lg_prof_interval:32,lg_prof_sample:19,prof_gdump:false,prof_accum:false,prof_leak:false,prof_fina
l:false"
    JEMALLOC_PROF_PRFIX=""

    # INFO, WARNING, ERROR, FATAL
    sys_log_level = INFO

    # ports for admin, web, heartbeat service
    be_port = 9060
    webserver_port = 8040
    heartbeat_service_port = 9050
    brpc_port = 8060
```

When using the ConfigMap to mount BE startup configuration, the key corresponding to the configuration must be `be.conf`. Write the ConfigMap to a file and deploy it to the namespace where the DorisCluster resource is deployed using the following command:

```
kubectl -n ${namespace} apply -f ${beConfigMapFile}.yaml
```

Here, ${namespace} refers to the namespace where the DorisCluster resource needs to be deployed, and ${beConfigMapFile} is the name of the ConfigMap file for BE.

step 2: Update the DorisCluster resource

To use the ConfigMap named `be-conf` for mounting the startup configuration, add the following config to the BE spec of the DorisCluster resource:

```
spec:
  feSpec:
    configMapInfo:
      configMapName: be-conf
      resolveKey: be.conf
```

> **Tip**
>
> Please use the startup configMap to mount files, when you want mount the file into the config directory in container, the config directory is ${DORIS_HOME}/conf.

Mounting multiple ConfigMaps

The Doris Operator supports mounting multiple ConfigMaps into different directories within the container, allowing flexible configuration management.

Mounting multiple ConfigMaps for FE
The following example demonstrates how to mount two ConfigMaps `test-fe1` and `test-fe2` to the directories “/etc/fe/config1/” and “/etc/fe/config2”,respectively, within the FE container:

```
spec:
  feSpec:
    configMaps:
    - configMapName: test-fe1
      mountPath: /etc/fe/config1
    - configMapName: test-fe2
      mountPath: /etc/fe/config2
```

Mounting multiple ConfigMaps for BE
Similarly, the following example shows how to mount two ConfigMaps `test-be1` and `test-be2` into the directories “/etc/be/config1” and “/etc/be/config2”, respectively, within the BE container:

```
  spec:
    beSpec:
```

```
    configMaps:
    - configMapName: test-be1
      mountPath: /etc/be/config1
    - configMapName: test-be2
      mountPath: /etc/be/config2
```

Persistent storage

Kubernetes provides the Persistent Volumes to persist data to physical storage. In Kubernetes, the Doris Operator automatically creates PersistentVolumeClaims associated with appropriate PersistentVolumes, based on the template that defined in the need deployed DorisCluster Resource.

Persistent storage for FE

In a Kubernetes-based Doris deployment, it is recommended to persist the following paths for FE:
1. Metadata: /opt/apache-doris/fe/doris-meta (default storage configuration for FE metadata). 2. Logs: /opt/apache-doris/fe/log (if log persistence is required).

Persistent metadata for FE

To persist FE metadata using the default storage configuration, add the following configuration to the DorisCluster resource:

```
spec:
  feSpec:
    persistentVolumes:
    - mountPath: /opt/apache-doris/fe/doris-meta
      name: meta
      persistentVolumeClaimSpec:
        # when use specific storageclass, the storageClassName should reConfig, example as annotation.
        storageClassName: ${your_storageclass}
        accessModes:
        - ReadWriteOnce
        resources:
          # notice: if the storage size less 5G, fe will not start normal.
          requests:
            storage: ${storageSize}
```

In the above configuration, ${your_storageclass} represents the name of the StorageClass you want to use, and ${storageSize} represents the storage size you want to allocation. The format is quantity expression, such as: 100Gi.

Persistent FE log

If your cluster lacks a centralized log collection system, persist the FE log directory by adding the following configuration to the DorisCluster resource:

```
spec:
  feSpec:
    persistentVolumes:
    - mountPath: /opt/apache-doris/fe/log
      name: log
```

```
    persistentVolumeClaimSpec:
      # when use specific storageclass, the storageClassName should reConfig, example as annotation.
      storageClassName: ${your_storageclass}
      accessModes:
      - ReadWriteOnce
      resources:
        # notice: if the storage size less 5G, fe will not start normal.
        requests:
          storage: ${storageSize}
```

In the above configuration, ${your_storageclass} represents the name of the StorageClass you want to use, and ${storageSize} represents the storage size you want to allocation. The format of ${storageSize} follows the quantity expression method of K8s, such as: 100Gi. Please replace them as needed when using.

> **Tip**
>
> If you have reconfigured meta_dir or `LOG_DIR` in the customized configuration file, please reconfigure the mountPath.

Persistent storage for BE

For BE nodes in a Doris deployment, it is recommended to persist the following paths:
1. Data Storage: /opt/apache-doris/be/storage (default storage for BE data). 2. Logs: /opt/apache-doris/be/log (if log persistence is required).

Persistent data

- **Using default storage configuration**
  To persist data uses the default storage configuration, update the DorisCluster resource with the following configuration:

  ```
  beSpec:
    persistentVolumes:
    - mountPath: /opt/apache-doris/be/storage
      name: be-storage
      persistentVolumeClaimSpec:
        storageClassName: ${your_storageclass}
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: ${storageSize}
  ```

  In the above configuration, ${your_storageclass} represents the name of the StorageClass you want to use, and ${storageSize} represents the storage size you want to use. The format of ${storageSize} follows the quantity expression method of K8s, such as: 100Gi. Please replace them as needed when using.

- **Customizing BE storage paths**
  To leverage multiple disks, you can configure multiple storage directories using storage_root_path. For

example, if storage_root_path=/home/disk1/doris.HDD;/home/disk2/doris.SSD, the configuration should include:

```
beSpec:
  persistentVolumes:
  - mountPath: /home/disk1/doris
    name: be-storage1
    persistentVolumeClaimSpec:
      storageClassName: ${your_storageclass}
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: ${storageSize}
  - mountPath: /home/disk2/doris
    name: be-storage2
    persistentVolumeClaimSpec:
      storageClassName: ${your_storageclass}
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: ${storageSize}
```

In the above configuration, ${your_storageclass} represents the name of the StorageClass you want to use, and ${storageSize} represents the storage size you want to use. The format of ${storageSize} follows the quantity expression method of K8s, such as: 100Gi. Please replace them as needed when using.

Persistent BE log

To persist BE logs when using the default configuration, update the DorisCluster resource DorisCluster resource as follows:

```
beSpec:
  persistentVolumes:
  - mountPath: /opt/apache-doris/be/log
    name: belog
    persistentVolumeClaimSpec:
      storageClassName: ${your_storageclass}
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: ${storageSize}
```

In the above configuration, ${your_storageclass} represents the name of the StorageClass you want to use, and ${storageSize} represents the storage size you want to use. The format of ${storageSize} follows the quantity expression method of K8s, such as: 100Gi. Please replace them as needed when using.

Access configuration

Kubernetes provides the use of Service as VIP (Virtual IP) and load balancer. There are three external exposure modes for Service: ClusterIP, NodePort, and LoadBalancer.

ClusterIP

Doris provides the ClusterIP access mode by default on Kubernetes. The ClusterIP access mode provides an internal IP address within the Kubernetes cluster to expose services through this internal IP. With the ClusterIP mode, services can only be accessed within the cluster.

Step 1: Configure ClusterIP

Doris provides the ClusterIP access mode by default on Kubernetes. You can use the ClusterIP access mode without any modification.

Step 2: Obtain the Service

After deploying the cluster, you can view the services exposed by the Doris Operator using the following command:

```
kubectl -n doris get svc
```

The returned result is as follows:

```
NAME                              TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)                              AGE
doriscluster-sample-be-internal   ClusterIP   None          <none>        9050/TCP                             9m
doriscluster-sample-be-service    ClusterIP   10.1.68.128   <none>        9060/TCP,8040/TCP,9050/TCP,8060/TCP  9m
doriscluster-sample-fe-internal   ClusterIP   None          <none>        9030/TCP                             14m
doriscluster-sample-fe-service    ClusterIP   10.1.118.16   <none>        8030/TCP,9020/TCP,9030/TCP,9010/TCP  14m
```

In the above results, there are two types of services for FE and BE, with suffixes of “internal” and “service” respectively: - The services with the “internal” suffix can only be used for internal communication within Doris, such as heartbeat, data exchange, and other operations, and are not for external use. - The services with the “service” suffix can be used by users.

Step 3: Access doris from inside the container

You can create a pod containing the mysql client in the current Kubernetes cluster using the following command:

```
kubectl run mysql-client --image=mysql:5.7 -it --rm --restart=Never --namespace=doris -- /bin/bash
```

From within the container in the cluster, you can access the Doris cluster using the service name with the “service” suffix that is exposed externally:

```
mysql -uroot -P9030 -hdoriscluster-sample-fe-service
```

NodePort

To access Doris from outside the Kubernetes cluster, you can use the NodePort service type. There are two ways to allocate a port for the NodePort: dynamic allocation and static allocation.

- Dynamic Allocation: If the port is not explicitly set, Kubernetes will automatically allocate an unused port from the default range (30000-32767) when the pod is created. - Static Allocation:I f a port is explicitly specified, Kubernetes will allocate that port if it is available, ensuring it remains fixed.

Doris exposes the following ports for external access:

Port Name	default value	Port Description
Query Port	9030	Used to access the Doris cluster via the MySQL protocol
HTTP Port	8030	The http server port on FE, used to view FE information
Web Server Port	8040	The http server port on BE, used to view BE information

Step 1: Configure NodePort

FE NodePort

- Dynamic Allocation: `yaml spec: feSpec: service: type: NodePort` - Static Allocation: `yaml spec: feSpec: service: type: NodePort servicePorts: - nodePort: 31001 targetPort: 8030 - nodePort: 31002 targetPort: 9030`

BE NodePort - Dynamic Allocation: `yaml spec: beSpec: service: type: NodePort` - Static Allocation: `yaml beSpec: service: type: NodePort servicePorts: - nodePort: 31006 targetPort: 8040` ##### Step 2: Obtain the service After deploying the cluster, you can view the services exposed by the Doris Operator using the following command:

```
kubectl get service
```

The returned result is as follows:

```
NAME                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
kubernetes                        ClusterIP   10.152.183.1     <none>        443/TCP
169d
doriscluster-sample-fe-internal   ClusterIP   None             <none>        9030/TCP
2d
doriscluster-sample-fe-service    NodePort    10.152.183.58    <none>
8030:31041/TCP,9020:30783/TCP,9030:31545/TCP,9010:31610/TCP   2d
doriscluster-sample-be-internal   ClusterIP   None             <none>        9050/TCP
2d
doriscluster-sample-be-service    NodePort    10.152.183.244   <none>
9060:30940/TCP,8040:32713/TCP,9050:30621/TCP,8060:30926/TCP   2d
```

Step 3: Access service using NodePort

To access Doris via NodePort, you need to know the Node IP and the mapped port. You can retrieve the node IPs using:

```
  kubectl get nodes -owide
```

Example output::

```
NAME   STATUS   ROLES          AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE          KERNEL-VERSION          CONTAINER-
RUNTIME
r60    Ready    control-plane   14d   v1.28.2   192.168.88.60   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
r61    Ready    <none>          14d   v1.28.2   192.168.88.61   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
r62    Ready    <none>          14d   v1.28.2   192.168.88.62   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
r63    Ready    <none>          14d   v1.28.2   192.168.88.63   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
```

You can then use the IP address of any node (e.g., 192.168.88.61, 192.168.88.62, or 192.168.88.63) along with the mapped port to access Doris. For example, using node 192.168.88.62 and port 31545:

```
  mysql -h 192.168.88.62 -P 31545 -uroot
```

LoadBalancer

LoadBalancer service type provides an additional load balancer, typically offered by cloud service providers. This mode is only available when deploying the Doris cluster on Kubernetes clusters managed by a cloud platform. ##### Step 1: Configure the LoadBalancer mode

FE LoadBalancer

```
spec:
  feSpec:
    service:
      type: LoadBalancer
```

BE LoadBalancer

```
spec:
  beSpec:
    service:
      type: LoadBalancer
```

Step 2: Obtain the service

After deploying the cluster, you can view the services exposed by the Doris Operator using the following command:

```
kubectl get service
```

The returned result is as follows:

```
NAME                             TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)                                         AGE
kubernetes                       ClusterIP      10.152.183.1     <none>
443/TCP                                         169d
doriscluster-sample-fe-internal  ClusterIP      None             <none>
9030/TCP                                        2d
doriscluster-sample-fe-service   LoadBalancer   10.152.183.58    ac4828493dgrftb884g67wg4tb68gyut-1137856348.us-east-
```

```
1.elb.amazonaws.com         8030:31041/TCP,9020:30783/TCP,9030:31545/TCP,9010:31610/TCP   2d
doriscluster-sample-be-internal   ClusterIP      None             <none>
9050/TCP                                                  2d
doriscluster-sample-be-service    LoadBalancer   10.152.183.244   ac4828493dgrftb884g67wg4tb68gyut-1137823345.us-east-
1.elb.amazonaws.com         9060:30940/TCP,8040:32713/TCP,9050:30621/TCP,8060:30926/TCP   2d
```

Step 3: Access service using LoadBalancer

To access Doris through the LoadBalancer, use the external IP (provided in the EXTERNAL-IP field) and the corresponding port. For example, using the `mysql` command:

```
mysql -h ac4828493dgrftb884g67wg4tb68gyut-1137856348.us-east-1.elb.amazonaws.com -P 31545 -uroot
```

Configuring the username and password for the management cluster

Managing Doris nodes requires connecting to the live FE nodes via the MySQL protocol using a username and password for administrative operations. Doris implements a permission management mechanism similar to RBAC, where the user must have the Node_priv permission to perform node management. By default, the Doris Operator deploys the cluster with the root user in passwordless mode.

The process of configuring the username and password can be divided into three scenarios:
- initializing the root user password during cluster deployment; - automatically setting a non-root user with management permissions in the root passwordless deployment; - setting the root user password after deploying the cluster in root passwordless mode.

To secure access, you must configure a username and password with Node_Priv permission in the DorisCluster resource after adding a password to the root user. There are two ways to set up the username and password for managing the cluster nodes: - Using environment variables - Using a Kubernetes Secret

Configuring the root user password during cluster deployment

To set the root user's password securely, Doris supports encrypting it in `fe.conf` using a two-stage SHA-1 encryption process. Here's how to set up the password.

Step 1: Generate the root encrypted password

Use the following methods to encrypt the root password using two-stage SHA-1 encryption:

- Java Code:

```
import org.apache.commons.codec.digest.DigestUtils;

public static void main( String[] args ) {
      //the original password
      String a = "123456";
      String b = DigestUtils.sha1Hex(DigestUtils.sha1(a.getBytes())).toUpperCase();
      //output the 2 stage encrypted password.
      System.out.println("*"+b);
  }
```

- Golang Code:

```
import (
"crypto/sha1"
"encoding/hex"
"fmt"
"strings"
)

func main() {
    //original password
    plan := "123456"
    //the first stage encryption.
    h := sha1.New()
    h.Write([]byte(plan))
    eb := h.Sum(nil)

    //the two stage encryption.
    h.Reset()
    h.Write(eb)
    teb := h.Sum(nil)
    dst := hex.EncodeToString(teb)
    tes := strings.ToUpper(fmt.Sprintf("%s", dst))
    //output the 2 stage encrypted password.
    fmt.Println("*"+tes)
}
```

  Configure the encrypted password into `fe.conf` according to the requirements of the configuration file format. Then, Then, distribute the configuration to the Kubernetes cluster using a ConfigMap, describes in the Cluster Parameter Configuration Section.

Step 2: Configure the DorisCluster resource

After setting the root password in fe.conf, Doris will automatically apply the password to the first FE node when it starts. For other nodes to join the cluster, specify the username and password in the DorisCluster resource so that Doris Operator can perform automatic node management. - Using environment variables

Configure the username root and password into the ".spec.adminUser.name" and ".spec.adminUser.password" fields in the DorisCluster resource. Doris Operator will automatically convert the following configuration into environment variables for the container to use. The auxiliary services inside the container will use the username and password configured by the environment variables to add themselves to the specified cluster. The configuration format is as follows:

```
spec:
  adminUser:
    name: root
    password: ${password}
```

Here, ${password} is the unencrypted password of root.

- Using secret

  To securely manage the username and password, you can use a Kubernetes Basic Authentication Secret. Configure the Secret to store the root username and password and reference it in the DorisCluster resource.

a. Configure the Required Secret

Configure the required Basic authentication Secret according to the following format:

```
stringData:
  username: root
  password: ${password}
```

Here, ${password} is the unencrypted password set for root.

b. Configure the DorisCluster Resource to be Deployed

Configure the DorisCluster to specify the required Secret in the following format:

```
spec:
  authSecret: ${secretName}
```

Here, ${secretName} is the name of the Secret containing the root username and password.

Automatically creating non-root management users and psswords during deployment (Recommended)

For enhanced security, it is recommended to create a non-root user for management during the first deployment, rather than using the root user. In this method, the username and password for the non-root user are configured through environment variables or Secrets. The Doris container's auxiliary services will automatically create the user in the database, set the password, and grant the necessary Node_priv permission. After deployment, Doris Operator will use the newly created non-root username and password to manage the cluster nodes.

- Using environment variables:

  To configure a non-root user, you can set the username and password using environment variables in the DorisCluster resource: `yaml spec: adminUser: name: ${DB_ADMIN_USER} password: ${DB_ADMIN_PASSWD}`

  Here, ${DB_ADMIN_USER} is the newly created username, and ${DB_ADMIN_PASSWD} is the password set for the newly created username.

- Using Secret: To securely manage the username and password, you can use a Kubernetes Secret for basic authentication.

  a. Configure the required secret

  ```
  stringData:
    username: ${DB_ADMIN_USER}
    password: ${DB_ADMIN_PASSWD}
  ```

  Here, ${DB_ADMIN_USER} is the newly created username, and ${DB_ADMIN_PASSWD} is the password set for the newly created username.
  Deploy the Secret to the Kubernetes cluster by running:

  ```
  kubectl -n ${namespace} apply -f ${secretFileName}.yaml
  ```

Here, ${namespace} is the namespace where the DorisCluster resource needs to be deployed, and ${secretFileName} is the file name of the Secret to be deployed.

b. Configure the DorisCluster resource

Update the DorisCluster resource according to the following format:

```
spec:
  authSecret: ${secretName}
```

Here, ${secretName} is the name of the deployed Basic authentication Secret.

> **Tip**
>
> After deployment, please set the root password. Doris Operator will switch to using the automatically newly created username and password to manage the nodes. Please avoid deleting the automatically created user.

Setting the root user password after cluster deployment

After deploying the Doris cluster and setting the root user's password, it's essential to create a management user with the necessary Node_priv permission to allow Doris Operator to automatically manage the cluster nodes. Using the root user for this purpose is not recommended. Instead, please refer to the User Creation and Permission Assignment Section to create a new user and grant Node_priv permission.

Step 1: Create a user with Node_priv permission

First, connect to the Doris database using the MySQL protocol, then create a new user with the required permissions: `shell CREATE USER '${DB_ADMIN_USER}' IDENTIFIED BY '${DB_ADMIN_PASSWD}';`

- ${DB_ADMIN_USER}: The name of the user you wish to create.
- ${DB_ADMIN_PASSWD}: The password for the newly created user.

step 2: Grant Node_priv permission to the new user

Grant the Node_priv permission to the newly created user:

```
GRANT NODE_PRIV ON *.*.* TO ${DB_ADMIN_USER};
```

${DB_ADMIN_USER}: The username you created in the previous step.
For more details on creating users, setting passwords, and granting permissions, refer to the CREATE-USER section.

step 3: Configure DorisCluster

- Using environment variables

Directly configure the new user's name and password in the DorisCluster resource:

```
spec:
  adminUser:
    name: ${DB_ADMIN_USER}
    password: ${DB_ADMIN_PASSWD}
```

Here, ${DB_ADMIN_USER} is the newly created username, and ${DB_ADIC_PASSWD} is the password set for the newly created user.

- Using Secret To securely manage the username and password, you can use Kubernetes Secrets.

  a. Create the required secret
  Create a Basic Authentication Secret for the new user:

```
stringData:
  username: ${DB_ADMIN_USER}
  password: ${DB_ADMIN_PASSWD}
```

Here, ${DB_ADMIN_USER} is the newly created username, and ${DB_ADMIN_PASSWD} is the password set for the newly created username.
Deploy the Secret to the Kubernetes cluster with:

```
kubectl -n ${namespace} apply -f ${secretFileName}.yaml
```

Here, ${namespace} is the namespace where the DorisCluster resource needs to be deployed, and ${secretFileName} is the file name of the Secret to be deployed.

  b. Update the DorisCluster resource
  Once the Secret is deployed, update the DorisCluster resource to specify the Secret:

```
spec:
  authSecret: ${secretName}
```

Here, ${secretName} is the name of the deployed Basic authentication Secret.

> **Tip**
>
> After setting the root password and configuring the new username and password for managing nodes after deployment, the existing services will be restarted once in a rolling manner.

Automatic Service Restart on Configuration Changes

Doris specifies startup parameters through configuration files. While most parameters can be modified through web interfaces and take effect immediately, certain parameters requiring service restart can now be automatically handled through Doris Operator's restart capability introduced in version 25.1.0.
To enable this functionality in a `DorisCluster` resource, configure:

```
spec:
  enableRestartWhenConfigChange: true
```

When this configuration is present, Doris Operator will:
1. Monitor changes to cluster startup configurations (mounted via ConfigMap, see Customizing Startup Configurations).
2. Automatically restart affected services when configurations change.

Example Usage

Support configmap monitoring and restart for FE and BE, Use FE usage as example. 1. Sample DorisCluster deployment specification: `yaml spec: enableRestartWhenConfigChange: true feSpec: image: apache/doris:fe-2.1.8 replicas: 1 configMapInfo: configMapName: fe-configmap` 2. Update FE service configurations.
When modifying values under the `fe.conf` key in the fe-configmap ConfigMap (containing FE service configurations), Doris Operator will automatically perform a rolling restart of FE services to apply changes.

Using Kerberos Authentication

The Doris Operator has supported Kerberos authentication for Doris (versions 2.1.9, 3.0.4, and later) in Kubernetes since version 25.2.0. To enable Kerberos authentication in Doris, both the krb5.conf file and keytab files are required. The Doris Operator mounts the krb5.conf file using a ConfigMap resource and mounts the keytab files using a Secret resource. The workflow for enabling Kerberos authentication is as follows:

1. Create a ConfigMap containing the krb5.conf file: `shell kubectl create -n ${namespace} configmap ${name} --from-file=krb5.conf` Replace ${namespace} with the namespace where the DorisCluster is deployed, and ${name} with the desired name for the ConfigMap.
2. Create a Secret containing the keytab files: `shell kubectl create -n ${namespace} secret generic ${name} --from-file=${xxx.keytab}` Replace ${namespace} with the namespace where the DorisCluster is deployed, and ${name} with the desired name for the Secret. If multiple keytab files need to be mounted, refer to the kubectl create Secret documentation to include them in a single Secret.
3. Configure the DorisCluster resource to specify the ConfigMap containing krb5.conf and the Secret containing keytab files: `yaml spec: kerberosInfo: krb5ConfigMap: ${krb5ConfigMapName} keytabSecretName: ${keytabSecretName} keytabPath: ${keytabPath}` ${krb5ConfigMapName}: Name of the ConfigMap containing the krb5.conf file. ${keytabSecretName}: Name of the Secret containing the keytab files. ${keytabPath}: The directory path in the container where the Secret mounts the keytab files. This path should match the directory specified by hadoop.kerberos.keytab when creating a catalog. For catalog configuration details, refer to the Hive Catalog configuration documentation.

Configure Shared Storage

As of version 25.4.0, the Doris Operator supports mounting shared storage with the ReadWriteMany access mode to all pods across multiple components. Before using this feature, ensure that the shared storage PersistentVolume and PersistentVolumeClaim resources have been created. Configure the DorisCluster resource as shown below before deploying the Doris cluster:

```
spec:
  sharedPersistentVolumeClaims:
  - mountPath: ${mountPath}
```

```
    persistentVolumeClaimName: ${sharedPVCName}
    supportComponents:
    - fe
    - be
```

- `${mountPath}` specifies the absolute path inside the container where the storage will be mounted.
- `${sharedPVCName}` refers to the name of the `PersistentVolumeClaim` to be mounted.
- `supportComponents` lists the names of the components that require the shared storage. In the example above, both the FE and BE components will mount the shared storage. If the supportComponents array is left empty, all deployed components will mount the shared storage by default.

> **Tip**
>
> The `mountPath` parameter can use `${DORIS_HOME}` as a prefix. When `${DORIS_HOME}` is used, it resolves to `/opt/apache-doris/fe` within FE containers and `/opt/apache-doris/be` within BE containers.

Configuring Probe Timeouts

DorisCluster provides two types of probe timeout configurations for each service: `startup probe timeout` and `liveness probe timeout`. If a service fails to start within the specified startup timeout period, it is considered to have failed and will be restarted. If a service becomes unresponsive for longer than the specified liveness timeout, the corresponding Pod will be automatically restarted.

Startup Probe Timeout

- FE Service Startup Timeout Configuration
  `spec: feSpec: startTimeout: 3600` The above configuration sets the FE service startup timeout to 3600 seconds.
- BE Service Startup Timeout Configuration
  `spec: beSpec: startTimeout: 3600` #### Liveness Probe Timeout
- FE Service Liveness Timeout Configuration
  `spec: feSpec: liveTimeout: 60` The above configuration sets the FE service liveness timeout to 60 seconds.
- BE Service Liveness Timeout Configuration
  `spec: beSpec: liveTimeout: 60` The above configuration sets the BE service liveness timeout to 60 seconds.

Deploy Doris Cluster

To deploy a Doris cluster on Kubernetes, ensure that the Doris Operator is deployed.
The deployment process for a Doris cluster consists of three steps: download the deployment template, custom the template and deploy cluster, and verify the cluster status. ### Step 1: Download the Doris deployment template

```
curl -O https://raw.githubusercontent.com/apache/doris-operator/master/doc/examples/doriscluster-sample.yaml
```

Step 2: Custom the template and deploy cluster

Perform customized configuration as needed according to the doc of Config Doris to Deploy. After the configuration is completed, deploy it with the following command:

```
kubectl apply -f doriscluster-sample.yaml
```

Step 3: Verify the cluster status

- Check the status of the cluster by checking the status of pods

  ```
  kubectl get pods
  ```

  Expected output:

  ```
  NAME                      READY   STATUS    RESTARTS   AGE
  doriscluster-sample-fe-0  1/1     Running   0          2m
  doriscluster-sample-be-0  1/1     Running   0          3m
  ```

- Check the status of the deployed resources

  ```
  kubectl get dcr -n doris
  ```

  Expected output:

  ```
  NAME                FESTATUS    BESTATUS    CNSTATUS   BROKERSTATUS
  doriscluster-sample  available   available
  ```

Access Cluster

Kubernetes provides the use of Service as VIP (Virtual IP) and load balancer. There are three external exposure modes for Service: ClusterIP, NodePort, and LoadBalancer. ### ClusterIP Doris provides the ClusterIP access mode by default on Kubernetes. The ClusterIP access mode provides an internal IP address within the Kubernetes cluster to expose services through this internal IP. With the ClusterIP mode, services can only be accessed within the cluster. #### Step 1: Obtain the Service After deploying the cluster, you can view the services exposed by the Doris Operator using the following command:

```
kubectl -n doris get svc
```

The returned result is as follows:

```
NAME                              TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)                               AGE
doriscluster-sample-be-internal   ClusterIP   None          <none>        9050/TCP                              9m
doriscluster-sample-be-service    ClusterIP   10.1.68.128   <none>        9060/TCP,8040/TCP,9050/TCP,8060/TCP   9m
doriscluster-sample-fe-internal   ClusterIP   None          <none>        9030/TCP                              14m
doriscluster-sample-fe-service    ClusterIP   10.1.118.16   <none>        8030/TCP,9020/TCP,9030/TCP,9010/TCP   14m
```

In the above results, there are two types of services for FE and BE, with suffixes of “internal” and “service” respectively: - The services with the “internal” suffix can only be used for internal communication within Doris, such as heartbeat, data exchange, and other operations, and are not for external use. - The services with the “service” suffix can be used by users.

Step 2: Access Doris

You can create a pod containing the mysql client in the current Kubernetes cluster using the following command:

```
kubectl run mysql-client --image=mysql:5.7 -it --rm --restart=Never --namespace=doris -- /bin/bash
```

From within the container in the cluster, you can access the Doris cluster using the service name with the “service” suffix that is exposed externally:

```
mysql -uroot -P9030 -hdoriscluster-sample-fe-service
```

NodePort

According to the DorisCluster access configuration section, after configuring the access mode to use NodePort, you can access FE using the MySQL protocol in root password-free mode. The steps are as follows: #### Step 1: Obtain the service After deploying the cluster, you can view the services exposed by the Doris Operator using the following command:

```
kubectl get service
```

The returned result is as follows:

```
NAME                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
kubernetes                        ClusterIP   10.152.183.1     <none>        443/TCP
169d
doriscluster-sample-fe-internal   ClusterIP   None             <none>        9030/TCP
2d
doriscluster-sample-fe-service    NodePort    10.152.183.58    <none>
8030:31041/TCP,9020:30783/TCP,9030:31545/TCP,9010:31610/TCP   2d
doriscluster-sample-be-internal   ClusterIP   None             <none>        9050/TCP
2d
doriscluster-sample-be-service    NodePort    10.152.183.244   <none>
9060:30940/TCP,8040:32713/TCP,9050:30621/TCP,8060:30926/TCP   2d
```

Step 2: Access Doris

To access Doris via NodePort, you need to know the Node IP and the mapped port. You can retrieve the node IPs using:

```
kubectl get nodes -owide
```

Example output::

```
NAME   STATUS   ROLES           AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE          KERNEL-VERSION          CONTAINER-
RUNTIME
r60    Ready    control-plane   14d   v1.28.2   192.168.88.60   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
r61    Ready    <none>          14d   v1.28.2   192.168.88.61   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
r62    Ready    <none>          14d   v1.28.2   192.168.88.62   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
r63    Ready    <none>          14d   v1.28.2   192.168.88.63   <none>        CentOS Stream 8   4.18.0-294.el8.x86_64
containerd://1.6.22
```

You can then use the IP address of any node (e.g., 192.168.88.61, 192.168.88.62, or 192.168.88.63) along with the mapped port to access Doris. For example, using node 192.168.88.62 and port 31545:

```
mysql -h 192.168.88.62 -P 31545 -uroot
```

LoadBalancer

According to the DorisCluster access configuration section, on a public cloud platform, after configuring the access mode to use LoadBalancer, you can access FE using the MySQL protocol in root password-free mode. The steps are as follows: #### Step 1: Obtain the service After deploying the cluster, you can view the services exposed by the Doris Operator using the following command:

```
kubectl get service
```

The returned result is as follows:

```
NAME                              TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)                                                 AGE
kubernetes                        ClusterIP      10.152.183.1     <none>
443/TCP                                                 169d
doriscluster-sample-fe-internal   ClusterIP      None             <none>
9030/TCP                                                2d
doriscluster-sample-fe-service    LoadBalancer   10.152.183.58    ac4828493dgrftb884g67wg4tb68gyut-1137856348.us-east-
1.elb.amazonaws.com         8030:31041/TCP,9020:30783/TCP,9030:31545/TCP,9010:31610/TCP   2d
doriscluster-sample-be-internal   ClusterIP      None             <none>
9050/TCP                                                2d
doriscluster-sample-be-service    LoadBalancer   10.152.183.244   ac4828493dgrftb884g67wg4tb68gyut-1137823345.us-east-
1.elb.amazonaws.com         9060:30940/TCP,8040:32713/TCP,9050:30621/TCP,8060:30926/TCP   2d
```

Step 2: Access Doris

To access Doris through the LoadBalancer, use the external IP (provided in the EXTERNAL-IP field) and the corresponding port. For example, using the `mysql` command:

```
mysql -h ac4828493dgrftb884g67wg4tb68gyut-1137856348.us-east-1.elb.amazonaws.com -P 31545 -uroot
```

StreamLoad Access to Doris Deployed on Kubernetes

Doris supports data import using the StreamLoad method. When the client and the Doris cluster are within the same local network, the client can directly use the Frontend (FE) address as the request endpoint. The FE service responds with an HTTP 301 status code and provides the Backend (BE) address, instructing the client to redirect the request to the BE for data import.

However, when Doris is deployed on Kubernetes, internal communication uses addresses that are only accessible within the Kubernetes cluster. If the FE returns a BE address that is only reachable internally via the 301 redirect mechanism, data import attempts from clients outside the Kubernetes cluster will fail.

To import data using StreamLoad from a client located outside the Kubernetes environment, you must configure the import address with a BE address that is externally accessible.

Configure External Access to the BE Service

To enable access to the BE service from outside the Kubernetes cluster, configure the service as either a NodePort or a LoadBalancer. Update the `DorisCluster` resource accordingly to apply these changes.

Configure the BE Proxy Address

As the description of NodePort or LoadBalancer to get an externally accessible address and the corresponding `web_server` port. Use this address and port as the request endpoint when importing data via StreamLoad.

Cluster Operation

How to enter the container when the pod crashes

In the k8s environment, the service will enter the `CrashLoopBackOff` state due to some unexpected things. You can view the pod status and pod_name under the specified namespace through the `kubectl get pod --namespace ${namespace}` command.

In this state, the cause of the service problem cannot be determined simply by using the describe and logs commands. When the service enters the `CrashLoopBackOff` state, there needs to be a mechanism that allows the pod deploying the service to enter the `running` state so that users can enter the container for debugging through exec.

doris-operator provides a `Debug` running mode. The following describes how to enter Debug mode for manual debugging when the service enters `CrashLoopBackOff`, and how to return to normal startup state after solving the problem.

Start Debug mode

When a pod of the service enters CrashLoopBackOff or cannot be started normally during normal operation, take the following steps to put the service into `Debug` mode and manually start the service to find the problem.

1. **Use the following command to add annotation to the pod with problems.**

```
$ kubectl annotate pod ${pod_name} --namespace ${namespace} apache.org.doris/runmode=debug
```

When the service is restarted next time, the service will detect the annotation that identifies the `Debug` mode startup, and will enter the `Debug` mode to start, and the pod status will be `running`.

2. **When the service enters `Debug` mode, the pod of the service is displayed in a normal state. Users can enter the inside of the pod through the following command**

```
$ kubectl --namespace ${namespace} exec -ti ${pod_name} bash
```

3. **Manually start the service under `Debug`. When the user enters the pod, manually execute the `start_xx.sh` script by modifying the port of the corresponding configuration file. The script directory is under `/opt/apache-doris/xx/bin`.**

FE needs to modify `query_port`, BE needs to modify `heartbeat_service_port` The main purpose is to avoid misleading the flow by accessing the crashed node through service in `Debug` mode.

Exit Debug mode

When the service locates the problem, it needs to exit the `Debug` operation. At this time, you only need to delete the corresponding pod according to the following command, and the service will start in the normal mode.

```
$ kubectl delete pod ${pod_name} --namespace ${namespace}
```

> **Tip**
>
> **After entering the pod, you need to modify the port information of the configuration file before you can manually start the corresponding Doris component.**
>
> - FE needs to modify the `query_port=9030` configuration with the default path: `/opt/apache-doris/fe/conf/fe.conf`.
> - BE needs to modify the `heartbeat_service_port=9050` configuration with the default path: `/opt/apache-doris/be/conf/be.conf`.

Upgrading doris cluster

This document describes how to use updates to upgrade an Apache Doris cluster based on a Doris Operator deployment.

Similar to conventionally deployed cluster upgrades, Doris clusters deployed by Doris Operator still require rolling upgrades from BE to FE nodes. Doris Operator is based on Kubernetes' Performing a Rolling Update provides rolling upgrade capabilities.

Things to note before upgrading

- It is recommended that the upgrade operation be performed during off-peak periods.
- During the rolling upgrade process, the connection to the closed node will fail, causing the request to fail. For this type of business, it is recommended to add retry capabilities to the client.
- Before upgrading, you can read the General Upgrade Manual to help you understand some principles and precautions during the upgrade. .
- The compatibility of data and metadata cannot be verified before upgrading. Therefore, cluster upgrade must avoid single copy of data and single FE FOLLOWER node in the cluster.
- Nodes will be restarted during the upgrade process, so unnecessary cluster balancing and replica repair logic may be triggered. Please shut it down first with the following command.

```
admin set frontend config("disable_balance" = "true");
admin set frontend config("disable_colocate_balance" = "true");
admin set frontend config("disable_tablet_scheduler" = "true");
```

- When upgrading Doris, please follow the principle of not upgrading across two or more key node versions. If you want to upgrade across multiple key node versions, upgrade to the latest key node version first, and then upgrade in sequence. If it is a non-key node version, You can ignore skipping. For details, please refer to Upgrade Version Instructions

Upgrade operation

The order of node types in the upgrade process is as follows. If a certain type of node does not exist, it will be skipped:

```
cn/be -> fe -> broker
```

It is recommended to modify the `image` of the corresponding cluster components in sequence and then apply the configuration. After the current type of component is fully upgraded and the status returns to normal, the rolling upgrade of the next type of node can be performed.

Upgrade BE

If you retain the cluster's crd (Doris Operator defines the abbreviation of `DorisCluster` type resource name) file, you can upgrade by modifying the configuration file and running the `kubectl apply` command.

1. Modify `spec.beSpec.image`

Change `apache/doris:be-2.1.8` to `apache/doris:be-2.1.9` `shell $ vim doriscluster-sample.yaml`

2. Save the changes and apply the changes to be upgraded:

```
$ kubectl apply -f doriscluster-sample.yaml -n doris
```

It can also be modified directly through `kubectl edit dcr`.

1. Check the dcr list under namespace 'doris' to obtain the `cluster_name` that needs to be updated.

```
$ kubectl get dcr -n doris
NAME                  FESTATUS    BESTATUS    CNSTATUS
doriscluster-sample   available   available
```

2. Modify, save and take effect

```
$ kubectl edit dcr doriscluster-sample -n doris
```

After entering the text editor, you will find `spec.beSpec.image` and change `apache/doris:be-2.1.8` to `apache/doris:be-2.1.9`

3. View the upgrade process and results:

```
$ kubectl get pod -n doris
```

When all Pods are rebuilt and enter the Running state, the upgrade is complete.

Upgrade FE

If you retain the cluster's crd (Doris Operator defines the abbreviation of the `DorisCluster` type resource name) file, you can upgrade by modifying the configuration file and running the `kubectl apply` command.

1. Modify `spec.feSpec.image`

Change `apache/doris:fe-2.1.8` to `apache/doris:fe-2.1.9` `shell $ vim doriscluster-sample.yaml`

2. Save the changes and apply the changes to be upgraded:

```
$ kubectl apply -f doriscluster-sample.yaml -n doris
```

It can also be modified directly through `kubectl edit dcr`.

1. Modify, save and take effect

```
$ kubectl edit dcr doriscluster-sample -n doris
```

After entering the text editor, you will find `spec.feSpec.image` and change `apache/doris:fe-2.1.8` to `apache/doris:fe-2.1.9`

2. View the upgrade process and results:

```
$ kubectl get pod -n doris
```

When all Pods are rebuilt and enter the Running state, the upgrade is complete.

After the upgrade is completed

Verify cluster node status

Access Doris through `mysql-client` through the method provided in the Access Doris Cluster document. Use SQL such as `show frontends` and `show backends` to view the version and status of each component.

```
mysql> show frontends\G;
*************************** 1. row ***************************
              Name: fe_13c132aa_3281_4f4f_97e8_655d01287425
              Host: doriscluster-sample-fe-0.doriscluster-sample-fe-internal.doris.svc.cluster.local
       EditLogPort: 9010
          HttpPort: 8030
         QueryPort: 9030
           RpcPort: 9020
ArrowFlightSqlPort: -1
              Role: FOLLOWER
          IsMaster: false
         ClusterId: 1779160761
              Join: true
             Alive: true
 ReplayedJournalId: 2422
     LastStartTime: 2024-02-19 06:38:47
     LastHeartbeat: 2024-02-19 09:31:33
          IsHelper: true
            ErrMsg:
           Version: doris-2.1.0
  CurrentConnected: Yes
*************************** 2. row ***************************
              Name: fe_f1a9d008_d110_4780_8e60_13d392faa54e
              Host: doriscluster-sample-fe-2.doriscluster-sample-fe-internal.doris.svc.cluster.local
       EditLogPort: 9010
          HttpPort: 8030
         QueryPort: 9030
           RpcPort: 9020
ArrowFlightSqlPort: -1
```

```
             Role: FOLLOWER
         IsMaster: true
        ClusterId: 1779160761
             Join: true
            Alive: true
 ReplayedJournalId: 2423
    LastStartTime: 2024-02-19 06:37:35
    LastHeartbeat: 2024-02-19 09:31:33
         IsHelper: true
           ErrMsg:
          Version: doris-2.1.0
  CurrentConnected: No
*************************** 3. row ***************************
             Name: fe_e42bf9da_006f_4302_b861_770d2c955a47
             Host: doriscluster-sample-fe-1.doriscluster-sample-fe-internal.doris.svc.cluster.local
       EditLogPort: 9010
          HttpPort: 8030
         QueryPort: 9030
           RpcPort: 9020
ArrowFlightSqlPort: -1
             Role: FOLLOWER
         IsMaster: false
        ClusterId: 1779160761
             Join: true
            Alive: true
 ReplayedJournalId: 2422
    LastStartTime: 2024-02-19 06:38:17
    LastHeartbeat: 2024-02-19 09:31:33
         IsHelper: true
           ErrMsg:
          Version: doris-2.1.0
  CurrentConnected: No
3 rows in set (0.02 sec)
```

If the `Alive` status of the FE node is true and the `Version` value is the new version, the FE node is upgraded successfully.

```
mysql> show backends\G;
*************************** 1. row ***************************
             BackendId: 10002
                  Host: doriscluster-sample-be-0.doriscluster-sample-be-internal.doris.svc.cluster.local
         HeartbeatPort: 9050
                BePort: 9060
              HttpPort: 8040
              BrpcPort: 8060
    ArrowFlightSqlPort: -1
         LastStartTime: 2024-02-19 06:37:56
         LastHeartbeat: 2024-02-19 09:32:43
                 Alive: true
  SystemDecommissioned: false
            TabletNum: 14
      DataUsedCapacity: 0.000
    TrashUsedCapcacity: 0.000
         AvailCapacity: 12.719 GB
         TotalCapacity: 295.167 GB
               UsedPct: 95.69 %
        MaxDiskUsedPct: 95.69 %
    RemoteUsedCapacity: 0.000
                   Tag: {"location" : "default"}
                ErrMsg:
               Version: doris-2.1.0
                Status: {"lastSuccessReportTabletsTime":"2024-02-19
09:31:48","lastStreamLoadTime":-1,"isQueryDisabled":false,"isLoadDisabled":false}
```

```
HeartbeatFailureCounter: 0
               NodeRole: mix
*************************** 2. row ***************************
              BackendId: 10003
                   Host: doriscluster-sample-be-1.doriscluster-sample-be-internal.doris.svc.cluster.local
          HeartbeatPort: 9050
                 BePort: 9060
               HttpPort: 8040
               BrpcPort: 8060
     ArrowFlightSqlPort: -1
          LastStartTime: 2024-02-19 06:37:35
          LastHeartbeat: 2024-02-19 09:32:43
                  Alive: true
   SystemDecommissioned: false
              TabletNum: 8
       DataUsedCapacity: 0.000
     TrashUsedCapcacity: 0.000
          AvailCapacity: 12.719 GB
          TotalCapacity: 295.167 GB
                UsedPct: 95.69 %
         MaxDiskUsedPct: 95.69 %
     RemoteUsedCapacity: 0.000
                    Tag: {"location" : "default"}
                 ErrMsg:
                Version: doris-2.1.0
                 Status: {"lastSuccessReportTabletsTime":"2024-02-19
09:31:43","lastStreamLoadTime":-1,"isQueryDisabled":false,"isLoadDisabled":false}
HeartbeatFailureCounter: 0
               NodeRole: mix
*************************** 3. row ***************************
              BackendId: 11024
                   Host: doriscluster-sample-be-2.doriscluster-sample-be-internal.doris.svc.cluster.local
          HeartbeatPort: 9050
                 BePort: 9060
               HttpPort: 8040
               BrpcPort: 8060
     ArrowFlightSqlPort: -1
          LastStartTime: 2024-02-19 08:50:36
          LastHeartbeat: 2024-02-19 09:32:43
                  Alive: true
   SystemDecommissioned: false
              TabletNum: 0
       DataUsedCapacity: 0.000
     TrashUsedCapcacity: 0.000
          AvailCapacity: 12.719 GB
          TotalCapacity: 295.167 GB
                UsedPct: 95.69 %
         MaxDiskUsedPct: 95.69 %
     RemoteUsedCapacity: 0.000
                    Tag: {"location" : "default"}
                 ErrMsg:
                Version: doris-2.1.0
                 Status: {"lastSuccessReportTabletsTime":"2024-02-19
09:32:04","lastStreamLoadTime":-1,"isQueryDisabled":false,"isLoadDisabled":false}
HeartbeatFailureCounter: 0
               NodeRole: mix
3 rows in set (0.01 sec)
```

If the `Alive` status of the BE node is true and the `Version` value is the new version, the BE node is upgraded successfully.

Restore cluster replica synchronization and balancing

After confirming that the status of each node is correct, execute the following SQL to restore cluster balancing and replica repair:

```
admin set frontend config("disable_balance" = "false");
admin set frontend config("disable_colocate_balance" = "false");
admin set frontend config("disable_tablet_scheduler" = "false");
```

Starting FE with `metadata_failure_recovery` Mode

When the Frontend (FE) service is unable to elect a leader and becomes unavailable, you can recover the cluster by selecting the node with the highest `VLSN` and force-starting it as the master using the recovery mechanism.

Starting in Recovery Mode in a Containerized Environment

1. Identify the node with the highest `VLSN`
   In Kubernetes, each time an FE Pod starts, it outputs the last 10 `VLSN` records of the node. An example is shown below: `the annotations value: the value not equal! debug /opt/apache-doris/fe/doris-meta/bdb/je.info.0:19:2025-08-05 03:42:47.650 UTC INFO [fe_f35530c4_3ff1_48fe_80d1_cc8e32dbc942] Replica-feeder fe_d8763579_92da_4d72_8c58_4e62b88bdff0 start stream at VLSN: 30 /opt/apache-doris/fe/doris-meta/bdb/je.info.0:21:2025-08-05 03:42:47.659 UTC INFO [fe_f35530c4_3ff1_48fe_80d1_cc8e32dbc942] Replica initialization completed. Replica VLSN: -1 Heartbeat master commit VLSN: 49 DTVLSN:0 Replica VLSN delta: 50 [Tue Aug 5 06:14:05 UTC 2025] start with meta run start_fe.sh with additional options: '--console'` In this example, the highest `VLSN` on the current node is 30, as indicated by the log prefix `start stream at VLSN:`.
2. Designate the Pod with the highest `VLSN` for recovery
   After identifying the Pod corresponding to the node with the highest `VLSN`, annotate it to enable the recovery mechanism: `kubectl annotate pod {podName} "selectdb.com.doris/recovery=true"` Upon restarting, the Pod will automatically append the `--metadata_failure_recovery` flag to its startup command and start in recovery mode.
3. Remove the annotation after recovery
   Once the FE service is running normally, make sure to remove the annotation added in Step 2 to avoid unexpected behavior during future restarts.

> **Note**
>
> 1. After adding the annotation, do not restart the Pod using kubectl delete pod, as this will remove the annotation. Instead, allow kubelet to restart it automatically or manually kill the process inside the container.
> 2. Starting FE in `metadata_failure_recovery` mode can take a long time due to extensive log replay. Before proceeding, increase the FE service's startup probe timeout, and delete all FE Pods before initiating the recovery startup.

Deploying on AWS

To facilitate a quick experience of Doris on AWS, we have provided a CloudFormation template (CFT) that allows for rapid cluster launch and operation. With this template, you can automatically configure AWS resources and launch a Doris cluster with minimal configuration required.

Alternatively, you can also purchase AWS resources independently and deploy the cluster manually using standard methods.

What’s AWS CloudFormation?

CloudFormation enables users to create a “stack of resources” in just one step. Resources refer to the items created by users, such as EC2 instances, VPCs, subnets, and more. A group of such resources is referred to as a stack. Users can write a template that easily allows them to create a resource stack according to their preferences in a single step. This is faster, more repeatable, and offers better consistency compared to manual creation and configuration. Additionally, templates can be placed into source code for version control, enabling their use for any purpose whenever needed.

What’s Doris on AWS CloudFormation?

Currently, Doris provides the Doris CloudFormation Template, which allows users to quickly create a cluster of the relevant Doris version on AWS by directly using this template, enabling them to experience the latest Doris features.

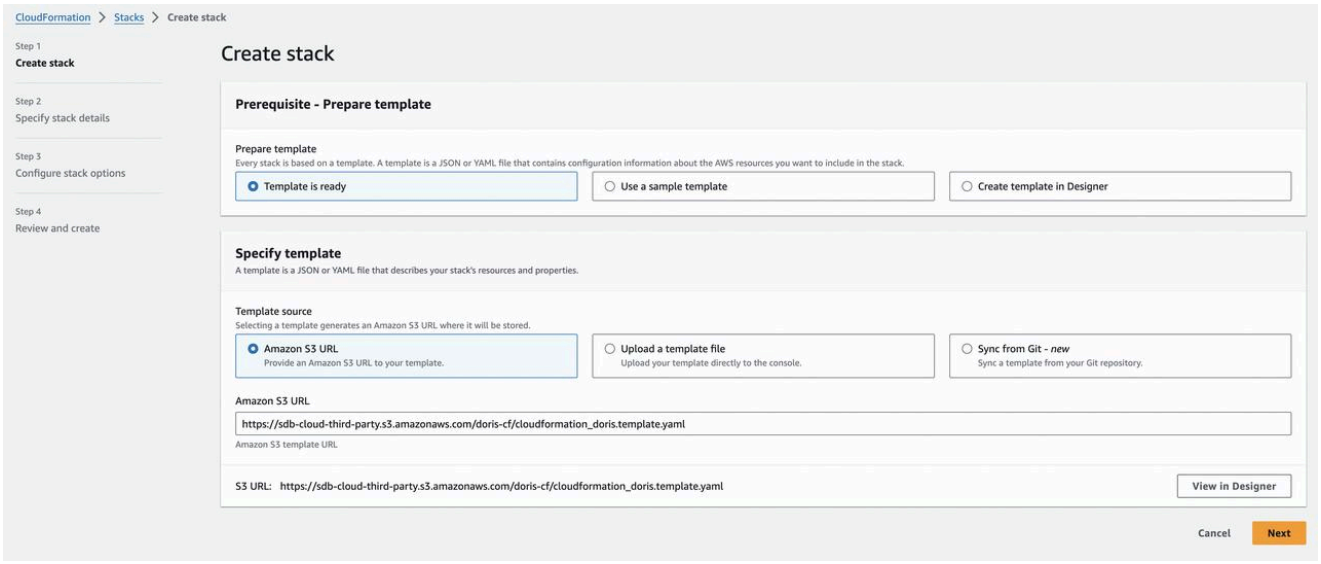
> ****Note: ** The template for building Doris clusters based on CloudFormation currently only supports the regions of us-east-1, us-west-1, and us-west-2. Doris on AWS CloudFormation is primarily intended for testing or experiencing purposes, and should not be used in production environments.**

Precautions for Use

- Determine the VPC and Subnet that will be deployed.
- Determine the key pair that will be used to log into the nodes.
- A VPC Endpoint Interface for S3 will be established during deployment.

Start Deployment

1. On the AWS console, navigate to CloudFormation and click on “Create stack”.

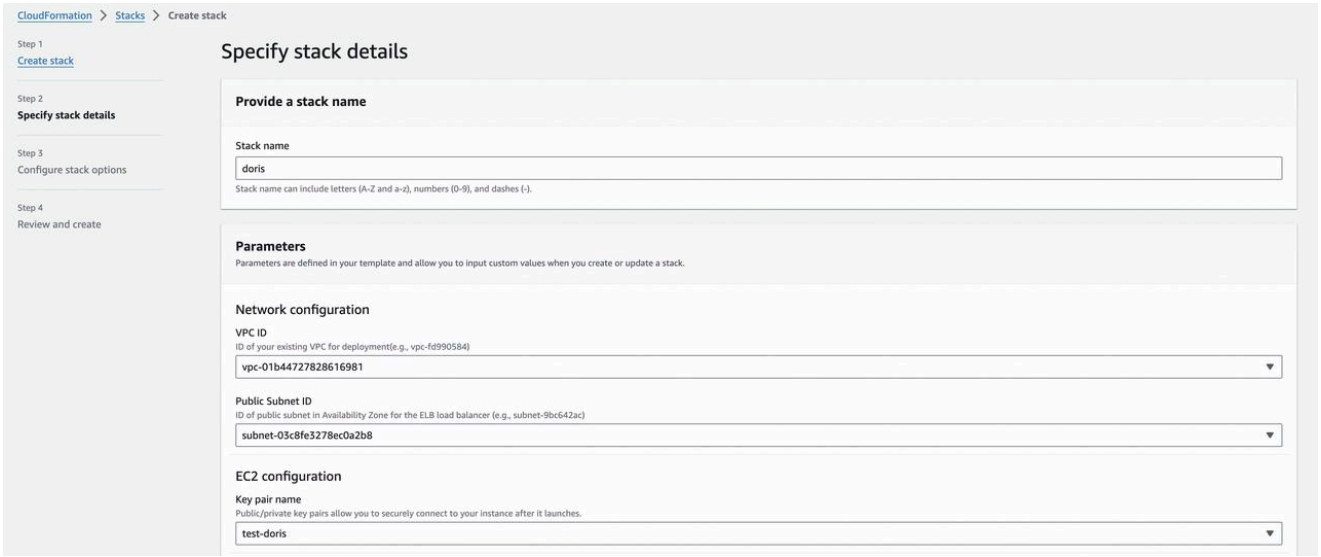


Start Deployment

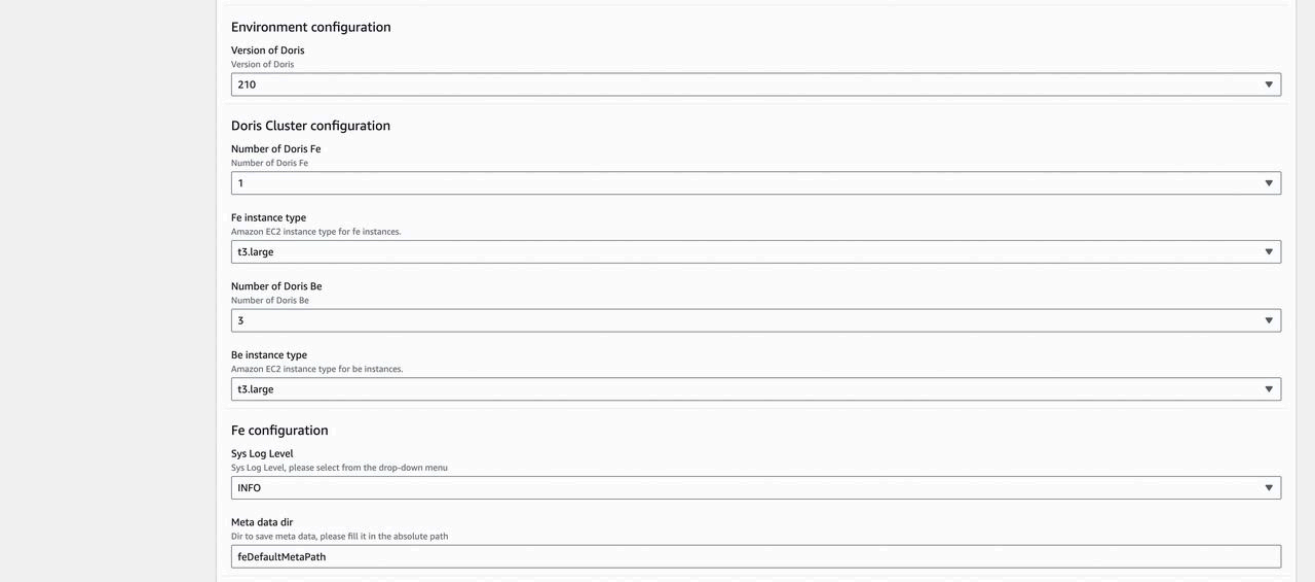
Select the “Amazon S3 URL Template source” option, and fill in the “Amazon S3 URL” field with the following template link:

https://sdb-cloud-third-party.s3.amazonaws.com/doris-cf/cloudformation_doris.template.yaml

2. Configure the specific parameters of the template



Configure the specific parameter



Configure the specific parameter



Configure the specific parameter

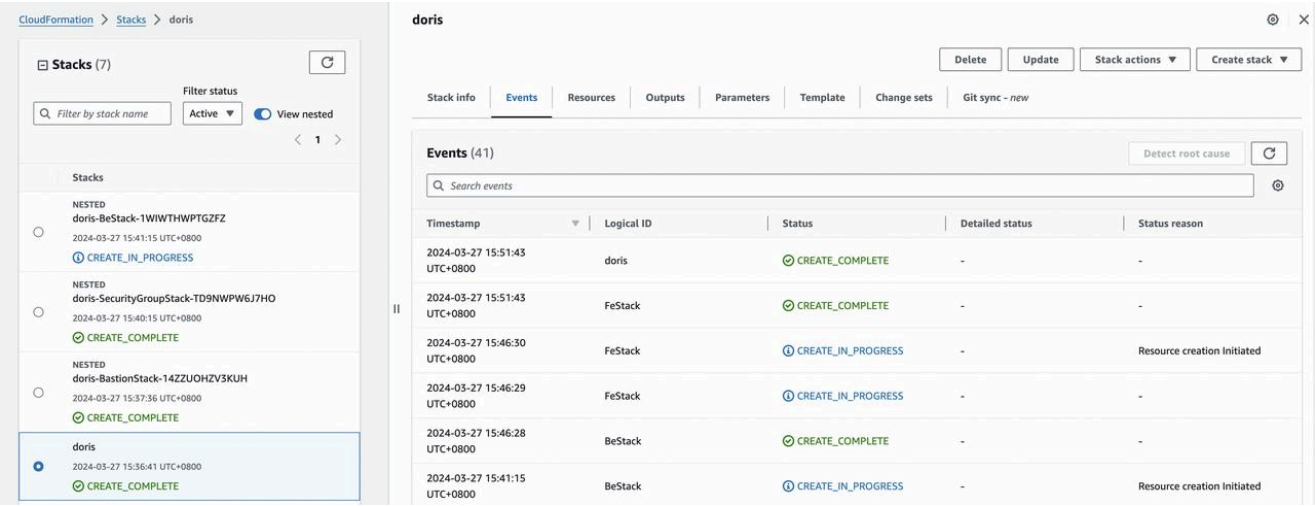
The main parameters are described as follows:

- **VPC ID**: The VPC where the deployment will be performed.
- **Subnet ID**: The subnet where the deployment will be deployed.
- **Key pair name**: The public/private key pairs used to connect to the deployed BE and FE nodes.
- **Version of Doris**: The version of Doris to be deployed, such as 2.1.0, 2.0.6, etc.
- **Number of Doris FE**: The number of FE nodes. The template defaults to selecting only 1 FE.
- **Fe instance type**: The node type of FE, and the default value can be used.
- **Number of Doris Be**: The number of BE nodes, which can be 1 or 3.
- **Be instance type**: The node type of BE, and the default value can be used.
- **Meta data dir**: The metadata directory of the FE node, and the default value can be used.
- **Sys log level:** Sets the level of system logs, and the default value of “info” can be used.
- **Volume type of Be nodes:** The volume type of EBS mounted on BE nodes. Each node is mounted with one disk by default. The default value can be used.

- **Volume size of Be nodes**: The size of EBS mounted on BE nodes, measured in GB. The default value can be used.

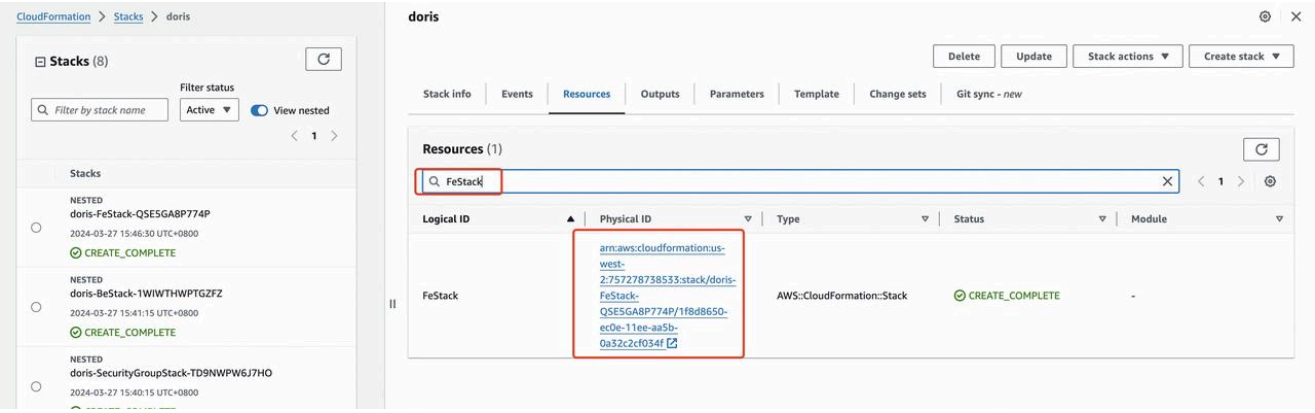
How to Connect to the Database

1. The display after successful deployment is as follows:

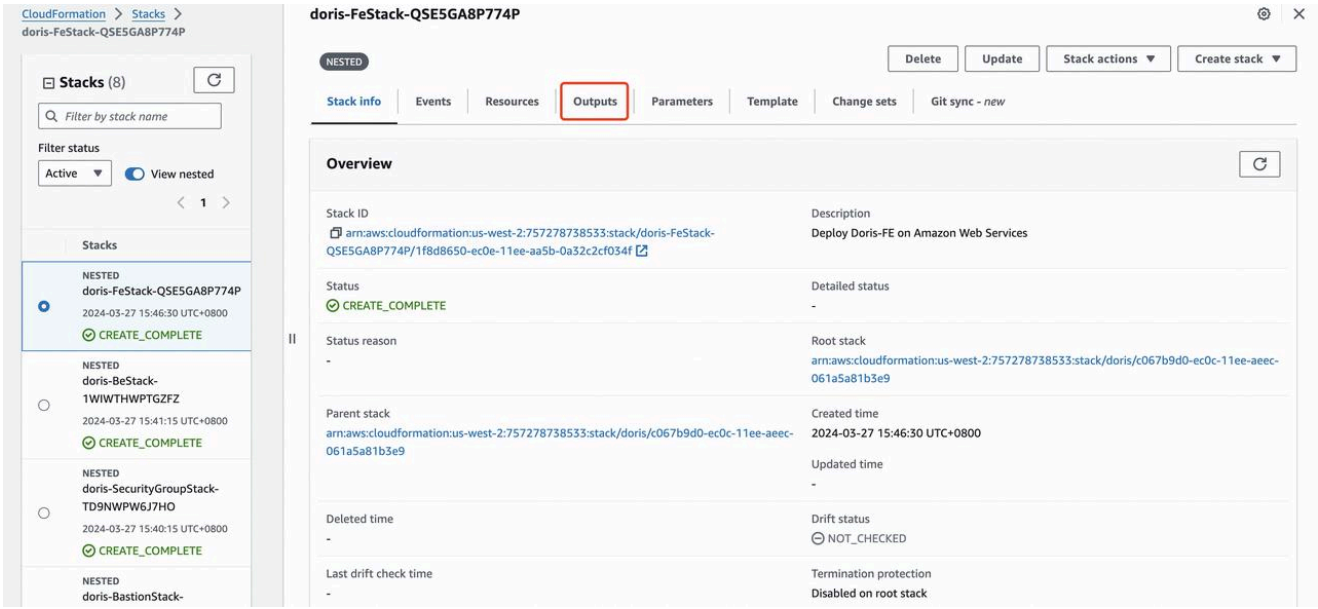


How to Connect to the Database

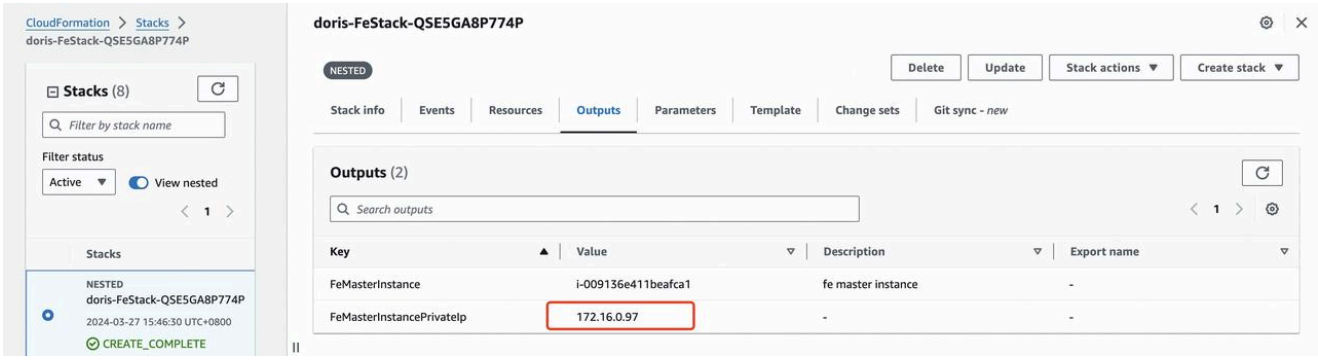
2. Next, find the connection address for FE as follows. In this example, you can view the address as 172.16.0.97 from the FE Outputs.



find the connection address for FE



find the connection address for FE



find the connection address for FE

3. To connect to the deployed Doris Cluster, here are some default values after deploying Doris using CloudFormation:

- **FE IP**: Obtain the IP address of FE following the steps in the previous section.
- **FE MySQL protocol port:** 9030
- **FE HTTP protocol port**: 8030
- **Default root password**: empty
- **Default admin password:** empty

Connecting by MySQL Protocol

Apache Doris adopts the MySQL network connection protocol. It is compatible with command-line tools, JDBC/ODBC drivers, and various visualization tools within the MySQL ecosystem. Additionally, Apache Doris comes with a built-in, easy-to-use Web UI. This guide is about how to connect to Doris using MySQL Client, MySQL JDBC Connector, DBeaver, and the built-in Doris Web UI.

MySQL Client

Download MySQL Client from the MySQL official website for Linux. Currently, Doris is primarily compatible with MySQL 5.7 and later clients.

Extract the downloaded MySQL client. In the `bin/` directory, find the `mysql` command-line tool. Execute the following command to connect to Doris:

```
# FE_IP represents the listening address of the FE node, while FE_QUERY_PORT represents the port of the MySQL protocol service of
the FE. This corresponds to the query_port parameter in fe.conf and it defaults to 9030.
mysql -h FE_IP -P FE_QUERY_PORT -u USER_NAME
```

After login, the following message will be displayed.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 236
Server version: 5.7.99 Doris version doris-2.0.3-rc06-37d31a5
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective
owners.                                     Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

MySQL JDBC Connector

Download the corresponding JDBC Connector from the official MySQL website.

Example of connection code:

```
String user = "user_name";
String password = "user_password";
String newUrl = "jdbc:mysql://FE_IP:FE_PORT/demo?
useUnicode=true&characterEncoding=utf8&useTimezone=true&serverTimezone=Asia/Shanghai&useSSL=false&allowPublicKeyRetrieval=true";
try {
    Connection myCon = DriverManager.getConnection(newUrl, user, password);
    Statement stmt = myCon.createStatement();
    ResultSet result = stmt.executeQuery("show databases");
    ResultSetMetaData metaData = result.getMetaData();
    int columnCount = metaData.getColumnCount();
    while (result.next()) {
        for (int i = 1; i <= columnCount; i++) {
            System.out.println(result.getObject(i));
        }
    }
} catch (SQLException e) {
```

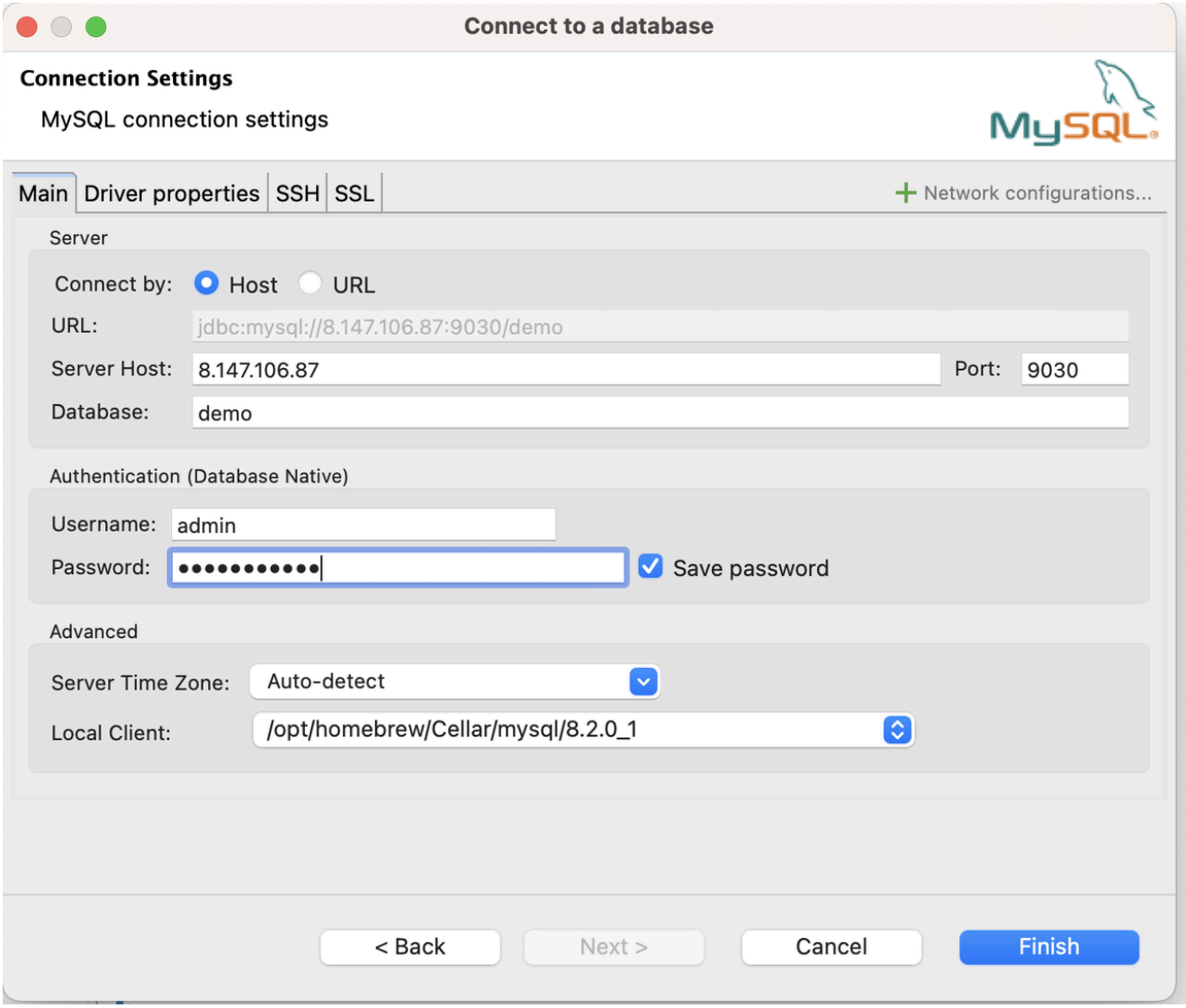
```
    log.error("get JDBC connection exception.", e);
}
```

If you need to initially change session variables when connecting, you can use the following format:

```
jdbc:mysql://FE_IP:FE_PORT/demo?sessionVariables=key1=val1,key2=val2
```

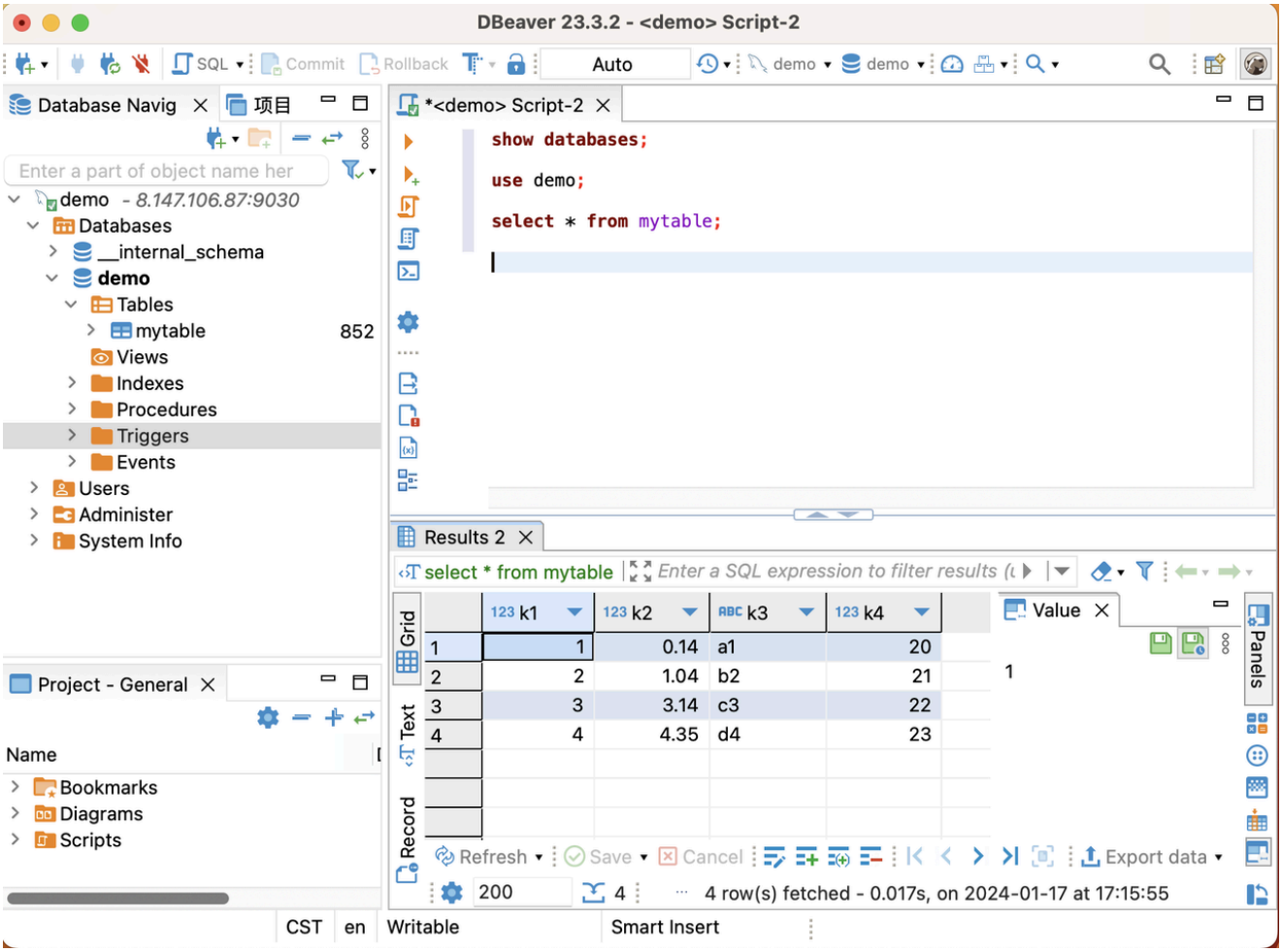
DBeaver

Create a MySQL connection to Apache Doris:



database-connect-dbeaver

Query in DBeaver:



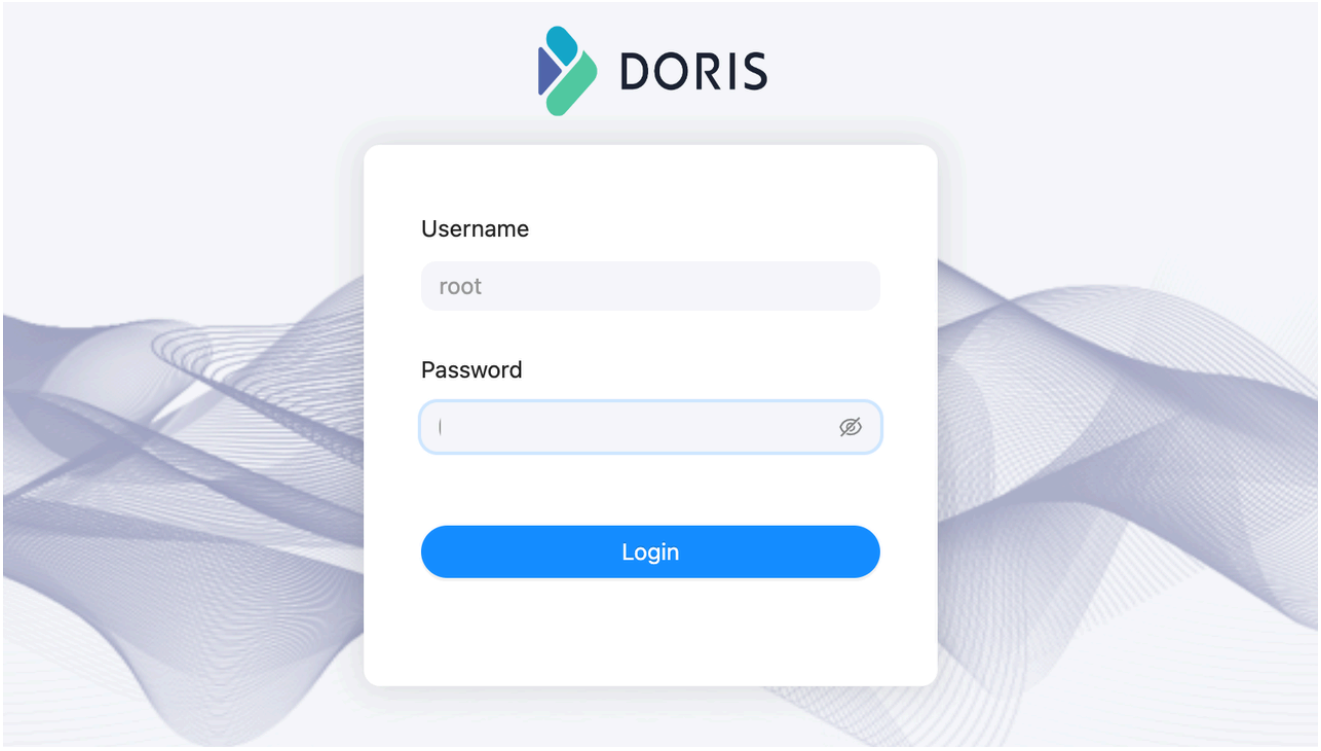
query-in-dbeaver

Built-in Web UI of Doris

Doris FE has a built-in Web UI. It allows users to perform SQL queries and view other related information without the need to install the MySQL client

To access the Web UI, simply enter the URL in a web browser: http://fe_ip:fe_port, for example, `http://172.20.63.118:8030`. This will open the built-in Web console of Doris.

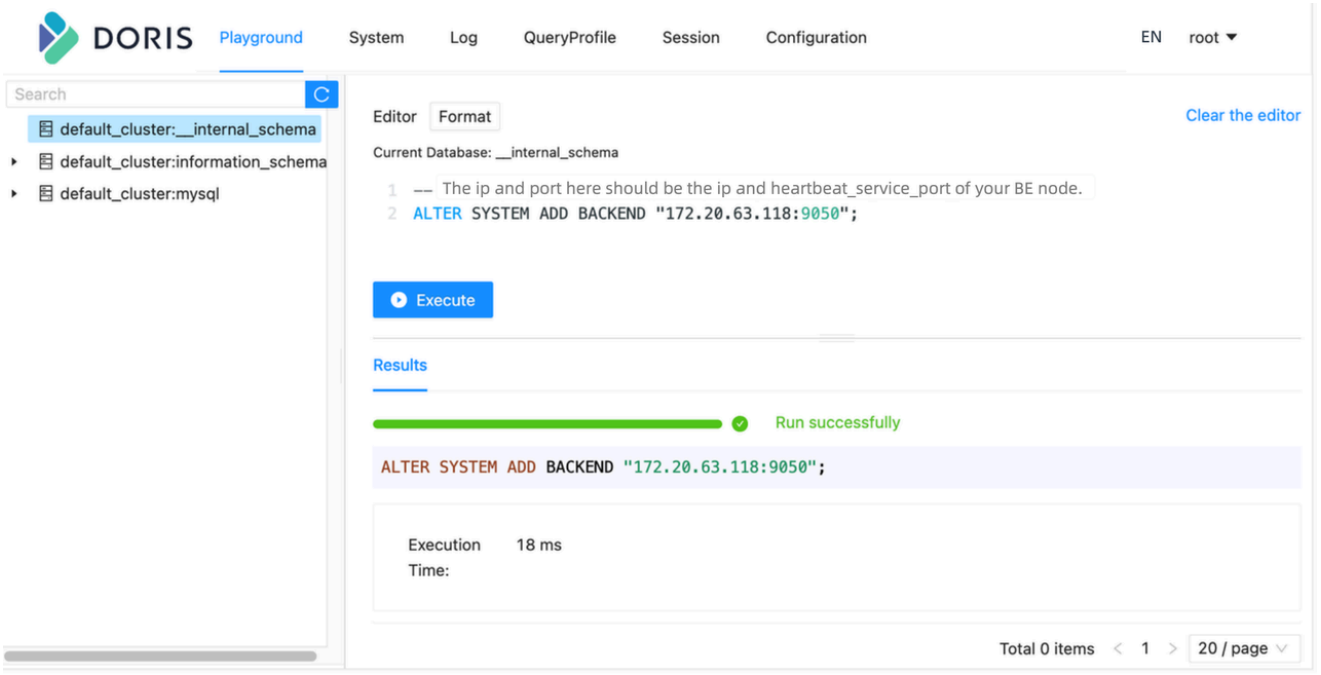
The built-in Web console is primarily intended for use by the root account of the cluster. By default, the root account password is empty after installation.



web-login-username-password

For example, you can execute the following command in the Playground to add a BE node.

```
ALTER SYSTEM ADD BACKEND "be_host_ip:heartbeat_service_port";
```



Doris-Web-UI-Playground-en

> **For successful execution of statements that are not related to specific databases/tables in the Playground, it is necessary to randomly select a database from the left-hand database panel. This limitation will be removed later.**

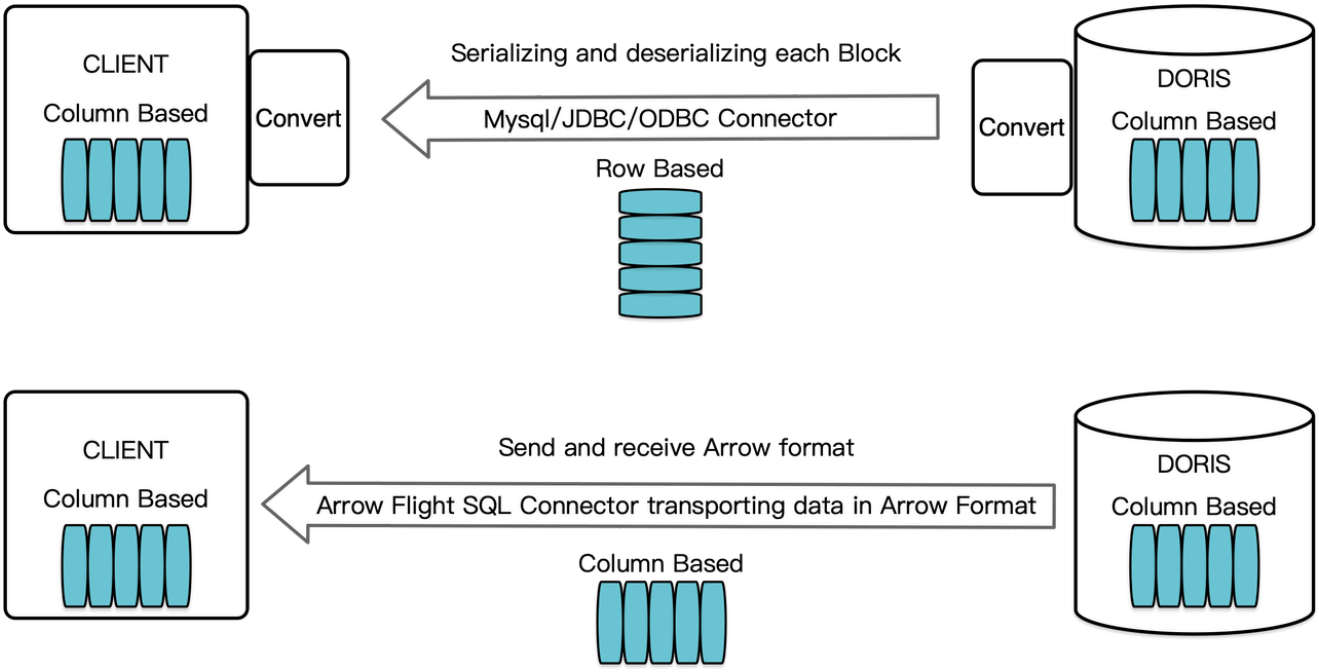
> The current built-in web console cannot execute SET type SQL statements. Therefore, the web console does not support statements like SET PASSWORD FOR ‘user’ = PASSWORD(‘user_password’).

Connecting by Arrow Flight SQL Protocol

Since Doris 2.1, a high-speed data link based on the Arrow Flight SQL protocol has been implemented, allowing SQL queries to rapidly retrieve large volumes of data from Doris in multiple languages. Arrow Flight SQL also provides a universal JDBC driver, supporting seamless interaction with databases that also follow the Arrow Flight SQL protocol. In some scenarios, performance can improve by up to a hundred times compared to data transfer solutions using MySQL Client or JDBC/ODBC drivers.

Implementation Principle

In Doris, query results are organized in columnar format as Blocks. In versions prior to 2.1, data could be transferred to the target client via MySQL Client or JDBC/ODBC drivers, but this required deserializing row-based Bytes into columnar format. By building a high-speed data transfer link based on Arrow Flight SQL, if the target client also supports Arrow columnar format, the entire transfer process avoids serialization and deserialization operations, completely eliminating the time and performance overhead associated with them.



Arrow_Flight_SQL

To install Apache Arrow, you can find detailed installation instructions in the official documentation Apache Arrow. For more information on how Doris implements the Arrow Flight protocol, you can refer to Doris support Arrow Flight SQL protocol.

Python Usage

Use Python's ADBC Driver to connect to Doris to achieve extremely fast data reading. The following steps use Python (version >= 3.9) ADBC Driver to perform a series of common database syntax operations, including DDL, DML, setting Session variables, and Show statements.

Install Library

The library is published on PyPI and can be easily installed in the following ways:

```
pip install adbc_driver_manager
pip install adbc_driver_flightsql
```

Import the following modules/libraries in the code to use the installed Library:

```
import adbc_driver_manager
import adbc_driver_flightsql
import adbc_driver_flightsql.dbapi as flight_sql

>>> print(adbc_driver_manager.__version__)
1.1.0
>>> print(adbc_driver_flightsql.__version__)
1.1.0
```

Connect to Doris

Create a client to interact with the Doris Arrow Flight SQL service. You need to provide Doris FE's Host, Arrow Flight Port, login username and password, and perform the following configuration. Modify the configuration parameters of Doris FE and BE:

- Modify arrow_flight_sql_port in fe/conf/fe.conf to an available port, such as 8070.
- Modify arrow_flight_sql_port in be/conf/be.conf to an available port, such as 8050.

`Note: The arrow_flight_sql_port port number configured in fe.conf and be.conf is different`

After modifying the configuration and restarting the cluster, searching for `Arrow Flight SQL service is started` in the fe/log/fe.log file indicates that the Arrow Flight Server of FE has been successfully started; searching for `Arrow Flight Service bind to host` in the be/log/be.INFO file indicates that the Arrow Flight Server of BE has been successfully started.

Assuming that the Arrow Flight SQL services of FE and BE in the Doris instance will run on ports 8070 and 8050 respectively, and the Doris username/password is "user"/"pass", the connection process is as follows:

```
conn = flight_sql.connect(uri="grpc://{FE_HOST}:{fe.conf:arrow_flight_sql_port}", db_kwargs={
            adbc_driver_manager.DatabaseOptions.USERNAME.value: "user",
            adbc_driver_manager.DatabaseOptions.PASSWORD.value: "pass",
        })
cursor = conn.cursor()
```

After the connection is completed, the returned Cursor can be used to interact with Doris through SQL to perform operations such as creating tables, obtaining metadata, importing data, and querying.

Create a table and get metadata

Pass Query to the cursor.execute() function to execute the table creation and metadata acquisition operations:

```
cursor.execute("DROP DATABASE IF EXISTS arrow_flight_sql FORCE;")
print(cursor.fetchallarrow().to_pandas())
```

```
cursor.execute("create database arrow_flight_sql;")
print(cursor.fetchallarrow().to_pandas())

cursor.execute("show databases;")
print(cursor.fetchallarrow().to_pandas())

cursor.execute("use arrow_flight_sql;")
print(cursor.fetchallarrow().to_pandas())

cursor.execute("""CREATE TABLE arrow_flight_sql_test
    (
         k0 INT,
         k1 DOUBLE,
         K2 varchar(32) NULL DEFAULT "" COMMENT "",
         k3 DECIMAL(27,9) DEFAULT "0",
         k4 BIGINT NULL DEFAULT '10',
         k5 DATE,
    )
    DISTRIBUTED BY HASH(k5) BUCKETS 5
    PROPERTIES("replication_num" = "1");""")
print(cursor.fetchallarrow().to_pandas())

cursor.execute("show create table arrow_flight_sql_test;")
print(cursor.fetchallarrow().to_pandas())
```

If StatusResult returns 0, it means that the Query is executed successfully (the reason for this design is to be compatible with JDBC).

```
  StatusResult
0            0

  StatusResult
0            0

                  Database
0         __internal_schema
1          arrow_flight_sql
..                      ...
507             udf_auth_db

[508 rows x 1 columns]

  StatusResult
0            0

  StatusResult
0            0
                  Table                                       Create Table
0  arrow_flight_sql_test  CREATE TABLE `arrow_flight_sql_test` (\n  `k0`...
```

Import data

Execute INSERT INTO to import a small amount of test data into the created table:

```
cursor.execute("""INSERT INTO arrow_flight_sql_test VALUES
        ('0', 0.1, "ID", 0.0001, 9999999999, '2023-10-21'),
        ('1', 0.20, "ID_1", 1.00000001, 0, '2023-10-21'),
        ('2', 3.4, "ID_1", 3.1, 123456, '2023-10-22'),
        ('3', 4, "ID", 4, 4, '2023-10-22'),
```

```
        ('4', 122345.54321, "ID", 122345.54321, 5, '2023-10-22');""")
print(cursor.fetchallarrow().to_pandas())
```

The following proves that the import was successful:

```
  StatusResult
0            0
```

If you need to import large amounts of data into Doris, you can use pydoris to perform Stream Load.

Execute a query

Then query the table imported above, including operations such as aggregation, sorting, and Set Session Variable.

```
cursor.execute("select * from arrow_flight_sql_test order by k0;")
print(cursor.fetchallarrow().to_pandas())

cursor.execute("set exec_mem_limit=2000;")
print(cursor.fetchallarrow().to_pandas())

cursor.execute("show variables like \"%exec_mem_limit%\";")
print(cursor.fetchallarrow().to_pandas())

cursor.execute("select k5, sum(k1), count(1), avg(k3) from arrow_flight_sql_test group by k5;")
print(cursor.fetch_df())
```

The result is as follows:

```
   k0            k1    K2               k3          k4          k5
0   0       0.10000    ID       0.000100000  9999999999  2023-10-21
1   1       0.20000  ID_1       1.000000010           0  2023-10-21
2   2       3.40000  ID_1       3.100000000      123456  2023-10-22
3   3       4.00000    ID       4.000000000           4  2023-10-22
4   4  122345.54321    ID  122345.543210000           5  2023-10-22

[5 rows x 6 columns]

  StatusResult
0            0

    Variable_name Value Default_Value Changed
0  exec_mem_limit  2000    2147483648       1

           k5  Nullable(Float64)_1  Int64_2 Nullable(Decimal(38, 9))_3
0  2023-10-22         122352.94321        3            40784.214403333
1  2023-10-21              0.30000        2                0.500050005

[2 rows x 5 columns]
```

Note: To fetch query results, you need to use `cursor.fetchallarrow()` to return the arrow format, or use `cursor.fetch_df()` to directly return the pandas dataframe, which will keep the data in column format. Do not use `cursor.fetchall()`, otherwise the column format data will be converted back to row format, which is essentially the same as using mysql-client. In fact, due to the extra column-to-row conversion operation on the client side, it may be slower than mysql-client.

Complete code

```
# Doris Arrow Flight SQL Test

# step 1, library is released on PyPI and can be easily installed.
# pip install adbc_driver_manager
# pip install adbc_driver_flightsql
import adbc_driver_manager
import adbc_driver_flightsql.dbapi as flight_sql

# step 2, create a client that interacts with the Doris Arrow Flight SQL service.
# Modify arrow_flight_sql_port in fe/conf/fe.conf to an available port, such as 8070.
# Modify arrow_flight_sql_port in be/conf/be.conf to an available port, such as 8050.
conn = flight_sql.connect(uri="grpc://{FE_HOST}:{fe.conf:arrow_flight_sql_port}", db_kwargs={
            adbc_driver_manager.DatabaseOptions.USERNAME.value: "root",
            adbc_driver_manager.DatabaseOptions.PASSWORD.value: "",
        })
cursor = conn.cursor()

# interacting with Doris via SQL using Cursor
def execute(sql):
    print("\n### execute query: ###\n " + sql)
    cursor.execute(sql)
    print("### result: ###")
    print(cursor.fetchallarrow().to_pandas())

# step3, execute DDL statements, create database/table, show stmt.
execute("DROP DATABASE IF EXISTS arrow_flight_sql FORCE;")
execute("show databases;")
execute("create database arrow_flight_sql;")
execute("show databases;")
execute("use arrow_flight_sql;")
execute("""CREATE TABLE arrow_flight_sql_test
    (
         k0 INT,
         k1 DOUBLE,
         K2 varchar(32) NULL DEFAULT "" COMMENT "",
         k3 DECIMAL(27,9) DEFAULT "0",
         k4 BIGINT NULL DEFAULT '10',
         k5 DATE,
    )
    DISTRIBUTED BY HASH(k5) BUCKETS 5
    PROPERTIES("replication_num" = "1");""")
execute("show create table arrow_flight_sql_test;")


# step4, insert into
execute("""INSERT INTO arrow_flight_sql_test VALUES
        ('0', 0.1, "ID", 0.0001, 9999999999, '2023-10-21'),
        ('1', 0.20, "ID_1", 1.00000001, 0, '2023-10-21'),
        ('2', 3.4, "ID_1", 3.1, 123456, '2023-10-22'),
        ('3', 4, "ID", 4, 4, '2023-10-22'),
        ('4', 122345.54321, "ID", 122345.54321, 5, '2023-10-22');""")


# step5, execute queries, aggregation, sort, set session variable
execute("select * from arrow_flight_sql_test order by k0;")
execute("set exec_mem_limit=2000;")
execute("show variables like \"%exec_mem_limit%\";")
execute("select k5, sum(k1), count(1), avg(k3) from arrow_flight_sql_test group by k5;")

# step6, close cursor
cursor.close()
```

Jdbc Connector with Arrow Flight SQL

The open source JDBC driver of Arrow Flight SQL protocol is compatible with the standard JDBC API, which can be used by most BI tools to access Doris through JDBC and supports high-speed transmission of Apache Arrow data. The usage is similar to connecting to Doris through the JDBC driver of MySQL protocol. You only need to replace the jdbc:mysql protocol in the link URL with the jdbc:arrow-flight-sql protocol. The query results are still returned in the JDBC ResultSet data structure.

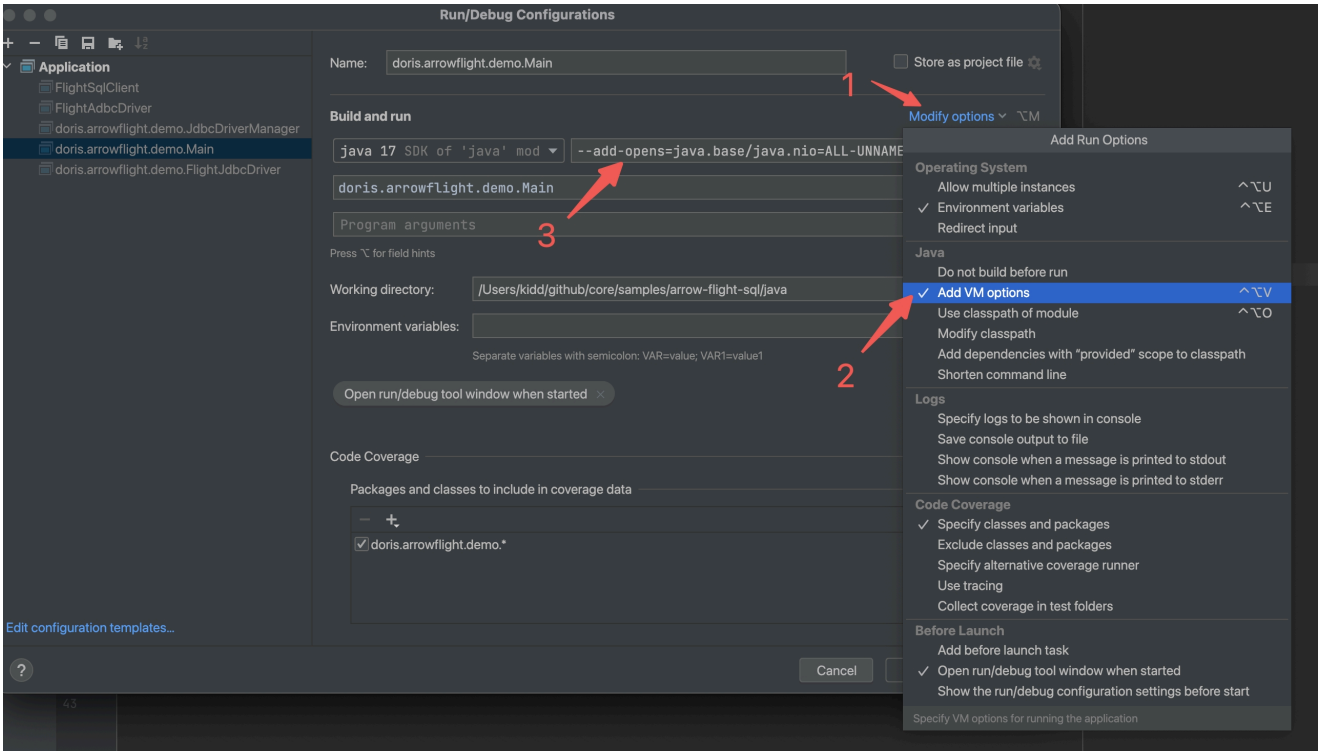
POM dependency:

```
<properties>
    <arrow.version>17.0.0</arrow.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.apache.arrow</groupId>
        <artifactId>flight-sql-jdbc-core</artifactId>
        <version>${arrow.version}</version>
    </dependency>
</dependencies>
```

Note: When using Java 9 or later, you must expose some JDK internal structures by adding `--add-opens=java.base/java.nio=ALL-UNNAMED` to the Java command, otherwise you may see some errors such as `module java.base does not "opens java.nio" to unnamed module` or `module java.base does not "opens java.nio" to org.apache.arrow.memory.core` or `java.lang.NoClassDefFoundError: Could not initialize class org.apache.arrow.memory.util.MemoryUtil (Internal; Prepare)`

```
# Directly on the command line
$ java --add-opens=java.base/java.nio=ALL-UNNAMED -jar ...
# Indirectly via environment variables
$ env _JAVA_OPTIONS="--add-opens=java.base/java.nio=ALL-UNNAMED" java -jar ...
```

If debugging in IntelliJ IDEA, you need to add `--add-opens=java.base/java.nio=ALL-UNNAMED` in `Build and run` of `Run/Debug Configurations`, refer to the picture below:



arrow-flight-sql-IntelliJ

The connection code example is as follows:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

Class.forName("org.apache.arrow.driver.jdbc.ArrowFlightJdbcDriver");
String DB_URL = "jdbc:arrow-flight-sql://{FE_HOST}:{fe.conf:arrow_flight_sql_port}?useServerPrepStmts=false"
        + "&cachePrepStmts=true&useSSL=false&useEncryption=false";
String USER = "root";
String PASS = "";

Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
Statement stmt = conn.createStatement();
ResultSet resultSet = stmt.executeQuery("select * from information_schema.tables;");
while (resultSet.next()) {
    System.out.println(resultSet.toString());
}

resultSet.close();
stmt.close();
conn.close();
```

Java Usage

In addition to using JDBC, similar to Python, JAVA can also create a Driver to read Doris and return data in Arrow format. The following are how to use AdbcDriver and JdbcDriver to connect to Doris Arrow Flight Server.

POM dependency:

```
<properties>
    <adbc.version>0.15.0</adbc.version>
```

```
</properties>

<dependencies>
    <dependency>
        <groupId>org.apache.arrow.adbc</groupId>
        <artifactId>adbc-driver-jdbc</artifactId>
        <version>${adbc.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.arrow.adbc</groupId>
        <artifactId>adbc-core</artifactId>
        <version>${adbc.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.arrow.adbc</groupId>
        <artifactId>adbc-driver-manager</artifactId>
        <version>${adbc.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.arrow.adbc</groupId>
        <artifactId>adbc-sql</artifactId>
        <version>${adbc.version}</version>
    </dependency>
    <dependency>
        <groupId>org.apache.arrow.adbc</groupId>
        <artifactId>adbc-driver-flight-sql</artifactId>
        <version>${adbc.version}</version>
    </dependency>
</dependencies>
```

ADBC Driver

The connection code example is as follows:

```
// 1. new driver
final BufferAllocator allocator = new RootAllocator();
FlightSqlDriver driver = new FlightSqlDriver(allocator);
Map<String, Object> parameters = new HashMap<>();
AdbcDriver.PARAM_URI.set(parameters, Location.forGrpcInsecure("{FE_HOST}", {fe.conf:arrow_flight_sql_port}).getUri().toString());
AdbcDriver.PARAM_USERNAME.set(parameters, "root");
AdbcDriver.PARAM_PASSWORD.set(parameters, "");
AdbcDatabase adbcDatabase = driver.open(parameters);

// 2. new connection
AdbcConnection connection = adbcDatabase.connect();
AdbcStatement stmt = connection.createStatement();

// 3. execute query
stmt.setSqlQuery("select * from information_schema.tables;");
QueryResult queryResult = stmt.executeQuery();
ArrowReader reader = queryResult.getReader();

// 4. load result
List<String> result = new ArrayList<>();
while (reader.loadNextBatch()) {
    VectorSchemaRoot root = reader.getVectorSchemaRoot();
    String tsvString = root.contentToTSVString();
    result.add(tsvString);
}
System.out.printf("batchs %d\n", result.size());

// 5. close
```

```
reader.close();
queryResult.close();
stmt.close();
connection.close();
```

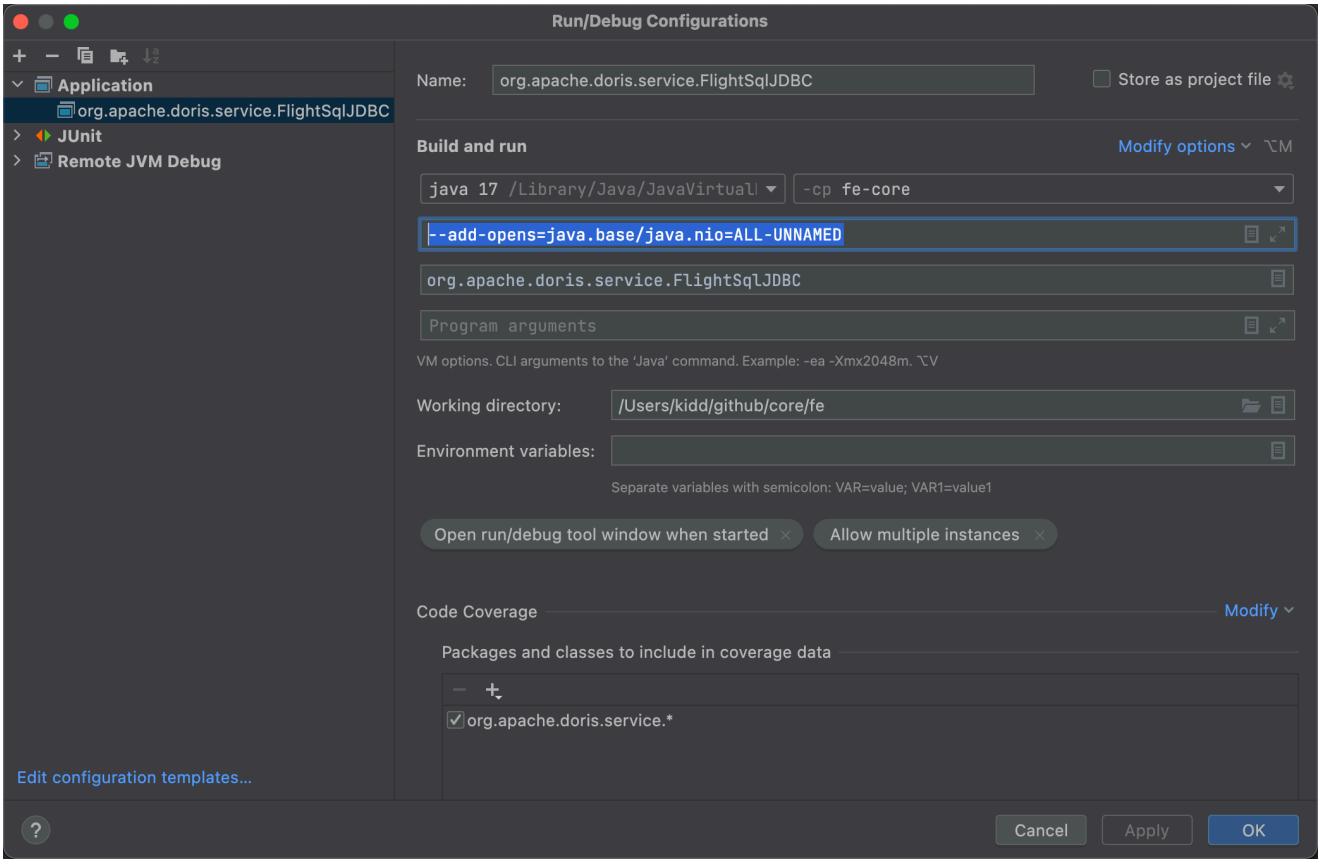
JDBC Driver

When using Java 9 or later, some JDK internals must be exposed by adding –add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED to the java command:

```
# Directly on the command line
$ java --add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED -jar ...
# Indirectly via environment variables
$ env _JAVA_OPTIONS="--add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED" java -jar ...
```

Otherwise, you may see some errors such as `module java.base does not "opens java.nio" to unnamed module` or `module java.base does not "opens java.nio" to org.apache.arrow.memory.core` or `ava.lang.NoClassDefFoundError: Could not initialize class org.apache.arrow.memory.util.MemoryUtil (Internal; Prepare)`

If you debug in IntelliJ IDEA, you need to add `--add-opens=java.base/java.nio=ALL-UNNAMED` in `Build and run` of `Run/Debug Configurations`, refer to the picture below:



IntelliJ IDEA

The connection code example is as follows:

```
final Map<String, Object> parameters = new HashMap<>();
AdbcDriver.PARAM_URI.set(
        parameters,"jdbc:arrow-flight-sql://{FE_HOST}:{fe.conf:arrow_flight_sql_port}?
```

```
useServerPrepStmts=false&cachePrepStmts=true&useSSL=false&useEncryption=false");
AdbcDriver.PARAM_USERNAME.set(parameters, "root");
AdbcDriver.PARAM_PASSWORD.set(parameters, "");
try (
        BufferAllocator allocator = new RootAllocator();
        AdbcDatabase db = new JdbcDriver(allocator).open(parameters);
        AdbcConnection connection = db.connect();
        AdbcStatement stmt = connection.createStatement()
) {
    stmt.setSqlQuery("select * from information_schema.tables;");
    AdbcStatement.QueryResult queryResult = stmt.executeQuery();
    ArrowReader reader = queryResult.getReader();
    List<String> result = new ArrayList<>();
    while (reader.loadNextBatch()) {
        VectorSchemaRoot root = reader.getVectorSchemaRoot();
        String tsvString = root.contentToTSVString();
        result.add(tsvString);
    }
    long etime = System.currentTimeMillis();
    System.out.printf("batchs %d\n", result.size());

    reader.close();
    queryResult.close();
    stmt.close();
}  catch (Exception e) {
    e.printStackTrace();
}
```

Choice of Jdbc and Java connection methods

JDBC/Java Arrow Flight SQL Sample is a JDBC/Java demo using Arrow FLight SQL. You can use it to test various connection methods for sending queries to Arrow Flight Server, helping you understand how to use Arrow FLight SQL and test performance. For expected execution results, see Add Arrow Flight Sql demo for Java.

Compared with the traditional `jdbc:mysql` connection method, the performance test of the Arrow Flight SQL connection method of Jdbc and Java can be found in Section 6.2 of GitHub Issue. Here are some usage suggestions based on the test conclusions.

1. For the above three Java Arrow Flight SQL connection methods, if the subsequent data analysis will be based on the row-based data format, then use jdbc:arrow-flight-sql, which will return data in the JDBC ResultSet format; if the subsequent data analysis can be based on the Arrow format or other column-based data formats, then use Flight AdbcDriver or Flight JdbcDriver to directly return data in the Arrow format, which will avoid row-column conversion and use the characteristics of Arrow to accelerate data parsing.

2. Whether parsing data in JDBC ResultSet or Arrow format, the time spent is longer than the time spent reading data. If the performance of Arrow Flight SQL is not as expected and the improvement is limited compared with `jdbc:mysql://`, you may want to analyze whether it takes too long to parse the data.

3. For all connection methods, JDK 17 reads data faster than JDK 1.8.

4. When reading a large amount of data, Arrow Flight SQL will use less memory than `jdbc:mysql://`, so if you are troubled by insufficient memory, you can also try Arrow Flight SQL.

5. In addition to the above three connection methods, you can also use the native FlightClient to connect to Arrow Flight Server, which can read multiple endpoints in parallel more flexibly. Flight AdbcDriver is also a link created based on FlightClient, which is simpler than using FlightClient directly.

Interaction with other big data components

Spark & Flink

Arrow Flight currently has no official plan to support Spark and Flink (GitHub Issue). Since version 24.0.0, Doris' own Spark Connector and Flink Connector have supported accessing Doris via Arrow Flight SQL, and it is expected that this will improve the reading performance several times.

The community previously referred to the open source Spark-Flight-Connector and used FlightClient in Spark to connect to Doris for testing. It was found that the data format conversion between Arrow and Doris Block is faster, which is 10 times the conversion speed between CSV format and Doris Block, and it has better support for complex types such as Map and Array. This is because the Arrow data format has a high compression rate and low network overhead during transmission. However, Doris Arrow Flight has not yet implemented multi-node parallel reading. It still aggregates query results to a BE node and returns them. For simple batch export of data, the performance may not be as fast as Doris Spark Connector, which supports Tablet-level parallel reading. If you want to use Arrow Flight SQL to connect to Doris in Spark, you can refer to the open-sourced Spark-Flight-Connector and Dremio-Flight-Connector to implement it yourself.

Support BI tools

Since Doris v2.1.8, BI tools such as DBeaver are supported to connect to Doris using the `arrow-flight-sql` protocol. For DBeaver's method of connecting to Doris using the `arrow-flight-sql` Driver, refer to: how-to-use-jdbc-driver-with-dbeaver-client, client-applications/clients/dbeaver/.

Extended Application

Multiple BEs return results in parallel

Doris will aggregate the results of a query on all BE nodes to one BE node by default. In Mysql/JDBC queries, FE will request query results from this aggregated data node. In Arrow Flight SQL queries, FE will wrap the IP/Port of this node in the Endpoint and return it to ADBC Client. ADBC Client will request the BE node corresponding to this Endpoint to pull data.

If the query is just a simple Select to pull data from Doris, without Join, Sort, Window Function and other operators with data Shuffle behavior, the query can be split according to Tablet granularity. Now Doris Spark/Flink Connector uses this method to implement parallel data reading, which is divided into two steps: 1. Execute `explain sql`, and the ScanOperator in the query plan returned by FE contains all Tablet ID Lists of Scan. 2. Split the original SQL into multiple SQLs based on the Tablet ID List above. Each SQL only reads part of the Tablet. The usage is similar to `SELECT * FROM t1 TABLET(10001,10002) limit 1000;`. The multiple SQLs after splitting can be executed in parallel. Refer to Support select table sample.

If the outermost layer of the query is aggregation, the SQL is similar to `select k1, sum(k2) from xxx group by k1`. After Doris v3.0.4, execute `set enable_parallel_result_sink=true;` to allow each BE node of a query to return query results independently. After receiving the Endpoint list returned by FE, ADBC Client pulls results from

multiple BE nodes in parallel. However, please note that when the aggregation result is very small, returning multiple BEs will increase the pressure on RPC. For specific implementation, please refer to support parallel result sink. In theory, except for the outermost query which is sorted, other queries can support each BE node to return results in parallel, but there is no need for this convenience at present, and no further implementation has been made.

Multiple BEs share the same IP accessible from outside the cluster

If there is a Doris cluster, its FE nodes can be accessed from outside the cluster, and all its BE nodes can only be accessed from within the cluster. This is fine when using Mysql Client and JDBC to connect to Doris to execute queries, and the query results will be returned by the Doris FE node. However, using Arrow Flight SQL to connect to Doris will not work because the ADBC Client needs to connect to the Doris BE node to pull the query results, but the Doris BE node is not allowed to be accessed from outside the cluster.

In a production environment, it is often inconvenient to expose the Doris BE node outside the cluster. However, you can add a reverse proxy (such as Nginx) to all Doris BE nodes. When the client outside the cluster connects to Nginx, it will be randomly routed to a Doris BE node. By default, the Arrow Flight SQL query results will be randomly saved on a Doris BE node. If it is different from the Doris BE node randomly routed by Nginx, data forwarding is required within the Doris BE node.

Starting from Doris v2.1.8, you can configure `public_host` and `arrow_flight_sql_proxy_port` in `be.conf` of all Doris BE nodes to the IP and port shared by multiple Doris BE nodes and accessible outside the cluster. The query results can be correctly forwarded and returned to the ADBC Client.

```
public_host={nginx ip}
arrow_flight_sql_proxy_port={nginx port}
```

FAQ

1. Q: Error `connection error: desc = "transport: Error while dialing: dial tcp <ip:arrow_flight_port>: i/o timeout"`.

A: If `<ip:arrow_flight_port>` in the error message is the IP and arrow-flight-port of the Doris FE node,

First check whether the arrow-flight-server of the Doris FE node is started normally. Searching for `Arrow Flight SQL service is started` in the fe/log/fe.log file indicates that the Arrow Flight Server of FE is started successfully.

If the arrow-flight-server of the Doris FE node is started normally, check whether the machine where the Client is located can `ping` the IP in the error message `<ip:arrow_flight_port>`. If it cannot be `ping`ed, you need to open an externally accessible IP for the Doris FE node and redeploy the cluster.

A: If the `<ip:arrow_flight_port>` in the error message is the IP and arrow-flight-port of the Doris BE node.

First check whether the arrow-flight-server of the Doris BE node is started normally. Searching for `Arrow Flight Service bind to host` in the be/log/be.INFO file indicates that the Arrow Flight Server of BE is started successfully.

If the arrow-flight-server of the Doris BE node starts normally, check whether the client machine can `ping` the IP in the `<ip:arrow_flight_port>` reported in the error message. If it cannot be `ping`, if it is known that the Doris BE node is in an intranet that cannot be accessed externally, the following two methods are used:

- Consider opening an externally accessible IP for each Doris BE node. Starting from Doris v2.1.8, you can configure `public_host` to this IP in the `be.conf` of this Doris BE node. Similarly, configure the `public_host` of all Doris BE nodes to the IP of the corresponding BE node that can be accessed by the client.
- Refer to the above section [Multiple BEs share the same IP that can be accessed externally by the cluster] to add a layer of reverse proxy for all Doris BE nodes.

If it is not clear whether Doris BE is completely in the intranet, check the connectivity between the client machine and other IPs of the machine where the Doris BE node is located. Execute `ifconfig` on the machine where the Doris BE node is located to return all the IPs of the current machine. One of the IPs should be the same as the IP in `<ip:arrow_flight_port>` and the same as the IP of the Doris BE node printed by `show backends`. `ping` the other IPs returned by `ifconfig` in turn. If the Doris BE node has an IP that can be accessed by the Client, refer to the above to configure this IP as `public_host`. If all IPs of the Doris BE node cannot be accessed by the Client, then the Doris BE node is completely in the intranet.

2. Q: When using JDBC or JAVA to connect to Arrow Flight SQL, an error message appears: `module java.base does not "opens java.nio" to unnamed module` or `module java.base does not "opens java.nio" to org.apache.arrow.memory.core` or `java.lang.NoClassDefFoundError: Could not initialize class org.apache.arrow.memory.util.MemoryUtil (Internal; Prepare)`

A: First check whether `JAVA_OPTS_FOR_JDK_17` in fe/conf/fe.conf contains `--add-opens=java.base/java.nio=ALL-UNNAMED`. If not, add it. Then refer to the notes in JDBC Connector with Arrow Flight SQL above and add `--add-opens=java.base/java.nio=ALL-UNNAMED` in the Java command. If debugging in IntelliJ IDEA, you need to add `--add-opens=java.base/java.nio=ALL-UNNAMED` in `Build and run` of `Run/Debug Configurations`.

3. Q: The ARM environment reports an error `get flight info statement failed, arrow flight schema timeout, TimeoutException: Waited 5000 milliseconds for io.grpc.stub.Client`.

A: If the Linux kernel version is <= 4.19.90, you need to upgrade to 4.19.279 or above, or recompile Doris BE in the environment of the lower version of the Linux kernel. For specific compilation methods, refer to the document

Cause: This is because there is a compatibility issue between the old version of the Linux kernel and Arrow. `cpp: arrow::RecordBatch::MakeEmpty()` will get stuck when constructing Arrow Record Batch, causing Doris BE's Arrow Flight Server to fail to respond to Doris FE's Arrow Flight Server's RPC request within 5000ms, causing FE to return rpc timeout failed to Client. When Spark and Flink read Doris, they also convert the query results into Arrow Record Batch and return them, so the same problem exists.

The Linux kernel version of kylinv10 SP2 and SP3 is only 4.19.90-24.4.v2101.ky10.aarch64 at most. The kernel version cannot be upgraded further. Doris BE can only be recompiled on kylinv10. If the problem still exists after compiling Doris BE with the new version of ldb_toolchain, you can try to compile it with the lower version of ldb_toolchain v0.17. If your ARM environment cannot connect to the external network, Huawei Cloud provides ARM + kylinv10, and Alibaba Cloud provides x86 + kylinv10

4. Q: Prepared statement passes parameters and reports errors.

A: Currently, `jdbc:arrow-flight-sql` and Java ADBC/JDBCDriver do not support prepared statement parameter passing. For example, `select * from xxx where id=?` will report an error `parameter ordinal 1 out of range`. This is a bug in Arrow Flight SQL (GitHub Issue).

5. Q: How to modify the batch size read by `jdbc:arrow-flight-sql` each time to improve performance in some scenarios.

A: By modifying `setTargetBatchSize` in the `makeJdbcConfig` method in the `org.apache.arrow.adbc.driver.jdbc.JdbcArrowReader` file, the default is 1024, and then saving the modified file to the local directory with the same path name, so as to overwrite the original file and take effect.

6. Q: ADBC v0.10, JDBC and Java ADBC/JDBCDriver do not support parallel reading.

A: The `stmt.executePartitioned()` method is not implemented. You can only use the native FlightClient to implement parallel reading of multiple endpoints, using the method `sqlClient=new FlightSqlClient, execute=sqlClient.execute(sql), endpoints=execute.getEndpoints(), for(FlightEndpoint endpoint: endpoints)`. In addition, the default AdbcStatement of ADBC V0.10 is actually JdbcStatement. After executeQuery, the row-format JDBC ResultSet is converted back to the Arrow column format. It is expected that Java ADBC will be fully functional by ADBC 1.0.0 GitHub Issue.

7. Q: Specify the database name in the URL.

A: As of Arrow v15.0, Arrow JDBC Connector does not support specifying database name in URL. For example, specifying connection to `test` database in `jdbc:arrow-flight-sql://{FE_HOST}:{fe.conf:arrow_flight_sql_port}/test?useServerPrepStmts=false` is invalid, and you can only execute SQL `use database` manually. Arrow v18.0 supports specifying database name in URL, but there are still bugs in actual testing.

8. Q: Python ADBC prints `Warning: Cannot disable autocommit; conn will not be DB-API 2.0 compliant`.

A: Ignore this Warning when using Python. This is a problem with Python ADBC Client and will not affect queries.

9. Q: Python reports an error `grpc: received message larger than max (20748753 vs. 16777216)`.

A: Refer to Python: grpc: received message larger than max (20748753 vs. 16777216) #2078 and add `adbc_driver_flightsql.DatabaseOptions.WITH_MAX_MSG_SIZE.value` in Database Option.

10. Q: Error `invalid bearer token` is reported.

A: Execute `SET PROPERTY FOR 'root' 'max_user_connections' = '10000';` to change the current maximum number of connections for the current user to 10000; add qe_max_connection=30000 and arrow_flight_token_cache_size=8000 in `fe.conf` and restart FE.

The connection between the ADBC Client and the Arrow Flight Server is essentially a long link, which requires Auth Token, Connection, and Session to be cached on the Server. After the connection is created, it will not be disconnected immediately at the end of a single query. The Client needs to send a close() request to clean it up, but in fact, the Client often does not send a close request, so the Auth Token, Connection, and Session will be saved on the Arrow Flight Server for a long time. By default, they will time out and disconnect after 3 days, or be eliminated according to LRU after the number of connections exceeds the limit of `arrow_flight_token_cache_size`.

As of Doris v2.1.8, Arrow Flight connections and Mysql/JDBC connections use the same connection limit, including the total number of connections of all FE users `qe_max_connection` and the number of connections of a single user `max_user_connections` in `UserProperty`. But the default `qe_max_connection` and `max_user_connections` are 1024 and 100 respectively. Arrow Flight SQL is often used to replace JDBC scenarios, but the JDBC

connection will be released immediately after the query ends. Therefore, when using Arrow Flight SQL, the default connection limit of Doris is too small, which often causes the connection number to exceed the limit of `arrow_flight_token_cache_size` and the connections still in use to be eliminated.

11. Q: Using JDBC or JAVA to connect Arrow Flight SQL to read Datatime type returns a timestamp instead of a formatted time.

A: Using JDBC or JAVA to connect Arrow Flight SQL to read Datatime type requires converting the timestamp yourself, refer to Add java parsing datetime type in arrow flight sql sample #48578. Using Python Arrow Flight SQL to read Datatime type returns the result of `2025-03-03 17:23:28Z`, while JDBC or JAVA returns `1740993808`.

12. Q: Using JDBC or Java JDBC Client to connect Arrow Flight SQL to read Array nested type returns an error `Configuration does not provide a mapping for array column 2`.

A: Refer to `sample/arrow-flight-sql` to use JAVA ADBC Client.

Python ADBC Client, JAVA ADBC Client, and Java JDBC DriverManager are all OK for reading Array nested types. Only when JDBC or Java JDBC Client is used to connect to Arrow Flight SQL, there is a problem. In fact, the compatibility of Arrow Flight JDBC is not guaranteed. It is not officially developed by Arrow, but by a third-party database company Dremio. Other compatibility issues have been found before, so it is recommended to use JAVA ADBC Client first.

2.1 Release Note

> Doris Arrow Flight is not perfect in versions v2.1.4 and earlier, so it is recommended to upgrade before use.

v2.1.9

1. Fix the problem of Doris data serialization to Arrow. Fix UT DataTypeSerDeArrowTest of Array/Map/Struct/Bitmap/HLL/Decimal256 types

- Failed to read `Decimal256` type;
- Subtle error in reading `DatetimeV2` type;
- Incorrect result in reading `DateV2` type;
- Error when reading `IPV4/IPV6` type result is NULL;

2. Fix the problem that Doris Arrow Flight SQL query fails and returns empty result, without returning real error information. Fix query result is empty and not return query error message

v2.1.8

1. Support BI tools such as DBeaver to connect to Doris using the `arrow-flight-sql` protocol, and support the correct display of metadata trees. Support arrow-flight-sql protocol getStreamCatalogs, getStreamSchemas, getStreamTables #46217.

2. When multiple BEs share the same IP that is accessible to the outside of the cluster, the query results can be correctly forwarded and returned to the ADBC Client. Arrow flight server supports data forwarding

when BE uses public vip

3. Support multiple endpoints to read in parallel. Arrow Flight support multiple endpoints
4. Fix query error `FE not found arrow flight schema`. Fix FE not found arrow flight schema
5. Fix error `BooleanBuilder::AppendValues` when reading columns that allow NULL. Fix Doris NULL column conversion to arrow batch
6. Fix `show processlist` displays duplicate Connection IDs. Fix arrow-flight-sql ConnectContext to use a unified ID #46284
7. Fix the problem that the time zone is lost when reading `Datetime` and `DatetimeV2` types, resulting in a datetime that is 8 hours less than the actual data. Fix time zone issues and accuracy issues #38215

v2.1.7

1. Fix frequent log printing `Connection wait_timeout`. Fix kill timeout FlightSqlConnection and FlightSqlConnectProcessor close
2. Fix Arrow Flight Bearer Token expiration from Cache. Fix Arrow Flight bearer token cache evict after expired

v2.1.6

1. Fix query error `0.0.0.0:xxx, connection refused`. Fix return result from FE Arrow Flight server error 0.0.0.0:xxx, connection refused
2. Fix query error `Reach limit of connections`. Fix exceed user property max connection cause Reach limit of connections #39127

In previous versions, execute `SET PROPERTY FOR 'root' 'max_user_connections' = '1024';` to modify the current maximum number of connections for the current user to 1024, which can be temporarily circumvented.

Because the previous version only limits the number of Arrow Flight connections to less than `qe_max_connection/2`, `qe_max_connection` is the total number of connections for all fe users, the default is 1024, and does not limit the number of Arrow Flight connections for a single user to less than `max_user_connections` in `UserProperty`, the default is 100, so when the number of Arrow Flight connections exceeds the upper limit of the current user's connection number, an error `Reach limit of connections` will be reported, so the current user's `max_user_connections` needs to be increased.

For details of the problem, see: Questions

3. Add Conf `arrow_flight_result_sink_buffer_size_rows` to support modifying the ArrowBatch size of query results returned in a single time, the default is 4096 * 8. Add config arrow_flight_result_sink_buffer_size_rows

v2.1.5

1. Fix the problem that Arrow Flight SQL query results are empty. Fix arrow flight result sink #36827

Doris v2.1.4 may report an error when reading large amounts of data. For details, see: Questions

3.0 Release Note

v3.0.5

1. Fix the problem of serializing Doris data to Arrow. Fix UT DataTypeSerDeArrowTest of Array/Map/Struct/Bitmap/HLL/Decimal256 types

- Failed to read `Decimal256` type;
- Subtle error in reading `DatetimeV2` type;
- Incorrect result in reading `DateV2` type;
- Error when reading `IPV4/IPV6` type result is NULL;

v3.0.4

1. Support DBeaver and other BI tools to connect to Doris using `arrow-flight-sql` protocol, and support correct display of metadata tree. Support arrow-flight-sql protocol getStreamCatalogs, getStreamSchemas, getStreamTables #46217.
2. Support multiple endpoints to read in parallel. Arrow Flight support multiple endpoints
3. Fix the error `BooleanBuilder::AppendValues` when reading columns that allow NULL. Fix Doris NULL column conversion to arrow batch
4. Fix `show processlist` to display duplicate Connection IDs. Fix arrow-flight-sql ConnectContext to use a unified ID #46284
5. Fix Doris Arrow Flight SQL query failed and returned empty results, without returning real error information. Fix query result is empty and not return query error message

v3.0.3

1. Fix query error `0.0.0.0:xxx, connection refused`. Fix return result from FE Arrow Flight server error 0.0.0.0:xxx, connection refused
2. Fix query error `Reach limit of connections`. Fix exceed user property max connection cause Reach limit of connections #39127

In previous versions, execute `SET PROPERTY FOR 'root' 'max_user_connections' = '1024';` to modify the current maximum number of connections for the current user to 1024, which can be temporarily circumvented.

Because the previous version only limits the number of Arrow Flight connections to less than `qe_max_connection/2`, `qe_max_connection` is the total number of connections for all fe users, the default is 1024, and does not limit the number of Arrow Flight connections for a single user to less than `max_user_connections` in `UserProperty`, the default is 100, so when the number of Arrow Flight connections exceeds the upper limit of the current user's connection number, an error `Reach limit of connections` will be reported, so the current user's `max_user_connections` needs to be increased.

For details of the problem, see: Questions

3. Fix frequent printing of log `Connection wait_timeout`. Fix kill timeout FlightSqlConnection and FlightSqlConnectProcessor close
4. Fix Arrow Flight Bearer Token being eliminated from Cache after expiration. Fix Arrow Flight bearer token cache evict after expired
5. When multiple BEs share the same IP address accessible from outside the cluster, the query results can be correctly forwarded and returned to the ADBC Client. Arrow flight server supports data forwarding when BE uses public vip
6. Fix the query error `FE not found arrow flight schema`. Fix FE not found arrow flight schema
7. Fix the problem that the time zone is lost when reading `Datetime` and `DatetimeV2` types, resulting in a datetime that is 8 hours less than the actual data. Fix time zone issues and accuracy issues #38215

v3.0.2

1. Added Conf `arrow_flight_result_sink_buffer_size_rows` to support modifying the ArrowBatch size of query results returned in a single transaction, the default is 4096 * 8. Add config arrow_flight_result_sink_buffer_size_rows

v3.0.1

1. Query results are missing, query result rows = actual number of rows / number of BEs Fix get Schema failed when enable_parallel_result_sink is false #37779

In Doris 3.0.0, if the outermost layer of the query is an aggregation, the SQL is similar to `select k1, sum(k2) from xxx group by k1`, you may encounter (query result rows = actual number of rows / number of BEs), which is a problem introduced by support parallel result sink. In Fix get Schema failed when enable_parallel_result_sink is false is a temporary fix, which will be officially fixed after Arrow Flight support multiple endpoints supports parallel reading of multiple endpoints.

Overview | Table Design

Overview

Creating tables

Users can use the CREATE TABLE statement to create a table in Doris. You can also use the CREATE TABLE LIKE or CREATE TABLE AS clause to derive the table definition from another table.

Table name

In Doris, table names are case-sensitive by default. You can configure lower_case_table_namesto make them case-insensitive during the initial cluster setup. The default maximum length for table names is 64 bytes, but you can change this by configuring table_name_length_limit. It is not recommended to set this value too high. For syntax on creating tables, please refer to CREATE TABLE.

Table property

In Doris, the CREATE TABLE statement can specify table properties, including:

- **buckets**: Determines the distribution of data within the table.
- **storage_medium**: Controls the storage method for data, such as using HDD, SSD, or remote shared storage.
- **replication_num**: Controls the number of data replicas to ensure redundancy and reliability.
- **storage_policy**: Controls the migration strategy for cold and hot data separation storage.

These properties apply to partitions, meaning that after a partition is created, it will have its own properties. Modifying table properties will only affect partitions created in the future and will not affect existing partitions. For more information about table properties, refer to ALTER TABLE PROPERTY. Dynamic partitions can have these properties set individually.

Notes

1. **Choose an appropriate data model**: The data model cannot be changed, so you need to select an appropriate data model when creating the table.
2. **Choose an appropriate number of buckets**: The number of buckets in an already created partition cannot be modified. You can modify the number of buckets by replacing the partition, or you can modify the number of buckets for partitions that have not yet been created in dynamic partitions.
3. **Column addition operations**: Adding or removing VALUE columns is a lightweight operation that can be completed in seconds. Adding or removing KEY columns or modifying data types is a heavyweight operation, and the completion time depends on the amount of data. For large datasets, it is recommended to avoid adding or removing KEY columns or modifying data types.

4. **Optimize storage strategy**: You can use tiered storage to store cold data on HDD or S3/HDFS.

Table Model Overview

When creating a table in Doris, it is necessary to specify the table model to define how data is stored and managed. Doris provides three table models: the **Duplicate Key Model**, **Unique Key Model** and **Aggregate Key Model**, which cater to different application scenarios. Each model has corresponding mechanisms for data deduplication, aggregation, and updates. Choosing the appropriate table model helps achieve business objectives while ensuring flexibility and efficiency in data processing.

Table Model Classification

Doris supports three types of table models:

- Duplicate Key Model: Allows the specified Key columns to be duplicated, and Doris's storage layer retains all written data. This model is suitable for situations where all original data records must be preserved.
- Unique Key Model: Ensures that each row has a unique Key value, and guarantees that there are no duplicate rows for a given Key column. The Doris storage layer retains only the latest written data for each key, making this model suitable for scenarios that involve data updates.
- Aggregate Key Model: Allows data to be aggregated based on the Key columns. The Doris storage layer retains aggregated data, reducing storage space and improving query performance. This model is typically used in situations where summary or aggregated information (such as totals or averages) is required.

After creating the table, the properties of the table model are confirmed and cannot be modified. Choosing the right model for the business is crucial:

- **Duplicate Key Model** is suitable for ad-hoc queries with any dimensions. Although it cannot leverage the benefits of pre-aggregation, it is not constrained by aggregation models and can take advantage of the columnar storage model (only reading relevant columns without needing to read all key columns).
- **Unique Key Model** is designed for scenarios where a unique key constraint is needed, ensuring the uniqueness of the key. However, it cannot utilize the query benefits brought by pre-aggregations such as ROLLUP.
- **Aggregate Key Model** can greatly reduce the data and computation required for aggregation queries through pre-aggregation, making it ideal for fixed-schema reporting queries. However, this model is not friendly to `count(*)` queries. Also, because the aggregation method for the Value columns is fixed, when performing other types of aggregation queries, semantic correctness must be considered.
- **Update partial columns**, please refer to the documentation for Partial Column Updates in Unique Key Model and Partial Column Updates in Aggregate Model for relevant usage advice.

Sort Key

In Doris, data is stored in a columnar format, and a table can be divided into Key columns and Value columns. The Key columns are used for grouping and sorting, while the Value columns are used for aggregation. Key columns can consist of one or more fields, and when creating a table, data is sorted and stored according to the columns of Aggregate Key, Unique Key, and Duplicate Key models.

Different table models require the specification of Key columns during table creation, each with a different significance: for the Duplicate Key model, the Key columns represent sorting, without any uniqueness constraints. In the Aggregate Key and Unique Key models, aggregation is performed based on the Key columns, which not only have sorting capabilities but also enforce uniqueness constraints.

Proper use of the Sort Key can provide the following benefits:

- **Accelerated Query Performance**: Sort keys help reduce the amount of data that needs to be scanned. For range queries or filtering queries, the sort key can directly locate the data. For queries that require sorting, the sort key can also accelerate the sorting process.
- **Data Compression Optimization**: Storing data in an ordered fashion based on the sort key improves compression efficiency, as similar data will be grouped together, significantly increasing the compression ratio and reducing storage space.
- **Reduced Deduplication Costs**: When using the Unique Key Model, the sort key allows Doris to perform deduplication more efficiently, ensuring data uniqueness.

When selecting a sort key, the following recommendations can be followed:

- The Key columns must come before all Value columns.
- Preferably choose integer types. This is because integer types are much more efficient in computation and lookup than strings.
- For selecting different lengths of integer types, follow the principle of choosing what is sufficient.
- For the length of `VARCHAR` and `STRING` types, follow the principle of choosing enough…

Table Model Comparison

	Duplicate Key Model	Unique Key Model	Aggregate Key Model
Key Column Uniqueness	Not Supported, Key columns can be duplicated	Supported	Supported
Synchronous Materialized View	Supported	Supported	Supported
Asynchronous Materialized View	Supported	Supported	Supported
UPDATE Statement	Not Supported	Supported	Not Supported
DELETE Statement	Partially Supported	Supported	Not Supported

	Duplicate Key Model	Unique Key Model	Aggregate Key Model
Full Row Update on Import	Not Supported	Supported	Not Supported
Partial Column Update on Import	Not Supported	Supported	Partially Supported

Duplicate Key Model

The **Duplicate Key Model** in Doris is the default table model, designed to store individual raw data records. The `Duplicate Key` specified during table creation determines the columns for sorting and storage, optimizing common queries. It is recommended to choose no more than three columns as the sort key. For more specific selection guidelines, refer to Sort Key. The Duplicate Key Model has the following characteristics:

- **Preserving Raw Data**: The Duplicate Key Model retains all original data, making it ideal for storing and querying raw data. It is recommended for use cases requiring detailed data analysis to avoid data loss.
- **No Deduplication or Aggregation**: Unlike the Aggregate and Primary Key models, the Duplicate Key Model does not perform deduplication or aggregation, fully retaining identical records.
- **Flexible Data Querying**: The Duplicate Key Model retains all original data, enabling detailed extraction and aggregation across any dimension for metadata auditing and fine-grained analysis.

Use Cases

In the Duplicate Key Model, data is generally only appended, and old data is not updated. The Duplicate Key Model is ideal for scenarios that require full raw data:

- **Log Storage**: Used for storing various types of application logs, such as access logs, error logs, etc. Each piece of data needs to be detailed for future auditing and analysis.
- **User Behavior Data**: When analyzing user behavior, such as click data or user access paths, it is necessary to retain detailed user actions. This helps in building user profiles and conducting detailed analysis of behavior patterns.
- **Transaction Data**: For storing transaction or order data, once a transaction is completed, there is typically no need for data changes…

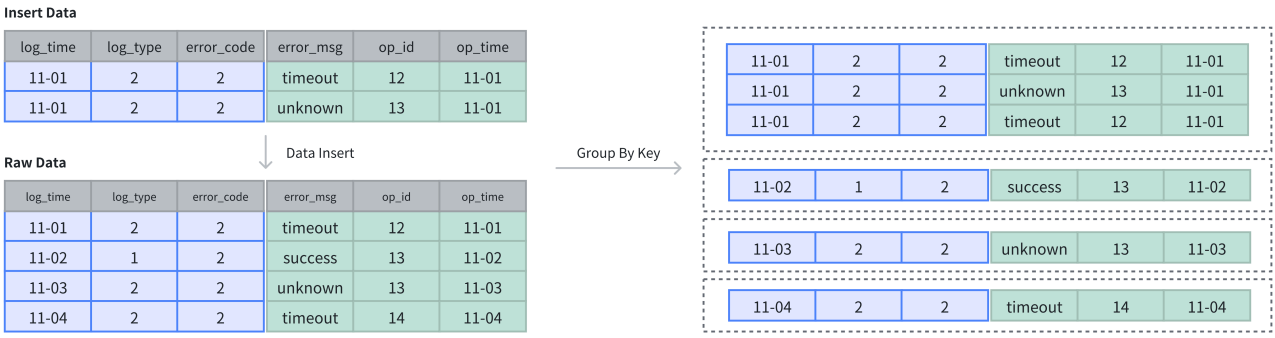
Table Creation Instructions

When creating a table, the **DUPLICATE KEY** keyword can be used to specify the Duplicate Key Model. The Duplicate Key table must specify the Key columns, which are used to sort the data during storage. In the following example, the Duplicate Key table stores log information and sorts the data based on the `log_time`, `log_type`, and `error_code` columns:

```
CREATE TABLE IF NOT EXISTS example_tbl_duplicate
(
    log_time        DATETIME       NOT NULL,
    log_type        INT            NOT NULL,
    error_code      INT,
    error_msg       VARCHAR(1024),
    op_id           BIGINT,
    op_time         DATETIME
)
DUPLICATE KEY(log_time, log_type, error_code)
DISTRIBUTED BY HASH(log_type) BUCKETS 10;
```

Data Insertion and Storage

In a Duplicate Key table, data is not deduplicated or aggregated; inserting data directly stores it. The Key columns in the Duplicate Key Model are used for sorting.



columnar_storage

In the example above, after inserting 2 rows into the initial 4 rows, the data is appended, resulting in a total of 6 rows.

```
-- 4 rows raw data
INSERT INTO example_tbl_duplicate VALUES
('2024-11-01 00:00:00', 2, 2, 'timeout', 12, '2024-11-01 01:00:00'),
('2024-11-02 00:00:00', 1, 2, 'success', 13, '2024-11-02 01:00:00'),
('2024-11-03 00:00:00', 2, 2, 'unknown', 13, '2024-11-03 01:00:00'),
('2024-11-04 00:00:00', 2, 2, 'unknown', 12, '2024-11-04 01:00:00');

-- insert into 2 rows
INSERT INTO example_tbl_duplicate VALUES
('2024-11-01 00:00:00', 2, 2, 'timeout', 12, '2024-11-01 01:00:00'),
('2024-11-01 00:00:00', 2, 2, 'unknown', 13, '2024-11-01 01:00:00');

-- check the rows of table
SELECT * FROM example_tbl_duplicate;
+---------------------+----------+------------+-----------+-------+---------------------+
| log_time            | log_type | error_code | error_msg | op_id | op_time             |
+---------------------+----------+------------+-----------+-------+---------------------+
| 2024-11-02 00:00:00 |        1 |          2 | success   |    13 | 2024-11-02 01:00:00 |
| 2024-11-01 00:00:00 |        2 |          2 | timeout   |    12 | 2024-11-01 01:00:00 |
| 2024-11-03 00:00:00 |        2 |          2 | unknown   |    13 | 2024-11-03 01:00:00 |
| 2024-11-04 00:00:00 |        2 |          2 | unknown   |    12 | 2024-11-04 01:00:00 |
| 2024-11-01 00:00:00 |        2 |          2 | unknown   |    13 | 2024-11-01 01:00:00 |
| 2024-11-01 00:00:00 |        2 |          2 | timeout   |    12 | 2024-11-01 01:00:00 |
+---------------------+----------+------------+-----------+-------+---------------------+
```

Unique Key Model

When data updates are required, use the **Unique Key Model**. It guarantees the uniqueness of the Key columns so that new data overwrites existing records with matching keys, ensuring that only the most up-to-date records are maintained. This model is ideal for update scenarios, enabling unique-key-level updates during data insertion. The Unique Key Model has the following characteristics:

- **Unique Key UPSERT**: During insertion, records with duplicate keys are updated, while new keys are inserted.
- **Automatic Deduplication**: The model ensures key uniqueness and automatically deduplicates data based on the unique key.
- **Optimized for High-frequency Updates**: It efficiently handles high-frequency updates while balancing update and query performance.

Use Cases

- **High-frequency Data Updates**: In upstream OLTP databases, where dimension tables are frequently updated, the Unique Key Model can efficiently synchronize the upstream updated records and perform efficient UPSERT operations.
- **Efficient Data Deduplication**: In scenarios such as advertising campaigns or customer relationship management systems, where deduplication is required based on user IDs, the Unique Key Model ensures efficient deduplication.
- **Partial Columns Updates**: In scenarios such as in user profiling where dynamic tags change frequently, or in order consumption scenarios where the transaction status needs to be updated. The Unique Key Model's partial column update capability allows for changes to specific columns.

Implementation Methods

In Doris, the Unique Key Model has two implementation methods:

- **Merge-on-write**: Starting from version 1.2, the default implementation of the Unique Key Model in Doris is the merge-on-write mode. In this mode, data is immediately merged for the same Key upon writing, ensuring that the data storage state after each write is the final merged result of the unique key, and only the latest result is stored. Merge-on-write provides a good balance between query and write performance, avoiding the need to merge multiple versions of data during queries and ensuring predicate pushdown to the storage layer. The merge-on-write model is recommended for most scenarios.
- **Merge-on-read**: Prior to version 1.2, Doris's Unique Key Model defaulted to merge-on-read mode. In this mode, data is not merged upon writing but is appended incrementally, retaining multiple versions within Doris. During queries or Compaction, data is merged by the same Key version. Merge-on-read is suitable for write-heavy and read-light scenarios, but during queries, multiple versions must be merged, and predicates cannot be pushed down, which may affect query speed.

In Doris, there are two types of update semantics for the Unique Key Model:

- **Full Row Upsert**: The default update semantic for the Unique Key Model is **full row UPSERT**, i.e., UPDATE OR INSERT. If the Key of the row exists, it will be updated; if it does not exist, new data will be inserted. In the full row UPSERT semantic, even if the user inserts data into specific columns using `INSERT INTO`, Doris will fill in the missing columns with NULL values or default values during the planner stage.
- **Partial Column Upsert**: If users want to update specific fields, they need to use the merge-on-write implementation and enable partial column updates support via specific parameters. Please refer to the documentation on Partial Column Updates.

Merge-on-write

Creating a Merge-on-write Table

To create a Unique Key table, use the `UNIQUE KEY` keyword. Enable merge-on-write mode by setting the `enable_unique_key_merge_on_write` attribute (default since Doris 2.1):

```
CREATE TABLE IF NOT EXISTS example_tbl_unique
(
    user_id         LARGEINT        NOT NULL,
    user_name       VARCHAR(50)     NOT NULL,
    city            VARCHAR(20),
    age             SMALLINT,
    sex             TINYINT
)
UNIQUE KEY(user_id, user_name)
DISTRIBUTED BY HASH(user_id) BUCKETS 10
PROPERTIES (
    "enable_unique_key_merge_on_write" = "true"
);
```

Merge-on-read

Creating a Merge-on-read Table

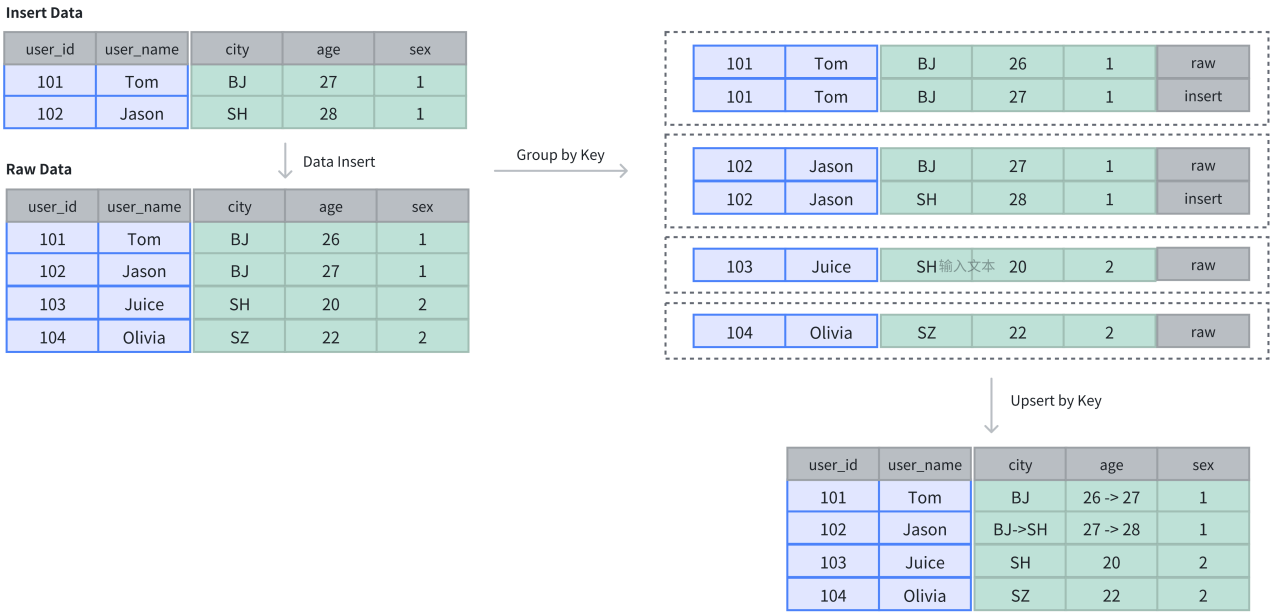
When creating a table, the `UNIQUE KEY` keyword can be used to specify a Unique Key table. The merge-on-read mode can be enabled by explicitly disabling the `enable_unique_key_merge_on_write` attribute. Before Doris version 2.1, the merge-on-read mode was enabled by default:

```
CREATE TABLE IF NOT EXISTS example_tbl_unique
(
    user_id         LARGEINT        NOT NULL,
    user_name       VARCHAR(50)     NOT NULL,
    city            VARCHAR(20),
    age             SMALLINT,
    sex             TINYINT
)
UNIQUE KEY(user_id, user_name)
DISTRIBUTED BY HASH(user_id) BUCKETS 10
PROPERTIES (
```

```
    "enable_unique_key_merge_on_write" = "false"
);
```

Data Insertion and Storage

In a Unique Key table, the Key columns serve both for sorting and deduplication. New insertions overwrite existing records with matching keys.



unique-key-model-insert

As shown in the example, there were 4 rows of data in the original table. After inserting 2 new rows, the newly inserted rows are updated based on the unique key:

```
-- insert into raw data
INSERT INTO example_tbl_unique VALUES
(101, 'Tom', 'BJ', 26, 1),
(102, 'Jason', 'BJ', 27, 1),
(103, 'Juice', 'SH', 20, 2),
(104, 'Olivia', 'SZ', 22, 2);

-- insert into data to update by key
INSERT INTO example_tbl_unique VALUES
(101, 'Tom', 'BJ', 27, 1),
(102, 'Jason', 'SH', 28, 1);

-- check updated data
SELECT * FROM example_tbl_unique;
+---------+-----------+------+------+------+
| user_id | user_name | city | age  | sex  |
+---------+-----------+------+------+------+
| 101     | Tom       | BJ   |   27 |    1 |
| 102     | Jason     | SH   |   28 |    1 |
| 104     | Olivia    | SZ   |   22 |    2 |
| 103     | Juice     | SH   |   20 |    2 |
+---------+-----------+------+------+------+
```

Notes

- The implementation mode for Unique Key tables is fixed at creation and cannot be changed via schema alterations.
- Under full row UPSERT semantics, if specific columns are omitted during insertion, Doris fills them with NULL or default values during planning.
- For partial column upsert, enable merge-on-write mode with the appropriate parameters. Refer to Partial Column Updates for guidance.
- When using a Unique table, the partition key must be included in the Key columns to ensure data uniqueness.

Aggregate Model

Doris's **Aggregate Key Model** is designed to efficiently handle aggregation operations in large-scale data queries. By performing pre-aggregation on the data, it reduces redundancy in computations and improves query performance. The model stores only aggregated data, omitting raw data, which saves storage space and enhances query performance.

Use Cases

- **Summarizing Detailed Data**: The Aggregate Key Model is used in scenarios like e-commerce platforms evaluating monthly sales, financial risk control calculating customer transaction totals, or advertising campaigns analyzing total ad clicks, for multidimensional summarization of detailed data.
- **No Need to Query Raw Detailed Data**: For use cases such as dashboard reports or user transaction behavior analysis, where the raw data is stored in a data lake and does not need to be retained in the database, only the aggregated data is stored.

Principle

Each data import creates a version in the Aggregate Key Model, and during the **Compaction** stage, versions are merged. When querying, data is aggregated by the primary key:

- **Data Import Stage**
    - Data is imported into the aggregate key table in batches, with each batch creating a new version.
    - Within each version, data with the same aggregation keys is pre-aggregated (e.g., sum, count, etc.).
- **Background File Merging Stage (Compaction)**
    - Multiple batches generate multiple version files, which are periodically merged into a larger version file.
    - During the merge process, data with the same aggregation key is re-aggregated to reduce redundancy and optimize storage.
- **Query Stage**
    - During queries, the system aggregates data with the same aggregation key from all versions to ensure accurate results.
    - This process ensures that aggregation operations are performed efficiently, even with large data volumes. The aggregated results are optimized for fast querying, providing a significant performance improvement over raw data queries

Table Creation Instructions

When creating a table, the **AGGREGATE KEY** keyword can be used to specify the Aggregate Key Model. The Aggregate Key Model must specify Key columns, which are used to aggregate Value columns during storage.

```
CREATE TABLE IF NOT EXISTS example_tbl_agg
(
    user_id             LARGEINT    NOT NULL,
    load_dt             DATE        NOT NULL,
    city                VARCHAR(20),
    last_visit_dt       DATETIME    REPLACE DEFAULT "1970-01-01 00:00:00",
    cost                BIGINT      SUM DEFAULT "0",
    max_dwell           INT         MAX DEFAULT "0",
)
AGGREGATE KEY(user_id, load_dt, city)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

In the example above, a fact table for user information and access behavior is defined, where `user_id`, `load_dt`, and `city` are used as Key columns for aggregation. During data import, the Key columns are aggregated into one row, and the Value columns are aggregated according to the specified aggregation types.

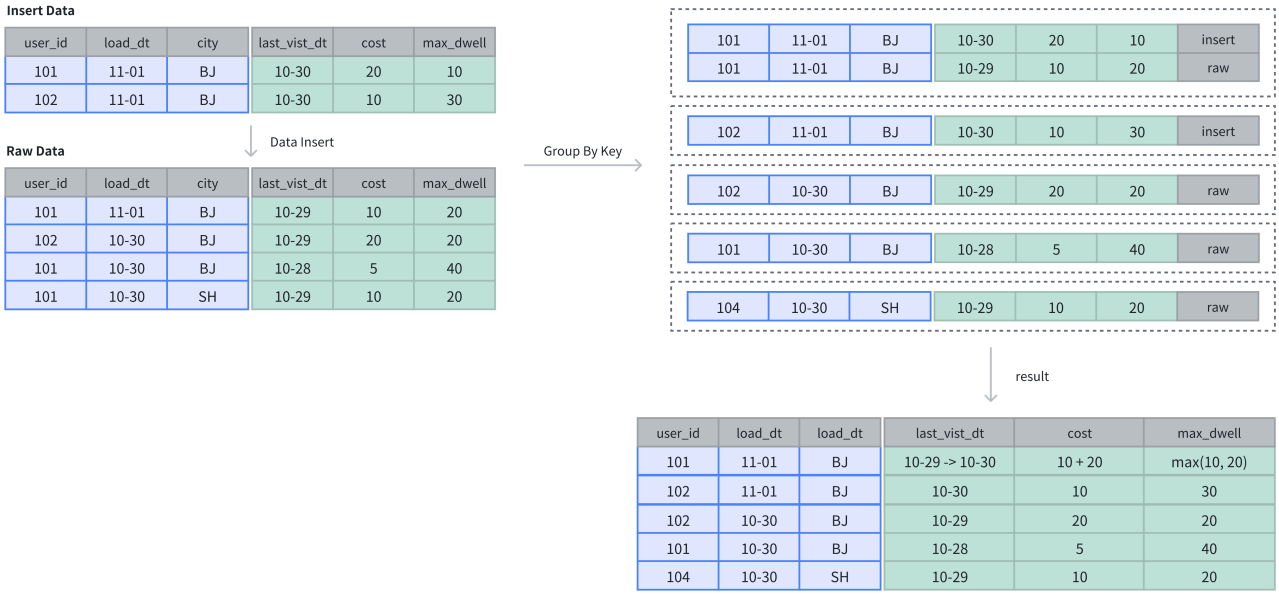
The following types of dimension aggregation are supported in the Aggregate Key Model:

Aggregation Method	Description
SUM	Sum, accumulates multiple Value rows.
REPLACE	Replacement, the Value in the next batch replaces the previously inserted Value.
MAX	Retain the maximum value.
MIN	Retain the minimum value.
REPLACE_IF_NOT_NULL	Replace non-null values. Unlike REPLACE, null values are not replaced.
HLL_UNION	Aggregation method for HLL type columns, using the HyperLogLog algorithm.
BITMAP_UNION	Aggregation method for BITMAP type columns, performing bitmap union aggregation.

> **Tip:**
>
> If the aggregation methods above do not meet your business requirements, consider using the `agg_state` type.

Data Insertion and Storage

In the Aggregate Key table, data is aggregated based on the primary key. After data insertion, aggregation operations are completed.



aggrate-key-model-insert

In the example above, there were originally 4 rows of data in the table. After inserting 2 rows, aggregation operations on the dimension columns are performed based on the Key columns:

```
-- 4 rows raw data
INSERT INTO example_tbl_agg VALUES
(101, '2024-11-01', 'BJ', '2024-10-29', 10, 20),
(102, '2024-10-30', 'BJ', '2024-10-29', 20, 20),
(101, '2024-10-30', 'BJ', '2024-10-28', 5, 40),
(101, '2024-10-30', 'SH', '2024-10-29', 10, 20);

-- insert into 2 rows
INSERT INTO example_tbl_agg VALUES
(101, '2024-11-01', 'BJ', '2024-10-30', 20, 10),
(102, '2024-11-01', 'BJ', '2024-10-30', 10, 30);

-- check the rows of table
SELECT * FROM example_tbl_agg;
+---------+------------+------+---------------------+------+----------------+
| user_id | load_date  | city | last_visit_dt       | cost | max_dwell      |
+---------+------------+------+---------------------+------+----------------+
| 102     | 2024-10-30 | BJ   | 2024-10-29 00:00:00 |   20 |             20 |
| 102     | 2024-11-01 | BJ   | 2024-10-30 00:00:00 |   10 |             30 |
| 101     | 2024-10-30 | BJ   | 2024-10-28 00:00:00 |    5 |             40 |
| 101     | 2024-10-30 | SH   | 2024-10-29 00:00:00 |   10 |             20 |
| 101     | 2024-11-01 | BJ   | 2024-10-30 00:00:00 |   30 |             20 |
+---------+------------+------+---------------------+------+----------------+
```

AGG_STATE

::: Info Tips: AGG_STATE is an experimental feature and is recommended for use in development and testing environments.

AGG_STATE cannot be used as a Key column. The aggregation function's signature must be declared when creating the table. Users don't need to specify length or default values. The data storage size depends on the function implementation.

```
set enable_agg_state = true;
CREATE TABLE aggstate(
    k1   int  NULL,
    v1   int  SUM,
    v2   agg_state<group_concat(string)> generic
)
AGGREGATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 3;
```

In this case, `agg_state` is used to declare the data type as `agg_state`, and `sum/group_concat` is the signature of the aggregation function. Note that `agg_state` is a data type, just like `int`, `array`, or `string`. `agg_state` can only be used with combinators such as state, merge, or union. `agg_state` represents the intermediate result of an aggregation function. For example, for the aggregation function `group_concat`, `agg_state` can represent the intermediate state of `group_concat('a', 'b', 'c')`, rather than the final result.

The `agg_state` type needs to be generated using the `state` function. For this table, you need to use `group_concat_state`:

```
insert into aggstate values(1, 1, group_concat_state('a'));
insert into aggstate values(1, 2, group_concat_state('b'));
insert into aggstate values(1, 3, group_concat_state('c'));
insert into aggstate values(2, 4, group_concat_state('d'));
```

The calculation method in the table is shown in the diagram below:

K1	V1	V2
1	sum(1, 2, 3)	group_concat('a', 'b', 'c')
2	sum(4)	group_concat('d')

state-func-group-concat-state-result-1

When querying the table, the merge operation can be used to merge multiple `state` values and return the final aggregation result. Since `group_concat` requires ordering, the result may be unstable.

```
select group_concat_merge(v2) from aggstate;
+-----------------------+
| group_concat_merge(v2) |
+-----------------------+
| d,c,b,a               |
+-----------------------+
```

If you do not want the final aggregation result, you can use `union` to combine multiple intermediate aggregation results and generate a new intermediate result.

```
insert into aggstate select 3, sum(v1), group_concat_union(v2) from aggstate;
```

The calculations in the table are as follows:

K1	V1	V2
1	sum(1, 2, 3)	group_concat('a', 'b', 'c')
2	sum(4)	group_concat('d')
3	sum(1, 2, 3, 4)	group_concat('a', 'b', 'c', 'd')

state-func-group-concat-state-result-2

The query result is as follows:

```
mysql> select sum(v1), group_concat_merge(v2) from aggstate;
+---------+------------------------+
| sum(v1) | group_concat_merge(v2) |
+---------+------------------------+
|      20 | c,b,a,d,c,b,a,d        |
+---------+------------------------+

mysql> select sum(v1), group_concat_merge(v2) from aggstate where k1 != 2;
+---------+------------------------+
| sum(v1) | group_concat_merge(v2) |
+---------+------------------------+
|      16 | c,b,a,d,c,b,a          |
+---------+------------------------+
```

Usage Notes

Suggestions for column types

Suggestions for column types when creating a table:

1. The Key column should precede all Value columns.
2. Whenever possible, choose integer types. This is because the calculation and search efficiency of integer types are much higher than that of strings.
3. For selecting integer types of different lengths, follow the principle of sufficiency.
4. For the lengths of VARCHAR and STRING types, also follow the principle of sufficiency.

Limitations of aggregate model

This section is about the limitations of the Aggregate Model.

The Aggregate Model only presents the aggregated data. That means we have to ensure the presentation consistency of data that has not yet been aggregated (for example, two different import batches). The following provides further explanation with examples.

Suppose that you have the following table schema:

ColumnName	Type	AggregationType	Comment
user_id	LARGEINT		User ID
date	DATE		Date when the data are imported
cost	BIGINT	SUM	Total user consumption

Assume that there are two batches of data that have been imported into the storage engine as follows:

batch 1

user_id	date	cost
10001	2017-11-20	50
10002	2017-11-21	39

batch 2

user_id	date	cost
10001	2017-11-20	1
10001	2017-11-21	5
10003	2017-11-22	22

As you can see, data about User 10001 in these two import batches have not yet been aggregated. However, in order to ensure that users can only query the aggregated data as follows:

user_id	date	cost
10001	2017-11-20	51
10001	2017-11-21	5
10002	2017-11-21	39
10003	2017-11-22	22

We have added an aggregation operator to the query engine to ensure the presentation consistency of data.

In addition, on the aggregate column (Value), when executing aggregate class queries that are inconsistent with the aggregate type, please pay attention to the semantics. For example, in the example above, if you execute the following query:

```
SELECT MIN(cost) FROM table;
```

The result will be 5, not 1.

Meanwhile, this consistency guarantee could considerably reduce efficiency in some queries.

Take the basic count (*) query as an example:

```
SELECT COUNT(*) FROM table;
```

In other databases, such queries return results quickly. Because in actual implementation, the models can get the query result by counting rows and saving the statistics upon import, or by scanning only one certain column of data to get count value upon query, with very little overhead. But in Doris's Aggregation Model, the overhead of such queries is **large**.

For the previous example:

batch 1

user_id	date	cost
10001	2017-11-20	50
10002	2017-11-21	39

batch 2

user_id	date	cost
10001	2017-11-20	1
10001	2017-11-21	5
10003	2017-11-22	22

Since the final aggregation result is:

user_id	date	cost
10001	2017-11-20	51
10001	2017-11-21	5

user_id	date	cost
10002	2017-11-21	39
10003	2017-11-22	22

The correct result of `select count (*) from table;` should be **4**. But if the model only scans the `user_id` column and operates aggregation upon query, the final result will be **3** (10001, 10002, 10003). And if it does not operate aggregation, the final result will be **5** (a total of five rows in two batches). Apparently, both results are wrong.

In order to get the correct result, we must read both the `user_id` and `date` column, and **performs aggregation** when querying. That is to say, in the `count (*)` query, Doris must scan all AGGREGATE KEY columns (in this case, `user_id` and `date`) and aggregate them to get the semantically correct results. That means if there are many aggregated columns, `count (*)` queries could involve scanning large amounts of data.

Therefore, if you need to perform frequent `count (*)` queries, we recommend that you simulate `count (*)` by adding a column of value 1 and aggregation type SUM. In this way, the table schema in the previous example will be modified as follows:

ColumnName	Type	AggregationType	Comment
user ID	BIGINT		User ID
date	DATE		Date when the data are imported
Cost	BIGINT	SUM	Total user consumption
count	BIGINT	SUM	For count queries

The above adds a count column, the value of which will always be **1**, so the result of `select count (*) from table;` is equivalent to that of `select sum (count) from table;` The latter is much more efficient than the former. However, this method has its shortcomings, too. That is, it requires that users will not import rows with the same values in the AGGREGATE KEY columns. Otherwise, `select sum (count) from table;` can only express the number of rows of the originally imported data, instead of the semantics of `select count (*) from table;`

Another method is to add a `cound` column of value 1 but aggregation type of REPLACE. Then `select sum (count) from table;` and `select count (*) from table;` could produce the same results. Moreover, this method does not require the absence of same AGGREGATE KEY columns in the import data.

MoW Unique Key Model

The Merge on Write implementation in the Unique Model does not impose the same limitation as the Aggregate Model. In Merge on Write, the model adds a `delete bitmap` for each imported rowset to mark the data being overwritten or deleted. With the previous example, after Batch 1 is imported, the data status will be as follows:

batch 1

user_id	date	cost	delete bit
10001	2017-11-20	50	false
10002	2017-11-21	39	false

After Batch 2 is imported, the duplicate rows in the first batch will be marked as deleted, and the status of the two batches of data is as follows

batch 1

user_id	date	cost	delete bit
10001	2017-11-20	50	**true**
10002	2017-11-21	39	false

batch 2

user_id	date	cost	delete bit
10001	2017-11-20	1	false
10001	2017-11-21	5	false
10003	2017-11-22	22	false

In queries, all data marked `true` in the `delete bitmap` will not be read, so there is no need for data aggregation. Since there are 4 valid rows in the above data, the query result should also be 4. This also enables minimum overhead since it only scans one column of data.

In the test environment, `count(*)` queries in Merge on Write of the Unique Model deliver 10 times higher performance than that of the Aggregate Model.

Duplicate model

The Duplicate Model does not impose the same limitation as the Aggregate Model because it does not involve aggregation semantics. For any columns, it can return the semantically correct results in `count (*)` queries.

Key columns

For the Duplicate, Aggregate, and Unique Models, the Key columns will be specified when the table is created, but there exist some differences: In the Duplicate Model, the Key columns of the table can be regarded as just “sorting columns”, but not unique identifiers. In Aggregate and Unique Models, the Key columns are both “sorting columns” and “unique identifier columns”.

Suggestions for choosing data model

Since the data model was established when the table was built, and **irrevocable thereafter, it is very important to select the appropriate data model**.

1. The Aggregate Model can greatly reduce the amount of data scanned and query computation by pre-aggregation. Thus, it is very suitable for report query scenarios with fixed patterns. But this model is unfriendly to `count (*)` queries. Meanwhile, since the aggregation method on the Value column is fixed, semantic correctness should be considered in other types of aggregation queries.
2. The Unique Model guarantees the uniqueness of primary key for scenarios requiring a unique primary key. The downside is that it cannot exploit the advantage brought by pre-aggregation such as ROLLUP

in queries. Users who have high-performance requirements for aggregate queries are recommended to use the newly added Merge on Write implementation since version 1.2.

3. The Duplicate Model is suitable for ad-hoc queries of any dimensions. Although it may not be able to take advantage of the pre-aggregation feature, it is not limited by what constrains the Aggregate Model and can give full play to the advantage of columnar storage (reading only the relevant columns, but not all Key columns).
4. If user need to use partial-update, please refer to document partial-update

Data Distribution Concept

In Doris, the core of **data distribution** is to efficiently map the rows of data written to the table onto various **data shards (Tablets)** in the underlying storage through reasonable partitioning and bucketing strategies. Through data distribution strategies, Doris can fully utilize the storage and computing capabilities of multiple nodes, thereby supporting efficient storage and querying of large-scale data.

Overview of Data Distribution

Data Writing

When writing data, Doris first allocates the rows of data to the corresponding partitions based on the table's partitioning strategy. Then, according to the bucketing strategy, the rows of data are further mapped to specific shards within the partition, determining the storage location of the data rows.

Query Execution

During query execution, Doris's optimizer will trim data based on partitioning and bucketing strategies to maximize the reduction of the scanning range. In cases involving JOIN or aggregation queries, data transfer across nodes (Shuffle) may occur. Reasonable partitioning and bucketing design can reduce Shuffle and fully utilize **Colocate Join** to optimize query performance.

Node and Storage Architecture

Node Types

A Doris cluster consists of the following two types of nodes:

- **FE Node (Frontend)**: Manages cluster metadata (such as tables and shards) and is responsible for SQL parsing and execution planning.
- **BE Node (Backend)**: Stores data and is responsible for executing computational tasks. The results from BE are aggregated and returned to FE, which then returns them to the user.

Data Shard (Tablet)

The data stored in the BE node is divided into shards, with each shard being the smallest unit of data management in Doris and the basic unit for data movement and replication.

Partitioning Strategy

Partitioning is the first layer of logical division for data organization, used to divide the data in the table into smaller subsets. Doris provides the following two **partition types** and three **partition modes**:

Partition Types

- **Range Partitioning**: Allocates data rows to corresponding partitions based on the value range of the partition column.
- **List Partitioning**: Allocates data rows to corresponding partitions based on specific values of the partition column.

Partition Modes

- **Manual Partitioning**: Users manually create partitions (e.g., specified during table creation or added via `ALTER` statements).
- **Dynamic Partitioning**: The system automatically creates partitions based on time scheduling rules, but does not create partitions on demand when writing data.
- **Automatic Partitioning**: The system automatically creates corresponding partitions as needed during data writing, but care should be taken to avoid generating too many partitions with dirty data.

Bucketing Strategy

Bucketing is the second layer of logical division for data organization, used to further divide data rows into smaller units within a partition. Doris supports the following two bucketing methods:

- **Hash Bucketing**: Distributes data rows evenly across shards by calculating the `crc32` hash value of the bucketing column and taking the modulus of the number of buckets.
- **Random Bucketing**: Randomly allocates data rows to shards. When using Random bucketing, the `load_to_single_tablet` option can be used to optimize the quick writing of small-scale data.

Data Distribution Optimization

Colocate Join

For large tables that frequently require JOIN or aggregation queries, the **Colocate** strategy can be enabled to place data with the same bucketing column values on the same physical node, reducing data transfer across nodes and significantly improving query performance.

Partition Pruning

During queries, Doris can prune irrelevant partitions through filtering conditions, thereby reducing the data scanning range and lowering I/O costs.

Bucketing Parallelism

During queries, a reasonable number of buckets can fully utilize the computational and I/O resources of the machines.

Data Distribution Goals

1. **Uniform Data Distribution** Ensure that data is evenly distributed across all BE nodes to avoid data skew that could overload certain nodes, thereby improving overall system performance.
2. **Optimize Query Performance** Reasonable partition pruning can significantly reduce the amount of data scanned, a reasonable number of buckets can enhance computational parallelism, and effective use of COLOCATE can lower Shuffle costs, improving JOIN and aggregation query efficiency.
3. **Flexible Data Management**
    - Time-based partitioning to store cold data (HDD) and hot data (SSD).
    - Regularly delete historical partitions to free up storage space.
4. **Control Metadata Scale** The metadata for each shard is stored in both FE and BE, so it is necessary to reasonably control the number of shards. The empirical recommendation is:
    - For every 10 million shards, FE requires at least 100GB of memory.
    - The number of shards handled by a single BE should be less than 20,000.
5. **Optimize Write Throughput**
    - The number of buckets should be reasonably controlled (recommended < 128) to avoid degrading write performance.
    - The number of partitions written at one time should be appropriate (recommended to write a small number of partitions at a time).

By carefully designing and managing partitioning and bucketing strategies, Doris can efficiently support the storage and query processing of large-scale data, meeting various complex business needs.

Manual partitioning

Partition columns

- Partition columns can be specified as one or multiple columns, and the partition columns must be KEY columns.
- When `allow_partition_column_nullable` is set to true, Range partition supports the use of NULL partition columns. List Partition does not support NULL partition columns at all times.
- Regardless of the type of partition column, double quotes are required when writing partition values.
- There is theoretically no upper limit on the number of partitions. However, each table is limited to 4096 partitions by default. If you want to exceed this limit, you can modify the FE configuration parameters `max_multi_partition_num` and `max_dynamic_partition_num`.
- When creating a table without partitioning, the system will automatically generate a full-range partition with the same name as the table name. This partition is not visible to users and cannot be deleted or modified.
- Overlapping ranges are not allowed when creating partitions.

RANGE partitioning

Partition columns are usually time columns for convenient management of old and new data. RANGE partitioning supports column types such as `DATE`, `DATETIME`, `TINYINT`, `SMALLINT`, `INT`, `BIGINT`, and `LARGEINT`.

Partition information supports the following four writing methods:

- `FIXED RANGE`: This method defines the partition as a left-closed, right-open interval.

```
PARTITION BY RANGE(col1[, col2, ...])
(
    PARTITION partition_name1 VALUES [("k1-lower1", "k2-lower1", "k3-lower1",...), ("k1-upper1", "k2-upper1", "k3-upper1", ...)),
    PARTITION partition_name2 VALUES [("k1-lower1-2", "k2-lower1-2", ...), ("k1-upper1-2", MAXVALUE, ))
)
```

For example:

```
PARTITION BY RANGE(`date`)
(
    PARTITION `p201701` VALUES [("2017-01-01"),  ("2017-02-01")),
    PARTITION `p201702` VALUES [("2017-02-01"), ("2017-03-01")),
    PARTITION `p201703` VALUES [("2017-03-01"), ("2017-04-01"))
)
```

- `LESS THAN`: This method only defines the upper bound of the partition. The lower bound is determined by the upper bound of the previous partition.

```
PARTITION BY RANGE(col1[, col2, ...])
(
    PARTITION partition_name1 VALUES LESS THAN MAXVALUE | ("value1", "value2", ...),
    PARTITION partition_name2 VALUES LESS THAN MAXVALUE | ("value1", "value2", ...)
)
```

For example:

```
PARTITION BY RANGE(`date`)
(
    PARTITION `p201701` VALUES LESS THAN ("2017-02-01"),
    PARTITION `p201702` VALUES LESS THAN ("2017-03-01"),
    PARTITION `p201703` VALUES LESS THAN ("2017-04-01")
)

PARTITION BY RANGE(`date`)
(
    PARTITION `p201701` VALUES LESS THAN ("2017-02-01"),
    PARTITION `p201702` VALUES LESS THAN ("2017-03-01"),
    PARTITION `p201703` VALUES LESS THAN ("2017-04-01")
    PARTITION `other` VALUES LESS THAN (MAXVALUE)
)
```

LIST partitioning

Partition columns based on LIST partitioning support data types such as `BOOLEAN`, `TINYINT`, `SMALLINT`, `INT`, `BIGINT`, `LARGEINT`, `DATE`, `DATETIME`, `CHAR`, and `VARCHAR`. Partition values are enumerated values. Only when the data is one of the enumerated values of the target partition, the partition can be hit.

Partitions support specifying the enumerated values contained in each partition through `VALUES IN (...)`.

For example:

```
PARTITION BY LIST(city)
(
    PARTITION `p_cn` VALUES IN ("Beijing", "Shanghai", "Hong Kong"),
    PARTITION `p_usa` VALUES IN ("New York", "San Francisco"),
    PARTITION `p_jp` VALUES IN ("Tokyo")
)
```

LIST partitioning also supports multi-column partitioning, for example:

```
PARTITION BY LIST(id, city)
(
    PARTITION p1_city VALUES IN (("1", "Beijing"), ("1", "Shanghai")),
    PARTITION p2_city VALUES IN (("2", "Beijing"), ("2", "Shanghai")),
    PARTITION p3_city VALUES IN (("3", "Beijing"), ("3", "Shanghai"))
)
```

NULL partitioning

Partition columns based on NULL partitioning must be not null columns by default. If you need to use null columns, set the session variable `allow_partition_column_nullable` to `true`. For LIST partitioning, the NULL

partitioning is supported, while for RANGE partitioning, null values will be assigned to the `less than` partition. The columns are as follows:

LIST partitioning

```
mysql> create table null_list(
    -> k0 varchar null
    -> )
    -> partition by list (k0)
    -> (
    -> PARTITION pX values in ((NULL))
    -> )
    -> DISTRIBUTED BY HASH(`k0`) BUCKETS 1
    -> properties("replication_num" = "1");
Query OK, 0 rows affected (0.11 sec)

mysql> insert into null_list values (null);
Query OK, 1 row affected (0.19 sec)

mysql> select * from null_list;
+------+
| k0   |
+------+
| NULL |
+------+
1 row in set (0.18 sec)
```

RANGE partitioning with the `less than` partition

```
mysql> create table null_range(
    -> k0 int null
    -> )
    -> partition by range (k0)
    -> (
    -> PARTITION p10 values less than (10),
    -> PARTITION p100 values less than (100),
    -> PARTITION pMAX values less than (maxvalue)
    -> )
    -> DISTRIBUTED BY HASH(`k0`) BUCKETS 1
    -> properties("replication_num" = "1");
Query OK, 0 rows affected (0.12 sec)

mysql> insert into null_range values (null);
Query OK, 1 row affected (0.19 sec)

mysql> select * from null_range partition(p10);
+------+
| k0   |
+------+
| NULL |
+------+
1 row in set (0.18 sec)
```

RANGE partitioning without the `less than` partition

```
mysql> create table null_range2(
    -> k0 int null
    -> )
    -> partition by range (k0)
    -> (
```

```
    -> PARTITION p200 values [("100"), ("200"))
    -> )
    -> DISTRIBUTED BY HASH(`k0`) BUCKETS 1
    -> properties("replication_num" = "1");
Query OK, 0 rows affected (0.13 sec)

mysql> insert into null_range2 values (null);
ERROR 5025 (HY000): Insert has filtered data in strict mode, tracking_url=......
```

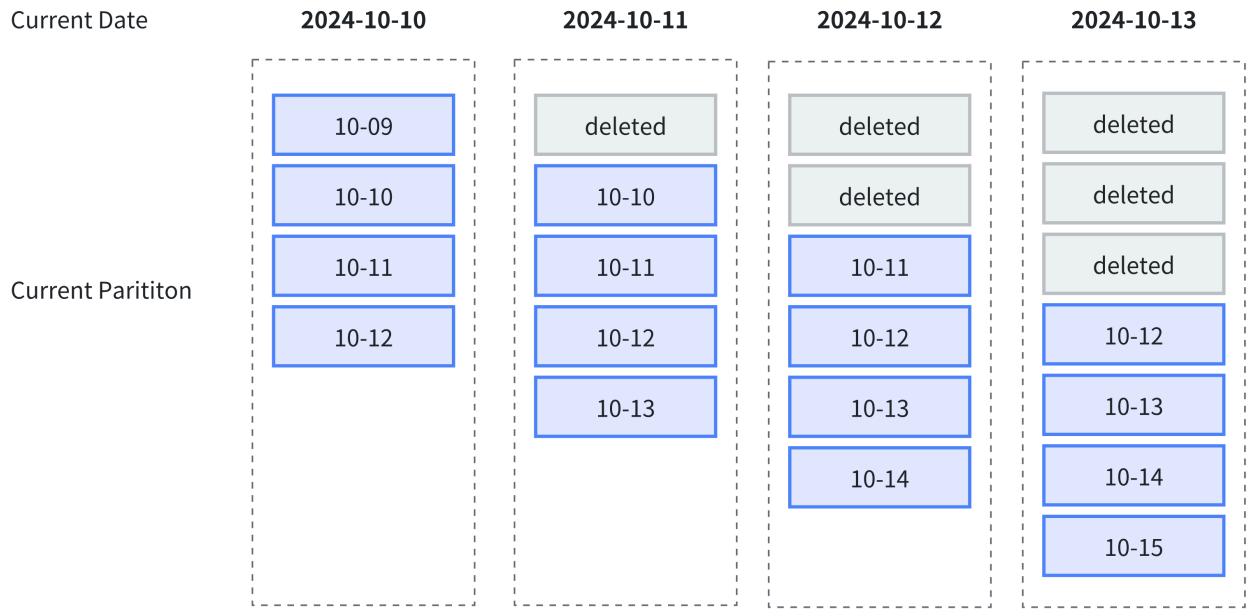
Dynamic Partitioning

Dynamic partitioning will add and remove partitions in a rolling manner according to predefined rules, thereby managing the lifecycle of table partitions (TTL) and reducing data storage pressure. In scenarios such as log management and time-series data management, dynamic partitioning can typically be used to roll-delete expired data.

The diagram below illustrates lifecycle management using dynamic partitioning, with the following rules specified:

- The dynamic partition scheduling unit `dynamic_partition.time_unit` is set to DAY, organizing partitions by day;
- The dynamic partition start offset `dynamic_partition.start` is set to -1, retaining the partition from one day ago;
- The dynamic partition end offset `dynamic_partition.end` is set to 2, retaining partitions for the next two days.

According to the above rules, as time progresses, a total of 4 partitions will always be retained: the partition from the past day, the current day partition, and the partitions for the next two days.



dynamic-partition

Usage Restrictions

When using dynamic partitioning, the following rules must be followed:

- Dynamic partitioning will fail when used simultaneously with Cross-Cluster Replication (CCR).
- Dynamic partitioning only supports Range type partitions on DATE/DATETIME columns.
- Dynamic partitioning only supports a single partition key.

Creating Dynamic Partitions

When creating a table, you can create a dynamic partitioned table by specifying the `dynamic_partition` property.

```
CREATE TABLE test_dynamic_partition(
    order_id    BIGINT,
    create_dt   DATE,
    username    VARCHAR(20)
)
DUPLICATE KEY(order_id)
PARTITION BY RANGE(create_dt) ()
DISTRIBUTED BY HASH(order_id) BUCKETS 10
PROPERTIES(
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-1",
    "dynamic_partition.end" = "2",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.create_history_partition" = "true"
);
```

In the example above, a dynamic partitioned table was created with the following specifications.

For detailed `dynamic_partition` parameters, refer to Dynamic Partition Parameter Description.

Managing Dynamic Partitions

Modifying Dynamic Partition Properties

> **Tip:**
>
> When using the ALTER TABLE statement to modify dynamic partitioning, the changes will not take effect immediately. The dynamic partitions will be polled and checked at intervals specified by the `dynamic_partition_check_interval_seconds` parameter to complete the necessary partition creation and deletion operations.

In the example below, the ALTER TABLE statement is used to modify a non-dynamic partitioned table to a dynamic partitioned table:

```
CREATE TABLE test_partition(
    order_id    BIGINT,
    create_dt   DATE,
    username    VARCHAR(20)
)
DUPLICATE KEY(order_id)
PARTITION BY RANGE(create_dt) ()
DISTRIBUTED BY HASH(order_id) BUCKETS 10;

ALTER TABLE test_partition SET (
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-1",
    "dynamic_partition.end" = "2",
    "dynamic_partition.prefix" = "p",
```

```
    "dynamic_partition.create_history_partition" = "true"
);
```

查看动态分区调度情况

通过 SHOW-DYNAMIC-PARTITION 可以查看当前数据库下，所有动态分区表的调度情况：

```
SHOW DYNAMIC PARTITION TABLES;
+-----------+--------+----------+-------------+------+--------+---------+-----------+----------------+---------------------+--------
+-----------------------+----------------------+-------------------------+
| TableName | Enable | TimeUnit | Start       | End  | Prefix | Buckets | StartOf   | LastUpdateTime | LastSchedulerTime   | State
| LastCreatePartitionMsg | LastDropPartitionMsg | ReservedHistoryPeriods  |
+-----------+--------+----------+-------------+------+--------+---------+-----------+----------------+---------------------+--------
+-----------------------+----------------------+-------------------------+
| d3        | true   | WEEK     | -3          | 3    | p      | 1       | MONDAY    | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | [2021-12-01,2021-12-31] |
| d5        | true   | DAY      | -7          | 3    | p      | 32      | N/A       | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | NULL                    |
| d4        | true   | WEEK     | -3          | 3    | p      | 1       | WEDNESDAY | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | NULL                    |
| d6        | true   | MONTH    | -2147483648 | 2    | p      | 8       | 3rd       | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | NULL                    |
| d2        | true   | DAY      | -3          | 3    | p      | 32      | N/A       | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | NULL                    |
| d7        | true   | MONTH    | -2147483648 | 5    | p      | 8       | 24th      | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | NULL                    |
+-----------+--------+----------+-------------+------+--------+---------+-----------+----------------+---------------------+--------
+-----------------------+----------------------+-------------------------+
7 rows in set (0.02 sec)
```

Historical Partition Management

When specifying the number of dynamic partitions using the `start` and `end` attributes, historical partitions are not created all at once to avoid long waiting times. Only partitions after the current time are created. If you need to create all partitions at once, you must enable the `create_history_partition` parameter.

For example, if the current date is 2024-10-11 and you set `start = -2` and `end = 2`:

- If `create_history_partition = true` is specified, all partitions are created immediately, resulting in five partitions: [10-09, 10-13].
- If `create_history_partition = false` is specified, only partitions from 10-11 onwards are created, resulting in three partitions: [10-11, 10-13].

Dynamic Partition Parameter Description

Dynamic Partition Property Parameters

Dynamic partition rule parameters are prefixed with `dynamic_partition` and can be set with the following rule parameters:

Parameter	Required	Description
`dynamic_partition.enable`	No	Whether to enable the dynamic partition feature. Can be set to TRUE or FALSE. If other required dynamic partition parameters are specified, it defaults to TRUE.
`dynamic_partition.time_unit`	Yes	The unit of dynamic partition scheduling. Can be set to `HOUR`, `DAY`, `WEEK`, `MONTH`, or `YEAR`, indicating partition creation or deletion by hour, day, week, month, or year respectively.
`dynamic_partition.start`	No	The starting offset for dynamic partitions, which is a negative number. The default value is -2147483648, meaning historical partitions are not deleted. Depending on the `time_unit` attribute, partitions before this offset based on the current day (week/month) will be deleted. Whether historical partitions after this offset up to the current time are created depends on `dynamic_partition.create_history_partition`.
`dynamic_partition.end`	Yes	The ending offset for dynamic partitions, which is a positive number. Depending on the `time_unit` attribute, partitions within the specified range ahead of the current day (week/month) are created in advance.
`dynamic_partition.prefix`	Yes	The prefix for dynamically created partition names.
`dynamic_partition.buckets`	No	The number of buckets corresponding to dynamically created partitions. Setting this parameter will override the number of buckets specified in `DISTRIBUTED`.
`dynamic_partition.replication_num`	No	The number of replicas corresponding to dynamically created partitions. If not specified, it defaults to the number of replicas specified when the table was created.
`dynamic_partition.create_history_partition`	No	Defaults to false. When set to true, Doris will automatically create all partitions according to the rules below. Additionally, the FE parameter `max_dynamic_partition_num` will limit the total number of partitions to avoid creating too many partitions at once. If the number of partitions to be created exceeds the `max_dynamic_partition_num` value, the operation will be prohibited. This parameter does not take effect if the `start` attribute is not specified.
`dynamic_partition.history_partition_num`	No	When `create_history_partition` is set to `true`, this parameter specifies the number of historical partitions to create. The default value is -1, meaning it is not set. This variable functions the same as `dynamic_partition.start`, and it is recommended to set only one of them simultaneously.
`dynamic_partition.start_day_of_week`	No	When `time_unit` is set to `WEEK`, this parameter specifies the starting day of the week. Values range from 1 to 7, where 1 represents Monday and 7 represents Sunday. The default is 1, meaning the week starts on Monday.

Parameter	Required	Description
dynamic_partition.start_day_of_month	No	When time_unit is set to MONTH, this parameter specifies the starting date of the month. Values range from 1 to 28, where 1 represents the first day of the month and 28 represents the 28th day. The default is 1, meaning the month starts on the first day. Starting on the 29th, 30th, or 31st is not supported to avoid ambiguities caused by leap years or leap months.
dynamic_partition.reserved_history_periods	No	The time range of historical partitions that need to be retained. When dynamic_partition.time_unit is set to "DAY/WEEK/MONTH/YEAR", it should be set in the format [yyyy-MM-dd,yyyy-MM-dd],[...,...]. When dynamic_partition.time_unit is set to "HOUR", it should be set in the format [yyyy-MM-dd HH:mm:ss,yyyy-MM-dd HH:mm:ss], [...,...]. If not set, it defaults to "NULL".
dynamic_partition.time_zone	No	The time zone for dynamic partitioning, which defaults to the server's system time zone, such as Asia/Shanghai. For more time zone settings, refer to Time Zone Management.

FE Configuration Parameters

Dynamic partition parameter configurations in FE can be modified in the FE configuration file or via the ADMIN SET FRONTEND CONFIG command:

Parameter	Default Value	Description
dynamic_partition_enable	false	Whether to enable Doris's dynamic partition feature. This parameter only affects partition operations of dynamic partition tables and does not affect regular tables.
dynamic_partition_check_interval_seconds	600	The execution frequency of the dynamic partition thread, in seconds.
max_dynamic_partition_num	500	Limits the maximum number of partitions that can be created when creating a dynamic partition table to avoid creating too many partitions at once.

Dynamic Partition Best Practices

Example 1: Partition by day, retain partitions for the past 7 days and the current day, and pre-create partitions for the next 3 days.

```
CREATE TABLE tbl1 (
    order_id    BIGINT,
    create_dt   DATE,
    username    VARCHAR(20)
)
PARTITION BY RANGE(create_dt) ()
DISTRIBUTED BY HASH(create_dt)
PROPERTIES (
```

```
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-7",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "32"
);
```

Example 2: Partition by month, do not delete historical partitions, and pre-create partitions for the next 2 months. Additionally, set the starting day to the 3rd of each month.

```
CREATE TABLE tbl1 (
    order_id    BIGINT,
    create_dt   DATE,
    username    VARCHAR(20)
)
PARTITION BY RANGE(create_dt) ()
DISTRIBUTED BY HASH(create_dt)
PROPERTIES (
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "MONTH",
    "dynamic_partition.end" = "2",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "8",
    "dynamic_partition.start_day_of_month" = "3"
);
```

Example 3: Partition by day, retain partitions for the past 10 days and the next 10 days, and retain historical data during the periods [2020-06-01, 2020-06-20] and [2020-10-31, 2020-11-15].

```
CREATE TABLE tbl1 (
    order_id    BIGINT,
    create_dt   DATE,
    username    VARCHAR(20)
)
PARTITION BY RANGE(create_dt) ()
DISTRIBUTED BY HASH(create_dt)
PROPERTIES (
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-10",
    "dynamic_partition.end" = "10",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "8",
    "dynamic_partition.reserved_history_periods"="[2020-06-01,2020-06-20],[2020-10-31,2020-11-15]"
);
```

Auto Partition

Application scenario

The Auto Partition feature supports automatic detection of whether the corresponding partition exists during the data import process. If it does not exist, the partition will be created automatically and imported normally.

The auto partition function mainly solves the problem that the user expects to partition the table based on a certain column, but the data distribution of the column is scattered or unpredictable, so it is difficult to accurately create the required partitions when building or adjusting the structure of the table, or the number of partitions is so large that it is too cumbersome to create them manually.

Take the time type partition column as an example, in dynamic partitioning, we support the automatic creation of new partitions to accommodate real-time data at specific time periods. For real-time user behavior logs and other scenarios, this feature basically meets the requirements. However, in more complex scenarios, such as dealing with non-real-time data, the partition column is independent of the current system time and contains a large number of discrete values. At this time, to improve efficiency we want to partition the data based on this column, but the data may actually involve the partition can not be grasped in advance, or the expected number of required partitions is too large. In this case, dynamic partitioning or manually created partitions cannot meet our needs, while Auto Partition covers such needs.

Suppose the table DDL is as follows:

```
CREATE TABLE `DAILY_TRADE_VALUE`
(
    `TRADE_DATE`              datev2 NOT NULL COMMENT 'TRADE_DATE',
    `TRADE_ID`                varchar(40) NOT NULL COMMENT 'TRADE_ID',
    ......
)
UNIQUE KEY(`TRADE_DATE`, `TRADE_ID`)
PARTITION BY RANGE(`TRADE_DATE`)
(
    PARTITION p_2000 VALUES [('2000-01-01'), ('2001-01-01')),
    PARTITION p_2001 VALUES [('2001-01-01'), ('2002-01-01')),
    PARTITION p_2002 VALUES [('2002-01-01'), ('2003-01-01')),
    PARTITION p_2003 VALUES [('2003-01-01'), ('2004-01-01')),
    PARTITION p_2004 VALUES [('2004-01-01'), ('2005-01-01')),
    PARTITION p_2005 VALUES [('2005-01-01'), ('2006-01-01')),
    PARTITION p_2006 VALUES [('2006-01-01'), ('2007-01-01')),
    PARTITION p_2007 VALUES [('2007-01-01'), ('2008-01-01')),
    PARTITION p_2008 VALUES [('2008-01-01'), ('2009-01-01')),
    PARTITION p_2009 VALUES [('2009-01-01'), ('2010-01-01')),
    PARTITION p_2010 VALUES [('2010-01-01'), ('2011-01-01')),
    PARTITION p_2011 VALUES [('2011-01-01'), ('2012-01-01')),
    PARTITION p_2012 VALUES [('2012-01-01'), ('2013-01-01')),
    PARTITION p_2013 VALUES [('2013-01-01'), ('2014-01-01')),
    PARTITION p_2014 VALUES [('2014-01-01'), ('2015-01-01')),
    PARTITION p_2015 VALUES [('2015-01-01'), ('2016-01-01')),
    PARTITION p_2016 VALUES [('2016-01-01'), ('2017-01-01')),
    PARTITION p_2017 VALUES [('2017-01-01'), ('2018-01-01')),
    PARTITION p_2018 VALUES [('2018-01-01'), ('2019-01-01')),
    PARTITION p_2019 VALUES [('2019-01-01'), ('2020-01-01')),
    PARTITION p_2020 VALUES [('2020-01-01'), ('2021-01-01')),
    PARTITION p_2021 VALUES [('2021-01-01'), ('2022-01-01'))
```

```
)
DISTRIBUTED BY HASH(`TRADE_DATE`) BUCKETS 10
PROPERTIES (
  "replication_num" = "1"
);
```

The table stores a large amount of business history data, partitioned based on the date the transaction occurred. As you can see when building the table, we need to manually create the partitions in advance. If the data range of the partitioned columns changes, for example, 2022 is added to the above table, we need to create a partition by ALTER-TABLE-PARTITION to make changes to the table partition. If such partitions need to be changed, or subdivided at a finer level of granularity, it is very tedious to modify them. At this point we can rewrite the table DDL using Auto Partition.

Syntax

When creating a table, use the following syntax to populate the `partitions_definition` section in the CREATE-TABLE statement.

1. AUTO RANGE PARTITION:

   ```
   AUTO PARTITION BY RANGE(<partition_expr>)
   <origin_partitions_definition>
   ```

   where

   ```
   partition_expr ::= date_trunc ( <partition_column>, '<interval>' )
   ```

2. AUTO LIST PARTITION:

   ```
   AUTO PARTITION BY LIST(`partition_col1` [, `partition_col2`, ...])
   <origin_partitions_definition>
   ```

Sample

1. AUTO RANGE PARTITION

   ```
   CREATE TABLE `date_table` (
       `TIME_STAMP` datev2 NOT NULL
   ) ENGINE=OLAP
   DUPLICATE KEY(`TIME_STAMP`)
   AUTO PARTITION BY RANGE (date_trunc(`TIME_STAMP`, 'month'))
   (
   )
   DISTRIBUTED BY HASH(`TIME_STAMP`) BUCKETS 10
   PROPERTIES (
   "replication_allocation" = "tag.location.default: 1"
   );
   ```

2. AUTO LIST PARTITION

```
CREATE TABLE `str_table` (
    `str` varchar not null
) ENGINE=OLAP
DUPLICATE KEY(`str`)
AUTO PARTITION BY LIST (`str`)
()
DISTRIBUTED BY HASH(`str`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

List Auto Partition supports multiple partition columns, which are written in the same way as normal List Partition: `AUTO PARTITION BY LIST (`col1`, `col2`, ...)`

Constraints

- In auto List Partition, the partition name length **must not exceed 50 characters**. This length is derived from the concatenation and escape of contents of partition columns on corresponding data rows, so the actual allowed length may be shorter.
- In auto Range Partition, the partition function only supports `date_trunc`, and the partition column supports only `DATE` or `DATETIME` types.
- In auto List Partition, function calls are not supported, and the partition column supports `BOOLEAN`, `TINYINT`, `SMALLINT`, `INT`, `BIGINT`, `LARGEINT`, `DATE`, `DATETIME`, `CHAR`, `VARCHAR` data types, with partition values being enumeration values.
- In auto List Partition, for every existing value in the partition column that does not correspond to a partition, a new independent partition will be created.

NULL value partition

When the session variable `allow_partition_column_nullable` is enabled:

- For Auto List Partition, the corresponding NULL value partition will be created automatically:

```
create table auto_null_list(
  k0 varchar null
)
auto partition by list (k0)
(
)
DISTRIBUTED BY HASH(`k0`) BUCKETS 1
properties("replication_num" = "1");

insert into auto_null_list values (null);

select * from auto_null_list;
+------+
| k0   |
+------+
| NULL |
+------+

select * from auto_null_list partition(pX);
+------+
| k0   |
+------+
```

```
| NULL |
+------+
```

- For Auto Range Partition, **null columns are not supported to be partition columns**.

```
CREATE TABLE `range_table_nullable` (
  `k1` INT,
  `k2` DATETIMEV2(3),
  `k3` DATETIMEV2(6)
) ENGINE=OLAP
DUPLICATE KEY(`k1`)
AUTO PARTITION BY RANGE (date_trunc(`k2`, 'day'))
()
DISTRIBUTED BY HASH(`k1`) BUCKETS 16
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);

ERROR 1105 (HY000): errCode = 2, detailMessage = AUTO RANGE PARTITION doesn't support NULL column
```

Example

When using Auto Partition, the example in the Application scenarios section can be rewritten as:

```
CREATE TABLE `DAILY_TRADE_VALUE`
(
    `TRADE_DATE`              datev2 NOT NULL COMMENT '交易日期',
    `TRADE_ID`                varchar(40) NOT NULL COMMENT '交易编号',
    ......
)
UNIQUE KEY(`TRADE_DATE`, `TRADE_ID`)
AUTO PARTITION BY RANGE (date_trunc(`TRADE_DATE`, 'year'))
(
)
DISTRIBUTED BY HASH(`TRADE_DATE`) BUCKETS 10
PROPERTIES (
  "replication_num" = "1"
);
```

Take the example of a table with only two columns, at which point the new table has no default partitions:

```
show partitions from `DAILY_TRADE_VALUE`;
Empty set (0.12 sec)
```

After inserting data and checking again, it is found that the table has created the corresponding partitions:

```
insert into `DAILY_TRADE_VALUE` values ('2012-12-13', 1), ('2008-02-03', 2), ('2014-11-11', 3);

show partitions from `DAILY_TRADE_VALUE`;
+-------------+----------------+----------------+---------------------+--------+-------------+-----------------------------------
-------------------------------------------+-----------------+---------+----------------+---------------+---------------------+----
-----------------+-------------------------+----------+------------+-------------------------+-----------+
| PartitionId | PartitionName   | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize | IsInMemory | ReplicaAllocation       | IsMutable |
+-------------+----------------+----------------+---------------------+--------+-------------+-----------------------------------
-------------------------------------------+-----------------+---------+----------------+---------------+---------------------+----
```

```
-----------------+-------------------------+----------+------------+-----------------------+-----------+
| 180060      | p20080101000000 | 2             | 2023-09-18 21:49:29 | NORMAL | TRADE_DATE   | [types: [DATEV2]; keys: [2008-01-
01]; ..types: [DATEV2]; keys: [2009-01-01]; ) | TRADE_DATE      | 10      | 1             | HDD           | 9999-12-31 23:59:59 |
| NULL                    | 0.000    | false      | tag.location.default: 1 | true      |
| 180039      | p20120101000000 | 2             | 2023-09-18 21:49:29 | NORMAL | TRADE_DATE   | [types: [DATEV2]; keys: [2012-01-
01]; ..types: [DATEV2]; keys: [2013-01-01]; ) | TRADE_DATE      | 10      | 1             | HDD           | 9999-12-31 23:59:59 |
| NULL                    | 0.000    | false      | tag.location.default: 1 | true      |
| 180018      | p20140101000000 | 2             | 2023-09-18 21:49:29 | NORMAL | TRADE_DATE   | [types: [DATEV2]; keys: [2014-01-
01]; ..types: [DATEV2]; keys: [2015-01-01]; ) | TRADE_DATE      | 10      | 1             | HDD           | 9999-12-31 23:59:59 |
| NULL                    | 0.000    | false      | tag.location.default: 1 | true      |
+-------------+-----------------+---------------+--------------------+--------+-------------+----------------------------------
-------------------------------------------+-----------------+---------+---------------+--------------+--------------------+----
-----------------+-------------------------+----------+------------+-----------------------+-----------+
```

It can be concluded that the partitions created by Auto Partition share the same functionality as partitions created by manual partitioning.

Conjunct with Dynamic Partition

Doris supports both Auto and Dynamic Partition. In this case, both functions are in effect:

1. Auto Partition will automatically create partitions on demand during data import;
2. Dynamic Partition will automatically create, recycle and dump partitions.

There is no conflict between the two syntaxes, just set the corresponding clauses/attributes at the same time. Please note that it is uncertain whether the partition in current period is created by Auto Partition or Dynamic Partition. Different creation methods will lead to different naming formats for the partitions.

> **This feature has been supported since Doris 2.1.7**

Best Practice

In scenarios where you need to set a limit on the partition lifecycle, you can **disable the creation of Dynamic Partition, leaving the creation of partitions to be completed by Auto Partition**, and complete the management of the partition lifecycle through the Dynamic Partition's function of dynamically reclaiming partitions:

```
create table auto_dynamic(
    k0 datetime(6) NOT NULL
)
auto partition by range (date_trunc(k0, 'year'))
(
)
DISTRIBUTED BY HASH(`k0`) BUCKETS 2
properties(
    "dynamic_partition.enable" = "true",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.start" = "-50",
    "dynamic_partition.end" = "0", --- Dynamic Partition No Partition Creation
    "dynamic_partition.time_unit" = "year",
    "replication_num" = "1"
);
```

This way we have both the flexibility of Auto Partition and consistency in partition names.

> **In some early versions prior to 2.1.7, this feature was not disabled but not recommended.**

Partition Management

> **Since 2.1.6, Doris supports the `partitions` table function and the `auto_partition_name` function, which can be used to easily find and manage partitions for data.**

When Auto Partition is enabled, partition names can be mapped to partitions using the `auto_partition_name` function.The `partitions` table function generates detailed partition information from partition names. Let's take the `DAILY_TRADE_VALUE` table as an example to see its current partition after we insert data:

```
select * from partitions("catalog"="internal","database"="optest","table"="DAILY_TRADE_VALUE") where PartitionName =
auto_partition_name('range', 'year', '2008-02-03');
+-------------+-----------------+----------------+---------------------+--------+--------------+-----------------------------------
-------------------------------------------+-----------------+---------+---------------+---------------+---------------------+----
----------------+--------------------------+-----------+------------+-----------------------+-----------+--------------------+---
-----------+
| PartitionId | PartitionName   | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize  | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+-----------------+----------------+---------------------+--------+--------------+-----------------------------------
-------------------------------------------+-----------------+---------+---------------+---------------+---------------------+----
----------------+--------------------------+-----------+------------+-----------------------+-----------+--------------------+---
-----------+
|      127095 | p20080101000000 |              2 | 2024-11-14 17:29:02 | NORMAL | TRADE_DATE   | [types: [DATEV2]; keys: [2008-01-
01]; ..types: [DATEV2]; keys: [2009-01-01]; ) | TRADE_DATE      |      10 |              1 | HDD           | 9999-12-31 23:59:59 |
| \N                      | 985.000 B |          0 | tag.location.default: 1 |         1 |                  1 | \N           |
+-------------+-----------------+----------------+---------------------+--------+--------------+-----------------------------------
-------------------------------------------+-----------------+---------+---------------+---------------+---------------------+----
----------------+--------------------------+-----------+------------+-----------------------+-----------+--------------------+---
-----------+
```

In this way the IDs and values of each partition can be precisely filtered for subsequent partition-specific operations (e.g. `insert overwrite partition`).

For a detailed grammar description, see: auto_partition_name，partitions。

Key points

- Similar to regular partitioned tables, aoto List Partition supports multi-column partitioning with no syntax differences.
- If partitions are created during data insertion or import processes, and the entire import process is not completed (fails or is canceled), the created partitions will not be automatically deleted.
- Tables using Auto Partition only differ in the method of partition creation, switching from manual to automatic. The original usage of the table and its created partitions remains the same as non-Auto Partition tables or partitions.

- To prevent the accidental creation of too many partitions, Apache Doris controls the maximum number of partitions an Auto Partition table can accommodate through the `max_auto_partition_num setting` in the FE configuration. This value can be adjusted if needed.
- When importing data into a table with Auto Partition enabled, the coordinator sends data with a polling interval different from regular tables. Refer to `olap_table_sink_send_interval_auto_partition_factor` in BE Configuration for details. This setting does not have an impact after `enable_memtable_on_sink_node` is enabled.
- When use insert-overwrite to load data into Auto Partition table, the behaviour is detailed in the INSERT OVERWRITE documentation.
- If metadata operations are involved when importing and creating partitions, the import process may fail.

Data Bucketing

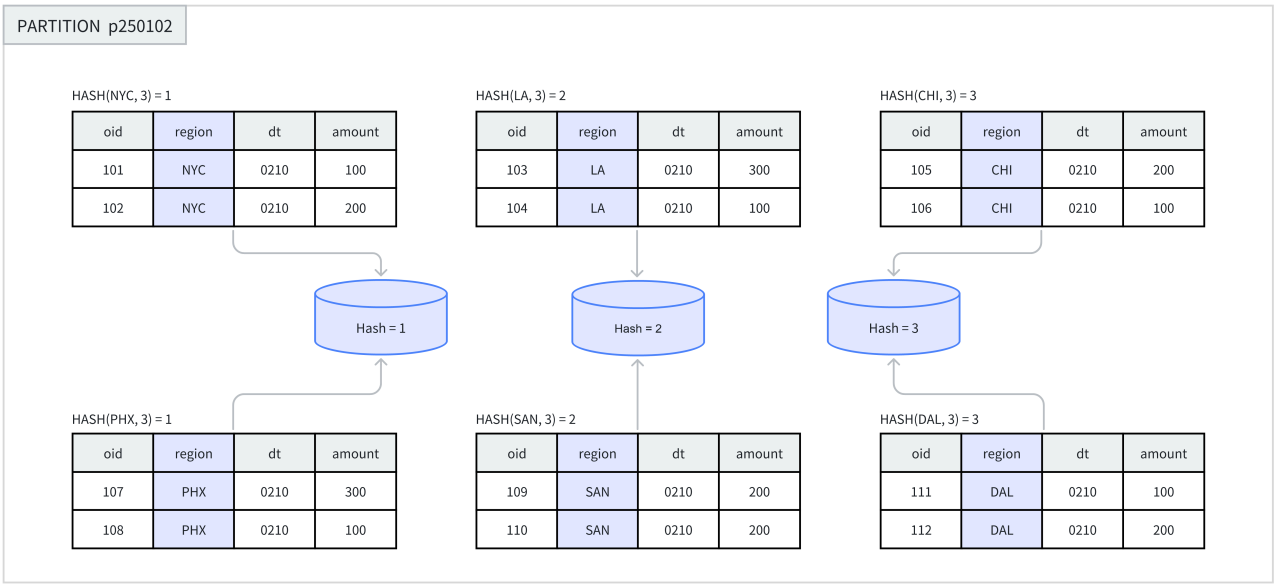
A partition can further be divided into different data buckets based on business logic. Each bucket will be stored as a physical data tablet. A reasonable bucket strategy can effectively reduce the amount of data scanned during queries, thereby improving query performance and increasing query concurrency.

Bucket Methods

Doris supports two bucket methods: Hash Bucketing and Random Bucketing.

Hash Bucketing

When creating a table or adding a partition, users need to select one or more columns as the bucket columns and specify the number of buckets. Within the same partition, the system performs a hash calculation based on the bucket key and the number of buckets. Data with the same hash value will be allocated to the same bucket. For example, in the figure below, partition p250102 is divided into three buckets based on the region column, and rows with the same hash value are placed into the same bucket.



hash-bucket

It is recommended to use Hash Bucketing in the following scenarios:

- When the business needs to frequently filter based on a certain field, you can use this field as the bucket key for Hash Bucketing to improve query efficiency.
- When the data distribution in the table is relatively uniform, Hash Bucketing is also a suitable choice.

The following example shows how to create a table with Hash Bucketing. For detailed syntax, please refer to the CREATE TABLE statement.

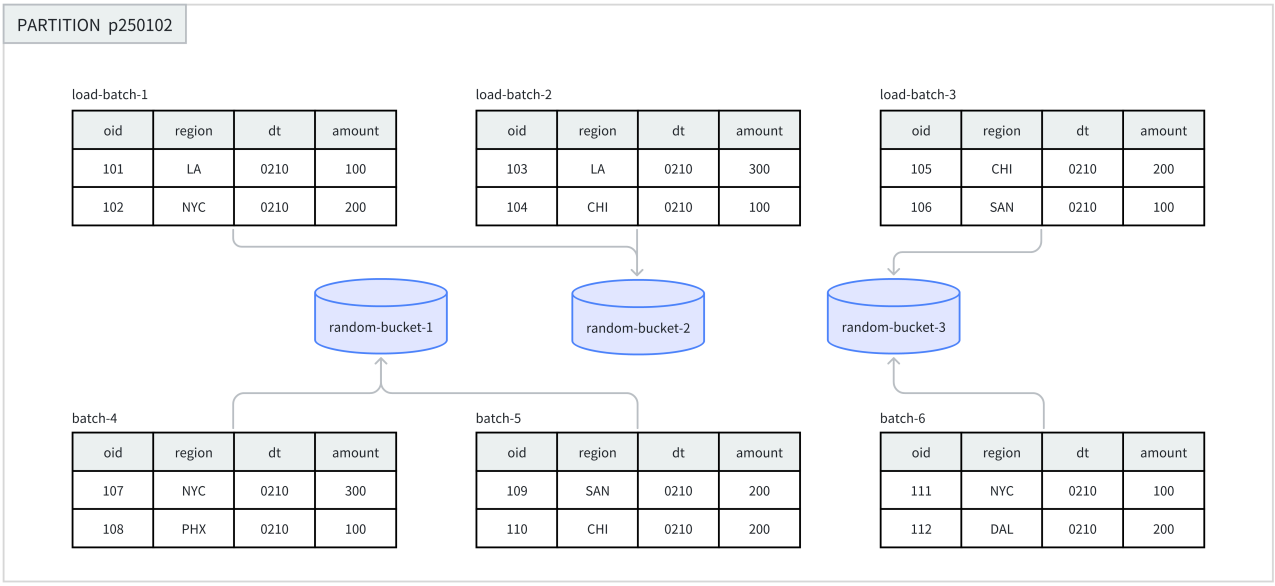
```
CREATE TABLE demo.hash_bucket_tbl(
    oid         BIGINT,
    dt          DATE,
    region      VARCHAR(10),
    amount      INT
)
DUPLICATE KEY(oid)
PARTITION BY RANGE(dt) (
    PARTITION p250101 VALUES LESS THAN("2025-01-01"),
    PARTITION p250102 VALUES LESS THAN("2025-01-02")
)
DISTRIBUTED BY HASH(region) BUCKETS 8;
```

In the example, `DISTRIBUTED BY HASH(region)` specifies the creation of Hash Bucketing and selects the `region` column as the bucket key. Meanwhile, `BUCKETS 8` specifies the creation of 8 buckets.

Random Bucketing

In each partition, Random Bucketing randomly distributes data to various buckets without relying on the hash value of a certain field. Random Bucketing ensures uniform data distribution, thus avoiding data skew caused by improper bucket key selection.

During data import, each batch of a single import job will be randomly written to a tablet, ensuring uniform data distribution. For example, in one operation, eight batches of data are randomly allocated to three buckets under partition `p250102`.



random-bucket

When using Random Bucketing, you can enable the single-tablet import mode (set `load_to_single_tablet` to `true`). During large-scale data imports, one batch of data will only be written to one data tablet, helping to improve the concurrency and throughput of data imports, reducing write amplification caused by data imports and Compaction, thereby ensuring cluster stability.

It is recommended to use Random Bucketing in the following scenarios:

- In scenarios of arbitrary dimension analysis, where the business does not frequently filter or join queries based on a specific column, you can choose Random Bucketing.
- When the data distribution of frequently queried columns or combinations of columns is extremely uneven, using Random Bucketing can avoid data skew.
- Random Bucketing cannot be pruned based on bucket keys and will scan all data in the hit partition, so it is not recommended for point query scenarios.
- Only DUPLICATE tables can use Random partitioning. UNIQUE and AGGREGATE tables cannot use Random Bucketing.

The following example shows how to create a table with Random Bucketing. For detailed syntax, please refer to the CREATE TABLE statement:

```
CREATE TABLE demo.random_bucket_tbl(
    oid         BIGINT,
    dt          DATE,
    region      VARCHAR(10),
    amount      INT
)
DUPLICATE KEY(oid)
PARTITION BY RANGE(dt) (
    PARTITION p250101 VALUES LESS THAN("2025-01-01"),
    PARTITION p250102 VALUES LESS THAN("2025-01-02")
)
DISTRIBUTED BY RANDOM BUCKETS 8;
```

In the example, the statement `DISTRIBUTED BY RANDOM` specifies the use of Random Bucketing. Creating Random Bucketing does not require selecting a bucket key, and the statement `BUCKETS 8` specifies the creation of 8 buckets.

Choosing Bucket Keys

> **Note**
>
> Only Hash Bucketing requires selecting bucket keys; Random Bucketing does not require selecting bucket keys.

Bucket keys can be one or more columns. If it is a DUPLICATE table, any Key column or Value column can be used as the bucket key. If it is an AGGREGATE or UNIQUE table, to ensure gradual aggregation, the bucket column must be a Key column.

Generally, you can choose bucket keys based on the following rules:

- **Using Query Filter Conditions:** Using query filter conditions for Hash Bucketing helps data pruning and reduces data scan volume;
- **Using High Cardinality Columns:** Selecting high cardinality (many unique values) columns for Hash Bucketing helps to evenly distribute data across each bucket;

- **High-Concurrency Point Query Scenarios:** It is recommended to select a single column or fewer columns for bucketing. Point queries may only trigger a scan of one bucket, and the probability of different queries triggering scans of different buckets is high, thereby reducing the IO impact between queries.
- **High-Throughput Query Scenarios:** It is recommended to select multiple columns for bucketing to make data more evenly distributed. If the query conditions cannot include the equality conditions of all bucket keys, it will increase query throughput and reduce the latency of single queries.

Choosing the Number of Buckets

In Doris, a bucket is stored as a physical file (tablet). The number of tablets in a table equals `partition_num` (number of partitions) multiplied by `bucket_num` (number of buckets). Once the number of Partitions is specified, it cannot be changed.

When determining the number of buckets, you need to consider machine expansion in advance. Starting from version 2.0, Doris supports automatically setting the number of buckets in partitions based on machine resources and cluster information.

Manually Setting the Number of Buckets

You can specify the number of buckets using the `DISTRIBUTED` statement:

```
-- Set hash bucket num to 8
DISTRIBUTED BY HASH(region) BUCKETS 8

-- Set random bucket num to 8
DISTRIBUTED BY RANDOM BUCKETS 8
```

When determining the number of buckets, two principles are usually followed: quantity and size. If there is a conflict between the two, the size principle is prioritized:

- **Size Principle:** Keep each tablet between **1 GB and 20 GB** compressed data (excluding index), or under **10 GB** for Unique Key tables. Too small a tablet may result in poor aggregation effect and increase metadata management pressure; too large a tablet is not conducive to replica migration and supplementation and will increase the cost of retrying Schema Change operations. You can check actual tablet sizes with `SHOW TABLETS FROM your_table`.
- **Quantity Principle:** Without considering expansion, it is recommended that the number of tablets for a table be slightly more than the number of disks in the entire cluster.

The bucket count should be a multiple of the number of BEs for even data distribution. Generally, the bucket count per partition should not exceed **128** — if you need more, consider partitioning the table first.

For example, assuming there are 10 BE machines with one disk per BE, you can follow the recommendations below for data bucketing:

Partition Size	Recommended Number of Buckets
< 1 GB	1 bucket

Partition Size	Recommended Number of Buckets
1-10 GB	10 buckets
10-200 GB	10-20 buckets
> 200 GB	Consider partitioning first

Data sizes refer to compressed data size in Doris. You can check actual sizes with `SHOW TABLETS FROM your_table`.

> **Note**
>
> The data volume of the table can be viewed using the `SHOW DATA` command. The result needs to be divided by the number of replicas and the data volume of the table.

Automatic Bucket Number Setting

The automatic bucket number calculation function will automatically predict the future partition size based on the partition size over a period of time, and determine the number of buckets accordingly.

```
-- Set hash bucket auto
DISTRIBUTED BY HASH(region) BUCKETS AUTO
properties("estimate_partition_size" = "20G")

-- Set random bucket auto
DISTRIBUTED BY HASH(region) BUCKETS AUTO
properties("estimate_partition_size" = "20G")
```

When creating buckets, you can adjust the estimated partition size through the `estimate_partition_size` attribute. This parameter is optional, and if not provided, Doris will default to 10GB. Please note that this parameter is not related to the future partition size calculated by the system based on historical partition data.

Maintaining Data Buckets

> **Note**
>
> Currently, Doris only supports modifying the number of buckets in newly added partitions and does not support the following operations:
>
> 1. Modifying the bucketing type
> 2. Modifying the bucket key
> 3. Modifying the number of buckets for existing buckets

When creating a table, the number of buckets for each partition is uniformly specified through the `DISTRIBUTED` statement. To cope with data growth or reduction, you can specify the number of buckets for the new partition when dynamically adding partitions. The following example shows how to modify the number of buckets in newly added partitions using the `ALTER TABLE` command:

```
-- Modify hash bucket table
ALTER TABLE demo.hash_bucket_tbl
ADD PARTITION p250103 VALUES LESS THAN("2025-01-03")
DISTRIBUTED BY HASH(region) BUCKETS 16;

-- Modify random bucket table
ALTER TABLE demo.random_bucket_tbl
ADD PARTITION p250103 VALUES LESS THAN("2025-01-03")
DISTRIBUTED BY RANDOM BUCKETS 16;

-- Modify dynamic partition table
ALTER TABLE demo.dynamic_partition_tbl
SET ("dynamic_partition.buckets"="16");
```

After modifying the number of buckets, you can use the SHOW PARTITION command to check the updated number of buckets.

Common Issues

1. Incomplete syntax error prompts may occur in longer table creation statements. Here are some possible syntax errors for manual troubleshooting:

    - Syntax structure errors. Please carefully read HELP CREATE TABLE and check the relevant syntax structure.
    - Reserved words. When user-defined names encounter reserved words, they need to be enclosed in backticks ``. It is recommended to use this symbol for all custom names.
    - Chinese characters or full-width characters. Non-UTF8 encoded Chinese characters or hidden full-width characters (spaces, punctuation, etc.) can cause syntax errors. It is recommended to use a text editor that displays invisible characters for inspection.

2. Failed to create partition [xxx]. Timeout

    Doris creates tables sequentially based on partition granularity. When a partition fails to create, this error may occur. Even if partitions are not used, when there is a problem with table creation, `Failed to create partition` may still be reported because, as mentioned earlier, Doris creates an unmodifiable default partition for tables without specified partitions.

    When encountering this error, it is usually because the BE encountered a problem when creating data tablets. You can troubleshoot by following these steps:

    - In the fe.log, search for the `Failed to create partition` log entry at the corresponding timestamp. In this log entry, you may find a series of number pairs similar to `{10001-10010}`. The first number in the pair represents the Backend ID, and the second number represents the Tablet ID. For example, this number pair indicates that the creation of Tablet ID 10010 on Backend ID 10001 failed.
    - Go to the be.INFO log of the corresponding Backend and search for Tablet ID-related logs within the corresponding time period to find error messages.
    - Here are some common tablet creation failure errors, including but not limited to:
        - The BE did not receive the relevant task. In this case, you cannot find Tablet ID-related logs in be.INFO or the BE reports success but actually fails. For these issues, please refer to the Installation and Deployment section to check the connectivity between FE and BE.
        - Pre-allocated memory failure. This may be because the byte length of a row in the table exceeds 100KB.
        - `Too many open files`. The number of open file handles exceeds the Linux system limit. You need to modify the handle limit of the Linux system.

- If there is a timeout when creating data tablets, you can also extend the timeout by setting `tablet_create_timeout_second=xxx` and `max_create_table_timeout_second=xxx` in the fe.conf file. By default, `tablet_create_timeout_second` is set to 1 second, and `max_create_table_timeout_second` is set to 60 seconds. The overall timeout is calculated as `min(tablet_create_timeout_second * replication_num, max_create_table_timeout_second)`. For specific parameter settings, please refer to the FE Configuration section.

3. The table creation command does not return results for a long time.

- Doris's table creation command is a synchronous command. The timeout for this command is currently set simply as (tablet num * replication num) seconds. If many data tablets are created and some of them fail to create, it may result in a long wait before returning an error.
- Under normal circumstances, the table creation statement should return within a few seconds or tens of seconds. If it exceeds one minute, it is recommended to cancel the operation directly and check the relevant errors in the FE or BE logs.

More Help

For more detailed information on data partitioning, you can refer to the CREATE TABLE command manual or enter `HELP CREATE TABLE;` in the MySQL client to get more help information.

Basic concepts

This document mainly introduces table creation and data partitioning in Doris, as well as potential problems and solutions encountered during table creation operations.

Row & Column

In Doris, data is logically described in the form of tables.

A table consists of rows and columns:

- Row: Represents a single line of user data;
- Column: Used to describe different fields in a row of data;
- Columns can be divided into two types: Key and Value. From a business perspective, Key and Value can correspond to dimension columns and metric columns respectively. The key columns in Apache Doris are those specified in the table creation statement, which are the columns following the keywords `unique key`, `aggregate key`, or `duplicate key`. The remaining columns are value columns. From the perspective of the aggregation model, rows with the same Key columns will be aggregated into a single row. The aggregation method for value columns is specified by the user during table creation. For more information on aggregation models, refer to the Doris Data Model.

Partition & Bucket

Doris uses a two-level partitioning and bucketing method to organize and manage data.

Partition

Partition refers to dividing the table into smaller, more manageable, non-overlapping subsets based on specific column values in the table. Each subset of data is called a partition. Each row of data belongs to exactly one specific partition. Partitions can be seen as the smallest logical management unit.

Currently, Doris supports two types of partitioning: Range and List. If no partition is specified during table creation, Doris will generate a default partition containing all the data in the table, which is transparent to the user.

Partitioning based on data distribution and query patterns offers several benefits:

- **Improved Query Performance**: Partitioning allows the system to prune irrelevant partitions based on the query conditions, reducing the amount of data scanned and significantly improving query efficiency. This is especially beneficial when handling large datasets, as the partition strategy can greatly reduce I/O overhead.
- **Flexible Management**: Partitioning allows data to be split based on logic such as time or geography, facilitating data archiving, cleaning, and backup. For example, partitioning by time can effectively manage historical and newly added data, supporting efficient time-based data maintenance strategies.

Bucket

Bucketing refers to further dividing the data within a partition into smaller, non-overlapping units according to some rule. Each row of data belongs to exactly one specific bucket. Unlike partitioning, which divides data based on specific column values, bucketing attempts to evenly distribute the data across predefined buckets, thereby reducing data skew. Bucketing improves query performance by ensuring even data distribution and enhancing data locality.

Currently, Doris supports two types of bucketing: Hash and Random.

A bucket corresponds to a data shard (Tablet) at the physical level, and data shards are physically stored independently. They are the smallest physical storage units for operations like data movement and replication.

Proper bucketing offers several advantages:

- **Even Data Distribution**: Bucketing evenly distributes data across buckets, reducing the risk of data concentration or skew, and preventing resource overload on specific nodes or storage devices.
- **Reduced Hotspots**: By distributing data evenly, bucketing helps to reduce the risk of overloading specific nodes or partitions, preventing hotspots, and improving system stability and processing capability.
- **Improved Concurrency Performance**: Bucketing enhances the performance of concurrent queries, especially when multiple query requests need to access different data within the same partition. The granularity of bucketing allows the system to efficiently process multiple requests in parallel, thereby improving throughput.

Example of creating a table

`CREATE TABLE` in Apache Doris is a synchronous command which returns the result once the SQL is executed. Successful returns indicate successful table creation. For more information, refer to CREATE-TABLE or input the `HELP CREATE TABLE` command.

The following code sample introduces how to create tables in Apache Doris by RANGE partitioning and Hash buckets.

```
-- Range Partition
CREATE TABLE IF NOT EXISTS example_range_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "User ID",
    `date` DATE NOT NULL COMMENT "Date when the data are imported",
    `timestamp` DATETIME NOT NULL COMMENT "Timestamp when the data are imported",
    `city` VARCHAR(20) COMMENT "User location city",
    `age` SMALLINT COMMENT "User age",
    `sex` TINYINT COMMENT "User gender",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "User last visit time",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "Total user consumption",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum user dwell time",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum user dwell time"
)
ENGINE=OLAP
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY RANGE(`date`)
(
    PARTITION `p201701` VALUES [("2017-01-01"),  ("2017-02-01")),
```

```
    PARTITION `p201702` VALUES [("2017-02-01"), ("2017-03-01")),
    PARTITION `p201703` VALUES [("2017-03-01"), ("2017-04-01"))
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
    "replication_num" = "1"
);
```

Here use Aggregate Key Model as an example. In Aggregate Key Model, all columns that are specified with an aggregation type (SUM, REPLACE, MAX, or MIN) are Value columns. The rest are the Key columns.

For more information about what fields can be set in the `PROPERTIES` section of `CREATE TABLE`, refer to CREATE-TABLE.

The default type of `ENGINE` is `OLAP`. Only OLAP is responsible for data management and storage by Apache Doris itself. Other engine types, such as MySQL, Broker and ES, are essentially just mappings to tables in other external databases or systems, allowing Apache Doris to read this data. However, Apache Doris itself does not create, manage, or store any tables or data for engine types except OLAP.

`IF NOT EXISTS` indicates that if the table has not been created before, it will be created. Note that this only checks if the table name exists and does not check if the schema of the new table is the same as the schema of an existing table. Therefore, if there is a table with the same name but a different schema, this command will also return successfully, but it does not mean that a new table with a new schema has been created.

Advanced Features and Examples

Doris supports advanced data partitioning methods, including Dynamic Partition, Auto Partition, and Auto Bucket, which enable more flexible data management. The following are examples of implementations: more convenient. Rewrite the above example with Auto Range Partition as follows:

Auto Partition

Auto Partition supports automatic creation of corresponding partitions according to user-defined rules during data import.

```
CREATE TABLE IF NOT EXISTS example_range_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "User ID",
    `date` DATE NOT NULL COMMENT "Date when the data are imported",
    `timestamp` DATETIME NOT NULL COMMENT "Timestamp when the data are imported",
    `city` VARCHAR(20) COMMENT "User location city",
    `age` SMALLINT COMMENT "User age",
    `sex` TINYINT COMMENT "User gender",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "User last visit time",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "Total user consumption",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum user dwell time",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum user dwell time"
)
ENGINE=OLAP
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
AUTO PARTITION BY RANGE(date_trunc(`date`, 'month')) --- Using months as partition granularity
()
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
```

```
(
    "replication_num" = "1"
);
```

As above, when the data is imported, Doris will automatically create the corresponding partitions as `date` with a granularity of month level. `2018-12-01` and `2018-12-31` will fall into the same partition, while `2018-11-12` will fall into the leading partition. Auto Partition also supports List partition, please check Auto Partition's documentation for more usage.

Dynamic Partition

Dynamic Partition is an automatic partition creation and recovery management method based on the real time, rewrite the above example with dynamic partitioning as follows:

```
CREATE TABLE IF NOT EXISTS example_range_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "User ID",
    `date` DATE NOT NULL COMMENT "Date when the data are imported",
    `timestamp` DATETIME NOT NULL COMMENT "Timestamp when the data are imported",
    `city` VARCHAR(20) COMMENT "User location city",
    `age` SMALLINT COMMENT "User age",
    `sex` TINYINT COMMENT "User gender",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "User last visit time",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "Total user consumption",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum user dwell time",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum user dwell time"
)
ENGINE=OLAP
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY RANGE(`date`)
()
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
    "replication_num" = "1",
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "WEEK", --- Partition granularity is week
    "dynamic_partition.start" = "-2", --- Retain two weeks forward
    "dynamic_partition.end" = "2", --- Two weeks after creation in advance
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "8"
);
```

Dynamic partition supports tiered storage, customised copy number and other features, see the dynamic partition documentation for details.

Auto&Dynamic Partition

> **Supported since Doris 2.1.7**

Auto Partition and Dynamic Partition each have their own advantages, and combining the two enables flexible on-demand partition creation and automatic reclamation:

```
CREATE TABLE IF NOT EXISTS example_range_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "User ID",
    `date` DATE NOT NULL COMMENT "Date when the data are imported",
    `timestamp` DATETIME NOT NULL COMMENT "Timestamp when the data are imported",
    `city` VARCHAR(20) COMMENT "User location city",
    `age` SMALLINT COMMENT "User age",
    `sex` TINYINT COMMENT "User gender",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "User last visit time",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "Total user consumption",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum user dwell time",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum user dwell time"
)
ENGINE=OLAP
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
AUTO PARTITION BY RANGE(date_trunc(`date`, 'month')) --- Using months as partition granularity
()
DISTRIBUTED BY HASH(`user_id`) BUCKETS 16
PROPERTIES
(
    "replication_num" = "1",
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "month", --- Both must have the same granularity
    "dynamic_partition.start" = "-2", --- Dynamic Partition automatically cleans up partitions that are more than two weeks old
    "dynamic_partition.end" = "0", --- Dynamic Partition does not create future partitions. it is left entirely to Auto Partition.
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "8"
);
```

For detailed suggestions on this feature, see Auto Partition Conjunct with Dynamic Partition.

Auto Bucket

When the user is not sure of a reasonable number of buckets, Auto Bucket for Doris to complete the estimation, and the user only needs to provide the estimated amount of table data:

```
CREATE TABLE IF NOT EXISTS example_range_tbl
(
    `user_id` LARGEINT NOT NULL COMMENT "User ID",
    `date` DATE NOT NULL COMMENT "Date when the data are imported",
    `timestamp` DATETIME NOT NULL COMMENT "Timestamp when the data are imported",
    `city` VARCHAR(20) COMMENT "User location city",
    `age` SMALLINT COMMENT "User age",
    `sex` TINYINT COMMENT "User gender",
    `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00" COMMENT "User last visit time",
    `cost` BIGINT SUM DEFAULT "0" COMMENT "Total user consumption",
    `max_dwell_time` INT MAX DEFAULT "0" COMMENT "Maximum user dwell time",
    `min_dwell_time` INT MIN DEFAULT "99999" COMMENT "Minimum user dwell time"
)
ENGINE=OLAP
AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)
PARTITION BY RANGE(`date`)
(
    PARTITION `p201701` VALUES LESS THAN ("2017-02-01"),
    PARTITION `p201702` VALUES LESS THAN ("2017-03-01"),
    PARTITION `p201703` VALUES LESS THAN ("2017-04-01"),
    PARTITION `p2018` VALUES [("2018-01-01"), ("2019-01-01"))
)
DISTRIBUTED BY HASH(`user_id`) BUCKETS AUTO
PROPERTIES
(
```

```
    "replication_num" = "1",
    "estimate_partition_size" = "2G" --- User estimate of the amount of data a partition will have, defaults to 10G if not provided
);
```

Note that this approach does not apply to cases where the amount of table data is particularly large.

View partitions

View the partiton information of a table by running the `show create table` command.

```
> show create table  example_range_tbl
+-------------------+-----------------------------------------------------------------------------------------------------------+
| Table             | Create Table                                                                                              |
+-------------------+-----------------------------------------------------------------------------------------------------------+
| example_range_tbl | CREATE TABLE `example_range_tbl` (                                                                        |
|                   |   `user_id` largeint(40) NOT NULL COMMENT 'User ID',                                                       |
|                   |   `date` date NOT NULL COMMENT 'Date when the data are imported',
|
|                   |   `timestamp` datetime NOT NULL COMMENT 'Timestamp when the data are imported',
|
|                   |   `city` varchar(20) NULL COMMENT 'User location city',
|
|                   |   `age` smallint(6) NULL COMMENT 'User age',                                                              |
|                   |   `sex` tinyint(4) NULL COMMENT 'User gender',                                                              |
|                   |   `last_visit_date` datetime REPLACE NULL DEFAULT "1970-01-01 00:00:00" COMMENT 'User last visit time', |
|                   |   `cost` bigint(20) SUM NULL DEFAULT "0" COMMENT 'Total user consumption',
|
|                   |   `max_dwell_time` int(11) MAX NULL DEFAULT "0" COMMENT 'Maximum user dwell time',
|
|                   |   `min_dwell_time` int(11) MIN NULL DEFAULT "99999" COMMENT 'Minimum user dwell time'
|
|                   | ) ENGINE=OLAP                                                                                             |
|                   | AGGREGATE KEY(`user_id`, `date`, `timestamp`, `city`, `age`, `sex`)                                       |
|                   | COMMENT 'OLAP'                                                                                            |
|                   | PARTITION BY RANGE(`date`)                                                                                |
|                   | (PARTITION p201701 VALUES [('0000-01-01'), ('2017-02-01')),                                               |
|                   | PARTITION p201702 VALUES [('2017-02-01'), ('2017-03-01')),                                                |
|                   | PARTITION p201703 VALUES [('2017-03-01'), ('2017-04-01')))                                                |
|                   | DISTRIBUTED BY HASH(`user_id`) BUCKETS 16                                                                 |
|                   | PROPERTIES (                                                                                              |
|                   | "replication_allocation" = "tag.location.default: 1",                                                     |
|                   | "is_being_synced" = "false",                                                                              |
|                   | "storage_format" = "V2",                                                                                  |
|                   | "light_schema_change" = "true",                                                                           |
|                   | "disable_auto_compaction" = "false",                                                                      |
|                   | "enable_single_replica_compaction" = "false"                                                              |
|                   | );                                                                                                        |
+-------------------+-----------------------------------------------------------------------------------------------------------+
```

Or run the `show partitions from your_table` command.

```
> show partitions from example_range_tbl
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------+-----------------+---------+----------------+---------------
+--------------------+--------------------+------------------------+----------+------------+------------------------+----------
-+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium
| CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize | IsInMemory | ReplicaAllocation       | IsMutable
|
```

```
+-------------+---------------+----------------+---------------------+--------+-------------+-------------------------------------
---------------------------------------+-----------------+---------+---------------+--------------
+---------------------+---------------------+-------------------------+----------+------------+-------------------------+----------
-+
| 28731       | p201701       | 1              | 2024-01-25 10:50:51 | NORMAL | date        | [types: [DATEV2]; keys: [0000-01-01];
..types: [DATEV2]; keys: [2017-02-01]; ) | user_id         | 16      | 1             | HDD
| 9999-12-31 23:59:59 |                     |                   | 0.000    | false      | tag.location.default: 1 | true      |
| 28732       | p201702       | 1              | 2024-01-25 10:50:51 | NORMAL | date        | [types: [DATEV2]; keys: [2017-02-01];
..types: [DATEV2]; keys: [2017-03-01]; ) | user_id         | 16      | 1             | HDD
| 9999-12-31 23:59:59 |                     |                   | 0.000    | false      | tag.location.default: 1 | true      |
| 28733       | p201703       | 1              | 2024-01-25 10:50:51 | NORMAL | date        | [types: [DATEV2]; keys: [2017-03-01];
..types: [DATEV2]; keys: [2017-04-01]; ) | user_id         | 16      | 1             | HDD
| 9999-12-31 23:59:59 |                     |                   | 0.000    | false      | tag.location.default: 1 | true      |
+-------------+---------------+----------------+---------------------+--------+-------------+-------------------------------------
---------------------------------------+-----------------+---------+---------------+--------------
+---------------------+---------------------+-------------------------+----------+------------+-------------------------+----------
-+
```

Alter partitions

You can add a new partition by running the `alter table add partition` command.

```
ALTER TABLE example_range_tbl ADD PARTITION p201704 VALUES LESS THAN("2020-05-01") DISTRIBUTED BY HASH(`user_id`) BUCKETS 5;
```

For more information about how to alter partitions, refer to ALTER-TABLE-PARTITION.

Partition Retrieval

The `partitions` table function(supported since 2.1.5) and the `information_schema.partitions` system table(supported since 2.1.7) record partition information for the cluster. The partition information can be extracted from the corresponding table for use when automatically managing partitions:

```
--- Find the partition with the corresponding value in the Auto Partition table.
mysql> select * from partitions("catalog"="internal", "database"="optest", "table"="DAILY_TRADE_VALUE") where PartitionName =
auto_partition_name('range', 'year', '2008-02-03');
+-------------+----------------+---------------+---------------------+--------+-------------+-------------------------------------
---------------------------------------+-----------------+---------+---------------+--------------+---------------------+----
----------------+-------------------------+----------+------------+-------------------------+----------+---------------------+---
-----------+
| PartitionId | PartitionName   | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize  | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+----------------+---------------+---------------------+--------+-------------+-------------------------------------
---------------------------------------+-----------------+---------+---------------+--------------+---------------------+----
----------------+-------------------------+----------+------------+-------------------------+----------+---------------------+---
-----------+
|      127095 | p20080101000000 |              2 | 2024-11-14 17:29:02 | NORMAL | TRADE_DATE   | [types: [DATEV2]; keys: [2008-01-
01]; ..types: [DATEV2]; keys: [2009-01-01]; ) | TRADE_DATE      |      10 |             1 | HDD           | 9999-12-31 23:59:59 |
| \N                      | 985.000 B |          0 | tag.location.default: 1 |         1 |                 1 | \N          |
+-------------+----------------+---------------+---------------------+--------+-------------+-------------------------------------
---------------------------------------+-----------------+---------+---------------+--------------+---------------------+----
----------------+-------------------------+----------+------------+-------------------------+----------+---------------------+---
-----------+
1 row in set (0.30 sec)

mysql> select * from information_schema.partitions where TABLE_SCHEMA='optest' and TABLE_NAME='list_table1' and
PARTITION_NAME=auto_partition_name('list', null);
+---------------+-------------+-------------+----------------+-------------------+---------------------------+--------------------
```

```
-----------+-----------------+--------------------+----------------------+-------------------------+----------------------+-------
------+---------------+-------------+-----------------+--------------+-----------+-------------+--------------------+-------------
--------+----------+------------------+-----------+-----------------+
| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME  | PARTITION_NAME | SUBPARTITION_NAME | PARTITION_ORDINAL_POSITION |
SUBPARTITION_ORDINAL_POSITION | PARTITION_METHOD | SUBPARTITION_METHOD | PARTITION_EXPRESSION | SUBPARTITION_EXPRESSION |
PARTITION_DESCRIPTION | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH | MAX_DATA_LENGTH | INDEX_LENGTH | DATA_FREE | CREATE_TIME |
UPDATE_TIME         | CHECK_TIME          | CHECKSUM | PARTITION_COMMENT | NODEGROUP | TABLESPACE_NAME |
+---------------+--------------+-------------+----------------+-------------------+---------------------------+--------------------
-----------+-----------------+--------------------+----------------------+-------------------------+----------------------+-------
------+---------------+-------------+-----------------+--------------+-----------+-------------+--------------------+-------------
--------+----------+------------------+-----------+-----------------+
| internal      | optest       | list_table1 | pX             | NULL              |                           0 |
0 | LIST             | NULL               | str                 | NULL                    | (NULL)                |          1 |
1266 |        1266 |              0 |            0 |         0 |           0 | 2024-11-14 19:58:45 | 0000-00-00 00:00:00 |        0
|                   |           |                 |
+---------------+--------------+-------------+----------------+-------------------+---------------------------+--------------------
-----------+-----------------+--------------------+----------------------+-------------------------+----------------------+-------
------+---------------+-------------+-----------------+--------------+-----------+-------------+--------------------+-------------
--------+----------+------------------+-----------+-----------------+
1 row in set (0.24 sec)

--- Find the partition that corresponds to the starting point
mysql> select * from information_schema.partitions where TABLE_NAME='DAILY_TRADE_VALUE' and PARTITION_DESCRIPTION like "[('2012-01-
01'),%";
+---------------+--------------+-------------------+-----------------+-------------------+---------------------------+-------------
-----------------+------------------+--------------------+----------------------+-------------------------+----------------------+-------
-----------+------------+---------------+-------------+-----------------+--------------+-----------+-------------+----------------
----+--------------------+----------+------------------+-----------+-----------------+
| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME        | PARTITION_NAME  | SUBPARTITION_NAME | PARTITION_ORDINAL_POSITION |
SUBPARTITION_ORDINAL_POSITION | PARTITION_METHOD | SUBPARTITION_METHOD | PARTITION_EXPRESSION | SUBPARTITION_EXPRESSION |
PARTITION_DESCRIPTION            | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH | MAX_DATA_LENGTH | INDEX_LENGTH | DATA_FREE |
CREATE_TIME | UPDATE_TIME         | CHECK_TIME          | CHECKSUM | PARTITION_COMMENT | NODEGROUP | TABLESPACE_NAME |
+---------------+--------------+-------------------+-----------------+-------------------+---------------------------+-------------
-----------------+------------------+--------------------+----------------------+-------------------------+----------------------+-------
-----------+------------+---------------+-------------+-----------------+--------------+-----------+-------------+----------------
----+--------------------+----------+------------------+-----------+-----------------+
| internal      | optest       | DAILY_TRADE_VALUE | p20120101000000 | NULL              |                           0 |
0 | RANGE            | NULL               | TRADE_DATE           | NULL                    | [('2012-01-01'), ('2013-01-01')) |
1 |           985 |         985 |              0 |            0 |         0 |           0 | 2024-11-14 17:29:02 | 0000-00-00
00:00:00 |        0 |                   |           |                 |
+---------------+--------------+-------------------+-----------------+-------------------+---------------------------+-------------
-----------------+------------------+--------------------+----------------------+-------------------------+----------------------+-------
-----------+------------+---------------+-------------+-----------------+--------------+-----------+-------------+----------------
----+--------------------+----------+------------------+-----------+-----------------+
1 row in set (0.65 sec)
```

Data Types

Apache Doris support standard SQL syntax, using MySQL Network Connection Protocol, highly compatible with MySQL syntax protocol. Therefore, in terms of data type support, Apache Doris aligns as closely as possible with MySQL-related data types

The list of data types supported by Doris is as follows:

Numeric data type §

Type Name	Storage (bytes)	Description
BOOLEAN	1	Boolean data type that stores only two values: 0 represents false, 1 represents true.
TINYINT	1	Integer value, signed range is from -128 to 127.
SMALLINT	2	Integer value, signed range is from -32768 to 32767.
INT	4	Integer value, signed range is from -2147483648 to 2147483647.
BIGINT	8	Integer value, signed range is from -9223372036854775808 to 9223372036854775807.
LARGEINT	16	Integer value, range is [-2^127 + 1 to 2^127 - 1].
FLOAT	4	Single precision floating point number, range is [-3.4 * 10^38 to 3.4 * 10^38].
DOUBLE	8	Double precision floating point number, range is [-1.79 * 10^308 to 1.79 * 10^308].
DECIMAL	4/8/16/32	An exact fixed-point number defined by precision (total number of digits) and scale (number of digits to the right of the decimal point). Format: DECIMAL(P[,S]), where P is precision and S is scale. The range for P is [1, MAX_P], where MAX_P=38 when `enable_decimal256`=false, and MAX_P=76 when `enable_decimal256`=true, and for S is [0, P]. The default value of `enable_decimal256` is false. Setting it to true can get more accurate results, but it will bring some performance loss. Storage requirements:

Datetime data type §

Type name	Storeage (bytes)	Description
DATE	4	DATE holds values for a calendar year, month and day, the supported range is ['0000-01-01', '9999-12-31']. Default print format: 'yyyy-MM-dd'.

Type name	Storeage (bytes)	Description
DATETIME	8	A DATE and TIME combination Format: DATETIME ([P]). The optional parameter P represents time precision, with a value range of [0,6], supporting up to 6 decimal places (microseconds). When not set, it is 0. The supported range is ['0000-01-01 00:00:00 [.000000]', '9999-12-31 23:59:59 [.999999]']. Default print format: 'yyy-MM-dd HH: mm: ss. SSSSSS'.

String data type §

Type name	Storeage (bytes)	Description
CHAR	M	A FIXED length string, the parameter M specifies the column length in characters. The range of M is from 1 to 255.
VARCHAR	Variable Length	A VARIABLE length string , the parameter M specifies the maximum string length in characters. The range of M is from 1 to 65533. The variable-length string is stored in UTF-8 encoding. English characters occupy 1 byte, and Chinese characters occupy 3 bytes.
STRING	Variable Length	A VARIABLE length string, default supports 1048576 bytes (1 MB), and a limit of maximum precision of 2147483643 bytes (2 GB). Size can be configured string_type_length_soft_limit_bytes adjusted through BE. String type can only be used in value column, not in key column and partition bucket column.

Semi-structured data type §

Type name	Storeage (bytes)	Description
ARRAY	Variable Length	Arrays composed of elements of type T cannot be used as key columns. Currently supported for use in tables with Duplicate and Unique models.
MAP	Variable Length	Maps consisting of elements of type K and V, cannot be used as Key columns. These maps are currently supported in tables using the Duplicate and Unique models.
STRUCT	Variable Length	A structure composed of multiple Fields can also be understood as a collection of multiple columns. It cannot be used as a Key. Currently, STRUCT can only be used in tables of Duplicate models. The name and number of Fields in a Struct are fixed and are always Nullable.
JSON	Variable Length	Binary JSON type, stored in binary JSON format, access internal JSON fields through JSON function. Supported up to 1048576 bytes (1MB) by default, and can be adjusted to a maximum of 2147483643 bytes (2GB). This limit can be modified through the BE configuration parameter 'jsonb_type_length_soft_limit_bytes'.
VARIANT	Variable Length	The VARIANT data type is dynamically adaptable, specifically designed for semi-structured data like JSON. It can store any JSON object and automatically splits JSON fields into subcolumns for improved storage efficiency and query performance. The length limits and configuration methods are the same as for the STRING type. However, the VARIANT type can only be used in value columns and cannot be used in key columns or partition / bucket columns.

Aggregation data type §

Type name	Storeage (bytes)	Description
HLL	Variable Length	HLL stands for HyperLogLog, is a fuzzy deduplication. It performs better than Count Distinct when dealing with large datasets. The error rate of HLL is typically around 1%, and sometimes it can reach 2%. HLL cannot be used as a key column, and the aggregation type is HLL_UNION when creating a table. Users do not need to specify the length or default value as it is internally controlled based on the aggregation level of the data. HLL columns can only be queried or used through the companion functions such as hll_union_agg, hll_raw_agg, hll_cardinality, and hll_hash.
BITMAP	Variable Length	BITMAP type can be used in Aggregate tables, Unique tables or Duplicate tables. - When used in a Unique table or a Duplicate table, BITMAP must be employed as non-key columns. - When used in an Aggregate table, BITMAP must also serve as non-key columns, and the aggregation type must be set to BITMAP_UNION during table creation. Users do not need to specify the length or default value as it is internally controlled based on the aggregation level of the data. BITMAP columns can only be queried or used through the companion functions such as bitmap_union_count, bitmap_union, bitmap_hash, and bitmap_hash64.
QUANTILE_STATE	Variable Length	A type used to calculate approximate quantile values. When loading, it performs pre-aggregation for the same keys with different values. When the number of values does not exceed 2048, it records all data in detail. When the number of values is greater than 2048, it employs the TDigest algorithm to aggregate (cluster) the data and store the centroid points after clustering. QUANTILE_STATE cannot be used as a key column and should be paired with the aggregation type QUANTILE_UNION when creating a table. Users do not need to specify the length or default value as it is internally controlled based on the aggregation level of the data. QUANTILE_STATE columns can only be queried or used through the companion functions such as QUANTILE_PERCENT, QUANTILE_UNION, and TO_QUANTILE_STATE.
AGG_STATE	Variable Length	Aggregate function can only be used with state/merge/union function combiners. AGG_STATE cannot be used as a key column. When creating a table, the signature of the aggregate function needs to be declared alongside. Users do not need to specify the length or default value. The actual data storage size depends on the function's implementation.

IP types §

Type Name	Storage (bytes)	Description
IPv4	4	It is used in conjunction with the `ipv4_*` family of functions.
IPv6	16	It is used in conjunction with the `ipv6_*` family of functions.

You can also view all the data types supported by Doris with the `SHOW DATA TYPES;`statement.

Column Compression

Doris adopts a **columnar storage** model to organize and store data, which is particularly suitable for analytical workloads and can significantly improve query efficiency. In columnar storage, each column of the table is stored independently, facilitating the application of compression techniques and thus improving storage efficiency. Doris provides various compression algorithms, allowing users to choose the appropriate compression method based on workload requirements to optimize storage and query performance.

Why Compression is Needed

In Doris, data compression mainly has the following two core objectives:

1. **Improve Storage Efficiency** Compression can significantly reduce the disk space required for data storage, allowing more data to be stored on the same physical resources.
2. **Optimize Performance** The volume of compressed data is smaller, requiring fewer I/O operations during queries, thereby accelerating query response times. Modern compression algorithms typically have very fast decompression speeds, which can enhance read efficiency while reducing storage space.

Supported Compression Algorithms

Doris supports various compression algorithms, each with different trade-offs between compression ratio and decompression speed, allowing users to choose the appropriate algorithm based on their needs:

Compression Type	Characteristics	Applicable Scenarios
No Compression	- No compression applied to data.	Suitable for scenarios where compression is not needed, such as when the data is already compressed or storage space is not an issue.
LZ4	- Very fast compression and decompression speeds. - Moderate compression ratio.	Suitable for scenarios with high decompression speed requirements, such as real-time queries or high-concurrency loads.
LZ4F (LZ4 Frame)	- Extended version of LZ4 supporting more flexible compression configurations. - Fast speed with moderate compression ratio.	Needed when fast compression is required with fine control over configurations.
LZ4HC (LZ4 High Compression)	- Higher compression ratio compared to LZ4, but slower compression speed. - Decompression speed is comparable to LZ4.	Needed when a higher compression ratio is required, while still focusing on decompression speed.
ZSTD (Zstandard)	- High compression ratio with flexible compression level adjustments. - Decompression speed remains fast even at high compression ratios.	Required for high storage efficiency demands, while balancing query performance.
Snappy	- Designed for fast decompression. - Moderate compression ratio.	Required for scenarios with high decompression speed and low CPU overhead demands.

Compression Type	Characteristics	Applicable Scenarios
Zlib	- Good balance between compression ratio and speed. - Slower compression and decompression speeds compared to other algorithms, but higher compression ratio.	Required for scenarios with high storage efficiency demands and insensitivity to decompression speed, such as archiving and cold data storage.

Compression Principles

Column Compression Due to the adoption of columnar storage, Doris can independently compress each column in the table. This method enhances compression efficiency because the data in the same column often has similar distribution characteristics.

Encoding Before Compression Before compressing data, Doris encodes the column data (e.g., **dictionary encoding**, **run-length encoding**, etc.) to transform the data into a form more suitable for compression, further enhancing compression efficiency.

Page Compression Doris adopts a **page**-level compression strategy. The data in each column is divided into multiple pages, and the data within each page is compressed independently. By compressing by page, Doris can efficiently handle large-scale datasets while ensuring high compression ratios and decompression performance.

Configurable Compression Strategies Users can specify the compression algorithm to be used when creating a table. This flexibility allows users to make the best choice between compression efficiency and performance based on specific workloads.

Factors Affecting Compression Effectiveness

Although different compression algorithms have their own advantages and disadvantages, the effectiveness of compression depends not only on the chosen algorithm but also on the following factors:

Order of Data

The order of data has a significant impact on compression effectiveness. For columns with high sequentiality (e.g., timestamps or continuous numeric columns), compression algorithms can typically achieve better results. The more regular the order of the data, the more repetitive patterns the compression algorithm can identify during compression, thus improving the compression ratio.

Data Redundancy

The more duplicate values in a data column, the more pronounced the compression effect. For example, using dictionary encoding on duplicate values can significantly reduce storage space. However, for data columns without obvious duplicates, the compression effect may not meet expectations.

Data Type

The type of data can also affect compression effectiveness. Generally, numeric data types (such as integers and floating-point numbers) are easier to compress than string data types. For data types with a wide range of

values, the effectiveness of the compression algorithm may be impacted.

Column Length

The length of data in a column can also affect compression effectiveness. Shorter columns are usually easier to compress than longer columns because compression algorithms can more efficiently find repetitive patterns in shorter data blocks.

Nulls

When the proportion of null values in a column is high, the compression algorithm may be more effective because it can encode these null values as a special pattern, reducing storage space.

How to Choose the Right Compression Algorithm

Choosing the right compression algorithm should be based on workload characteristics:

- For **high-performance real-time analysis** scenarios, it is recommended to use **LZ4** or **Snappy**.
- For scenarios prioritizing **storage efficiency**, it is recommended to use **ZSTD** or **Zlib**.
- For scenarios that need to balance speed and compression ratio, **LZ4F** can be chosen.
- For **archiving or cold data storage** scenarios, it is advisable to use **Zlib** or **LZ4HC**.

Setting Compression in Doris

When creating a table, you can specify the compression algorithm to determine how the data is stored:

```sql CREATE TABLE example_table ( id INT, name STRING, age INT ) DUPLICATE KEY(id) DISTRIBUTED BY HASH(id) BUCKETS 10 PROPERTIES ( “compression” = “zstd” );
```

Index Overview

Database indexes are used to accelerate queries. To speed up different query scenarios, Doris supports various rich indexes.

Index Types and Principles

From the perspective of accelerating queries and their principles, Doris indexes are categorized into two main types: point query indexes and skip indexes. - **Point Query Indexes:** Commonly used to speed up point queries, the principle is to locate which rows satisfy the WHERE conditions through the index and directly read those rows. Point query indexes are very effective when the number of rows meeting the conditions is small. Doris's point query indexes include Prefix Index and Inverted Index.

- **Prefix Index:** Doris stores data in an ordered manner according to the sort key and creates a sparse prefix index every 1024 rows. The key in the index is the value of the sorted column in the first row of the current 1024 rows. If the query involves a sorted column, the system will find the first row of the relevant 1024-row group and start scanning from there.
- **Inverted Index:** For columns with an inverted index, an posting list is created mapping each value to a set of row ids. For equality queries, it first finds the set of row ids from the posting list, then directly reads the data of those rows, avoiding row-by-row scanning. Inverted indexes can also accelerate range filtering and fulltext search. The algorithms are more complex, but the basic principles are similar. (Note: the previous BITMAP index has been replaced by the more powerful inverted index.)
- **Skip Indexes:** Commonly used to accelerate analysis, the principle is to determine data blocks that do not satisfy the WHERE conditions through the index and skip these blocks, only reading the data blocks that may satisfy the conditions and then performing a row-by-row filter to finally get the rows that meet the conditions. Skip indexes are more effective when the number of rows meeting the conditions is large. Doris's skip indexes include ZoneMap indexes, BloomFilter indexes, and NGram BloomFilter indexes.
    - **ZoneMap Index:** Automatically maintains statistics for each column, recording the maximum, minimum, and whether there are NULL values for each data file (Segment) and data block (Page). For equality queries, range queries, and IS NULL, it can determine whether the data file and data block can contain the data that meets the conditions based on the maximum value, minimum value, and whether there are NULL values. If not, Doris skips reading the corresponding file or data block, reducing IO and accelerating queries.
    - **BloomFilter Index:** Stores values of the indexed column in a BloomFilter data structure, which can quickly determine whether a value is in the BloomFilter with very low storage space. For equality queries, if the value is not in the BloomFilter, the corresponding data file or data block can be skipped, reducing IO and accelerating queries.
    - **NGram BloomFilter Index:** Used to accelerate text LIKE queries. The principle is similar to the BloomFilter index, but instead of storing the original text values, it performs NGram tokenization of the text and stores each token in the BloomFilter. For LIKE queries, the LIKE pattern is also tokenized using NGram. If any token is not in the BloomFilter, the corresponding data file or data block does not meet the LIKE condition and can be skipped.

Among the above indexes, the prefix index and ZoneMap index are built-in indexes automatically maintained by Doris, requiring no user management. Inverted indexes, BloomFilter indexes, and NGram BloomFilter indexes need to be manually created and managed by the user based on the scenario.

- Comparison of characteristics of different types of indexes

Type	Index	Advantages	Limitations
Point Query	Prefix Index	Built-in index, best performance. Only one prefix index per table.	Only one prefix index per table.
Point Query	Inverted Index	Supports tokenization and keyword matching. Building index on any column. Multi-condition combination and accelerating more functions.	Large index storage space, similar to raw data.
Skip	ZoneMap Index	Built-in index, small index storage space. Only one prefix index per table.	Only one prefix index per table.
Skip	BloomFilter Index	More precise than ZoneMap, medium index space.	Supports few query types. Only supports equal (not others: not equal, range, LIKE, MATCH).
Skip	NGram BloomFilter	Supports LIKE acceleration, medium index space.	Supports few query types. Only supports LIKE acceleration.

- List of operators and functions for index acceleration

Operator or Function	Prefix Index	Inverted Index	ZoneMap Index	BloomFilter Index	NGram BloomFilter Index
=	YES	YES	YES	YES	NO
!=	YES	YES	NO	NO	NO
IN	YES	YES	YES	YES	NO
NOT IN	YES	YES	NO	NO	NO
>, >=, <, <=, BETWEEN	YES	YES	YES	NO	NO
IS NULL	YES	YES	YES	NO	NO
IS NOT NULL	YES	YES	NO	NO	NO
LIKE	NO	NO	NO	NO	YES
MATCH, MATCH_*	NO	YES	NO	NO	NO
array_contains	NO	YES	NO	NO	NO
array_overlaps	NO	YES	NO	NO	NO
is_ip_address_in_range	NO	YES	NO	NO	NO

Index Design Guidelines

The design and optimization of database table indexes are closely related to data distribution and queries, requiring testing and optimization based on the actual scenario. Although there is no “silver bullet,” Doris continuously strives to reduce the difficulty of using indexes. Users can follow these simple guidelines for index selection and testing.

1. Specify the most frequently used filter condition as the KEY to automatically create a prefix index, as it has the best filtering effect. However, only one prefix index can be created per table, so it should be used for the most frequent filter condition.
2. For non-key fields that require filter acceleration, the first choice is to create an inverted index due to its broad applicability and multi-condition combination. The second choice includes the following two indexes:
    - If there is a need for string LIKE matching, add an NGram BloomFilter index.
    - If index storage space is critical, replace the inverted index with a BloomFilter index.
3. If performance is not as expected, analyze the amount of data filtered by the index and the time consumed through QueryProfile. Refer to the detailed documentation of each index for specifics.

Sort Key and Prefix Index

Index Principles

Doris stores data in a structure similar to SSTable (Sorted String Table). This structure is an ordered data structure that can be sorted and stored according to one or more specified columns. In such a data structure, looking up conditions on all or part of the sorted columns is highly efficient.

In the Aggregate, Unique, and Duplicate data models, the underlying data storage is sorted according to the columns specified in the CREATE TABLE statement under AGGREGATE KEY, UNIQUE KEY, and DUPLICATE KEY respectively. These keys are referred to as sort keys. With sort keys, Doris can quickly locate the required data without scanning the entire table by specifying conditions on the sorted columns during a query, thereby reducing search complexity and speeding up the query.

Based on the sort keys, Doris introduces a prefix index. The prefix index is a sparse index. The data in the table forms a logical data block (Data Block) according to the corresponding number of rows. Each logical data block stores an index entry in the prefix index table, where the length of the index entry does not exceed 36 bytes. The entry content is the prefix composed of the sorted columns of the first row in the data block. When looking up the prefix index table, it helps determine the starting row number of the logical data block where the row data is located. Because the prefix index is relatively small, it can be fully cached in memory, allowing for quick data block location and significantly improving query efficiency.

> **The first 36 bytes of a row in a data block are used as the prefix index for that row. When encountering a VARCHAR type, the prefix index is directly truncated. If the first column is VARCHAR, even if it does not reach 36 bytes, it will be directly truncated, and the subsequent columns will not be included in the prefix index.**

Use Cases

Prefix indexes can speed up equality queries and range queries.

Managing Indexes

There is no specific syntax to define a prefix index. When creating a table, the first 36 bytes of the table's Key are automatically taken as the prefix index.

Recommendations for Prefix Index Selection

> **Because the Key definition of a table is unique, a table has only one set of prefix indexes. Therefore, it is important to choose an appropriate prefix index when designing the table structure. The following recommendations can be considered:**
>
> 1. Choose the fields most commonly used in WHERE filtering conditions as the Key.

2. Place the more frequently used fields at the front, as prefix indexes are only effective for fields in the WHERE condition that are part of the Key's prefix.

For queries that use other columns not covered by the prefix index as conditions, the efficiency may not meet the requirements. There are two solutions: 1. Create an inverted index on the columns that require accelerated queries, as a table can have many inverted indexes. 2. For DUPLICATE tables, multi-prefix indexes can be indirectly achieved by creating corresponding strongly consistent materialized views with adjusted column orders. For more details, refer to query acceleration/materialized views.

Using Indexes

Prefix indexes are used to accelerate equality and range queries in the WHERE clause. They automatically take effect when applicable, and there is no special syntax required.

The acceleration effect of the prefix index can be analyzed using the following metrics in the Query Profile: - RowsKeyRangeFiltered: The number of rows filtered by the prefix index, which can be compared with other Rows values to analyze the filtering effect of the index.

Example Usage

- Suppose the sorted columns of a table are as follows, then the prefix index would be: user_id (8 Bytes) + age (4 Bytes) + message (prefix 20 Bytes).

ColumnName	Type
user_id	BIGINT
age	INT
message	VARCHAR(100)
max_dwell_time	DATETIME
min_dwell_time	DATETIME

- Suppose the sorted columns of a table are as follows, then the prefix index would be user_name (20 Bytes). Even if it does not reach 36 bytes, it is directly truncated due to encountering VARCHAR, and subsequent columns are not included.

ColumnName	Type
user_name	VARCHAR(20)
age	INT
message	VARCHAR(100)
max_dwell_time	DATETIME
min_dwell_time	DATETIME

- When our query condition is the prefix of the prefix index, it can significantly speed up the query. For example, in the first example, executing the following query:

```
SELECT * FROM table WHERE user_id = 1829239 AND age = 20;
```

This query will be much more efficient than the following query:

```
SELECT * FROM table WHERE age = 20;
```

Therefore, choosing the correct column order when creating a table can greatly improve query efficiency.

Inverted Index

Indexing Principle

Inverted Index is a commonly used indexing technique in the field of information retrieval. It divides text into individual words and constructs a word -> document IDs index, allowing for quick searches to determine which documents contain a specific word.

Starting from version 2.0.0, Doris supports inverted indexes, which can be used for full-text searches on text types, equality, and range queries on normal numerical and date types, and quickly filtering rows that meet the criteria from massive amounts of data.

In Doris's implementation of the inverted index, each row in the table corresponds to a document, and each column corresponds to a field in the document. Thus, using an inverted index, you can quickly locate rows containing specific keywords, thereby accelerating the WHERE clause.

Unlike other indexes in Doris, the inverted index uses independent files at the storage layer, corresponding one-to-one with data files but physically stored independently. This approach allows for creating and deleting indexes without rewriting data files, significantly reducing processing overhead.

Usage Scenarios

Inverted indexes have a wide range of applications and can accelerate equality, range, and full-text searches (keyword matching, phrase matching, etc.). A table can have multiple inverted indexes, and the conditions of multiple inverted indexes can be combined arbitrarily during queries.

The functionality of inverted indexes is briefly introduced as follows:

1. Accelerate full-text searches for string types - Support for keyword search, including matching multiple keywords simultaneously `MATCH_ALL` and matching any one keyword `MATCH_ANY`.

- Support for phrase queries `MATCH_PHRASE`
- Support for tokenized regular expression queries `MATCH_REGEXP`
- Support for English, Chinese, and Unicode tokenizers

2. Accelerate normal equality and range queries, covering and replacing the functionality of BITMAP index

- Support for fast filtering of string, numerical, and datetime types for =, !=, >, >=, <, <=
- Support for fast filtering of string, numerical, and datetime array types for `array_contains`

3. Support for comprehensive logical combinations

- Not only supports acceleration for AND conditions but also for OR and NOT conditions
- Supports arbitrary logical combinations of multiple conditions with AND, OR, NOT

4. Flexible and efficient index management

- Support for defining inverted indexes when creating a table
- Support for adding inverted indexes to existing tables, with incremental index construction without rewriting existing data in the table
- Support for deleting inverted indexes from existing tables without rewriting existing data in the table

> **There are some limitations to using inverted indexes:**
>
> 1. Floating-point types FLOAT and DOUBLE, which have precision issues, do not support inverted indexes due to inaccurate precision. The solution is to use the precisely accurate DECIMAL type, which supports inverted indexes.
> 2. Some complex data types do not yet support inverted indexes, including MAP, STRUCT, JSON, HLL, BITMAP, QUANTILE_STATE, AGG_STATE.
> 3. DUPLICATE and UNIQUE table models with Merge-on-Write enabled support building inverted indexes on any column. However, AGGREGATE and UNIQUE models without Merge-on-Write enabled only support building inverted indexes on Key columns, as non-Key columns cannot have inverted indexes. This is because these two models require reading all data for merging, so indexes cannot be used for pre-filtering.

Managing Indexes

Defining Inverted Indexes When Creating a Table

In the table creation statement, after the COLUMN definition, is the index definition:

```
CREATE TABLE table_name
(
  column_name1 TYPE1,
  column_name2 TYPE2,
  column_name3 TYPE3,
  INDEX idx_name1(column_name1) USING INVERTED [PROPERTIES(...)] [COMMENT 'your comment'],
  INDEX idx_name2(column_name2) USING INVERTED [PROPERTIES(...)] [COMMENT 'your comment']
)
table_properties;
```

Syntax explanation:

1. `idx_column_name(column_name)` **is mandatory,** `column_name` **is the name of the column for the index, must be a column defined earlier,** `idx_column_name` **is the index name, must be unique at the table level, recommended naming convention: prefix** `idx_` **before the column name**

2. `USING INVERTED` **is mandatory to specify that the index type is an inverted index**

3. `PROPERTIES` **is optional to specify additional properties of the inverted index, currently supported properties are:**

parser: specifies the tokenizer

- By default, it is unspecified, meaning no tokenization
    - `english`: English tokenization, suitable for columns with English text, uses spaces and punctuation for tokenization, high performance
        - `chinese`: Chinese tokenization, suitable for columns with mainly Chinese text, lower performance than English tokenization
            - `unicode`: Unicode tokenization, suitable for mixed Chinese and English, and mixed multilingual texts. It can tokenize email prefixes and suffixes, IP addresses, and mixed character and number strings, and can tokenize Chinese by characters.

  Tokenization results can be verified using the `TOKENIZE` SQL function, see the following sections for details.

parser_mode

Specifies the tokenization mode, currently supported modes for `parser = chinese` are:

- fine_grained: fine-grained mode, tends to generate shorter, more words, e.g., '武汉市长江大桥' will be tokenized into '武汉', '武汉市', '市长', '长江', '长江大桥', '大桥'
    - coarse_grained: coarse-grained mode, tends to generate longer, fewer words, e.g., '武汉市长江大桥' will be tokenized into '武汉市', '长江大桥'
        - default coarse_grained

support_phrase

Specifies whether the index supports MATCH_PHRASE phrase query acceleration

- true: supported, but the index requires more storage space
    - false: not supported, more storage efficient, can use MATCH_ALL to query multiple keywords
        - From versions 2.0.14, 2.1.5 and 3.0.1, the default is true if parser is set. Otherwise default to false.

  For example, the following example specifies Chinese tokenization, coarse-grained mode, and supports phrase query acceleration.

```
 INDEX idx_name(column_name) USING INVERTED PROPERTIES("parser" = "chinese", "parser_mode" = "coarse_grained",
"support_phrase" = "true")
```

char_filter

Specifies preprocessing the text before tokenization, usually to affect tokenization behavior

char_filter_type: specifies different functional char_filters (currently only supports char_replace)

char_replace replaces each char in the pattern with a char in the replacement

- char_filter_pattern: characters to be replaced
    - char_filter_replacement: replacement character array, optional, defaults to a space character

  For example, the following example replaces dots and underscores with spaces, thus treating them as word separators, affecting tokenization behavior.

```
 INDEX idx_name(column_name) USING INVERTED PROPERTIES("parser" = "unicode", "char_filter_type" = "char_replace",
"char_filter_pattern" = "._", "char_filter_replacement" = " ")
```

ignore_above

Specifies the length limit for non-tokenized string indexes (parser not specified)

- Strings longer than the length set by ignore_above will not be indexed. For string arrays, ignore_above applies to each array element separately, and elements longer than ignore_above will not be indexed.
    - Default is 256, unit is bytes

lower_case

Whether to convert tokens to lowercase for case-insensitive matching

- true: convert to lowercase
    - false: do not convert to lowercase
        - From version 2.1.2, the default is true, automatically converting to lowercase. Earlier versions default to false.

stopwords

Specifying the stopword list to use, which will affect the behavior of the tokenizer

- The default built-in stopword list includes meaningless words such as ‘is,’ ‘the,’ ‘a,’ etc. When writing or querying, the tokenizer will ignore words that are in the stopword list.
    - none: Use an empty stopword list

4. `COMMENT` **is optional for specifying index comments**

Adding Inverted Indexes to Existing Tables

1. ADD INDEX

Supports both `CREATE INDEX` and `ALTER TABLE ADD INDEX` syntax. The parameters are the same as those used when defining indexes during table creation.

```
-- Syntax 1
CREATE INDEX idx_name ON table_name(column_name) USING INVERTED [PROPERTIES(...)] [COMMENT 'your comment'];
-- Syntax 2
ALTER TABLE table_name ADD INDEX idx_name(column_name) USING INVERTED [PROPERTIES(...)] [COMMENT 'your comment'];
```

2. BUILD INDEX

The `CREATE / ADD INDEX` operation only adds the index definition. New data written after this operation will generate inverted indexes, but existing data requires using `BUILD INDEX` to trigger indexing:

```
-- Syntax 1, by default, builds the index for all partitions in the table
BUILD INDEX index_name ON table_name;
-- Syntax 2, you can specify partitions, one or more
BUILD INDEX index_name ON table_name PARTITIONS(partition_name1, partition_name2);
```

To check the progress of `BUILD INDEX`, use `SHOW BUILD INDEX`:

```
SHOW BUILD INDEX [FROM db_name];
-- Example 1, view the progress of all BUILD INDEX tasks
SHOW BUILD INDEX;
-- Example 2, view the progress of BUILD INDEX tasks for a specific table
SHOW BUILD INDEX where TableName = "table1";
```

To cancel `BUILD INDEX`, use `CANCEL BUILD INDEX`:

```
CANCEL BUILD INDEX ON table_name;
CANCEL BUILD INDEX ON table_name (job_id1, job_id2, ...);
```

> `BUILD INDEX` **creates an asynchronous task executed by multiple threads on each BE. The number of threads can be set using the BE config** `alter_index_worker_count`**, with a default value of 3.**
>
> In versions before and 2.1.4, `BUILD INDEX` would keep retrying until it succeeded. Starting from this version, failure and timeout mechanisms prevent endless retries.
>
> 1. If the majority of replicas for a tablet fail to `BUILD INDEX`, the entire `BUILD INDEX` operation fails.
> 2. If the time exceeds `alter_table_timeout_second`, the `BUILD INDEX` operation times out.
> 3. Users can trigger `BUILD INDEX` multiple times; indexes that have already been built successfully will not be rebuilt.

Deleting Inverted Indexes from Existing Tables

```
-- Syntax 1
DROP INDEX idx_name ON table_name;
-- Syntax 2
ALTER TABLE table_name DROP INDEX idx_name;
```

> `DROP INDEX` **deletes the index definition, so new data will no longer write to the index. This creates an asynchronous task to perform the index deletion, executed by multiple threads on each BE. The number of threads can be set using the BE parameter** `alter_index_worker_count`**, with a default value of 3.**

Viewing Inverted Index

– Syntax 1: The INDEX section in the table schema with USING INVERTED indicates an inverted index SHOW CREATE TABLE table_name;

– Syntax 2: IndexType as INVERTED indicates an inverted index SHOW INDEX FROM table_name;

Using Indexes

Accelerating Queries with Inverted Indexes

```
-- 1. Full-text search keyword matching using MATCH_ANY and MATCH_ALL
SELECT * FROM table_name WHERE column_name MATCH_ANY | MATCH_ALL 'keyword1 ...';

-- 1.1 Rows in the content column containing keyword1
SELECT * FROM table_name WHERE content MATCH_ANY 'keyword1';

-- 1.2 Rows in the content column containing keyword1 or keyword2; you can add more keywords
SELECT * FROM table_name WHERE content MATCH_ANY 'keyword1 keyword2';

-- 1.3 Rows in the content column containing both keyword1 and keyword2; you can add more keywords
SELECT * FROM table_name WHERE content MATCH_ALL 'keyword1 keyword2';
```

```
-- 2. Full-text search phrase matching using MATCH_PHRASE

-- 2.1 Rows in the content column containing both keyword1 and keyword2, where keyword2 must immediately follow keyword1
-- 'keyword1 keyword2', 'wordx keyword1 keyword2', 'wordx keyword1 keyword2 wordy' all match because they contain 'keyword1
keyword2' with keyword2 immediately following keyword1
-- 'keyword1 wordx keyword2' does not match because there is a word between keyword1 and keyword2
-- 'keyword2 keyword1' does not match because the order is reversed
-- To use MATCH_PHRASE, you need to enable "support_phrase" = "true" in PROPERTIES.
SELECT * FROM table_name WHERE content MATCH_PHRASE 'keyword1 keyword2';

-- 2.2 Rows in the content column containing both keyword1 and keyword2, with a slop (maximum word distance) of 3
-- 'keyword1 keyword2', 'keyword1 a keyword2', 'keyword1 a b c keyword2' all match because the slop is 0, 1, and 3 respectively, all
within 3
-- 'keyword1 a b c d keyword2' does not match because the slop is 4, exceeding 3
-- 'keyword2 keyword1', 'keyword2 a keyword1', 'keyword2 a b c keyword1' also match because when slop > 0, the order of keyword1 and
keyword2 is not required. To enforce the order, Doris provides a + sign after slop
SELECT * FROM table_name WHERE content MATCH_PHRASE 'keyword1 keyword2 ~3';
-- To enforce order, use a positive sign with slop; 'keyword1 a b c keyword2' matches, while 'keyword2 a b c keyword1' does not
SELECT * FROM table_name WHERE content MATCH_PHRASE 'keyword1 keyword2 ~3+';

-- 2.3 Prefix matching the last word keyword2, with a default limit of 50 prefixes (controlled by session variable
inverted_index_max_expansions)
-- It is necessary to ensure that keyword1 and keyword2 remain adjacent in the original text after tokenization, with no other words
in between.
-- 'keyword1 keyword2abc' matches because keyword1 is identical and keyword2abc is a prefix of keyword2
-- 'keyword1 keyword2' also matches because keyword2 is a prefix of keyword2
-- 'keyword1 keyword3' does not match because keyword3 is not a prefix of keyword2
-- 'keyword1 keyword3abc' does not match because keyword3abc is not a prefix of keyword2
SELECT * FROM table_name WHERE content MATCH_PHRASE_PREFIX 'keyword1 keyword2';

-- 2.4 If only one word is provided, it defaults to a prefix query with a limit of 50 prefixes (controlled by session variable
inverted_index_max_expansions)
SELECT * FROM table_name WHERE content MATCH_PHRASE_PREFIX 'keyword1';

-- 2.5 Regular expression matching on tokenized words, with a default limit of 50 matches (controlled by session variable
inverted_index_max_expansions)
```

```
-- Similar to MATCH_PHRASE_PREFIX but with regex instead of prefix
SELECT * FROM table_name WHERE content MATCH_REGEXP 'key.*';

-- 3. Normal equality, range, IN, and NOT IN queries using standard SQL syntax, for example:
SELECT * FROM table_name WHERE id = 123;
SELECT * FROM table_name WHERE ts > '2023-01-01 00:00:00';
SELECT * FROM table_name WHERE op_type IN ('add', 'delete');

-- 4. Full-text search across multiple columns using the multi_match function
-- Parameters:
--   First N parameters are column names to search
--   Second-to-last parameter specifies match mode: 'any'/'all'/'phrase'/'phrase_prefix'
--   Last parameter is the keyword or phrase to search for

-- 4.1 Rows where 'keyword1' appears in ANY of col1,col2,col3 (OR logic)
select * FROM table_name WHERE multi_match(col1, col2, col3, 'any', 'keyword1');

-- 4.2 Rows where 'keyword1' appears in ALL of col1,col2,col3 (AND logic)
select * FROM table_name WHERE multi_match(col1, col2, col3, 'all', 'keyword1');

-- 4.3 Rows where the exact phrase 'keyword1' appears in ANY of col1,col2,col3 (exact phrase match)
select * FROM table_name WHERE multi_match(col1, col2, col3, 'phrase', 'keyword1');

-- 4.4 Rows where a phrase starting with 'keyword1' appears in ANY of col1,col2,col3 (phrase prefix match)
-- For example, will match content like "keyword123"
select * FROM table_name WHERE multi_match(col1, col2, col3, 'phrase_prefix', 'keyword1');
```

Analyzing Index Acceleration Effects Through Profiles

Inverted query acceleration can be toggled using the session variable `enable_inverted_index_query`, which is set to true by default. To verify the acceleration effect of the index, it can be set to false to turn it off.

The acceleration effect of the inverted index can be analyzed using the following metrics in the Query Profile: - RowsInvertedIndexFiltered: The number of rows filtered by the inverted index, which can be compared with other Rows values to analyze the filtering effect of the index. - InvertedIndexFilterTime: The time consumed by the inverted index. - InvertedIndexSearcherOpenTime: The time taken to open the inverted index. - InvertedIndexSearcherSearchTime: The time taken for internal queries within the inverted index.

Verifying Tokenization Effects Using Tokenization Functions

To check the actual effect of tokenization or to tokenize a piece of text, you can use the `TOKENIZE` function for verification.

The first parameter of the `TOKENIZE` function is the text to be tokenized, and the second parameter specifies the tokenization parameters used when creating the index.

SELECT TOKENIZE('I love Doris','"parser"="english"'); +----------+ | tokenize('I love Doris', '"parser"="english"') | +----------+ | ["i", "love", "doris"] | +----------+

```
## Usage Example

Demonstrating the creation of an inverted index, full-text search, and regular queries using 1 million records from HackerNews. This includes a simple performance comparison with queries without indexing.

### Table Creation
```

```sql
CREATE DATABASE test_inverted_index;

USE test_inverted_index;

-- Create a table with an inverted index on the comment field
--   USING INVERTED specifies the index type as an inverted index
--   PROPERTIES("parser" = "english") specifies using the "english" tokenizer; other options include "chinese" for Chinese
tokenization and "unicode" for mixed-language tokenization. If the "parser" parameter is not specified, no tokenization is applied.

CREATE TABLE hackernews_1m
(
    `id` BIGINT,
    `deleted` TINYINT,
    `type` String,
    `author` String,
    `timestamp` DateTimeV2,
    `comment` String,
    `dead` TINYINT,
    `parent` BIGINT,
    `poll` BIGINT,
    `children` Array<BIGINT>,
    `url` String,
    `score` INT,
    `title` String,
    `parts` Array<INT>,
    `descendants` INT,
    INDEX idx_comment (`comment`) USING INVERTED PROPERTIES("parser" = "english") COMMENT 'inverted index for comment'
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES ("replication_num" = "1");
```

Data Import

Importing Data via Stream Load

```
wget https://qa-build.oss-cn-beijing.aliyuncs.com/regression/index/hacknernews_1m.csv.gz

curl --location-trusted -u root: -H "compress_type:gz" -T hacknernews_1m.csv.gz -XPUT
http://127.0.0.1:8030/api/test_inverted_index/hackernews_1m/_stream_load
{
    "TxnId": 2,
    "Label": "a8a3e802-2329-49e8-912b-04c800a461a6",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 1000000,
    "NumberLoadedRows": 1000000,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 130618406,
    "LoadTimeMs": 8988,
    "BeginTxnTimeMs": 23,
    "StreamLoadPutTimeMs": 113,
    "ReadDataTimeMs": 4788,
    "WriteDataTimeMs": 8811,
    "CommitAndPublishTimeMs": 38
}
```

Confirm Data Import Success with SQL count()

```
SELECT count() FROM hackernews_1m;
+---------+
| count() |
+---------+
| 1000000 |
+---------+
```

Queries

01 Full-Text Search

- Using `LIKE` to match and count rows containing 'OLAP' in the `comment` column took 0.18s.

  ```
  SELECT count() FROM hackernews_1m WHERE comment LIKE '%OLAP%';
  +---------+
  | count() |
  +---------+
  |      34 |
  +---------+
  ```

- Using full-text search with `MATCH_ANY` based on the inverted index to count rows containing 'OLAP' in the `comment` column took 0.02s, resulting in a 9x speedup. The performance improvement would be even more significant on larger datasets.

  The difference in the number of results is due to the inverted index normalizing the terms by converting them to lowercase, among other processes, hence `MATCH_ANY` yields more results than `LIKE`.

  ```
  SELECT count() FROM hackernews_1m WHERE comment MATCH_ANY 'OLAP';
  +---------+
  | count() |
  +---------+
  |      35 |
  +---------+
  ```

- Similarly, comparing the performance for counting occurrences of 'OLTP', 0.07s vs 0.01s. Due to caching, both `LIKE` and `MATCH_ANY` improved, but the inverted index still provided a 7x speedup.

  ```
  SELECT count() FROM hackernews_1m WHERE comment LIKE '%OLTP%';
  +---------+
  | count() |
  +---------+
  |      48 |
  +---------+

  SELECT count() FROM hackernews_1m WHERE comment MATCH_ANY 'OLTP';
  +---------+
  | count() |
  +---------+
  |      51 |
  +---------+
  ```

- Counting rows where both 'OLAP' and 'OLTP' appear took 0.13s vs 0.01s, a 13x speedup.

To require multiple terms to appear simultaneously (AND relationship), use `MATCH_ALL 'keyword1 keyword2 ...'`.

```
SELECT count() FROM hackernews_1m WHERE comment LIKE '%OLAP%' AND comment LIKE '%OLTP%';
+---------+
| count() |
+---------+
|      14 |
+---------+


SELECT count() FROM hackernews_1m WHERE comment MATCH_ALL 'OLAP OLTP';
+---------+
| count() |
+---------+
|      15 |
+---------+
```

- Counting rows where either ‘OLAP’ or ‘OLTP’ appears took 0.12s vs 0.01s, a 12x speedup.

  To require any one or more of multiple terms to appear (OR relationship), use `MATCH_ANY 'keyword1 keyword2 ...'`.

```
SELECT count() FROM hackernews_1m WHERE comment LIKE '%OLAP%' OR comment LIKE '%OLTP%';
+---------+
| count() |
+---------+
|      68 |
+---------+

SELECT count() FROM hackernews_1m WHERE comment MATCH_ANY 'OLAP OLTP';
+---------+
| count() |
+---------+
|      71 |
+---------+
```

02 Standard Equality and Range Queries

- Range query on a `DateTime` type column

```
SELECT count() FROM hackernews_1m WHERE timestamp > '2007-08-23 04:17:00';
+---------+
| count() |
+---------+
|  999081 |
+---------+
```

- Adding an inverted index for the `timestamp` column

```
-- For date-time types, USING INVERTED does not require specifying a parser
-- CREATE INDEX is one syntax for creating an index, another method will be shown later
CREATE INDEX idx_timestamp ON hackernews_1m(timestamp) USING INVERTED;
```

```
BUILD INDEX idx_timestamp ON hackernews_1m;
```

- Checking the index creation progress. From the difference between `FinishTime` and `CreateTime`, we can see that building the inverted index for 1 million rows on the `timestamp` column took only 1 second.

```
SHOW ALTER TABLE COLUMN;
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----
-----------+---------------+----------+------+----------+---------+
| JobId | TableName     | CreateTime              | FinishTime              | IndexName     | IndexId | OriginIndexId |
SchemaVersion | TransactionId | State    | Msg  | Progress | Timeout |
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----
-----------+---------------+----------+------+----------+---------+
| 10030 | hackernews_1m | 2023-02-10 19:44:12.929 | 2023-02-10 19:44:13.938 | hackernews_1m | 10031   | 10008         |
1:1994690496  | 3             | FINISHED |      | NULL     | 2592000 |
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----
-----------+---------------+----------+------+----------+---------+
```

```
-- If the table has no partitions, PartitionName defaults to TableName
SHOW BUILD INDEX;
+-------+---------------+---------------+-----------------------------------------------------------------+-----------------------
--+-------------------------+---------------+----------+------+----------+
| JobId | TableName     | PartitionName | AlterInvertedIndexes                                            | CreateTime
| FinishTime              | TransactionId | State    | Msg  | Progress |
+-------+---------------+---------------+-----------------------------------------------------------------+-----------------------
--+-------------------------+---------------+----------+------+----------+
| 10191 | hackernews_1m | hackernews_1m | [ADD INDEX idx_timestamp (`timestamp`) USING INVERTED],  | 2023-06-26
15:32:33.894 | 2023-06-26 15:32:34.847 | 3             | FINISHED |      | NULL     |
+-------+---------------+---------------+-----------------------------------------------------------------+-----------------------
--+-------------------------+---------------+----------+------+----------+
```

- After the index is created, range queries use the same query syntax. Doris will automatically recognize the index for optimization. However, due to the small dataset, the performance difference is not significant.

```
SELECT count() FROM hackernews_1m WHERE timestamp > '2007-08-23 04:17:00';
+---------+
| count() |
+---------+
|  999081 |
+---------+
```

- Performing similar operations on a numeric column `parent` with an equality match query.

```
SELECT count() FROM hackernews_1m WHERE parent = 11189;
+---------+
| count() |
+---------+
|       2 |
+---------+

-- For numeric types, USING INVERTED does not require specifying a parser
-- ALTER TABLE t ADD INDEX is the second syntax for creating an index
ALTER TABLE hackernews_1m ADD INDEX idx_parent(parent) USING INVERTED;


-- Execute BUILD INDEX to create the inverted index for existing data
BUILD INDEX idx_parent ON hackernews_1m;


SHOW ALTER TABLE COLUMN;
```

```
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----
-----------+---------------+----------+------+----------+---------+
| JobId | TableName     | CreateTime              | FinishTime              | IndexName     | IndexId | OriginIndexId |
SchemaVersion | TransactionId | State    | Msg  | Progress | Timeout |
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----
-----------+---------------+----------+------+----------+---------+
| 10030 | hackernews_1m | 2023-02-10 19:44:12.929 | 2023-02-10 19:44:13.938 | hackernews_1m | 10031   | 10008         |
1:1994690496  | 3             | FINISHED |      | NULL     | 2592000 |
| 10053 | hackernews_1m | 2023-02-10 19:49:32.893 | 2023-02-10 19:49:33.982 | hackernews_1m | 10054   | 10008         |
1:378856428   | 4             | FINISHED |      | NULL     | 2592000 |
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----
-----------+---------------+----------+------+----------+---------+

SHOW BUILD INDEX;
+-------+---------------+---------------+-----------------------------------------------------------+-------------------------+---
---------------------+---------------+----------+------+----------+
| JobId | TableName     | PartitionName | AlterInvertedIndexes                                      | CreateTime              |
FinishTime              | TransactionId | State    | Msg  | Progress |
+-------+---------------+---------------+-----------------------------------------------------------+-------------------------+---
---------------------+---------------+----------+------+----------+
| 11005 | hackernews_1m | hackernews_1m | [ADD INDEX idx_parent (`parent`) USING INVERTED],  | 2023-06-26 16:25:10.167 |
2023-06-26 16:25:10.838 | 1002          | FINISHED |      | NULL     |
+-------+---------------+---------------+-----------------------------------------------------------+-------------------------+---
---------------------+---------------+----------+------+----------+


SELECT count() FROM hackernews_1m WHERE parent = 11189;
+---------+
| count() |
+---------+
|       2 |
+---------+
```

- Creating an inverted index for the string column `author` without tokenization. Equality queries can also leverage the index for speedup.

  ```sql SELECT count() FROM hackernews_1m WHERE author = ‘faster’; +———+ | count() | +———+ | 20 | +———+

  – Here, USING INVERTED is used without tokenizing the `author` column, treating it as a single term ALTER TABLE hackernews_1m ADD INDEX idx_author(author) USING INVERTED;

  – Execute BUILD INDEX to add the inverted index for existing data BUILD INDEX idx_author ON hackernews_1m;

Creating an incremental index for 1 million author records took only 1.5 seconds.

```
SHOW ALTER TABLE COLUMN;
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----------
--+---------------+----------+------+----------+---------+
| JobId | TableName | CreateTime | FinishTime | IndexName | IndexId | OriginIndexId |
SchemaVersion | TransactionId | State | Msg | Progress | Timeout |
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+----------
--+---------------+----------+------+----------+---------+
| 10030 | hackernews_1m | 2023-02-10 19:44:12.929 | 2023-02-10 19:44:13.938 | hackernews_1m | 10031 | 10008 | 1:1994690496
| 3 | FINISHED | | NULL | 2592000 |
| 10053 | hackernews_1m | 2023-02-10 19:49:32.893 | 2023-02-10 19:49:33.982 | hackernews_1m | 10054 | 10008 | 1:378856428
| 4 | FINISHED | | NULL | 2592000 |
| 10076 | hackernews_1m | 2023-02-10 19:54:20.046 | 2023-02-10 19:54:21.521 | hackernews_1m | 10077 | 10008 | 1:1335127701
| 5 | FINISHED | | NULL | 2592000 |
```
```

```
+-------+---------------+-------------------------+-------------------------+---------------+---------+---------------+-------------
--+---------------+----------+------+----------+---------+
```

```
SHOW BUILD INDEX ORDER BY CreateTime DESC LIMIT 1;
+-------+---------------+---------------+-----------------------------------------------------+-------------------------+------------
-------------+---------------+----------+------+----------+
| JobId | TableName     | PartitionName | AlterInvertedIndexes                                | CreateTime              | FinishTime
| TransactionId | State    | Msg  | Progress |
+-------+---------------+---------------+-----------------------------------------------------+-------------------------+------------
-------------+---------------+----------+------+----------+
| 13006 | hackernews_1m | hackernews_1m | [ADD INDEX idx_author (`author`) USING INVERTED],  | 2023-06-26 17:23:02.610 | 2023-06-26
17:23:03.755 | 3004          | FINISHED |      | NULL     |
+-------+---------------+---------------+-----------------------------------------------------+-------------------------+------------
-------------+---------------+----------+------+----------+
```

– After creating the index, string equality matches also showed significant acceleration.

```
SELECT count() FROM hackernews_1m WHERE author = 'faster';
+---------+
| count() |
+---------+
|      20 |
+---------+
```

BloomFilter Index

Indexing Principles

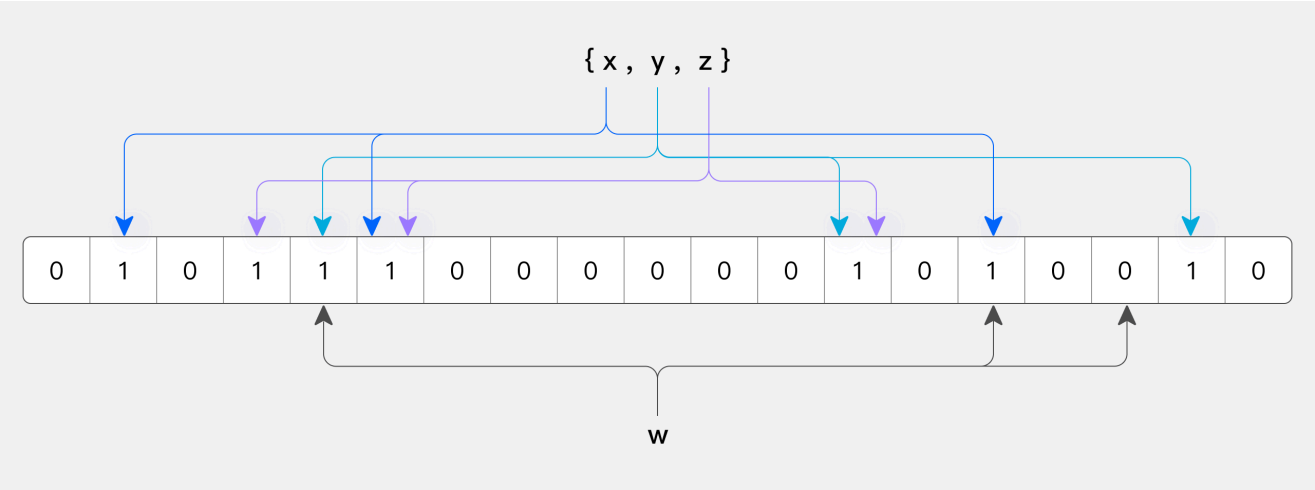
The BloomFilter index is a type of skip index based on BloomFilter. Its principle is to use BloomFilter to skip data blocks that do not meet the specified conditions for equality queries, thereby reducing IO and accelerating queries.

BloomFilter is a fast lookup algorithm proposed by Bloom in 1970, which uses multiple hash functions. It is commonly used in scenarios where quick determination of whether an element belongs to a set is needed without requiring 100% accuracy. BloomFilter has the following characteristics:

- A space-efficient probabilistic data structure used to check if an element is in a set.
- For a membership check, BloomFilter returns one of two results: possibly in the set or definitely not in the set.

A BloomFilter consists of a very long binary bit array and a series of hash functions. The bit array is initially all set to 0. When an element is to be checked, it is hashed by a series of hash functions to generate a series of values, and the bits at these positions in the array are set to 1.

The figure below shows an example of a BloomFilter with m=18 and k=3 (where m is the size of the bit array and k is the number of hash functions). Elements x, y, and z in the set are hashed by 3 different hash functions into the bit array. When querying element w, if any bit calculated by the hash functions is 0, then w is not in the set. Conversely, if all bits are 1, it only indicates that w may be in the set, but not definitely, due to possible hash collisions.



Bloom_filter

Thus, if all bits at the calculated positions are 1, it only indicates that the element may be in the set, not definitely, due to possible hash collisions. This is the “false positive” nature of BloomFilter. Therefore, a BloomFilter-based index can only skip data that does not meet the conditions but cannot precisely locate data that does.

The Doris BloomFilter index is built on a per-page basis, with each data block storing a BloomFilter. During writing, each value in the data block is hashed into the corresponding BloomFilter. During querying, for equality

conditions, it is checked if the BloomFilter of each data block contains the value. If not, the data block is skipped, reducing IO and speeding up the query.

Use Cases

BloomFilter indexes can accelerate equality queries (including = and IN) and are effective for high cardinality fields, such as unique id fields like userid.

> **BloomFilter has the following limitations:**
>
> 1. It has no effect on queries other than in and =, such as !=, NOT IN, >, <, etc.
> 2. It does not support BloomFilter indexing on columns of type Tinyint, Float, Double.
> 3. It has limited acceleration effect on low cardinality fields. For example, a “gender” field with only two values will likely be included in almost every data block, making the BloomFilter index meaningless.
>
> To check the effect of a BloomFilter index on a query, you can analyze relevant metrics in the Query Profile.
>
> - BlockConditionsFilteredBloomFilterTime is the time consumed by the BloomFilter index.
> - RowsBloomFilterFiltered is the number of rows filtered out by the BloomFilter. You can compare it with other Rows values to analyze the filtering effect of the BloomFilter index.

Managing Indexes

Creating a BloomFilter Index When Creating a Table

Due to historical reasons, the syntax for defining BloomFilter indexes differs from the general INDEX syntax used for inverted indexes. BloomFilter indexes are specified in the table’s PROPERTIES using “bloom_filter_columns”, which can specify one or more fields.

```
PROPERTIES (
"bloom_filter_columns" = "column_name1,column_name2"
);
```

Viewing BloomFilter Indexes

```
SHOW CREATE TABLE table_name;
```

Adding or Removing BloomFilter Indexes on an Existing Table

Modify the table’s bloom_filter_columns property via ALTER TABLE to add or remove BloomFilter indexes.

Add a BloomFilter index for column_name3

```
ALTER TABLE table_name SET ("bloom_filter_columns" = "column_name1,column_name2,column_name3");
```

Remove the BloomFilter index for column_name1

```
ALTER TABLE table_name SET ("bloom_filter_columns" = "column_name2,column_name3");
```

Using Indexes

BloomFilter index is used to accelerate equality queries in the WHERE clause. It automatically takes effect when applicable, and there is no special syntax required.

The acceleration effect of the BloomFilter index can be analyzed using the following metrics in the Query Profile: - RowsBloomFilterFiltered: The number of rows filtered by the BloomFilter index, which can be compared with other Rows values to analyze the filtering effect of the index. - BlockConditionsFilteredBloomFilterTime: The time consumed by the BloomFilter inverted index.

Usage Example

Here is an example of how to create a BloomFilter index in Doris.

The BloomFilter index in Doris is created by adding the “bloom_filter_columns” property in the CREATE TABLE statement, with k1, k2, k3 being the key columns for the BloomFilter index. For example, the following creates BloomFilter indexes on saler_id and category_id.

```
CREATE TABLE IF NOT EXISTS sale_detail_bloom  (
    sale_date date NOT NULL COMMENT "Sale date",
    customer_id int NOT NULL COMMENT "Customer ID",
    saler_id int NOT NULL COMMENT "Salesperson",
    sku_id int NOT NULL COMMENT "Product ID",
    category_id int NOT NULL COMMENT "Product category",
    sale_count int NOT NULL COMMENT "Sales quantity",
    sale_price DECIMAL(12,2) NOT NULL COMMENT "Unit price",
    sale_amt DECIMAL(20,2)  COMMENT "Total sales amount"
)
DUPLICATE KEY(sale_date, customer_id, saler_id, sku_id, category_id)
DISTRIBUTED BY HASH(saler_id) BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"bloom_filter_columns"="saler_id,category_id"
);
```

N-Gram BloomFilter Index

Indexing Principles

n-gram tokenization is a method of splitting a sentence or a piece of text into multiple adjacent word groups. The NGram BloomFilter index, similar to the BloomFilter index, is a skip index based on BloomFilter.

Unlike the BloomFilter index, the NGram BloomFilter index is used to accelerate text LIKE queries. Instead of storing the original text values, it tokenizes the text using NGram and stores each token in the BloomFilter. For LIKE queries, the pattern in LIKE '%pattern%' is also tokenized using NGram. Each token is checked against the BloomFilter, and if any token is not found, the corresponding data block does not meet the LIKE condition and can be skipped, reducing IO and accelerating the query.

Use Cases

The NGram BloomFilter index can only accelerate string LIKE queries, and the number of consecutive characters in the LIKE pattern must be greater than or equal to the N defined in the NGram index.

> **- NGram BloomFilter only supports string columns and can only accelerate LIKE queries.**
>
> - NGram BloomFilter index and BloomFilter index are mutually exclusive, meaning a column can only have one or the other.
> - The performance analysis of the NGram BloomFilter index is similar to that of the BloomFilter index.

Managing Indexes

Creating an NGram BloomFilter Index

The index definition follows the COLUMN definition in the CREATE TABLE statement:

```
INDEX `idx_column_name` (`column_name`) USING NGRAM_BF PROPERTIES("gram_size"="3", "bf_size"="1024") COMMENT 'username ngram_bf
index'
```

Explanation of the syntax:

1. `idx_column_name(column_name)` is mandatory. `column_name` is the column to be indexed and must appear in the column definitions above. `idx_column_name` is the index name, which must be unique at the table level. It is recommended to name it with a prefix `idx_` followed by the column name.
2. `USING NGRAM_BF` is mandatory and specifies that the index type is an NGram BloomFilter index.
3. `PROPERTIES` is optional and is used to specify additional properties for the NGram BloomFilter index. The supported properties are:

- **gram_size**: The N in NGram, specifying the number of consecutive characters to form a token. For example, 'This is a simple ngram example' with N = 3 would be tokenized into 'This is a', 'is a simple', 'a simple ngram', 'simple ngram example' (4 tokens).
- **bf_size**: The size of the BloomFilter in bits. bf_size determines the size of the index corresponding to each data block. The larger this value, the more storage space it occupies, but the lower the probability of hash collisions.

It is recommended to set **gram_size** to the minimum length of the string in LIKE queries but not less than 2. Generally, "gram_size"="3", "bf_size"="1024" is recommended, then adjust based on the Query Profile.

4. `COMMENT` is optional and specifies an index comment.

Viewing NGram BloomFilter Index

– Syntax 1: The INDEX section in the table schema with USING NGRAM_BF indicates an inverted index

```
SHOW CREATE TABLE table_name;
```

– Syntax 2: IndexType as NGRAM_BF indicates an inverted index

```
SHOW INDEX FROM table_name;
```

Deleting an NGram BloomFilter Index

```
ALTER TABLE table_ngrambf DROP INDEX idx_ngrambf;
```

Modifying an NGram BloomFilter Index

```
CREATE INDEX idx_column_name2(column_name2) ON table_ngrambf USING NGRAM_BF PROPERTIES("gram_size"="3", "bf_size"="1024") COMMENT
'username ngram_bf index';

ALTER TABLE table_ngrambf ADD INDEX idx_column_name2(column_name2) USING NGRAM_BF PROPERTIES("gram_size"="3", "bf_size"="1024")
COMMENT 'username ngram_bf index';
```

Using Indexes

To use NGram BloomFilter index, you need to set the following parameters (enable_function_pushdown is false by default):

```
SET enable_function_pushdown = true;
```

NGram BloomFilter index is used to accelerate LIKE queries, for example: SELECT count() FROM table1 WHERE message LIKE '%error%';

The acceleration effect of the BloomFilter index (including NGram) can be analyzed using the following metrics in the Query Profile: - RowsBloomFilterFiltered: The number of rows filtered by the BloomFilter index, which

can be compared with other Rows values to analyze the filtering effect of the index. - BlockConditionsFilteredBloomFilterTime: The time consumed by the BloomFilter inverted index.

Usage Example

This section demonstrates the usage and effectiveness of the NGram BloomFilter index using a dataset of Amazon product reviews, `amazon_reviews`.

Table Creation

```
CREATE TABLE `amazon_reviews` (
  `review_date` int(11) NULL,
  `marketplace` varchar(20) NULL,
  `customer_id` bigint(20) NULL,
  `review_id` varchar(40) NULL,
  `product_id` varchar(10) NULL,
  `product_parent` bigint(20) NULL,
  `product_title` varchar(500) NULL,
  `product_category` varchar(50) NULL,
  `star_rating` smallint(6) NULL,
  `helpful_votes` int(11) NULL,
  `total_votes` int(11) NULL,
  `vine` boolean NULL,
  `verified_purchase` boolean NULL,
  `review_headline` varchar(500) NULL,
  `review_body` string NULL
) ENGINE=OLAP
DUPLICATE KEY(`review_date`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`review_date`) BUCKETS 16
PROPERTIES (
  "replication_allocation" = "tag.location.default: 1",
  "compression" = "ZSTD"
);
```

Data Import

Download the dataset using wget or other tools from the following URLs:

```
https://datasets-documentation.s3.eu-west-3.amazonaws.com/amazon_reviews/amazon_reviews_2010.snappy.parquet
https://datasets-documentation.s3.eu-west-3.amazonaws.com/amazon_reviews/amazon_reviews_2011.snappy.parquet
https://datasets-documentation.s3.eu-west-3.amazonaws.com/amazon_reviews/amazon_reviews_2012.snappy.parquet
https://datasets-documentation.s3.eu-west-3.amazonaws.com/amazon_reviews/amazon_reviews_2013.snappy.parquet
https://datasets-documentation.s3.eu-west-3.amazonaws.com/amazon_reviews/amazon_reviews_2014.snappy.parquet
https://datasets-documentation.s3.eu-west-3.amazonaws.com/amazon_reviews/amazon_reviews_2015.snappy.parquet
```

Import the data using stream load:

```
curl --location-trusted -u root: -T amazon_reviews_2010.snappy.parquet -H "format:parquet"
http://127.0.0.1:8030/api/${DB}/amazon_reviews/_stream_load
curl --location-trusted -u root: -T amazon_reviews_2011.snappy.parquet -H "format:parquet"
http://127.0.0.1:8030/api/${DB}/amazon_reviews/_stream_load
curl --location-trusted -u root: -T amazon_reviews_2012.snappy.parquet -H "format:parquet"
http://127.0.0.1:8030/api/${DB}/amazon_reviews/_stream_load
curl --location-trusted -u root: -T amazon_reviews_2013.snappy.parquet -H "format:parquet"
http://127.0.0.1:8030/api/${DB}/amazon_reviews/_stream_load
```

```
curl --location-trusted -u root: -T amazon_reviews_2014.snappy.parquet -H "format:parquet"
http://127.0.0.1:8030/api/${DB}/amazon_reviews/_stream_load
curl --location-trusted -u root: -T amazon_reviews_2015.snappy.parquet -H "format:parquet"
http://127.0.0.1:8030/api/${DB}/amazon_reviews/_stream_load
```

> **The data file may exceed 10 GB, and you may need to adjust the streaming_road_max_mb in be.conf to prevent exceeding the upload size limit of the stream load. You can dynamically adjust it by following the steps below:**
>
> ```
> curl -X POST http://{be_ip}:{be_http_port}/api/update_config?streaming_load_max_mb=32768
> ```
>
> Every BE needs to execute the above command.

Run a count query to confirm successful data import:

```
mysql> SELECT COUNT(*) FROM amazon_reviews;
+-----------+
| count(*)  |
+-----------+
| 135589433 |
+-----------+
```

Querying

First, run the query without any index. The WHERE clause contains a LIKE condition, and the query takes 7.60 seconds:

```
SELECT
    product_id,
    any_value(product_title),
    AVG(star_rating) AS rating,
    COUNT(*) AS count
FROM
    amazon_reviews
WHERE
    review_body LIKE '%is super awesome%'
GROUP BY
    product_id
ORDER BY
    count DESC,
    rating DESC,
    product_id
LIMIT 5;
```

Results:

```
+------------+----------------------------------------+--------------------+-------+
| product_id | any_value(product_title)               | rating             | count |
+------------+----------------------------------------+--------------------+-------+
| B00992CF6W | Minecraft                              | 4.8235294117647056 |    17 |
| B009UX2YAC | Subway Surfers                         | 4.7777777777777777 |     9 |
| B00DJFIMW6 | Minion Rush: Despicable Me Official Game |              4.875 |     8 |
| B00867O0CM | Temple Run                             |                  5 |     6 |
```

```
| B00KWVZ75O | Angry Birds Epic RPG                    |                5 |     6 |
+------------+-----------------------------------------+------------------+-------+
5 rows in set (7.60 sec)
```

Next, add an NGram BloomFilter index and run the same query again. The query takes 0.93 seconds, an 8x performance improvement:

```
ALTER TABLE amazon_reviews ADD INDEX review_body_ngram_idx(review_body) USING NGRAM_BF PROPERTIES("gram_size"="10",
"bf_size"="10240");
```

```
+------------+-----------------------------------------+-------------------+-------+
| product_id | any_value(product_title)                | rating            | count |
+------------+-----------------------------------------+-------------------+-------+
| B00992CF6W | Minecraft                               | 4.8235294117647056 |    17 |
| B009UX2YAC | Subway Surfers                          | 4.7777777777777777 |     9 |
| B00DJFIMW6 | Minion Rush: Despicable Me Official Game |              4.875 |     8 |
| B00867O0CM | Temple Run                              |                5 |     6 |
| B00KWVZ75O | Angry Birds Epic RPG                    |                5 |     6 |
+------------+-----------------------------------------+-------------------+-------+
5 rows in set (0.93 sec)
```

Schema Change

Users can modify the schema of Doris tables through the Alter Table operation. Schema changes mainly involve column modifications and index changes. This article mainly introduces column-related schema changes; for index-related changes, please refer to Table Index to understand the different methods of changing indexes.

Principles Introduction

Doris supports two types of schema change operations: lightweight schema change and heavyweight schema change. The differences mainly lie in the complexity of the execution process, execution speed, and resource consumption.

Feature	Lightweight Schema Change	Heavyweight Schema Change
Execution Speed	Seconds (almost real-time)	Minutes, hours, days (depends on the amount of data in the table; the larger the data, the slower the execution)
Data Rewrite Needed	No	Yes, involves rewriting data files
System Performance Impact	Minimal	May impact system performance, especially during data conversion
Resource Consumption	Low	High, will consume computing resources to reorganize data, and the storage space occupied by the table's data involved in the process will double.
Operation Types	Add, delete value columns, rename columns, modify VARCHAR length	Modify column data types, change primary keys, modify column order, etc.

Lightweight Schema Change

Lightweight schema change refers to simple schema modification operations that do not involve data rewriting. These operations are usually performed at the metadata level and only require modifying the table's metadata without involving physical modifications to data files. Lightweight schema change operations can typically be completed in seconds and do not significantly impact system performance. Lightweight schema changes include:

- Adding or deleting value columns
- Renaming columns
- Modifying the length of VARCHAR columns (except for UNIQUE and DUP table key columns).

Heavyweight Schema Change

Heavyweight schema change involves rewriting or converting data files, and these operations are relatively complex, usually requiring the assistance of Doris's Backend (BE) to perform actual data modifications or reorganizations. Heavyweight schema change operations typically involve deep changes to the table's data

structure and may affect the physical layout of storage. All operations that do not support lightweight schema changes fall under heavyweight schema changes, such as:

- Changing the data type of a column
- Modifying the order of columns

Heavyweight operations will start a task in the background for data conversion. The background task will convert each tablet of the table, rewriting the original data into new data files on a tablet basis. During the data conversion process, a “double write” phenomenon may occur, where new data is simultaneously written to both the new tablet and the old tablet. After the data conversion is complete, the old tablet will be deleted, and the new tablet will replace it.

Job Management

View Jobs

Users can view the progress of schema change jobs through the `SHOW ALTER TABLE COLUMN` command. This command allows users to see the currently executing or completed schema change jobs. When a schema change job involves materialized views, this command will display multiple rows, each corresponding to a materialized view. An example is as follows:

```
mysql > SHOW ALTER TABLE COLUMN\G;
*************************** 1. row ***************************
        JobId: 20021
    TableName: tbl1
   CreateTime: 2019-08-05 23:03:13
   FinishTime: 2019-08-05 23:03:42
    IndexName: tbl1
      IndexId: 20022
OriginIndexId: 20017
SchemaVersion: 2:792557838
TransactionId: 10023
        State: FINISHED
          Msg:
     Progress: NULL
      Timeout: 86400
1 row in set (0.00 sec)
```

Cancel Jobs

If the job status is not FINISHED or CANCELLED, you can cancel the schema change job using the following command:

```
CANCEL ALTER TABLE COLUMN FROM tbl_name;
```

Usage Examples

Rename Column

```
ALTER TABLE [database.]table RENAME COLUMN old_column_name new_column_name;
```

For specific syntax, refer to ALTER TABLE RENAME.

Add a Column

- If an aggregate model adds a value column, the `agg_type` must be specified.
- If a non-aggregate model (such as DUPLICATE KEY) adds a key column, the KEY keyword must be specified.

Adding a column to a non-aggregate table

1. Create table statement

```
CREATE TABLE IF NOT EXISTS example_db.my_table(
    col1 int,
    col2 int,
    col3 int,
    col4 int,
    col5 int
) DUPLICATE KEY(col1, col2, col3)
DISTRIBUTED BY RANDOM BUCKETS 10;
```

2. Add a key column `key_col` after `col1` in `example_db.my_table`

```
ALTER TABLE example_db.my_table ADD COLUMN key_col INT KEY DEFAULT "0" AFTER col1;
```

3. Add a value column `value_col` after `col4` in `example_db.my_table`

```
ALTER TABLE example_db.my_table ADD COLUMN value_col INT DEFAULT "0" AFTER col4;
```

Adding a column to an aggregate table

1. Create table statement

```
CREATE TABLE IF NOT EXISTS example_db.my_table(
    col1 int,
    col2 int,
    col3 int,
    col4 int SUM,
    col5 varchar(32) REPLACE DEFAULT "abc"
) AGGREGATE KEY(col1, col2, col3)
DISTRIBUTED BY HASH(col1) BUCKETS 10;
```

2. Add a key column `key_col` after `col1` in `example_db.my_table`

```
ALTER TABLE example_db.my_table ADD COLUMN key_col INT DEFAULT "0" AFTER col1;
```

3. Add a value column `value_col` of SUM aggregation type after `col4` in `example_db.my_table`

```
ALTER TABLE example_db.my_table ADD COLUMN value_col INT SUM DEFAULT "0" AFTER col4;
```

Add Multiple Columns

- If an aggregate model adds a value column, the `agg_type` must be specified.
- If an aggregate model adds a key column, the KEY keyword must be specified.

Adding multiple columns to an aggregate table

1. Create table statement

```
CREATE TABLE IF NOT EXISTS example_db.my_table(
    col1 int,
    col2 int,
    col3 int,
    col4 int SUM,
    col5 varchar(32) REPLACE DEFAULT "abc"
) AGGREGATE KEY(col1, col2, col3)
DISTRIBUTED BY HASH(col1) BUCKETS 10;
```

2. Add multiple columns to `example_db.my_table` (aggregate model)

```
ALTER TABLE example_db.my_table
ADD COLUMN (c1 INT DEFAULT "1", c2 FLOAT SUM DEFAULT "0");
```

Delete Column

- Partition columns cannot be deleted.
- UNIQUE key columns cannot be deleted.

To delete a column from `example_db.my_table`

1. Create table statement

```
CREATE TABLE IF NOT EXISTS example_db.my_table(
    col1 int,
    col2 int,
    col3 int,
    col4 int SUM,
    col5 varchar(32) REPLACE DEFAULT "abc"
) AGGREGATE KEY(col1, col2, col3)
DISTRIBUTED BY HASH(col1) BUCKETS 10;
```

2. Delete the `col4` column from `example_db.my_table`

```
ALTER TABLE example_db.my_table DROP COLUMN col4;
```

Modify Column Type and Position

- If an aggregate model modifies a value column, the `agg_type` must be specified.
- If a non-aggregate type modifies a key column, the **KEY** keyword must be specified.

- Only the type of the column can be modified; other attributes of the column must remain the same.
- Partition columns and bucket columns cannot be modified.
- The following type conversions are currently supported (users need to be aware of precision loss):
    - TINYINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE types can be converted to larger numeric types.
    - TINTINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE/DECIMAL can be converted to VARCHAR.
    - VARCHAR supports modifying the maximum length.
    - VARCHAR/CHAR can be converted to TINTINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE.
    - VARCHAR/CHAR can be converted to DATE (currently supports six formats: “%Y-%m-%d”, “%y-%m-%d”, “%Y%m%d”, “%y%m%d”, “%Y/%m/%d”, “%y/%m/%d”).
    - DATETIME can be converted to DATE (only retains year-month-day information, e.g., `2019-12-09 21:47:05` <–> `2019-12-09`).
    - DATE can be converted to DATETIME (hours, minutes, and seconds are automatically set to zero, e.g., `2019-12-09` <–> `2019-12-09 00:00:00`).
    - FLOAT can be converted to DOUBLE.
    - INT can be converted to DATE (if the INT type data is invalid, the conversion fails, and the original data remains unchanged).
    - All types except DATE and DATETIME can be converted to STRING, but STRING cannot be converted to any other type.

1. Create table statement

```
CREATE TABLE IF NOT EXISTS example_db.my_table(
    col0 int,
    col1 int DEFAULT "1",
    col2 int,
    col3 varchar(32),
    col4 int SUM,
    col5 varchar(32) REPLACE DEFAULT "abc"
) AGGREGATE KEY(col0, col1, col2, col3)
DISTRIBUTED BY HASH(col0) BUCKETS 10;
```

2. Modify the type of key column `col1` to BIGINT and move it after column `col2`

```
ALTER TABLE example_db.my_table
MODIFY COLUMN col1 BIGINT KEY DEFAULT "1" AFTER col2;
```

Note: Whether modifying a key column or a value column, the complete column information must be declared.

3. Modify the maximum length of the `val5` column in the base table. The original `val5` was (val5 VARCHAR(32) REPLACE DEFAULT “abc”)

```
ALTER TABLE example_db.my_table
MODIFY COLUMN col5 VARCHAR(64) REPLACE DEFAULT "abc";
```

Note: Only the type of the column can be modified; other attributes of the column must remain the same.

4. Modify the length of a field in a key column

```
ALTER TABLE example_db.my_table
MODIFY COLUMN col3 varchar(50) KEY NULL comment 'to 50';
```

Reorder

- All columns must be listed.
- Value columns must be after key columns.

1. Create table statement

```
CREATE TABLE IF NOT EXISTS example_db.my_table(
    k1 int DEFAULT "1",
    k2 int,
    k3 varchar(32),
    k4 date,
    v1 int SUM,
    v2 int MAX,
) AGGREGATE KEY(k1, k2, k3, k4)
DISTRIBUTED BY HASH(k1) BUCKETS 10;
```

2. Reorder the columns in `example_db.my_table`

```
ALTER TABLE example_db.my_table
ORDER BY (k3,k1,k2,k4,v2,v1);
```

Limitations

- A table can only have one schema change job running at the same time.
- Partition columns and bucket columns cannot be modified.
- If an aggregate table has value columns aggregated using the REPLACE method, key columns cannot be deleted.
- Unique tables cannot delete key columns.
- When adding value columns with aggregation types of SUM or REPLACE, the default value of that column has no meaning for historical data.
- Because historical data has lost detailed information, the value of the default cannot actually reflect the aggregated value.

- When modifying column types, all fields except Type must be supplemented with the original column's information.
- Note that, except for the new column type, aggregation method, Nullable attribute, and default value must be supplemented according to the original information.
- Modifying aggregation types, Nullable attributes, and default values is not supported.

Related Configurations

FE Configuration

- `alter_table_timeout_second`: The default timeout for jobs, 86400 seconds.

BE Configuration

- `alter_tablet_worker_count`: The number of threads used on the BE side to execute historical data conversion. The default is 3. If you want to speed up schema change jobs, you can appropriately increase this parameter and restart BE. However, too many conversion threads may increase IO pressure and affect other operations.
- `alter_index_worker_count`: The number of threads used on the BE side to execute historical data index building (Note: currently only supports inverted indexes). The default is 3. If you want to speed up index change jobs, you can appropriately increase this parameter and restart BE. However, too many threads may increase IO pressure and affect other operations.

Auto-Increment Column

In Doris, the auto increment column is a feature that automatically generates a unique numeric value, commonly used to create unique identifiers for each row of data, such as primary keys. Each time a new record is inserted, the auto increment column automatically assigns an incrementing value, eliminating the need for manually specifying the number. By using Doris's auto increment column, data uniqueness and consistency are ensured, simplifying the data insertion process, reducing human error, and improving data management efficiency. This makes the auto increment column an ideal choice for scenarios requiring unique identifiers, such as user IDs and more.

Functionality

For tables with an auto-increment column, Doris processes data writes as follows:

- **Auto-Population (Column Excluded)**: If the written data does not include the auto-increment column, Doris generates and populates unique values for this column.
- **Partial Specification (Column Included)**:
    - **Null Values**: Doris replaces null values in the written data with system-generated unique values.
    - **Non-Null Values**: User-provided values remain unchanged.

  :::caution Attention User-provided non-null values can disrupt the uniqueness of the auto-increment column. :::

Uniqueness

Doris guarantees **table-wide uniqueness** for values it generates in the auto-increment column. However:

- **Guaranteed Uniqueness**: This applies only to system-generated values.
- **User-Provided Values**: Doris does not validate or enforce uniqueness for values specified by users in the auto-increment column. This may result in duplicate entries.

Density

Auto-increment values generated by Doris are generally **dense** but with some considerations:

- **Potential Gaps**: Gaps may appear due to performance optimizations. Each backend node (BE) pre-allocates a block of unique values for efficiency, and these blocks do not overlap between nodes.
- **Non-Chronological Values**: Doris does not guarantee that values generated in later writes are larger than those from earlier writes.

  :::info Note Auto-increment values cannot be used to infer the chronological order of writes. :::

Syntax

To use auto-increment columns, you need to add the `AUTO_INCREMENT` attribute to the corresponding column during table creation (CREATE-TABLE). To manually specify the starting value for an auto-increment column, you can do so by using the `AUTO_INCREMENT(start_value)` statement when creating the table. If not specified, the default starting value is 1.

Examples

1. Creating a duplicate table with an auto-increment column as the key column.

```
CREATE TABLE `demo`.`tbl` (
        `id` BIGINT NOT NULL AUTO_INCREMENT,
        `value` BIGINT NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);

2. Creating a duplicate table with an auto-increment column as the key column, and setting the starting value to 100.

```sql
CREATE TABLE `demo`.`tbl` (
      `id` BIGINT NOT NULL AUTO_INCREMENT(100),
      `value` BIGINT NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);
```

3. Creating a duplicate table with an auto-increment column as one of the value columns.

```
CREATE TABLE `demo`.`tbl` (
    `uid` BIGINT NOT NULL,
    `name` BIGINT NOT NULL,
    `id` BIGINT NOT NULL AUTO_INCREMENT,
    `value` BIGINT NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(`uid`, `name`)
DISTRIBUTED BY HASH(`uid`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);
```

4. Creating a unique table with an auto-increment column as the key column.

```
CREATE TABLE `demo`.`tbl` (
      `id` BIGINT NOT NULL AUTO_INCREMENT,
      `name` varchar(65533) NOT NULL,
      `value` int(11) NOT NULL
) ENGINE=OLAP
UNIQUE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
```
```

```
PROPERTIES (
"replication_allocation" = "tag.location.default: 3",
"enable_unique_key_merge_on_write" = "true"
);
```

5. Creating a unique table with an auto-increment column as one of the value columns.

```
CREATE TABLE `demo`.`tbl` (
      `text` varchar(65533) NOT NULL,
      `id` BIGINT NOT NULL AUTO_INCREMENT,
) ENGINE=OLAP
UNIQUE KEY(`text`)
DISTRIBUTED BY HASH(`text`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 3",
"enable_unique_key_merge_on_write" = "true"
);
```

Constraints and Limitations

- Auto-increment columns can only be used in Duplicate or Unique model tables.
- A table can have only one auto-increment column.
- The auto-increment column must be of type `BIGINT` and cannot be `NULL`.
- The manually specified starting value for an auto-increment column must be 0 or greater.

Usage

Loading

Consider the table below:

```
CREATE TABLE `demo`.`tbl` (
    `id` BIGINT NOT NULL AUTO_INCREMENT,
    `name` varchar(65533) NOT NULL,
    `value` int(11) NOT NULL
) ENGINE=OLAP
UNIQUE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 3",
"enable_unique_key_merge_on_write" = "true"
);
```

When using the insert into statement to write data without including the auto-increment column `id`, Doris automatically generates and fills unique values for the column.

```
insert into tbl(name, value) values("Bob", 10), ("Alice", 20), ("Jack", 30);

select * from tbl order by id;
+------+-------+-------+
| id   | name  | value |
+------+-------+-------+
|    1 | Bob   |    10 |
|    2 | Alice |    20 |
```

```
|    3 | Jack  |    30 |
+------+-------+-------+
```

Similarly, when using stream load to load the file `test.csv` without specifying the auto-increment column `id`, Doris will automatically populate the `id` column with generated values.

test.csv:

```
Tom, 40
John, 50
```

```
curl --location-trusted -u user:passwd -H "columns:name,value" -H "column_separator:," -T ./test1.csv http://{host}:
{port}/api/{db}/tbl/_stream_load
```

```
select * from tbl order by id;
+------+-------+-------+
| id   | name  | value |
+------+-------+-------+
|    1 | Bob   |    10 |
|    2 | Alice |    20 |
|    3 | Jack  |    30 |
|    4 | Tom   |    40 |
|    5 | John  |    50 |
+------+-------+-------+
```

When writing data using the `INSERT INTO` statement and specifying the auto-increment column `id`, any null values in the written data for that column will be replaced with generated values.

```
insert into tbl(id, name, value) values(null, "Doris", 60), (null, "Nereids", 70);

select * from tbl order by id;
+------+---------+-------+
| id   | name    | value |
+------+---------+-------+
|    1 | Bob     |    10 |
|    2 | Alice   |    20 |
|    3 | Jack    |    30 |
|    4 | Tom     |    40 |
|    5 | John    |    50 |
|    6 | Doris   |    60 |
|    7 | Nereids |    70 |
+------+---------+-------+
```

Partial Update

When performing a partial update on a merge-on-write Unique table with an auto-increment column:

If the auto-increment column is a key column, users must explicitly specify it during partial updates. As a result, the target columns for partial updates must include the auto-increment column. In this case, the behavior aligns with that of standard partial updates.

```
CREATE TABLE `demo`.`tbl2` (
  `id` BIGINT NOT NULL AUTO_INCREMENT,
  `name` varchar(65533) NOT NULL,
```

```
  `value` int(11) NOT NULL DEFAULT "0"
  ) ENGINE=OLAP
  UNIQUE KEY(`id`)
  DISTRIBUTED BY HASH(`id`) BUCKETS 10
  PROPERTIES (
  "replication_allocation" = "tag.location.default: 3",
  "enable_unique_key_merge_on_write" = "true"
  );

insert into tbl2(id, name, value) values(1, "Bob", 10), (2, "Alice", 20), (3, "Jack", 30);

select * from tbl2 order by id;
+------+-------+-------+
| id   | name  | value |
+------+-------+-------+
|    1 | Bob   |    10 |
|    2 | Alice |    20 |
|    3 | Jack  |    30 |
+------+-------+-------+

set enable_unique_key_partial_update=true;
set enable_insert_strict=false;
insert into tbl2(id, name) values(1, "modified"), (4, "added");

select * from tbl2 order by id;
+------+----------+-------+
| id   | name     | value |
+------+----------+-------+
|    1 | modified |    10 |
|    2 | Alice    |    20 |
|    3 | Jack     |    30 |
|    4 | added    |     0 |
+------+----------+-------+
```

When the auto-increment column is a non-key column and no value is provided, its value will be derived from existing rows in the table. If a value is specified for the auto-increment column, null values in the written data will be replaced with generated values, while non-null values will remain unchanged. These records will then be processed according to the semantics of partial updates.

```
CREATE TABLE `demo`.`tbl3` (
  `id` BIGINT NOT NULL,
  `name` varchar(100) NOT NULL,
  `score` BIGINT NOT NULL,
  `aid` BIGINT NOT NULL AUTO_INCREMENT
  ) ENGINE=OLAP
  UNIQUE KEY(`id`)
  DISTRIBUTED BY HASH(`id`) BUCKETS 1
  PROPERTIES (
  "replication_allocation" = "tag.location.default: 3",
  "enable_unique_key_merge_on_write" = "true"
  );


insert into tbl3(id, name, score) values(1, "Doris", 100), (2, "Nereids", 200), (3, "Bob", 300);

select * from tbl3 order by id;
+------+---------+-------+------+
| id   | name    | score | aid  |
+------+---------+-------+------+
|    1 | Doris   |   100 |    0 |
|    2 | Nereids |   200 |    1 |
|    3 | Bob     |   300 |    2 |
```

```
+------+---------+-------+------+

set enable_unique_key_partial_update=true;
set enable_insert_strict=false;
insert into tbl3(id, score) values(1, 999), (2, 888);


select * from tbl3 order by id;
+------+---------+-------+------+
| id   | name    | score | aid  |
+------+---------+-------+------+
|    1 | Doris   |   999 |    0 |
|    2 | Nereids |   888 |    1 |
|    3 | Bob     |   300 |    2 |
+------+---------+-------+------+


insert into tbl3(id, aid) values(1, 1000), (3, 500);

select * from tbl3 order by id;
+------+---------+-------+------+
| id   | name    | score | aid  |
+------+---------+-------+------+
|    1 | Doris   |   999 | 1000 |
|    2 | Nereids |   888 |    1 |
|    3 | Bob     |   300 |  500 |
+------+---------+-------+------+
```

Usage Scenarios

Dictionary Encoding

Using bitmaps for audience analysis in user profiling involves creating a user dictionary, where each user is assigned a unique integer as their dictionary value. Aggregating these dictionary values can improve the performance of bitmap operations.

For example, in an offline UV (Unique Visitors) and PV (Page Views) analysis scenario, consider a detailed user behavior table:

```
CREATE TABLE `demo`.`dwd_dup_tbl` (
    `user_id` varchar(50) NOT NULL,
    `dim1` varchar(50) NOT NULL,
    `dim2` varchar(50) NOT NULL,
    `dim3` varchar(50) NOT NULL,
    `dim4` varchar(50) NOT NULL,
    `dim5` varchar(50) NOT NULL,
    `visit_time` DATE NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(`user_id`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 32
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);
```

Using the auto-increment column to create the following dictionary table:

```
CREATE TABLE `demo`.`dictionary_tbl` (
    `user_id` varchar(50) NOT NULL,
```

```
    `aid` BIGINT NOT NULL AUTO_INCREMENT
) ENGINE=OLAP
UNIQUE KEY(`user_id`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 32
PROPERTIES (
"replication_allocation" = "tag.location.default: 3",
"enable_unique_key_merge_on_write" = "true"
);
```

Write the `user_id` values from existing data into the dictionary table to map `user_id` to corresponding integer values:

```
insert into dictionary_tbl(user_id)
select user_id from dwd_dup_tbl group by user_id;
```

Alternatively, write only the `user_id` values from incremental data into the dictionary table.

```
insert into dictionary_tbl(user_id)
select dwd_dup_tbl.user_id from dwd_dup_tbl left join dictionary_tbl
on dwd_dup_tbl.user_id = dictionary_tbl.user_id where dwd_dup_tbl.visit_time > '2023-12-10' and dictionary_tbl.user_id is NULL;
```

In practical applications, Flink connectors can be used to write data into Doris.

To store aggregated results for the statistical dimensions `dim1`, `dim3`, and `dim5`, create the following table:

```
CREATE TABLE `demo`.`dws_agg_tbl` (
    `dim1` varchar(50) NOT NULL,
    `dim3` varchar(50) NOT NULL,
    `dim5` varchar(50) NOT NULL,
    `user_id_bitmap` BITMAP BITMAP_UNION NOT NULL,
    `pv` BIGINT SUM NOT NULL
) ENGINE=OLAP
AGGREGATE KEY(`dim1`,`dim3`,`dim5`)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 32
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);
```

Save the aggregated data into the results table.

```
insert into dws_agg_tbl
select dwd_dup_tbl.dim1, dwd_dup_tbl.dim3, dwd_dup_tbl.dim5, BITMAP_UNION(TO_BITMAP(dictionary_tbl.aid)), COUNT(1)
from dwd_dup_tbl INNER JOIN dictionary_tbl on dwd_dup_tbl.user_id = dictionary_tbl.user_id;
```

Execute UV and PV queries with the following statement:

```
select dim1, dim3, dim5, user_id_bitmap as uv, pv from dws_agg_tbl;
```

Efficient Pagination

Pagination is often required when displaying data on a page. Traditional pagination usually involves using `LIMIT`, `OFFSET`, and `ORDER BY` in SQL queries. For example, consider the following business table designed for display:

```
CREATE TABLE `demo`.`records_tbl` (
    `key` int(11) NOT NULL COMMENT "",
    `name` varchar(26) NOT NULL COMMENT "",
    `address` varchar(41) NOT NULL COMMENT "",
    `city` varchar(11) NOT NULL COMMENT "",
    `nation` varchar(16) NOT NULL COMMENT "",
    `region` varchar(13) NOT NULL COMMENT "",
    `phone` varchar(16) NOT NULL COMMENT "",
    `mktsegment` varchar(11) NOT NULL COMMENT ""
) DUPLICATE KEY (`key`, `name`)
DISTRIBUTED BY HASH(`key`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);
```

Assuming 100 records are displayed per page, the following SQL query can be used to fetch data for the first page:

```
select * from records_tbl order by `key`, `name` limit 100;
```

To fetch data for the second page, you can use the following query:

```
select * from records_tbl order by `key`, `name` limit 100 offset 100;
```

However, when performing deep pagination queries (with large offsets), this method can be inefficient, as it reads all data into memory for sorting before processing, even if only a small number of rows are needed. By using an auto-increment column, each row is assigned a unique value, enabling the use of a query like `WHERE unique_value > x LIMIT y` to filter out a large portion of the data in advance, making pagination more efficient.

To illustrate this, an auto-increment column is added to the business table, giving each row a unique identifier:

```
CREATE TABLE `demo`.`records_tbl2` (
    `key` int(11) NOT NULL COMMENT "",
    `name` varchar(26) NOT NULL COMMENT "",
    `address` varchar(41) NOT NULL COMMENT "",
    `city` varchar(11) NOT NULL COMMENT "",
    `nation` varchar(16) NOT NULL COMMENT "",
    `region` varchar(13) NOT NULL COMMENT "",
    `phone` varchar(16) NOT NULL COMMENT "",
    `mktsegment` varchar(11) NOT NULL COMMENT "",
    `unique_value` BIGINT NOT NULL AUTO_INCREMENT
) DUPLICATE KEY (`key`, `name`)
DISTRIBUTED BY HASH(`key`) BUCKETS 10
PROPERTIES (
    "replication_num" = "3"
);
```

For pagination with 100 records per page, the following SQL query can be used to fetch the data for the first page:

```
select * from records_tbl2 order by unique_value limit 100;
```

By recording the maximum value of `unique_value` from the returned results, let's assume it is 99. The following query can then be used to fetch data for the second page:

```
select * from records_tbl2 where unique_value > 99 order by unique_value limit 100;
```

If directly querying data from a later page and it's inconvenient to retrieve the maximum value of `unique_value` from the previous page's results (for example, when fetching data starting from the 101st page), the following query can be used:

```
select key, name, address, city, nation, region, phone, mktsegment
from records_tbl2, (select unique_value as max_value from records_tbl2 order by unique_value limit 1 offset 9999) as previous_data
where records_tbl2.unique_value > previous_data.max_value
order by records_tbl2.unique_value limit 100;
```

Tiered Storage Overview

To help users reduce storage costs, Doris provides flexible options for cold data management.

Cold Data Options	Applicable Conditions	Features
Compute-Storage Separation	Users have the capability to deploy a compute-storage separation setup	- Data is stored as a single replica in object storage - Local caching accelerates hot data access - Independent scaling of storage and compute resources significantly reduces costs
Local Tiering	In the compute-storage integrated mode, users want to further optimize local storage resources	- Supports cooling cold data from SSD to HDD - Fully utilizes the tiered characteristics of local storage to save high-performance storage costs
Remote Tiering	In the compute-storage integrated mode, users want to reduce costs using affordable object storage or HDFS	- Cold data is stored as a single replica in object storage or HDFS - Hot data continues to use local storage - Cannot be combined with local tiering for the same table

With the above options, Doris can flexibly adapt to different deployment scenarios, achieving a balance between query efficiency and storage cost.

Tiered Storage of SSD and HDD

Doris supports tiered storage between different disk types (SSD and HDD), combining dynamic partitioning features to dynamically migrate data from SSD to HDD based on the characteristics of hot and cold data. This approach reduces storage costs while maintaining high performance for hot data reads and writes.

Dynamic Partitioning and Tiered Storage

By configuring dynamic partitioning parameters of a table, users can set which partitions are stored on SSD and automatically migrate to HDD after cooling.

- **Hot Partitions**: Recently active partitions, prioritized to be stored on SSD to ensure high performance.
- **Cold Partitions**: Partitions that are accessed less frequently, which will gradually migrate to HDD to reduce storage costs.

For more information on dynamic partitioning, please refer to: Data Partitioning - Dynamic Partitioning.

Parameter Description

`dynamic_partition.hot_partition_num`

- **Function**:
  - Specifies how many of the most recent partitions are hot partitions, which are stored on SSD, while the remaining partitions are stored on HDD.
- **Note**:
  - `"dynamic_partition.storage_medium" = "HDD"` must be set simultaneously; otherwise, this parameter will not take effect.
  - If there are no SSD devices in the storage path, this configuration will cause partition creation to fail.

Example Description:

Assuming the current date is **2021-05-20**, with daily partitioning, the dynamic partitioning configuration is as follows:

```
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.hot_partition_num" = 2
"dynamic_partition.start" = -3
"dynamic_partition.end" = 3
```

The system will automatically create the following partitions and configure their storage medium and cooling time:

```
p20210517: ["2021-05-17", "2021-05-18") storage_medium=HDD storage_cooldown_time=9999-12-31 23:59:59
p20210518: ["2021-05-18", "2021-05-19") storage_medium=HDD storage_cooldown_time=9999-12-31 23:59:59
p20210519: ["2021-05-19", "2021-05-20") storage_medium=SSD storage_cooldown_time=2021-05-21 00:00:00
p20210520: ["2021-05-20", "2021-05-21") storage_medium=SSD storage_cooldown_time=2021-05-22 00:00:00
p20210521: ["2021-05-21", "2021-05-22") storage_medium=SSD storage_cooldown_time=2021-05-23 00:00:00
```

```
p20210522: ["2021-05-22", "2021-05-23") storage_medium=SSD storage_cooldown_time=2021-05-24 00:00:00
p20210523: ["2021-05-23", "2021-05-24") storage_medium=SSD storage_cooldown_time=2021-05-25 00:00:00
```

`dynamic_partition.storage_medium`

- **Function**:
  - Specifies the final storage medium for dynamic partitions. The default is HDD, but SSD can be selected.
- **Note**:
  - When set to SSD, the `hot_partition_num` attribute will no longer take effect, and all partitions will default to SSD storage medium with a cooling time of 9999-12-31 23:59:59.

Example

1. Create a table with dynamic_partition

```
CREATE TABLE tiered_table (k DATE)
PARTITION BY RANGE(k)()
DISTRIBUTED BY HASH (k) BUCKETS 5
PROPERTIES
(
    "dynamic_partition.storage_medium" = "hdd",
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.hot_partition_num" = "2",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "5",
    "dynamic_partition.create_history_partition"= "true",
    "dynamic_partition.start" = "-3"
);
```

2. Check storage medium of partitions

```
SHOW PARTITIONS FROM tiered_table;
```

You should have 7 partitions, 5 of which use SSD as the storage medium, while the other 2 use HDD.

```
p20210517: ["2021-05-17", "2021-05-18") storage_medium=HDD storage_cooldown_time=9999-12-31 23:59:59
p20210518: ["2021-05-18", "2021-05-19") storage_medium=HDD storage_cooldown_time=9999-12-31 23:59:59
p20210519: ["2021-05-19", "2021-05-20") storage_medium=SSD storage_cooldown_time=2021-05-21 00:00:00
p20210520: ["2021-05-20", "2021-05-21") storage_medium=SSD storage_cooldown_time=2021-05-22 00:00:00
p20210521: ["2021-05-21", "2021-05-22") storage_medium=SSD storage_cooldown_time=2021-05-23 00:00:00
p20210522: ["2021-05-22", "2021-05-23") storage_medium=SSD storage_cooldown_time=2021-05-24 00:00:00
p20210523: ["2021-05-23", "2021-05-24") storage_medium=SSD storage_cooldown_time=2021-05-25 00:00:00
```

Remote Storage

Overview

Remote storage supports placing cold data in external storage (such as object storage, HDFS).

> **Note**
>
> The data in remote storage has only one copy, and the reliability of the data depends on the reliability of the remote storage. You need to ensure that the remote storage has erasure coding (EC) or multi-replica technology to ensure data reliability.

Usage

Saving Cold Data to S3 Compatible Storage

Step 1: Create S3 Resource.

```
CREATE RESOURCE "remote_s3"
PROPERTIES
(
    "type" = "s3",
    "s3.endpoint" = "bj.s3.com",
    "s3.region" = "bj",
    "s3.bucket" = "test-bucket",
    "s3.root.path" = "path/to/root",
    "s3.access_key" = "bbb",
    "s3.secret_key" = "aaaa",
    "s3.connection.maximum" = "50",
    "s3.connection.request.timeout" = "3000",
    "s3.connection.timeout" = "1000"
);
```

> **When creating the S3 RESOURCE, a link verification to the S3 remote will be performed to ensure the correctness of the RESOURCE creation.**

Step 2: Create STORAGE POLICY.

Then create a STORAGE POLICY associated with the RESOURCE created above:

```
CREATE STORAGE POLICY test_policy
PROPERTIES(
    "storage_resource" = "remote_s3",
    "cooldown_ttl" = "1d"
);
```

Step 3: Use STORAGE POLICY when creating a table.

```
CREATE TABLE IF NOT EXISTS create_table_use_created_policy
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048)
)
UNIQUE KEY(k1)
DISTRIBUTED BY HASH (k1) BUCKETS 3
PROPERTIES(
    "enable_unique_key_merge_on_write" = "false",
    "storage_policy" = "test_policy"
);
```

> **Note**
>
> If the UNIQUE table is set with `"enable_unique_key_merge_on_write" = "true"`, this feature cannot be used.

Saving Cold Data to HDFS

Step 1: Create HDFS RESOURCE:

```
CREATE RESOURCE "remote_hdfs" PROPERTIES (
        "type"="hdfs",
        "fs.defaultFS"="fs_host:default_fs_port",
        "hadoop.username"="hive",
        "hadoop.password"="hive",
        "root_path"="/my/root/path",
        "dfs.nameservices" = "my_ha",
        "dfs.ha.namenodes.my_ha" = "my_namenode1, my_namenode2",
        "dfs.namenode.rpc-address.my_ha.my_namenode1" = "nn1_host:rpc_port",
        "dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
        "dfs.client.failover.proxy.provider.my_ha" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
    );
```

Step 2: Create STORAGE POLICY.

```
CREATE STORAGE POLICY test_policy PROPERTIES (
    "storage_resource" = "remote_hdfs",
    "cooldown_ttl" = "300"
)
```

Step 3: Use STORAGE POLICY to create a table.

```
CREATE TABLE IF NOT EXISTS create_table_use_created_policy (
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048)
)
UNIQUE KEY(k1)
DISTRIBUTED BY HASH (k1) BUCKETS 3
PROPERTIES(
"enable_unique_key_merge_on_write" = "false",
```

```
"storage_policy" = "test_policy"
);
```

> **Note**
>
> If the UNIQUE table is set with `"enable_unique_key_merge_on_write" = "true"`, this feature cannot be used.

Cooling Existing Tables to Remote Storage

In addition to new tables supporting the setting of remote storage, Doris also supports setting remote storage for an existing table or PARTITION.

For an existing table, set remote storage by associating the created STORAGE POLICY with the table:

```
ALTER TABLE create_table_not_have_policy set ("storage_policy" = "test_policy");
```

For an existing PARTITION, set remote storage by associating the created STORAGE POLICY with the PARTITION:

```
ALTER TABLE create_table_partition MODIFY PARTITION (*) SET("storage_policy"="test_policy");
```

> **Note that if the user specifies different Storage Policies for the entire Table and some Partitions when creating the table, the Storage Policy set for the Partition will be ignored, and all Partitions of the table will use the table's Policy. If you need a Partition's Policy to differ from others, you can modify it using the method described above for associating a Storage Policy with an existing Partition.**
>
> For more details, please refer to the Docs directory under RESOURCE, STORAGE POLICY, CREATE TABLE, ALTER TABLE, etc.

Configuring Compaction

- The BE parameter `cold_data_compaction_thread_num` can set the concurrency for executing remote storage Compaction, with a default of 2.
- The BE parameter `cold_data_compaction_interval_sec` can set the time interval for executing remote storage Compaction, with a default of 1800 seconds, which is half an hour.

Limitations

- Tables using remote storage do not support backup.
- Modifying the location information of remote storage, such as endpoint, bucket, or path, is not supported.
- Unique model tables with Merge-on-Write enabled do not support remote storage.

- Storage policies support creation, modification, and deletion. Before deleting a storage policy, ensure that no tables are referencing it.
- Once a storage policy is set, it cannot be unset.

Cold Data Space

Viewing

Method 1: You can view the size uploaded to the object by each BE through `show proc '/backends'`, in the RemoteUsedCapacity item, this method has a slight delay.

Method 2: You can view the size of each tablet occupied by the table through `show tablets from tableName`, in the RemoteDataSize item.

Garbage Collection

There may be situations that generate garbage data on remote storage:

1. Rowset upload fails but some segments are successfully uploaded.
2. The uploaded rowset did not reach consensus in multiple replicas.
3. Rowsets participating in compaction after compaction is completed.

Garbage data will not be cleaned up immediately. The BE parameter `remove_unused_remote_files_interval_sec` can set the time interval for garbage collection on remote storage, with a default of 21600 seconds, which is 6 hours.

Query and Performance Optimization

To optimize query performance and save object storage resources, local Cache has been introduced. When querying data from remote storage for the first time, Doris will load the data from remote storage to the local disk of the BE for caching. The Cache has the following characteristics:

- The Cache is actually stored on the local disk of the BE and does not occupy memory space.
- The Cache is managed through LRU and does not support TTL.

For specific configurations, please refer to Data Cache.

FAQ

1. `ERROR 1105 (HY000): errCode = 2, detailMessage = Failed to create repository: connect to s3 failed: Unable to marshall request to JSON: host must not be null.`

The S3 SDK defaults to using the virtual-hosted style method. However, some object storage systems (such as MinIO) may not have virtual-hosted style access enabled or supported. In this case, we can add the `use_path_style` parameter to force the use of the path style method:

```
CREATE RESOURCE "remote_s3"
PROPERTIES
(
    "type" = "s3",
    "s3.endpoint" = "bj.s3.com",
    "s3.region" = "bj",
    "s3.bucket" = "test-bucket",
    "s3.root.path" = "path/to/root",
    "s3.access_key" = "bbb",
    "s3.secret_key" = "aaaa",
    "s3.connection.maximum" = "50",
    "s3.connection.request.timeout" = "3000",
    "s3.connection.timeout" = "1000",
    "use_path_style" = "true"
);
```

2. What happens after modifying parameters related to cooldown time?

   Changes to cooldown-related parameters only take effect for data that has not yet been cooled to remote storage. For data that has already been cooled to remote storage, the changes do not apply. For example, if you change `cooldown_ttl` from 21 days to 7 days, data that is already in remote storage will not be moved back to local storage;

Hybrid Row-Columnar Storage

Hybrid Row-Columnar Storage

Doris uses columnar storage by default, with each column stored contiguously. Columnar storage offers excellent performance for analytical scenarios (such as aggregation, filtering, sorting, etc.), as it only reads the necessary columns, reducing unnecessary IO. However, in point query scenarios (such as `SELECT *`), all columns need to be read, requiring an IO operation for each column, which can lead to IOPS becoming a bottleneck, especially for wide tables with many columns (e.g., hundreds of columns).

To address the IOPS bottleneck in point query scenarios, starting from version 2.0.0, Doris supports Hybrid Row-Columnar Storage. When users create tables, they can specify whether to enable row storage. With row storage enabled, each row only requires one IO operation for point queries (such as `SELECT *`), significantly improving performance.

The principle of row storage is that an additional column is added during storage. This column concatenates all the columns of the corresponding row and stores them using a special binary format.

Syntax

When creating a table, specify whether to enable row storage, and the storage compression unit size `page_size` in the table's `PROPERTIES`.

Property	Default	Description
`"store_row_column"`	`false`	Enable row storage for all columns.
`"row_store_page_size"`	`16384` (16 KB)	Size of the row-store page in bytes.

A page is the smallest unit for storage read and write operations, and `page_size` refers to the size of a row-store page. This means that reading a single row requires generating a page IO. The larger this value is, the better the compression effect and the lower the storage space usage. However, the IO overhead during point queries increases, resulting in lower performance (because each IO operation reads at least one page). Conversely, the smaller the value, the higher the storage space usage and the better the performance for point queries. The default value of 16KB is a balanced choice in most cases. If you prioritize query performance, you can configure a smaller value, such as 4KB or even lower. If you prioritize storage space, you can configure a larger value, such as 64KB or even higher.

> **Version 2.1 does not support the `row_store_columns` property to specify partial columns for row storage.**

Example

The example below creates an 8-column table with row storage enabled. To optimize for high-concurrency point query performance, the `row_store_page_size` is configured to 4KB.

```
CREATE TABLE `tbl_point_query` (
     `key` int(11) NULL,
     `v1` decimal(27, 9) NULL,
     `v2` varchar(30) NULL,
     `v3` varchar(30) NULL,
     `v4` date NULL,
     `v5` datetime NULL,
     `v6` float NULL,
     `v7` datev2 NULL
) ENGINE=OLAP
UNIQUE KEY(`key`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`key`) BUCKETS 1
PROPERTIES (
     "enable_unique_key_merge_on_write" = "true",
     "light_schema_change" = "true",
     "store_row_column" = "true",
     "row_store_page_size" = "4096"
);
```

Query:

```
SELECT * FROM tbl_point_query WHERE key = 100;
```

For more information on point query usage, please refer to High-Concurrent Point Query.

Notice

1. Enabling row storage will increase the storage space used. The increase in storage space is related to the data characteristics and is generally 2 to 10 times the size of the original table. The exact space usage needs to be tested with actual data.
2. The `page_size` of row storage also affects the storage space. You can adjust it based on the table attribute parameter `row_store_page_size`.
3. Modifying the `store_row_column` property via `ALTER TABLE` is not supported.

Best Practices | Table Design

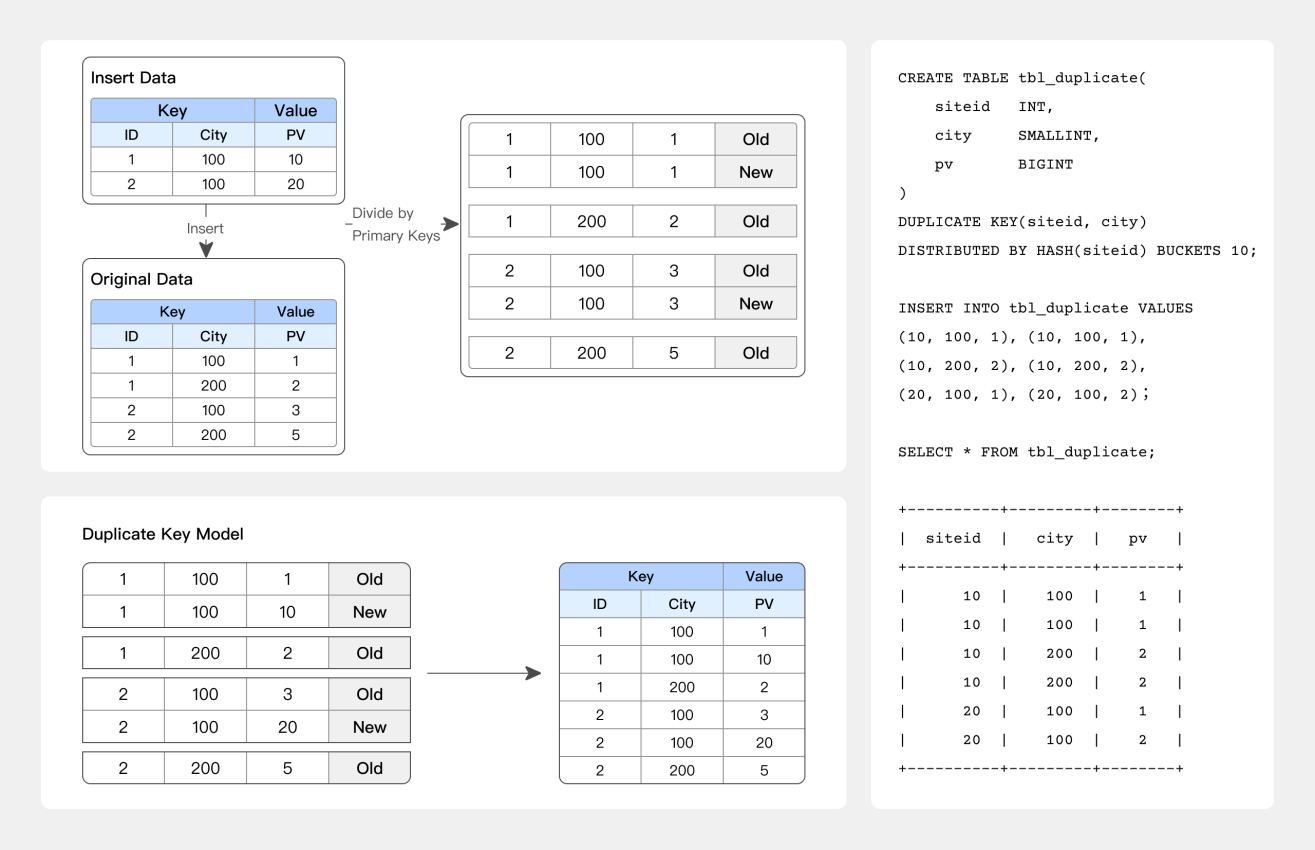
Best Practices

Data models

> Doris arranges data in three models: DUPLICATE KEY model, UNIQUE KEY model, and AGGREGATE KEY model.

> ****Recommendations****
>
> Because the data model is determined and **immutable** during table creation, it is important to select the most suitable data model.
>
> 1. The Duplicate Key model is a good fit for ad-hoc queries across any dimensions. It cannot leverage the benefits of pre-aggregation, but it is also not constrained by the limitations of an aggregation model, so it can take advantage of columnar storage (only reading the relevant columns without the need to read all key columns).
> 2. The Aggregate Key model, through pre-aggregation, can largely reduce the amount of data scanned and the computational workload for aggregate queries. It is particularly suitable for reporting queries with fixed patterns. However, this model is not friendly for count(*) queries. Additionally, since the aggregation method on the Value columns are fixed, users should pay extra attention to semantic correctness when performing other types of aggregate queries.
> 3. The Unique Key model is designed for scenarios that require unique primary key constraints. It can ensure the uniqueness of primary keys. The downside is that it cannot reap the benefits brought by materialization and pre-aggregation. For users with high-performance requirements for aggregate queries, it is recommended to use the Merge-on-Write feature of the Unique Key model since Doris 1.2.
> 4. Users with partial column update requirements might select from the following data models:
>   1. Unique Key model (Merge-on-Write mode)
>   2. Aggregate Key model (aggregated by REPLACE_IF_NOT_NULL)

DUPLICATE KEY model



duplicate-key-model-example

When only the sorting columns are specified, rows with the same key will not be merged.

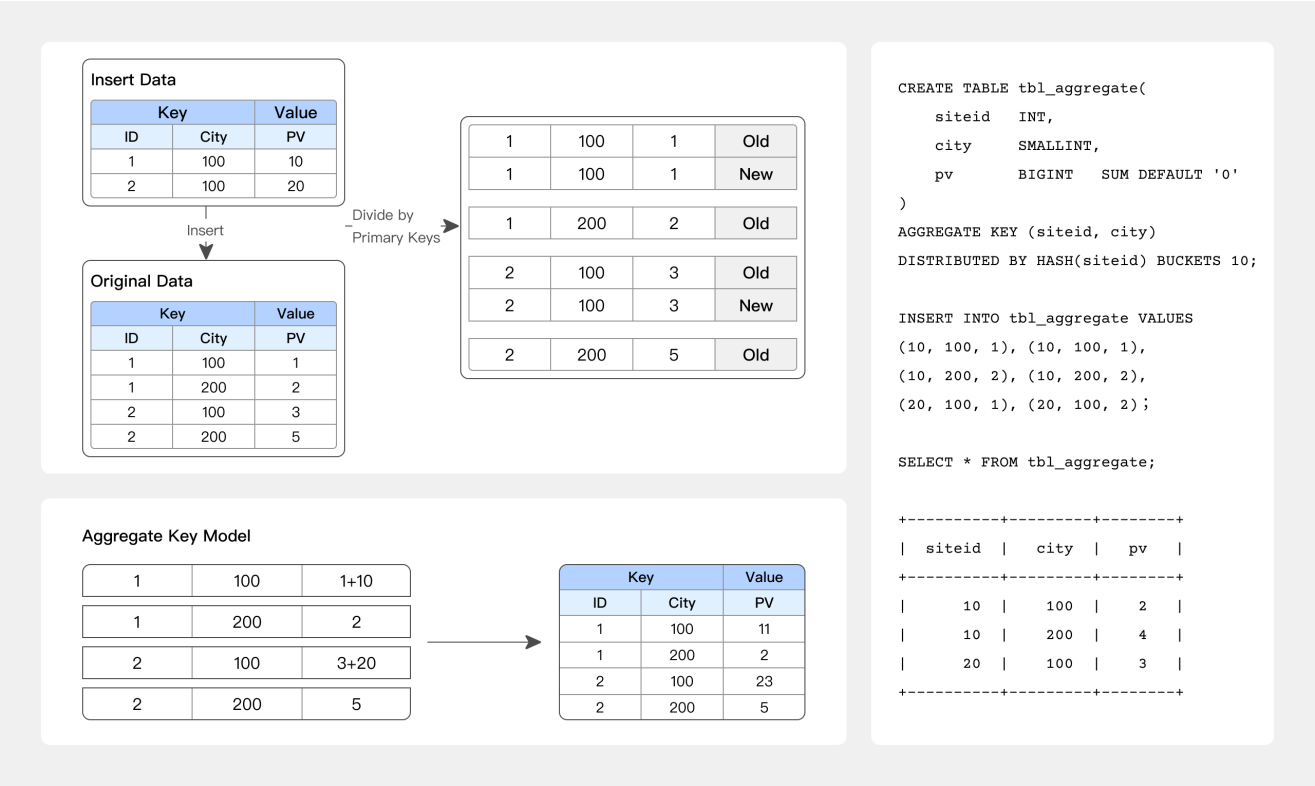
This is applicable to analytical business scenarios where data does not require pre-aggregation:

- Analysis of raw data
- Analysis of log or time series data where only new data is appended.

Best practice

```
-- For example, log analysis that allows only appending new data with replicated KEYs.
CREATE TABLE session_data
(
    visitorid   SMALLINT,
    sessionid   BIGINT,
    visittime   DATETIME,
    city        CHAR(20),
    province    CHAR(20),
    ip          varchar(32),
    brower      CHAR(20),
    url         VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid) -- Used solely for specifying sorting columns, rows with the same KEY will not be merged.
DISTRIBUTED BY HASH(sessionid, visitorid) BUCKETS 10;
```

AGGREGATE KEY model



aggregate-key-model-example

Old and new records with the same AGGREGATE KEY will be aggregated. The currently supported aggregation methods are as follows:

1. SUM: calculates the sum by accumulating the values of multiple rows;
2. REPLACE: replaces the value in the previously imported rows with the value from the next batch of data;
3. MAX: retains the maximum value;
4. MIN: retains the minimum value;
5. REPLACE_IF_NOT_NULL: replaces non-null values. Unlike REPLACE, it does not replace null values;
6. HLL_UNION: aggregates columns of HLL type using the HyperLogLog algorithm;
7. BITMAP_UNION: aggregates columns of BITMAP type using bitmap union aggregation;

This is suitable for reporting and multi-dimensional analysis scenarios such as:

- Website traffic analysis
- Multi-dimensional analysis of data reports.

Best practice

```
-- Example of website traffic analysis
CREATE TABLE site_visit
(
    siteid      INT,
    city        SMALLINT,
    username    VARCHAR(32),
    pv BIGINT   SUM DEFAULT '0' -- PV caculation
)
AGGREGATE KEY(siteid, city, username) -- Rows with the same KEY will be merged, and non-key columns will be aggregated based on the
```

```
specified aggregation function.
DISTRIBUTED BY HASH(siteid) BUCKETS 10;
```

UNIQUE KEY model

The new record will replace the old record that has the same UNIQUE KEY with it. Before Doris 1.2, the UNIQUE KEY model was implemented the same way as the REPLACE aggregation of the AGGREGATE KEY model. However, Since Doris 1.2, we have introduced the Merge-on-Write implementation for the UNIQUE KEY model, which provides better performance for aggregate queries.

This is suitable for analytical business scenarios that require updates, such as:

- Deduplicated order analysis
- Real-time synchronization of inserts, updates, and deletes.

Best practice

```
-- Example of deduplicated order analysis
CREATE TABLE sales_order
(
    orderid     BIGINT,
    status      TINYINT,
    username    VARCHAR(32),
    amount      BIGINT DEFAULT '0'
)
UNIQUE KEY(orderid) -- Rows of the same KEY will be merged
DISTRIBUTED BY HASH(orderid) BUCKETS 10;
```

Index

> Indexes can facilitate quick filtering and searching of data. Currently, Doris supports two types of indexes:
>
> 1. Built-in smart indexes, including prefix index and ZoneMap index.
> 2. User-created secondary indexes, including inverted index, BloomFilter index, Ngram BloomFilter index, and Bitmap index.

Prefix index

Prefix indexes are built-in indexes in the Aggregate, Unique, and Duplicate data models. The underlying data storage is sorted and stored based on the columns specified as the AGGREGATE KEY, UNIQUE KEY, or DUPLICATE KEY in their respective table creation statements. Prefix index, built on top of the sorted data, allows for quick data querying based on given prefix columns.

Prefix indexes are sparse indexes and cannot locate the exact row where a key is present. Instead, they can only roughly identify the range where the key may exist, and then use binary search algorithms to accurately locate the position of the key.

> ****Recommendations****

1. When creating a table, **the correct column order can greatly improve query efficiency**.
    1. Since the column order is specified during table creation, a table can only have one type of prefix index. However, this may not be efficient enough for queries based on the columns without prefix index. In such cases, users can adjust the column order by creating materialized views.
2. The first field in a prefix index should always be the field of the longest query conditions and should be a high-cardinality field.
    1. Bucketing field: It should have relatively even data distribution and be frequently used, preferably a high-cardinality field.
    2. Int(4) + Int(4) + varchar(50): The prefix index length is only 28.
    3. Int(4) + varchar(50) + Int(4): The prefix index length is only 24.
    4. varchar(10) + varchar(50): The prefix index length is only 30.
    5. Prefix index (36 characters): The first field delivers the best query performance. If a varchar field is encountered, the prefix index will automatically truncate it to the first 20 characters.
    6. If possible, include the most frequently used query fields in the prefix index. Otherwise, specify them as the bucketing fields.
3. The field lengths in the prefix index should be as explicit as possible because Doris can only utilize the prefix index for the first 36 bytes.
4. If it is difficult to design a partitioning, bucketing, and prefix index strategy for your data range, consider introducing inverted indexes for acceleration.

ZoneMap index

ZoneMap index is index information automatically maintained on a per-column basis in the columnar storage format. It includes information such as Min/Max values and the number of Null values. During data querying, the ZoneMap index is utilized to select the data range to scan based on the filtered fields using range conditions.

For example, when filtering the “age” field with the following query statement:

```
SELECT * FROM table WHERE age > 0 and age < 51;
```

If the Short Key Index is not hit, the ZoneMap index will be used to determine the data range, known as the “ordinary” range, that needs to be scanned based on the query conditions for the “age” field. This reduces the number of pages that need to be scanned.

Inverted index

Doris supports inverted indexes since version 2.0.0. Inverted index can be used for full-text searches on text data and range queries on regular numeric and date types. It enables fast filtering of rows that meet the conditions from massive amounts of data.

Best practice

```
-- Inverted index can be specified during table creation or added later. This is an example of specifying it during table creation:
CREATE TABLE table_name
(
  columns_difinition,
```

```
  INDEX idx_name1(column_name1) USING INVERTED [PROPERTIES("parser" = "english|unicode|chinese")] [COMMENT 'your comment']
  INDEX idx_name2(column_name2) USING INVERTED [PROPERTIES("parser" = "english|unicode|chinese")] [COMMENT 'your comment']
  INDEX idx_name3(column_name3) USING INVERTED [PROPERTIES("parser" = "chinese", "parser_mode" = "fine_grained|coarse_grained")]
[COMMENT 'your comment']
  INDEX idx_name4(column_name4) USING INVERTED [PROPERTIES("parser" = "english|unicode|chinese", "support_phrase" = "true|false")]
[COMMENT 'your comment']
  INDEX idx_name5(column_name4) USING INVERTED [PROPERTIES("char_filter_type" = "char_replace", "char_filter_pattern" = "._"),
"char_filter_replacement" = " "] [COMMENT 'your comment']
  INDEX idx_name5(column_name4) USING INVERTED [PROPERTIES("char_filter_type" = "char_replace", "char_filter_pattern" = "._")]
[COMMENT 'your comment']
)
table_properties;

-- Example: keyword matching in full-text searches, implemented by MATCH_ANY MATCH_ALL
SELECT * FROM table_name WHERE column_name MATCH_ANY | MATCH_ALL 'keyword1 ...';
```

****Recommendations****

1. If it is difficult to design a partitioning, bucketing, and prefix index strategy for your data range, consider introducing inverted indexes for acceleration.

****Restrictions****

1. Different data models have different restrictions on inverted index.
    1. Aggregate KEY model: only allows inverted index for Key columns
    2. Unique KEY model: allows inverted index for any column after enabling Merge-on-Write
    3. Duplicate KEY model: allows inverted index for any column

BloomFilter index

Doris supports adding BloomFilter indexes to fields with high value distinctiveness, making it suitable for scenarios that involve equivalence queries on columns with high cardinality.

Best practice

```
-- Example: add "bloom_filter_columns"="k1,k2,k3" in the PROPERTIES of the table creation statement.
-- To create BloomFilter index for saler_id and category_id in the table.
CREATE TABLE IF NOT EXISTS sale_detail_bloom  (
    sale_date date NOT NULL COMMENT "Sale data",
    customer_id int NOT NULL COMMENT "Customer ID",
    saler_id int NOT NULL COMMENT "Saler ID",
    sku_id int NOT NULL COMMENT "SKU ID",
    category_id int NOT NULL COMMENT "Category ID",
    sale_count int NOT NULL COMMENT "Sale count",
    sale_price DECIMAL(12,2) NOT NULL COMMENT "Sale price",
    sale_amt DECIMAL(20,2)  COMMENT "Sale amount"
)
Duplicate  KEY(sale_date, customer_id,saler_id,sku_id,category_id)
DISTRIBUTED BY HASH(saler_id) BUCKETS 10
PROPERTIES (
"bloom_filter_columns"="saler_id,category_id"
);
```

****Restrictions****

1. BloomFilter indexes are not supported for columns of type Tinyint, Float, and Double.
2. BloomFilter indexes can only accelerate filtering using “in” and “=” operators.
3. BloomFilter indexes must be built on high-cardinality columns (above 5000) in query conditions that involve “in” or “=” operators.
    1. BloomFilter indexes are suitable for non-prefix filtering.
    2. Queries will filter based on the high-frequency values in the column, and the filtering conditions are mostly “in” and “=”.
    3. Unlike Bitmap indexes, BloomFilter indexes are suitable for high-cardinality columns, such as UserID. If created on low-cardinality columns like “gender”, each block will contain almost all values, rendering the BloomFilter index meaningless.
    4. It is suitable for cases with data cardinality around half of the total range.
    5. For high-cardinality columns with equality (=) queries, such as ID numbers, using BloomFilter indexes can greatly accelerate performance.

Ngram BloomFilter index

Since 2.0.0, Doris has introduced the NGram BloomFilter index to improve the performance of “**LIKE**” queries.

Best practice

```
-- Example of creating NGram BloomFilter index in table creation statement
CREATE TABLE `nb_table` (
  `siteid` int(11) NULL DEFAULT "10" COMMENT "",
  `citycode` smallint(6) NULL COMMENT "",
  `username` varchar(32) NULL DEFAULT "" COMMENT "",
  INDEX idx_ngrambf (`username`) USING NGRAM_BF PROPERTIES("gram_size"="3", "bf_size"="256") COMMENT 'username ngram_bf index'
) ENGINE=OLAP
AGGREGATE KEY(`siteid`, `citycode`, `username`) COMMENT "OLAP"
DISTRIBUTED BY HASH(`siteid`) BUCKETS 10;

-- PROPERTIES("gram_size"="3", "bf_size"="256"), representing the number of grams and the byte size of the BloomFilter
-- The number of grams is determined according to the query cases and is typically set to the length of the majority of query strings. The number of bytes in the BloomFilter can be determined after testing. Generally, a larger number of bytes leads to better filtering results, and it is recommended to start with a value of 256 for testing and evaluating the effectiveness. However, it's important to note that a larger number of bytes also increases the storage cost of the index.
-- With high data cardinality, there is no need to set a large BloomFilter size. Conversely, with low data cardinality, increase the BloomFilter size to enhance filtering efficiency.
```

****Restrictions****

1. NGram BloomFilter index only supports string columns.
2. NGram BloomFilter indexes and BloomFilter indexes are mutually exclusive, meaning that only one of them can be set for the same column.
3. The sizes of the NGram and the BloomFilter can both be optimized based on the actual situation. If the NGram size is relatively small, you may increase the BloomFilter size.

4. For data at the scale of billions or above, if there is a need for fuzzy matching, it is recommended to use inverted indexes or NGram BloomFilter.

Bitmap index

To accelerate data queries, Doris supports users in adding Bitmap indexes to certain fields. This is suitable for scenarios involving equivalence or range queries on columns with lower cardinality.

Best practice

```
-- Example: create Bitmap index for siteid on bitmap_table
CREATE INDEX [IF NOT EXISTS] bitmap_index_name ON
bitmap_table (siteid)
USING BITMAP COMMENT 'bitmap_siteid';
```

****Restrictions****

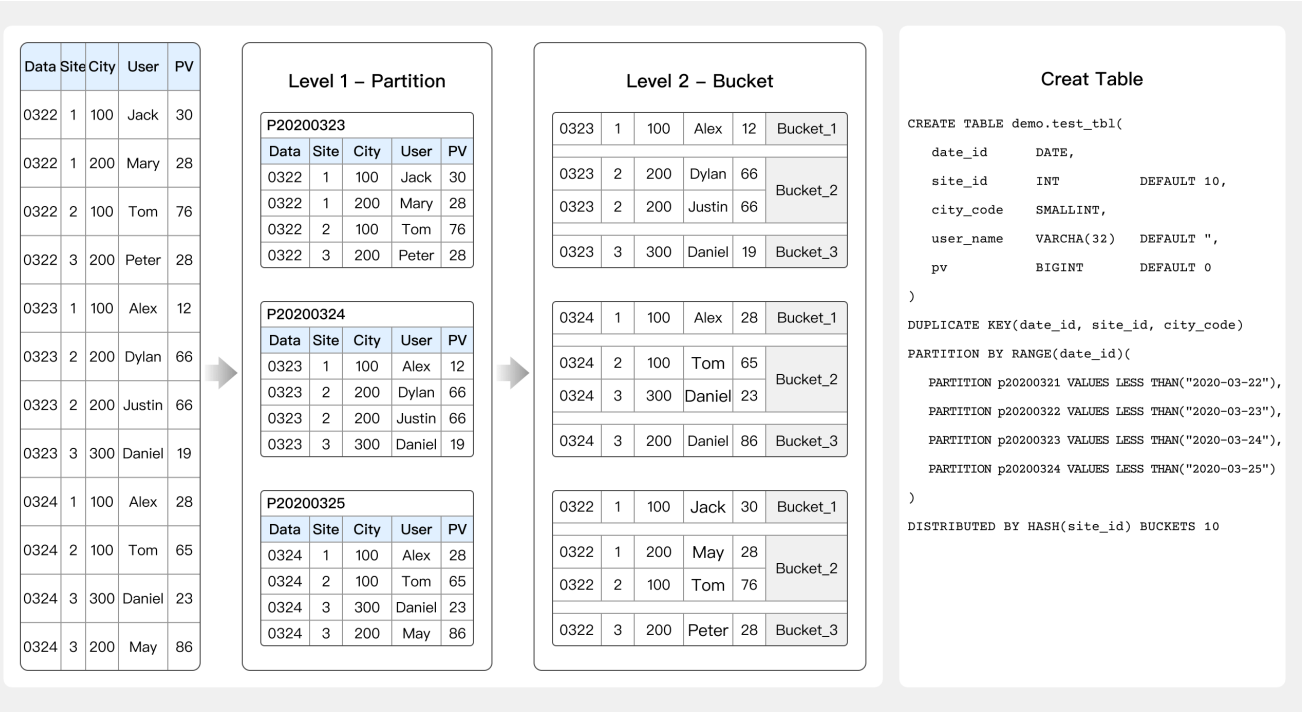
1. Bitmap indexes can only be created on a single column.
2. Bitmap indexes can be applied to all columns in the `Duplicate` and `Unique` Key models, as well as the key columns in the `Aggregate` Key model.
3. Bitmap indexes support the following data types:
    1. `TINYINT`
    2. `SMALLINT`
    3. `INT`
    4. `BIGINT`
    5. `CHAR`
    6. `VARCHAR`
    7. `DATE`
    8. `DATETIME`
    9. `LARGEINT`
    10. `DECIMAL`
    11. `BOOL`
4. Bitmap indexes only take effect under Segment V2. The storage format of tables with Bitmap indexes will be automatically converted to V2 format by default.
5. Bitmap indexes should be constructed within a certain cardinality range. It is not suitable for extremely high or low cardinality cases.
    1. It is recommended for columns with a cardinality between 100 and 100,000, such as the occupation field or city field. If the duplication rate is too high, there won't be significant advantages to build Bitmap indexes compared to other types of indexes. If the duplication rate is too low, Bitmap indexes can significantly reduce space efficiency and performance. Specific types of queries, such as count, OR, and AND operations, only require bitwise operations.
    2. Bitmap indexes are more suitable for orthogonal queries.

Field type

Doris supports various field types, including precise deduplication with BITMAP, fuzzy deduplication with HLL, semi-structured data types such as ARRAY/MAP/JSON, as well as common numeric, string, and time types.

> ****Recommendations****
>
> 1. VARCHAR
>   1. Variable-length string with a length range of 1-65533 bytes. It is stored in UTF-8 encoding, where English characters typically occupy 1 byte.
>   2. There is often a misunderstanding about the performance difference between varchar(255) and varchar(65533). If the data stored in both cases is the same, the performance will be the same as well. When creating a table, if you are unsure about the maximum length of the field, it is recommended to use varchar(65533) to prevent import errors caused by excessively long strings.
> 2. STRING
>   1. Variable-length string with a default size of 1048576 bytes (1MB), which can be increased to 2147483643 bytes (2GB). It is stored in UTF-8 encoding, where English characters typically occupy 1 byte.
>   2. It can only be used in value columns but not key columns or partitioning columns.
>   3. It is suitable for storing large text content. However, if such a requirement does not exist, it is recommended to use VARCHAR. STRING columns have limitations as they cannot be used in key columns or partitioning columns.
> 3. Numeric fields: Choose the appropriate data type based on the required precision. There is no special restrictions on this.
> 4. Time fields: Note that if there is a high precision requirement (timestamp accurate to milliseconds), you need to specify the use of datetime(6). Otherwise, such timestamps are not supported by default.
> 5. It is recommended to use the JSON data type instead of string type for storing JSON data.

Create table



create-table-example

Considerations in creating a table include the setting of data partitions and buckets in addition to data model, index, and field types.

Best practice

```
-- Take Merge-on-Write tables in the Unique Key model as an example:
-- Merge-on-Write in the Unique Key model is implemented in a different way from the Aggregate Key model. The performance of it is similar to that on the Duplicate Key model.
-- In use cases requiring primary key constraints, the Aggregate Key model can deliver much better query performance compared to the Duplicate Key model, especially in aggregate queries and queries that involve filtering a large amount of data using indexes.

-- For non-partitioned tables
CREATE TABLE IF NOT EXISTS tbl_unique_merge_on_write
(
    `user_id` LARGEINT NOT NULL COMMENT "Use ID",
    `username` VARCHAR(50) NOT NULL COMMENT "Username",
    `register_time` DATE COMMENT "User registration time",
    `city` VARCHAR(20) COMMENT "User city",
    `age` SMALLINT COMMENT "User age",
    `sex` TINYINT COMMENT "User gender",
    `phone` LARGEINT COMMENT "User phone number",
    `address` VARCHAR(500) COMMENT "User address"
)
UNIQUE KEY(`user_id`, `username`)
-- Data volume of 3~5G
DISTRIBUTED BY HASH(`user_id`) BUCKETS 10
PROPERTIES (
-- In Doris 1.2.0, as a new feature, Merge-on-Write is disabled by default. Users can enable it by adding the following property.
"enable_unique_key_merge_on_write" = "true"
);

-- For partitioned tables
CREATE TABLE IF NOT EXISTS tbl_unique_merge_on_write_p
(
```

```
    `user_id` LARGEINT NOT NULL COMMENT "Use ID",
    `username` VARCHAR(50) NOT NULL COMMENT "Username",
    `register_time` DATE COMMENT "User registration time",
    `city` VARCHAR(20) COMMENT "User city",
    `age` SMALLINT COMMENT "User age",
    `sex` TINYINT COMMENT "User gender",
    `phone` LARGEINT COMMENT "User phone number",
    `address` VARCHAR(500) COMMENT "User address"
)
UNIQUE KEY(`user_id`, `username`, `register_time`)
PARTITION BY RANGE(`register_time`) (
    PARTITION p00010101_1899 VALUES [('0001-01-01'), ('1900-01-01')),
    PARTITION p19000101 VALUES [('1900-01-01'), ('1900-01-02')),
    PARTITION p19000102 VALUES [('1900-01-02'), ('1900-01-03')),
    PARTITION p19000103 VALUES [('1900-01-03'), ('1900-01-04')),
    PARTITION p19000104_1999 VALUES [('1900-01-04'), ('2000-01-01')),
    FROM ("2000-01-01") TO ("2022-01-01") INTERVAL 1 YEAR,
    PARTITION p30001231 VALUES [('3000-12-31'), ('3001-01-01')),
    PARTITION p99991231 VALUES [('9999-12-31'), (MAXVALUE))
)
-- Data volume of 3~5G
DISTRIBUTED BY HASH(`user_id`) BUCKETS 10
PROPERTIES (
-- In Doris 1.2.0, as a new feature, Merge-on-Write is disabled by default. Users can enable it by adding the following property.
"enable_unique_key_merge_on_write" = "true",
-- The unit for dynamic partition scheduling can be specified as HOUR, DAY, WEEK, MONTH, or YEAR.
"dynamic_partition.time_unit" = "MONTH",
-- The starting offset for dynamic partitioning is specified as a negative number. Depending on the value of the time_unit, it uses
the current day (week/month) as the reference point, partitions prior to this offset will be deleted (TTL). If not specified, the
default value is -2147483648, indicating that historical partitions will not be deleted.
"dynamic_partition.start" = "-3000",
-- The ending offset for dynamic partitioning is specified as a positive number. Depending on the value of the time_unit, it uses
the current day (week/month) as the reference point. Create the corresponding partitions of the specified range in advance.
"dynamic_partition.end" = "10",
-- The prefix for names of the dynamically created partitions (required).
"dynamic_partition.prefix" = "p",
-- The number of buckets corresponding to the dynamically created partitions.
"dynamic_partition.buckets" = "10",
"dynamic_partition.enable" = "true",
-- The following is the number of replicas corresponding to dynamically created partitions. If not specified, the default value will
be the replication factor specified when creating the table, which is typically 3.
"dynamic_partition.replication_num" = "3",
"replication_num" = "3"
);

-- View existing partitions
-- The actual number of created partitions is determined by a combination of dynamic_partition.start, dynamic_partition.end, and the
settings of PARTITION BY RANGE.
show partitions from tbl_unique_merge_on_write_p;
```

****Restrictions****

1. The database character set should be specified as UTF-8 since only UTF-8 is supported.
2. The replication factor for tables must be 3 (if not specified, it defaults to 3).
3. The data volume of an individual tablet (**Tablet Count = Partition Count * Bucket Count * Replication Factor**) theoretically has no upper or lower bounds, except for small tables (in the range of hundreds of megabytes to one gigabyte), where it should be ensured to be within the range of 1 GB to 10 GB:

1. If the data volume of an individual tablet is too small, it can result in poor data aggregation performance and increased metadata management overhead.
2. If the data volume is too large, it hinders replica migration, synchronization, and increases the cost of schema changes or materialization operations (these operations are retried at the granularity of a tablet).

4. For data exceeding 500 million records, **partitioning and bucketing** strategies must be implemented:

    1. **Recommendations for bucketing:**
        1. For large tables, each tablet should be in the range of 1 GB to 10 GB to prevent the generation of too many small files.
        2. For dimension tables of approximately 100 megabytes, the number of tablets should be controlled within the range of 3 to 5. This ensures a certain level of concurrency without generating excessive small files.
    2. If partitioning is not feasible and the data grows rapidly without the possibility of dynamic time-based partitioning, it is advisable to increase the number of buckets to accommodate the data volume based on the data retention period (180 days). It is still recommended to keep the size of each bucket between 1 GB and 10 GB.
    3. Apply salting to the bucketing field and use the same salting strategy for queries in order to leverage bucket pruning capabilities.
    4. Random bucketing:
        1. If an OLAP table does not have fields that need to be updated, setting the data bucketing mode to RANDOM can avoid severe data skew. During data ingestion, each batch of data is randomly assigned to a tablet for writing.
        2. When the bucketing mode for a table is set to RANDOM, since there is no bucketing column, querying the table will scan all buckets in the hit partitions instead of querying specific buckets based on the values of the bucketing column. This setting is suitable for overall aggregation and analysis queries rather than high-concurrency point queries.
        3. If an OLAP table has a random distribution of data, setting the `load_to_single_tablet` parameter to true during data ingestion allows each task to write to a single tablet. This improves concurrency and throughput during large-scale data ingestion. It can also reduce the write amplification caused by data ingestion and compaction and ensure cluster stability.
    5. Dimension tables, which grow slowly, can use a single partition and apply bucketing based on commonly used query conditions (where the data distribution of the bucketing field is relatively even).
    6. Fact tables: We recommend using DATE or DATETIME as the partitioning column. In most cases, one partition per day is sufficient. For the bucketing strategy, use frequently queried columns with relatively even data distribution as the bucketing columns.

5. For scenarios where there is a large amount of historical partitioned data but the historical data is relatively small, unbalanced, or queried infrequently, you can use the following approach to place the data in special partitions. You can create historical partitions for historical data of small sizes (e.g., yearly partitions, monthly partitions). For example, you can create historical partitions for data `FROM ("2000-01-01") TO ("2022-01-01") INTERVAL 1 YEAR`:

```
PARTITION p00010101_1899 VALUES [('0001-01-01'), ('1900-01-01')),
PARTITION p19000101 VALUES [('1900-01-01'), ('1900-01-02')),
...
PARTITION p19000104_1999 VALUES [('1900-01-04'), ('2000-01-01')),
FROM ("2000-01-01") TO ("2022-01-01") INTERVAL 1 YEAR,
PARTITION p30001231 VALUES [('3000-12-31'), ('3001-01-01')),
PARTITION p99991231 VALUES [('9999-12-31'), (MAXVALUE))
```

Loading Overview

Apache Doris offers various methods for importing and integrating data, allowing you to import data from various sources into the database. These methods can be categorized into four types:

- **Real-Time Writing**: Data is written into Doris tables in real-time via HTTP or JDBC, suitable for scenarios requiring immediate analysis and querying.
    - For small amounts of data (once every 5 minutes), you can use JDBC INSERT.
    - For higher concurrency or frequency (more than 20 concurrent writes or multiple writes per minute), you can enable Group Commit and use JDBC INSERT or Stream Load.
    - For high throughput, you can use Stream Load via HTTP.
- **Streaming Synchronization**: Real-time data streams (e.g., Flink, Kafka, transactional databases) are imported into Doris tables, ideal for real-time analysis and querying.
    - You can use Flink Doris Connector to write Flink's real-time data streams into Doris.
    - You can use Routine Load or Doris Kafka Connector for Kafka's real-time data streams. Routine Load pulls data from Kafka to Doris and supports CSV and JSON formats, while Kafka Connector writes data to Doris, supporting Avro, JSON, CSV, and Protobuf formats.
    - You can use Flink CDC or Datax to write transactional database CDC data streams into Doris.
- **Batch Import**: Data is batch-loaded from external storage systems (e.g., Object Storage, HDFS, local files, NAS) into Doris tables, suitable for non-real-time data import needs.
    - You can use Broker Load to write files from Object Storage and HDFS into Doris.
    - You can use INSERT INTO SELECT to synchronously load files from Object Storage, HDFS, and NAS into Doris, and you can perform the operation asynchronously using a JOB.
    - You can use Stream Load or Doris Streamloader to write local files into Doris.
- **External Data Source Integration**: Query and partially import data from external sources (e.g., Hive, JDBC, Iceberg) into Doris tables.
    - You can create a Catalog to read data from external sources and use INSERT INTO SELECT to synchronize this data into Doris, with asynchronous execution via JOB.

Each import method in Doris is an implicit transaction by default. For more information on transactions, refer to Transactions.

Quick Overview of Import Methods

Doris import process mainly involves various aspects such as data sources, data formats, import methods, error handling, data transformation, and transactions. You can quickly browse the scenarios suitable for each import method and the supported file formats in the table below.

Import Method	Use Case	Supported File Formats	Import Mode
Stream Load	Importing local files or push data in applications via HTTP.	csv, json, parquet, orc	Synchronous
Broker Load	Importing from object storage, HDFS, etc.	csv, json, parquet, orc	Asynchronous
INSERT INTO VALUES	Writing data via JDBC.	SQL	Synchronous
INSERT INTO SELECT	Importing from an external source like a table in a catalog or files in Object Storage, HDFS.	SQL	Synchronous, Asynchronous via Job
Routine Load	Real-time import from Kafka	csv, json	Asynchronous
MySQL Load	Importing from local files.	csv	Synchronous
Group Commit	Writing with high frequency.	Depending on the import method used	-

local file

Doris provides multiple ways to load data from local sources:

1. Stream Load

Load local files or data streams into Doris via HTTP protocol. Supports CSV, JSON, Parquet, and ORC formats. For more information, refer to the Stream Load documentation.

2. Streamloader Tool

The Streamloader tool is a dedicated client tool for loading data into the Doris database, based on Stream Load. It can provide multi-file and concurrent load capabilities, reducing the time required for loading large volumes of data. For more documentation, refer to Streamloader.

3. MySQL Load

Doris is compatible with MySQL protocol and supports using the standard LOAD DATA syntax to load local files, suitable for loading CSV files.

Using Stream Load to Load Data

Step 1: Prepare Data

Create a CSV file named `streamload_example.csv` with the following content:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create Table

Create a table in Doris with the following syntax:

```
CREATE TABLE testdb.test_streamload(
    user_id BIGINT NOT NULL COMMENT "User ID",
    name VARCHAR(20) COMMENT "User Name",
    age INT COMMENT "User Age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load Data

Submit a Stream Load job using `curl`:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Example of load result:

```
{
    "TxnId": 3,
    "Status": "Success",
    "NumberTotalRows": 10,
    "NumberLoadedRows": 10
}
```

Step 4: Check Loaded Data

```
mysql> SELECT COUNT(*) FROM testdb.test_streamload;
+----------+
| count(*) |
+----------+
|       10 |
+----------+
```

Using Streamloader Tool to Load Data

Step 1: Prepare Data

Create a CSV file named `streamloader_example.csv` with the same content as above.

Step 2: Create Table

Create the table in Doris with the same syntax as above.

Step 3: Load Data

Use the Streamloader tool to load data:

```
doris-streamloader --source_file="streamloader_example.csv" --url="http://localhost:8330" --header="column_separator:," --
db="testdb" --table="test_streamload"
```

Example of load result:

```
Load Result: {
    "Status": "Success",
    "TotalRows": 10,
```

```
    "LoadedRows": 10
}
```

Step 4: Check Loaded Data

```
mysql> SELECT COUNT(*) FROM testdb.test_streamload;
+----------+
| count(*) |
+----------+
|       10 |
+----------+
```

Using MySQL Load to Load Data

Step 1: Prepare Data

Create a file named `client_local.csv` with the following sample data:

```
1,10
2,20
3,30
4,40
5,50
6,60
```

Step 2: Connect to MySQL Client

```
mysql --local-infile -h <fe_ip> -P <fe_query_port> -u root -D testdb
```

Step 3: Load Data

Execute the MySQL Load command:

```
LOAD DATA LOCAL
INFILE 'client_local.csv'
INTO TABLE testdb.t1
COLUMNS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

Step 4: Check Loaded Data

If the load is successful, the result will be displayed as follows:

```
Query OK, 6 rows affected (0.17 sec)
Records: 6  Deleted: 0  Skipped: 0  Warnings: 0
```

Kafka

Doris provides the following methods to load data from Kafka:

- **Using Routine Load to consume Kafka data**

Doris continuously consumes data from Kafka Topics through Routine Load. After submitting a Routine Load job, Doris generates load tasks in real-time to consume messages from the specified Topic in the Kafka cluster. Routine Load supports CSV and JSON formats, with Exactly-Once semantics, ensuring that data is neither lost nor duplicated. For more documentation, please refer to Routine Load.

- **Doris Kafka Connector to consume Kafka data**

The Doris Kafka Connector is a tool for loading Kafka data streams into the Doris database. Users can easily load various serialization formats (such as JSON, Avro, Protobuf) through the Kafka Connect plugin, and it supports parsing data formats from the Debezium component. For more documentation, please refer to Doris Kafka Connector.

In most cases, you can directly choose Routine Load for loading data without the need to integrate external components to consume Kafka data. When you need to load data in Avro or Protobuf formats, or data collected from upstream databases via Debezium, you can use the Doris Kafka Connector.

Using Routine Load to consume Kafka data

Usage Restrictions

1. Supported message formats are CSV and JSON. Each CSV message is one line, and the line does not contain a newline character at the end;
2. By default, it supports Kafka version 0.10.0.0 and above. If you need to use older versions (such as 0.9.0, 0.8.2, 0.8.1, 0.8.0), you need to modify the BE configuration to set `kafka_broker_version_fallback` to a compatible older version, or set `property.broker.version.fallback` when creating the Routine Load. Using older versions may result in some new features being unavailable, such as setting Kafka partition offsets based on time.

Operation Example

In Doris, create a persistent Routine Load load task through the CREATE ROUTINE LOAD command, which can be divided into single-table load and multi-table load. For detailed syntax, please refer to CREATE ROUTINE LOAD.

Single Table Load

Step 1: Prepare Data

In Kafka, sample data is as follows:

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-routine-load-csv --from-beginning
1,Emily,25
```

Step 2: Create Table in Database

Create the table to be loaded in Doris, with the following syntax:

```
CREATE TABLE testdb.test_routineload_tbl(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Create Routine Load job to load data into a single table

In Doris, use the CREATE ROUTINE LOAD command to create the load job:

```
CREATE ROUTINE LOAD testdb.example_routine_load_csv ON test_routineload_tbl
COLUMNS TERMINATED BY ",",
COLUMNS(user_id, name, age)
FROM KAFKA(
    "kafka_broker_list" = "192.168.88.62:9092",
    "kafka_topic" = "test-routine-load-csv",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

Step 4: Check Loaded Data

```
select * from test_routineload_tbl;
+-----------+----------------+------+
| user_id   | name           | age  |
+-----------+----------------+------+
|  1        | Emily          | 25   |
+-----------+----------------+------+
```

Multi-Table Load

In scenarios where multiple tables need to be loaded simultaneously, the data in Kafka must include table name information, formatted as: `table_name|data`. For example, when loading CSV data, the format should be: `table_name|val1,val2,val3`. Please note that the table name must exactly match the table name in Doris; otherwise, the loading will fail, and the column_mapping configuration introduced later is not supported.

Step 1: Prepare Data

In Kafka, sample data is as follows:

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-multi-table-load --from-beginning
test_multi_table_load1|1,Emily,25
test_multi_table_load2|2,Benjamin,35
```

Step 2: Create Tables in Database

Create the tables to be loaded in Doris, with the following syntax:

Table 1:

```
CREATE TABLE test_multi_table_load1(
    user_id            BIGINT       NOT NULL COMMENT "User ID",
    name               VARCHAR(20)           COMMENT "User Name",
    age                INT                   COMMENT "User Age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Table 2:

```
CREATE TABLE test_multi_table_load2(
    user_id            BIGINT       NOT NULL COMMENT "User ID",
    name               VARCHAR(20)           COMMENT "User Name",
    age                INT                   COMMENT "User Age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Create Routine Load job to load data into multiple tables

In Doris, use the CREATE ROUTINE LOAD command to create the load job:

```
CREATE ROUTINE LOAD example_multi_table_load
COLUMNS TERMINATED BY ","
FROM KAFKA(
    "kafka_broker_list" = "192.168.88.62:9092",
    "kafka_topic" = "test-multi-table-load",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

Step 4: Check Loaded Data

```
select * from test_multi_table_load1;
+---------+----------------+------+
| user_id | name           | age  |
+---------+----------------+------+
|  1      | Emily          | 25   |
+---------+----------------+------+

select * from test_multi_table_load2;
+---------+----------------+------+
| user_id | name           | age  |
+---------+----------------+------+
|  2      | Benjamin       | 35   |
+---------+----------------+------+
```

Configure Security Authentication

For methods of configuring Kafka with authentication, please refer to Kafka Security Authentication.

Using Doris Kafka Connector to consume Kafka data

The Doris Kafka Connector is a tool for loading Kafka data streams into the Doris database. Users can easily load various serialization formats (such as JSON, Avro, Protobuf) through the Kafka Connect plugin, and it supports parsing data formats from the Debezium component.

Start in Distributed Mode

Distributed mode provides scalability and automatic fault tolerance for Kafka Connect. In this mode, multiple worker processes can be started using the same `group.id`, which will coordinate the execution of connectors and tasks across all available worker processes.

1. Create a plugins directory under `$KAFKA_HOME` and place the downloaded doris-kafka-connector jar package inside.
2. Configure `config/connect-distributed.properties`:

```
# Modify kafka server address
bootstrap.servers=127.0.0.1:9092

# Modify group.id, which needs to be consistent across the same cluster
group.id=connect-cluster

# Modify to the created plugins directory
# Note: Please fill in the direct path of Kafka here. For example: plugin.path=/opt/kafka/plugins
plugin.path=$KAFKA_HOME/plugins

# It is recommended to increase Kafka's max.poll.interval.ms time to over 30 minutes, default is 5 minutes
# To avoid Stream Load data load consumption timeout, causing the consumer to be kicked out of the consumption group
max.poll.interval.ms=1800000
consumer.max.poll.interval.ms=1800000
```

3. Start:

```
$KAFKA_HOME/bin/connect-distributed.sh -daemon $KAFKA_HOME/config/connect-distributed.properties
```

4. Consume Kafka data:

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"test-doris-sink-cluster",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
    "topics":"topic_test",
    "doris.topic2table.map": "topic_test:test_kafka_tbl",
    "buffer.count.records":"10000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"5000000",
    "doris.urls":"10.10.10.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test_db",
    "key.converter":"org.apache.kafka.connect.storage.StringConverter",
    "value.converter":"org.apache.kafka.connect.storage.StringConverter"
```

```
  }
}'
```

Operate Kafka Connect

```
# View connector status
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/status -X GET
# Delete current connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster -X DELETE
# Pause current connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/pause -X PUT
# Resume current connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/resume -X PUT
# Restart tasks within the connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/tasks/0/restart -X POST
```

For an introduction to Distributed mode, please refer to Distributed Workers.

Load Ordinary Data

1. Load sample data:

In Kafka, sample data is as follows:

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-data-topic --from-beginning
{"user_id":1,"name":"Emily","age":25}
{"user_id":2,"name":"Benjamin","age":35}
{"user_id":3,"name":"Olivia","age":28}
{"user_id":4,"name":"Alexander","age":60}
{"user_id":5,"name":"Ava","age":17}
{"user_id":6,"name":"William","age":69}
{"user_id":7,"name":"Sophia","age":32}
{"user_id":8,"name":"James","age":64}
{"user_id":9,"name":"Emma","age":37}
{"user_id":10,"name":"Liam","age":64}
```

2. Create the table to be loaded:

Create the table to be loaded in Doris, with the following syntax:

```
CREATE TABLE test_db.test_kafka_connector_tbl(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 12;
```

3. Create the load task:

On the machine where Kafka Connect is deployed, submit the following load task via curl command:

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"test-doris-sink-cluster",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
```

```
    "tasks.max":"10",
    "topics":"test-data-topic",
    "doris.topic2table.map": "test-data-topic:test_kafka_connector_tbl",
    "buffer.count.records":"10000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"5000000",
    "doris.urls":"10.10.10.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test_db",
    "key.converter":"org.apache.kafka.connect.storage.StringConverter",
    "value.converter":"org.apache.kafka.connect.storage.StringConverter"
  }
}'
```

Load Data Collected by Debezium Component

1. The MySQL database has the following table:

```
CREATE TABLE test.test_user (
  user_id int NOT NULL ,
  name varchar(20),
  age int,
  PRIMARY KEY (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

insert into test.test_user values(1,'zhangsan',20);
insert into test.test_user values(2,'lisi',21);
insert into test.test_user values(3,'wangwu',22);
```

2. Create the table to be loaded in Doris:

```
CREATE TABLE test_db.test_user(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
UNIQUE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 12;
```

3. Deploy the Debezium connector for MySQL component, refer to: Debezium connector for MySQL.

4. Create the doris-kafka-connector load task:

Assuming the data from the MySQL table collected by Debezium is in the `mysql_debezium.test.test_user` Topic:

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"test-debezium-doris-sink",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
    "tasks.max":"10",
    "topics":"mysql_debezium.test.test_user",
    "doris.topic2table.map": "mysql_debezium.test.test_user:test_user",
    "buffer.count.records":"10000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"5000000",
```

```
    "doris.urls":"10.10.10.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test_db",
    "converter.mode":"debezium_ingestion",
    "enable.delete":"true",
    "key.converter":"org.apache.kafka.connect.json.JsonConverter",
    "value.converter":"org.apache.kafka.connect.json.JsonConverter"
  }
}'
```

Load Data in AVRO Serialization Format

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"doris-avro-test",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
    "topics":"avro_topic",
    "tasks.max":"10",
    "doris.topic2table.map": "avro_topic:avro_tab",
    "buffer.count.records":"100000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"10000000",
    "doris.urls":"10.10.10.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test",
    "load.model":"stream_load",
    "key.converter":"io.confluent.connect.avro.AvroConverter",
    "key.converter.schema.registry.url":"http://127.0.0.1:8081",
    "value.converter":"io.confluent.connect.avro.AvroConverter",
    "value.converter.schema.registry.url":"http://127.0.0.1:8081"
  }
}'
```

Load Data in Protobuf Serialization Format

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"doris-protobuf-test",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
    "topics":"proto_topic",
    "tasks.max":"10",
    "doris.topic2table.map": "proto_topic:proto_tab",
    "buffer.count.records":"100000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"10000000",
    "doris.urls":"10.10.10.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test",
    "load.model":"stream_load",
    "key.converter":"io.confluent.connect.protobuf.ProtobufConverter",
    "key.converter.schema.registry.url":"http://127.0.0.1:8081",
```

```
    "value.converter":"io.confluent.connect.protobuf.ProtobufConverter",
    "value.converter.schema.registry.url":"http://127.0.0.1:8081"
  }
}'
```

Flink

Using the Flink Doris Connector, you can load data generated by Flink (such as data read from Kafka or MySQL) into Doris in real-time.

Limitations

Requires a user-deployed Flink cluster.

Loading Data Using Flink

For detailed steps on loading data using Flink, please refer to Flink-Doris-Connector. The following steps demonstrate how to quickly load data using Flink.

Step 1: Create Table

```
CREATE TABLE `students` (
  `id` INT NULL,
  `name` VARCHAR(256) NULL,
  `age` INT NULL
) ENGINE=OLAP
UNIQUE KEY(`id`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

Step 2: Load Data Using Flink

Run bin/sql-client.sh to open the FlinkSQL console

```
CREATE TABLE student_sink (
    id INT,
    name STRING,
    age INT
    )
    WITH (
      'connector' = 'doris',
      'fenodes' = '10.16.10.6:28737',
      'table.identifier' = 'test.students',
      'username' = 'root',
      'password' = '',
      'sink.label-prefix' = 'doris_label'
);

INSERT INTO student_sink values(1,'zhangsan',123)
```

Step 3: Verify Loaded Data

```
select * from test.students;
+------+----------+------+
| id   | name     | age  |
+------+----------+------+
|  1   | zhangsan |  123 |
+------+----------+------+
```

HDFS | Data Source

HDFS

Doris provides two ways to load files from HDFS: - Use HDFS Load to load HDFS files into Doris, which is an asynchronous load method. - Use TVF to load HDFS files into Doris, which is a synchronous load method.

load with HDFS Load

Use HDFS Load to import files on HDFS. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file hdfsload_example.csv The file is stored on HDFS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_hdfsload(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using HDFS Load

```
LOAD LABEL hdfs_load_2022_04_01
(
    DATA INFILE("hdfs://127.0.0.1:8020/tmp/hdfsload_example.csv")
    INTO TABLE test_hdfsload
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
with HDFS
(
    "fs.defaultFS" = "hdfs://127.0.0.1:8020",
    "hadoop.username" = "user"
```

```
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_hdfsload;
```

Results:

```
mysql> select * from test_hdfsload;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file hdfsload_example.csv The file is stored on HDFS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_hdfsload(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
```

```
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

```
INSERT INTO test_hdfsload
SELECT * FROM hdfs (
    "uri" = "hdfs://127.0.0.1:8020/tmp/hdfsload_example.csv",
    "fs.defaultFS" = "hdfs://127.0.0.1:8020",
    "hadoop.username" = "doris",
    "format" = "csv",
    "csv_schema" = "user_id:int;name:string;age:int"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_hdfsload;
```

Results:

```
mysql> select * from test_hdfsload;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Amazon S3

Doris provides two ways to load files from AWS S3: - Use S3 Load to load S3 files into Doris, which is an asynchronous load method. - Use TVF to load S3 files into Doris, which is a synchronous load method.

load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on S3 and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data with S3 Load

```
LOAD LABEL s3_load_2022_04_01
(
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "S3",
    "s3.endpoint" = "https://s3.us-west-2.amazonaws.com",
    "s3.region" = "us-west-2",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>"
```

```
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on S3 and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
```

```
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "s3.endpoint" = "https://s3.us-west-2.amazonaws.com",
    "s3.region" = "us-west-2",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Google Cloud Storage | Data Source

Google Cloud Storage

Doris provides two ways to load files from Google Cloud Storage: - Use S3 Load to load Google Cloud Storage files into Doris, which is an asynchronous load method. - Use TVF to load Google Cloud Storage files into Doris, which is a synchronous load method.

load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Google Cloud Storage and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using S3 Load

```
LOAD LABEL s3_load_2022_04_01
(
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "GCP",
```

```
    "s3.endpoint" = "storage.us-west2.rep.googleapis.com",
    "s3.region" = "US-WEST2",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Google Cloud Storage and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
```

```
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "provider" = "GCP",
    "s3.endpoint" = "storage.us-west2.rep.googleapis.com",
    "s3.region" = "US-WEST2",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Azure Storage

Doris provides two ways to load files from Azure Storage: - Use S3 Load to load Azure Storage files into Doris, which is an asynchronous load method. - Use TVF to load Azure Storage files into Doris, which is a synchronous load method.

load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Azure Storage and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using S3 Load

> **Caution**
>
> When importing data from Azure Storage with S3 Load, note the following:
>
> - Azure Storage requires HTTPS transmission by default (`Secure transfer required: Enabled`). To access Azure Storage properly, set `s3_client_http_scheme = https` in Doris `be.conf`.
> - The `s3.region` setting in Azure S3 properties can be omitted.

```
LOAD LABEL s3_load_2022_04_01
(
```

```
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "AZURE",
    "s3.endpoint" = "StorageAccountA.blob.core.windows.net",
    "s3.region" = "westus3",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Azure Storage and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
```

```
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

> **Caution**
>
> When importing data from Azure Storage with TVF, note the following:
>
> - Azure Storage requires HTTPS transmission by default (`Secure transfer required: Enabled`). To access Azure Storage properly, set `s3_client_http_scheme = https` in Doris `be.conf`.
> - The `s3.region` setting in Azure S3 properties can be omitted.

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "provider" = "AZURE",
    "s3.endpoint" = "StorageAccountA.blob.core.windows.net",
    "s3.region" = "westus3",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
```

```
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Alibaba Cloud OSS

Doris provides two ways to load files from Alibaba Cloud OSS: - Use S3 Load to load Alibaba Cloud OSS files into Doris, which is an asynchronous load method. - Use TVF to load Alibaba Cloud OSS files into Doris, which is a synchronous load method.

load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Alibaba Cloud OSS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using S3 Load

> **Caution**
>
> The endpoint URLs for public and internal access are different. If the server and OSS are in the same region, it is recommended to use the internal endpoint link.
>
> - internal endpoint: `oss-cn-hangzhou-internal.aliyuncs.com`
> - public endpoint: `oss-cn-hangzhou.aliyuncs.com`

```
LOAD LABEL s3_load_2022_04_01
(
```

```
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "OSS",
    "s3.endpoint" = "oss-cn-hangzhou.aliyuncs.com",
    "s3.region" = "oss-cn-hangzhou",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Alibaba Cloud OSS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
```

```
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

> **Caution**
>
> The endpoint URLs for public and internal access are different. If the server and OSS are in the same region, it is recommended to use the internal endpoint link.
>
> - internal endpoint: `oss-cn-hangzhou-internal.aliyuncs.com`
> - public endpoint: `oss-cn-hangzhou.aliyuncs.com`

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "provider" = "OSS",
    "s3.endpoint" = "oss-cn-hangzhou.aliyuncs.com",
    "s3.region" = "oss-cn-hangzhou",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
```

```
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Huawei Cloud OBS

Doris provides two ways to load files from Huawei Cloud OBS: - Use S3 Load to load Huawei Cloud OBS files into Doris, which is an asynchronous load method. - Use TVF to load Huawei Cloud OBS files into Doris, which is a synchronous load method.

Load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Huawei Cloud OBS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using S3 Load

```
LOAD LABEL s3_load_2022_04_01
(
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "OBS",
    "s3.endpoint" = "obs.cn-north-1.myhuaweicloud.com",
    "s3.region" = "cn-north-1",
    "s3.access_key" = "<your-ak>",
```

```
    "s3.secret_key" = "<your-sk>"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Result:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Huawei Cloud OBS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
```

```
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "provider" = "OBS",
    "s3.endpoint" = "obs.cn-north-1.myhuaweicloud.com",
    "s3.region" = "cn-north-1",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Tencent Cloud COS

Doris provides two ways to load files from Tencent Cloud COS: - Use S3 Load to load Tencent Cloud COS files into Doris, which is an asynchronous load method. - Use TVF to load Tencent Cloud COS files into Doris, which is a synchronous load method.

Load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Tencent Cloud COS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using S3 Load

```
LOAD LABEL s3_load_2022_04_01
(
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "COS",
    "s3.endpoint" = "cos.ap-beijing.myqcloud.com",
    "s3.region" = "ap-beijing",
    "s3.access_key" = "<your-ak>",
```

```
    "s3.secret_key" = "<your-sk>"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on Tencent Cloud COS and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
```

```
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "provider" = "COS",
    "s3.endpoint" = "cos.ap-beijing.myqcloud.com",
    "s3.region" = "ap-beijing",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

MinIO | Data Source

MinIO

Doris provides two ways to load files from MinIO: - Use S3 Load to load MinIO files into Doris, which is an asynchronous load method. - Use TVF to load MinIO files into Doris, which is a synchronous load method.

load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on MinIO and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using S3 Load

> **Caution**
>
> When importing data from MinIO with S3 Load, note the following:
>
> - If MinIO is deployed in a local network without TLS, explicitly add `http://` in the endpoint, for example: `"s3.endpoint" = "http://localhost:9000"`.
> - The S3 SDK uses virtual-hosted style by default, while MinIO does not enable it by default. Add `"use_path_style" = "true"` to force path-style access.

```
LOAD LABEL s3_load_2022_04_05
(
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "S3",
    "s3.endpoint" = "play.min.io:9000",
    "s3.region" = "us-east-1",
    "s3.access_key" = "myminioadmin",
    "s3.secret_key" = "minio-secret-key-change-me",
    "use_path_style" = "true"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on MinIO and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
```

```
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

> **Caution**
>
> When importing data from MinIO with TVF, note the following:
>
> - If MinIO is deployed in a local network without TLS, explicitly add `http://` in the endpoint, for example: `"s3.endpoint" = "http://localhost:9000"`.
> - The S3 SDK uses virtual-hosted style by default, while MinIO does not enable it by default. Add `"use_path_style" = "true"` to force path-style access.

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "provider" = "S3",
    "s3.endpoint" = "play.min.io:9000",
    "s3.region" = "us-east-1",
    "s3.access_key" = "myminioadmin",
    "s3.secret_key" = "minio-secret-key-change-me",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int",
    "use_path_style" = "true"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
```

```
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

S3 Compatible Storage

Doris provides two ways to load files from S3 Compatible Storage: - Use S3 Load to load S3 Compatible Storage files into Doris, which is an asynchronous load method. - Use TVF to load S3 Compatible Storage files into Doris, which is a synchronous load method.

> **Caution**
>
> The S3 SDK uses the virtual-hosted style by default. However, some object storage systems may not enable or support virtual-hosted style access. In this case, we can add the `use_path_style` parameter to force the use of the path style.

Load with S3 Load

Use S3 Load to import files on object storage. For detailed steps, please refer to the Broker Load Manual

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on S3 Compatible Storage and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using S3 Load

```
LOAD LABEL s3_load_2022_04_01
(
```

```
    DATA INFILE("s3://your_bucket_name/s3load_example.csv")
    INTO TABLE test_s3load
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
)
WITH S3
(
    "provider" = "S3",
    "s3.endpoint" = "play.min.io:9000",
    "s3.region" = "us-east-1",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "use_path_style" = "true"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Load with TVF

Step 1: Prepare the data

Create a CSV file s3load_example.csv The file is stored on S3 Compatible Storage and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
```

```
8,James,64
9,Emma,37
10,Liam,64
```

Step 2: Create a table in Doris

```
CREATE TABLE test_s3load(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

Step 3: Load data using TVF

```
INSERT INTO test_s3load
SELECT * FROM S3
(
    "uri" = "s3://your_bucket_name/s3load_example.csv",
    "format" = "csv",
    "provider" = "S3",
    "s3.endpoint" = "play.min.io:9000",
    "s3.region" = "us-east-1",
    "s3.access_key" = "<your-ak>",
    "s3.secret_key" = "<your-sk>",
    "column_separator" = ",",
    "csv_schema" = "user_id:int;name:string;age:int",
    "use_path_style" = "true"
);
```

Step 4: Check the imported data

```
SELECT * FROM test_s3load;
```

Results:

```
mysql> select * from test_s3load;
+---------+-----------+------+
| user_id | name      | age  |
+---------+-----------+------+
|       5 | Ava       |   17 |
|      10 | Liam      |   64 |
|       7 | Sophia    |   32 |
|       9 | Emma      |   37 |
|       1 | Emily     |   25 |
|       4 | Alexander |   60 |
|       2 | Benjamin  |   35 |
|       3 | Olivia    |   28 |
|       6 | William   |   69 |
|       8 | James     |   64 |
+---------+-----------+------+
10 rows in set (0.04 sec)
```

Snowflake

During the migration from Snowflake to Doris, object storage is typically used as an intermediate medium. The core process is as follows: First, export data to object storage using Snowflake's COPY INTO statement. Then, use Doris' S3 Load feature to read data from the object storage and load it into Doris. For details, refer to S3 Load.

Considerations

Before migration, select Doris' data model, partitioning, and bucketing strategies based on Snowflake's table structure. For more table creation strategies, refer to Load Best Practices.

Data type mapping

Snowflake	Doris	Comment
NUMBER(p, s)/DECIMAL(p, s)/NUMERIC(p,s)	DECIMAL(p, s)	
INT/INTEGER	INT	
TINYINT/BYTEINT	TINYINT	
SMALLINT	SMALLINT	
BIGINT	BIGINT	
FLOAT/FLOAT4/FLOAT8/DOUBLE/DOUBLE PRECISION/REAL	DOUBLE	
VARCHAR/STRING/TEXT	VARCHAR/STRING	VARCHAR maximum length is 65535
CHAR/CHARACTER/NCHAR	CHAR	
BINARY/VARBINARY	STRING	
BOOLEAN	BOOLEAN	
DATE	DATE	
DATETIME/TIMESTAMP/TIMESTAMP_NTZ	DATETIME	TIMESTAMP is a configurable alias (default: TIMESTAMP_NTZ)
TIME	STRING	Cast to String when exporting from Snowflake
VARIANT	VARIANT	
ARRAY	ARRAY<T>	
OBJECT	JSON	
GEOGRAPHY/GEOMETRY	STRING	

1. Create Table

To migrate a Snowflake table to Doris, first create the Doris table.

Assume we have the following table and data in Snowflake:

```
CREATE OR REPLACE TABLE sales_data (
    order_id      INT PRIMARY KEY,
    customer_name VARCHAR(128),
    order_date    DATE,
    amount        DECIMAL(10,2),
    country       VARCHAR(48)
)
CLUSTER BY (order_date);

INSERT INTO sales_data VALUES
(1, 'Alice', '2025-04-08', 99.99, 'USA'),
(2, 'Bob', '2025-04-08', 149.50, 'Canada'),
(3, 'Charlie', '2025-04-09', 75.00, 'UK'),
(4, 'Diana', '2025-04-10', 200.00, 'Australia');
```

Based on this structure, create a Doris Primary Key partitioned table aligned with Snowflake's clustering key, partitioned by day:

```
CREATE TABLE `sales_data` (
  order_id      INT,
  order_date    DATE NOT NULL,
  customer_name VARCHAR(128),
  amount        DECIMAL(10,2),
  country       VARCHAR(48)
) ENGINE=OLAP
UNIQUE KEY(`order_id`,`order_date`)
PARTITION BY RANGE(`order_date`) (
PARTITION p20250408 VALUES [('2025-04-08'), ('2025-04-09')),
PARTITION p20250409 VALUES [('2025-04-09'), ('2025-04-10')),
PARTITION p20250410 VALUES [('2025-04-10'), ('2025-04-11'))
)
DISTRIBUTED BY HASH(`order_id`) BUCKETS 16
PROPERTIES (
 "dynamic_partition.enable" = "true",
 "dynamic_partition.time_unit" = "DAY",
 "dynamic_partition.end" = "5",
 "dynamic_partition.prefix" = "p",
 "dynamic_partition.buckets" = "16",
 "replication_num" = "1"
);
```

2. Export Data from Snowflake

2.1. **Export to S3 Parquet Files via COPY INTO**

Snowflake supports exporting to AWS S3，GCS，AZURE，**Export data partitioned by Doris' partition fields**. Example for AWS S3:

```
```sql
CREATE FILE FORMAT my_parquet_format TYPE = parquet;
```
```

```
CREATE OR REPLACE STAGE external_stage
URL='s3://mybucket/sales_data'
CREDENTIALS=(AWS_KEY_ID='<ak>' AWS_SECRET_KEY='<sk>')
FILE_FORMAT = my_parquet_format;

COPY INTO @external_stage from sales_data PARTITION BY (CAST(order_date AS VARCHAR)) header=true;
```

2.2. **Verify Exported Files on S3**

Exported files are organized into **subdirectories by partition** on S3:

```
![snowflake_s3_out_en](/images/data-operate/snowflake_s3_out_en.png)

![snowflake_s3_out2_en](/images/data-operate/snowflake_s3_out2_en.png)
```

3. Load Data to Doris

S3 Load is an asynchronous data load method. After execution, Doris actively pulls data from the data source. The data source supports object storage compatible with the S3 protocol, including (AWS S3，GCS，AZURE，etc)

This method is suitable for scenarios involving large volumes of data that require asynchronous processing in the background. For data imports that need to be handled synchronously, refer to TVF Load。

*Note: For **Parquet/ORC format files that contain complex types (Struct/Array/Map)**, TVF Load must be used.*

3.1. **Load a Single Partition**

```
LOAD LABEL sales_data_2025_04_08
(
    DATA INFILE("s3://mybucket/sales_data/2025_04_08/*")
    INTO TABLE sales_data
    FORMAT AS "parquet"
    (order_id, order_date, customer_name, amount, country)
)
WITH S3
(
    "provider" = "S3",
    "s3.endpoint" = "https://s3.ap-southeast-1.amazonaws.com",
    "s3.access_key" = "<ak>",
    "s3.secret_key" = "<sk>",
    "s3.region" = "ap-southeast-1"
);
```

3.2. **Check Load Status via SHOW LOAD**

Since S3 Load import is submitted asynchronously, you can check the status of a specific label using SHOW LOAD:

```
mysql> show load where label = "sales_data_2025_04_08"\G
*************************** 1. row ***************************
         JobId: 17956078
         Label: label_sales_data_2025_04_08
```

```
         State: FINISHED
      Progress: 100.00% (1/1)
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=2
      TaskInfo: cluster:s3.ap-southeast-1.amazonaws.com; timeout(s):3600; max_filter_ratio:0.0; priority:NORMAL
      ErrorMsg: NULL
    CreateTime: 2025-04-10 17:50:53
  EtlStartTime: 2025-04-10 17:50:54
 EtlFinishTime: 2025-04-10 17:50:54
 LoadStartTime: 2025-04-10 17:50:54
LoadFinishTime: 2025-04-10 17:50:54
           URL: NULL
    JobDetails: {"Unfinished backends":{"5eec1be8612d4872-91040ff1e7208a4f":[]},"ScannedRows":2,"TaskNumber":1,"LoadBytes":91,"All
backends":{"5eec1be8612d4872-91040ff1e7208a4f":[10022]},"FileNumber":1,"FileSize":1620}
 TransactionId: 766228
  ErrorTablets: {}
          User: root
       Comment:
1 row in set (0.00 sec)
```

3.3. **Handle Load Errors**

When there are multiple load tasks, you can use the following statement to query the dates and reasons for data load failures.

```
mysql> show load where state='CANCELLED' and label like "label_test%"\G
*************************** 1. row ***************************
         JobId: 18312384
         Label: label_test123
         State: CANCELLED
      Progress: 100.00% (3/3)
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=4; dpp.norm.ALL=0
      TaskInfo: cluster:s3.ap-southeast-1.amazonaws.com; timeout(s):1440; max_filter_ratio:0.0; priority:NORMAL
      ErrorMsg: type:ETL_QUALITY_UNSATISFIED; msg:quality not good enough to cancel
    CreateTime: 2025-04-15 17:32:59
  EtlStartTime: 2025-04-15 17:33:02
 EtlFinishTime: 2025-04-15 17:33:02
 LoadStartTime: 2025-04-15 17:33:02
LoadFinishTime: 2025-04-15 17:33:02
           URL: http://10.16.10.6:28747/api/_load_error_log?file=__shard_2/error_log_insert_stmt_7602ccd7c3a4854-
95307efca7bfe342_7602ccd7c3a4854_95307efca7bfe342
    JobDetails: {"Unfinished backends":{"7602ccd7c3a4854-95307efca7bfe341":[]},"ScannedRows":4,"TaskNumber":1,"LoadBytes":188,"All
backends":{"7602ccd7c3a4854-95307efca7bfe341":[10022]},"FileNumber":3,"FileSize":4839}
 TransactionId: 769213
  ErrorTablets: {}
          User: root
       Comment:
```

As shown in the example above, the issue is a **data quality error**(ETL_QUALITY_UNSATISFIED). To view the detailed error, you need to visit the URL provided in the result. For example, the data exceeded the defined length of the country column in the table schema: `python [root@VM-10-6-centos ~]$ curl "http://10.16.10.6:28747/api/_load_error_log?file=__shard_2/error_log_insert_stmt_7602ccd7c3a4854-95307efca7bfe342_7602ccd7c3a4854_95307efca7bfe342" Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [USA] schema length: 1; actual length: 3; . src line []; Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [Canada] schema length: 1; actual length: 6; . src line []; Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [UK] schema length: 1; actual length: 2; . src line [];`

```
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str:
[Australia] schema length: 1; actual length: 9; . src line [];
```

For data quality errors, if you want to allow skipping erroneous records, you can set a fault tolerance rate in the Properties section of the S3 Load task. For details, refer to Import Configuration Parameters。

3.4. **Load data for multiple partitions**

When migrating a large volume of historical data, it is recommended to use a batch load strategy. Each batch corresponds to one or a few partitions in Doris. It is recommended to keep the data size under 100GB per batch to reduce system load and lower the cost of retries in case of load failures.

You can refer to the script s3_load_demo.sh, which can poll the partition directory on S3 and submit the S3 Load task to Doris to achieve batch load.

BigQuery

During the process of migrating BigQuery, it is usually necessary to use object storage as an intermediate medium. The core process is as follows: First, use BigQuery's Export statement to export data to GCS (Google Cloud Storage); then, use Doris's S3 Load function to read data from the object storage and load it into Doris. For details, please refer to S3 Load.

Considerations

1. Before the migration, it is necessary to select Doris' Data Model, as well as the strategies for Partitioning and Bucketing according to the table structure of BigQuery. For more table creation strategies, please refer to Load Best Practices.
2. When BigQuery exports data in JSON type, it does not support exporting in Parquet format. You can export it in JSON format instead.
3. When BigQuery exports data of the Time type, it is necessary to export it after casting it to the String type.

Data type mapping

BigQuery	Doris	Comment
Array	Array	
BOOLEAN	BOOLEAN	
DATE	DATE	
DATETIME/TIMESTAMP	DATETIME	
JSON	JSON	
INT64	BIGINT	
NUMERIC	DECIMAL	
FLOAT64	DOUBLE	
STRING	VARCHAR/STRING	VARCHAR maximum length is 65535
STRUCT	STRUCT	
TIME	STRING	
OTHER	UNSUPPORTED	

1. Create Table

When migrating a BigQuery table to Doris, it is necessary to create a Doris table first.

Suppose we already have the following table and data in BigQuery.

```
CREATE OR REPLACE TABLE test.sales_data (
    order_id      INT64,
    customer_name STRING,
    order_date    DATE,
    amount        NUMERIC(10,2),
    country       STRING
)
PARTITION BY  order_date;


INSERT INTO test.sales_data (order_id, customer_name, order_date, amount, country) VALUES
(1, 'Alice', '2025-04-08', 99.99, 'USA'),
(2, 'Bob', '2025-04-08', 149.50, 'Canada'),
(3, 'Charlie', '2025-04-09', 75.00, 'UK'),
(4, 'Diana', '2025-04-10', 200.00, 'Australia');
```

According to this table structure, a Doris primary key partitioned table can be created. The partition field should be the same as that in BigQuery, and the table should be partitioned on a daily basis.

```
CREATE TABLE `sales_data` (
  order_id      INT,
  order_date    DATE NOT NULL,
  customer_name VARCHAR(128),
  amount        DECIMAL(10,2),
  country       VARCHAR(48)
) ENGINE=OLAP
UNIQUE KEY(`order_id`,`order_date`)
PARTITION BY RANGE(`order_date`) (
PARTITION p20250408 VALUES [('2025-04-08'), ('2025-04-09')),
PARTITION p20250409 VALUES [('2025-04-09'), ('2025-04-10')),
PARTITION p20250410 VALUES [('2025-04-10'), ('2025-04-11'))
)
DISTRIBUTED BY HASH(`order_id`) BUCKETS 16
PROPERTIES (
 "dynamic_partition.enable" = "true",
 "dynamic_partition.time_unit" = "DAY",
 "dynamic_partition.end" = "5",
 "dynamic_partition.prefix" = "p",
 "dynamic_partition.buckets" = "16",
 "replication_num" = "1"
);
```

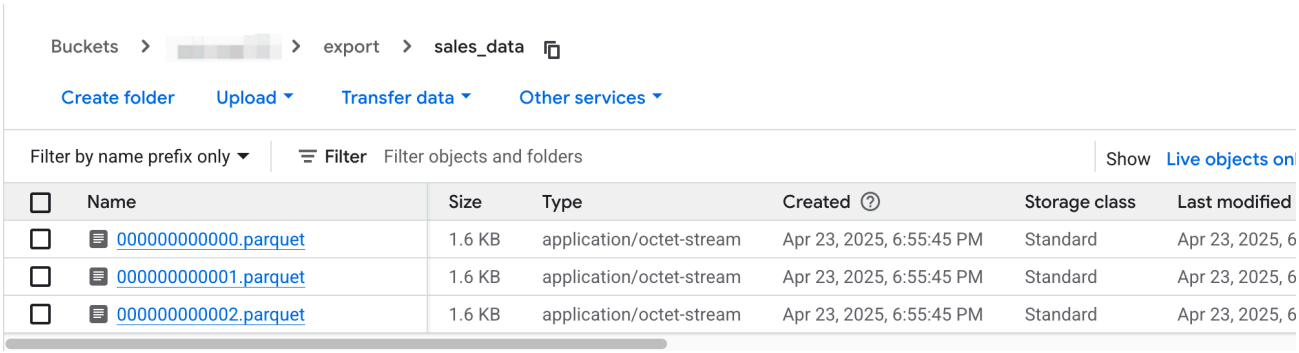
2. Export BigQuery Data

2.1. **Export to GCS Parquet format file through Export method**

```
EXPORT DATA
  OPTIONS (
    uri = 'gs://mybucket/export/sales_data/*.parquet',
    format = 'PARQUET')
AS (
  SELECT *
  FROM test.sales_data
);
```

2.2. **View the exported files on GCS**

The above command will export the data of sales_data to GCS, and each partition will generate one or more files with increasing file names. For details, please refer to exporting-data, as follows:



gcs_export

3. Load Data to Doris

S3 Load is an asynchronous data load method. After execution, Doris actively pulls data from the data source. The data source supports object storage compatible with the S3 protocol, including (AWS S3，GCS，AZURE，etc)

This method is suitable for scenarios involving large volumes of data that require asynchronous processing in the background. For data imports that need to be handled synchronously, refer to TVF Load。

*Note: For **Parquet/ORC format files that contain complex types (Struct/Array/Map)**, TVF Load must be used.*

3.1. **Loading data from a single file**

```
LOAD LABEL sales_data_2025_04_08
(
    DATA INFILE("s3://mybucket/export/sales_data/000000000000.parquet")
    INTO TABLE sales_data
    FORMAT AS "parquet"
    (order_id, order_date, customer_name, amount, country)
)
WITH S3
(
    "provider" = "GCP",
    "s3.endpoint" = "storage.asia-southeast1.rep.googleapis.com",
    "s3.region" = "asia-southeast1",
    "s3.access_key" = "<ak>",
    "s3.secret_key" = "<sk>"
);
```

3.2. **Check Load Status via SHOW LOAD**

Since S3 Load import is submitted asynchronously, you can check the status of a specific label using SHOW LOAD:

```
mysql> show load where label = "sales_data_2025_04_08"\G
*************************** 1. row ***************************
        JobId: 17956078
        Label: label_sales_data_2025_04_08
        State: FINISHED
```

```
      Progress: 100.00% (1/1)
          Type: BROKER
      EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=2
      TaskInfo: cluster:storage.asia-southeast1.rep.googleapis.com; timeout(s):3600; max_filter_ratio:0.0; priority:NORMAL
      ErrorMsg: NULL
    CreateTime: 2025-04-10 17:50:53
  EtlStartTime: 2025-04-10 17:50:54
EtlFinishTime: 2025-04-10 17:50:54
LoadStartTime: 2025-04-10 17:50:54
LoadFinishTime: 2025-04-10 17:50:54
          URL: NULL
    JobDetails: {"Unfinished backends":{"5eec1be8612d4872-91040ff1e7208a4f":[]},"ScannedRows":2,"TaskNumber":1,"LoadBytes":91,"All
backends":{"5eec1be8612d4872-91040ff1e7208a4f":[10022]},"FileNumber":1,"FileSize":1620}
TransactionId: 766228
  ErrorTablets: {}
          User: root
      Comment:
1 row in set (0.00 sec)
```

3.3. **Handle Load Errors**

When there are multiple load tasks, you can use the following statement to query the dates and reasons for data load failures.

```
mysql> show load where state='CANCELLED' and label like "label_test%"\G
*************************** 1. row ***************************
        JobId: 18312384
        Label: label_test123
        State: CANCELLED
      Progress: 100.00% (3/3)
          Type: BROKER
      EtlInfo: unselected.rows=0; dpp.abnorm.ALL=4; dpp.norm.ALL=0
      TaskInfo: cluster:storage.asia-southeast1.rep.googleapis.com; timeout(s):14400; max_filter_ratio:0.0; priority:NORMAL
      ErrorMsg: type:ETL_QUALITY_UNSATISFIED; msg:quality not good enough to cancel
    CreateTime: 2025-04-15 17:32:59
  EtlStartTime: 2025-04-15 17:33:02
EtlFinishTime: 2025-04-15 17:33:02
LoadStartTime: 2025-04-15 17:33:02
LoadFinishTime: 2025-04-15 17:33:02
          URL: http://10.16.10.6:28747/api/_load_error_log?file=__shard_2 error_log_insert_stmt_7602ccd7c3a4854-
95307efca7bfe342_7602ccd7c3a4854_95307efca7bfe342
    JobDetails: {"Unfinished backends":{"7602ccd7c3a4854-95307efca7bfe341":[]},"ScannedRows":4,"TaskNumber":1,"LoadBytes":188,"All
backends":{"7602ccd7c3a4854-95307efca7bfe341":[10022]},"FileNumber":3,"FileSize":4839}
TransactionId: 769213
  ErrorTablets: {}
          User: root
      Comment:
```

As shown in the example above, the issue is a **data quality error**(ETL_QUALITY_UNSATISFIED). To view the detailed error, you need to visit the URL provided in the result. For example, the data exceeded the defined length of the country column in the table schema:

```
[root@VM-10-6-centos ~]$ curl "http://10.16.10.6:28747/api/_load_error_log?file=__shard_2/error_log_insert_stmt_7602ccd7c3a4854-
95307efca7bfe342_7602ccd7c3a4854_95307efca7bfe342"
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [USA] schema length: 1;
actual length: 3; . src line [];
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [Canada] schema length: 1;
actual length: 6; . src line [];
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [UK] schema length: 1;
actual length: 2; . src line [];
```

```
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [Australia] schema length:
1; actual length: 9; . src line [];
```

For data quality errors, if you want to allow skipping erroneous records, you can set a fault tolerance rate in the Properties section of the S3 Load task. For details, refer to Load Configuration Parameters。

3.4. **Loading data from multiple files**

When migrating a large volume of historical data, it is recommended to use a batch load strategy. Each batch corresponds to one or a few partitions in Doris. It is recommended to keep the data size under 100GB per batch to reduce system load and lower the cost of retries in case of load failures.

You can refer to the script s3_load_file_demo.sh, which can split the file list under the specified directory on the object storage and submit multiple S3 Load tasks to Doris in batches to achieve the effect of batch load.

Redshift

During the process of migrating Redshift, it is usually necessary to use object storage as an intermediate medium. The core process is as follows: First, use the UNLOAD statement of Redshift to export data to the object storage; then, use the S3 Load function of Doris to read data from the object storage and import it into Doris. For details, please refer to S3 Load。

Considerations

1. Before the migration, it is necessary to select Doris' Data Model, as well as the strategies for Partitioning and Bucketing according to the table structure of Redshift. For more table creation strategies, please refer to Load Best Practices.
2. When Redshift exports data of the Time type, it is necessary to export it after casting it to the Varchar type.

Data type mapping

Redshift	Doris	Comment
SMALLINT	SMALLINT	
INTEGER	INT	
BIGINT	BIGINT	
DECIMAL	DECIMAL	
REAL	FLOAT	
DOUBLE PRECISION	DOUBLE	
BOOLEAN	BOOLEAN	
CHAR	CHAR	
VARCHAR	VARCHAR/STRING	VARCHAR maximum length is 65535
DATE	DATE	
TIMESTAMP	DATETIME	
TIME/TIMEZ	STRING	
SUPER	VARIANT	
OTHER	UNSUPPORTED	

1. Create Table

To migrate a Redshift table to Doris, first create the Doris table.

Assume we have the following table and data in Redshift:

```
CREATE TABLE sales_data (
    order_id      INTEGER,
    customer_name VARCHAR(128),
    order_date    DATE,
    amount        DECIMAL(10,2),
    country       VARCHAR(48)
)
DISTSTYLE AUTO;

INSERT INTO sales_data VALUES
(1, 'Alice', '2025-04-08', 99.99, 'USA'),
(2, 'Bob', '2025-04-08', 149.50, 'Canada'),
(3, 'Charlie', '2025-04-09', 75.00, 'UK'),
(4, 'Diana', '2025-04-10', 200.00, 'Australia');
```

According to this table structure, a Doris primary key partitioned table can be created. The partition field should be selected according to the business scenario. Here, the partition field is “order_date”, and it is partitioned on a daily basis.

```
CREATE TABLE `sales_data` (
  order_id      INT,
  order_date    DATE NOT NULL,
  customer_name VARCHAR(128),
  amount        DECIMAL(10,2),
  country       VARCHAR(48)
) ENGINE=OLAP
UNIQUE KEY(`order_id`,`order_date`)
PARTITION BY RANGE(`order_date`) (
PARTITION p20250408 VALUES [('2025-04-08'), ('2025-04-09')),
PARTITION p20250409 VALUES [('2025-04-09'), ('2025-04-10')),
PARTITION p20250410 VALUES [('2025-04-10'), ('2025-04-11'))
)
DISTRIBUTED BY HASH(`order_id`) BUCKETS 16
PROPERTIES (
 "dynamic_partition.enable" = "true",
 "dynamic_partition.time_unit" = "DAY",
 "dynamic_partition.end" = "5",
 "dynamic_partition.prefix" = "p",
 "dynamic_partition.buckets" = "16",
 "replication_num" = "1"
);
```

2. Export Data from Redshift

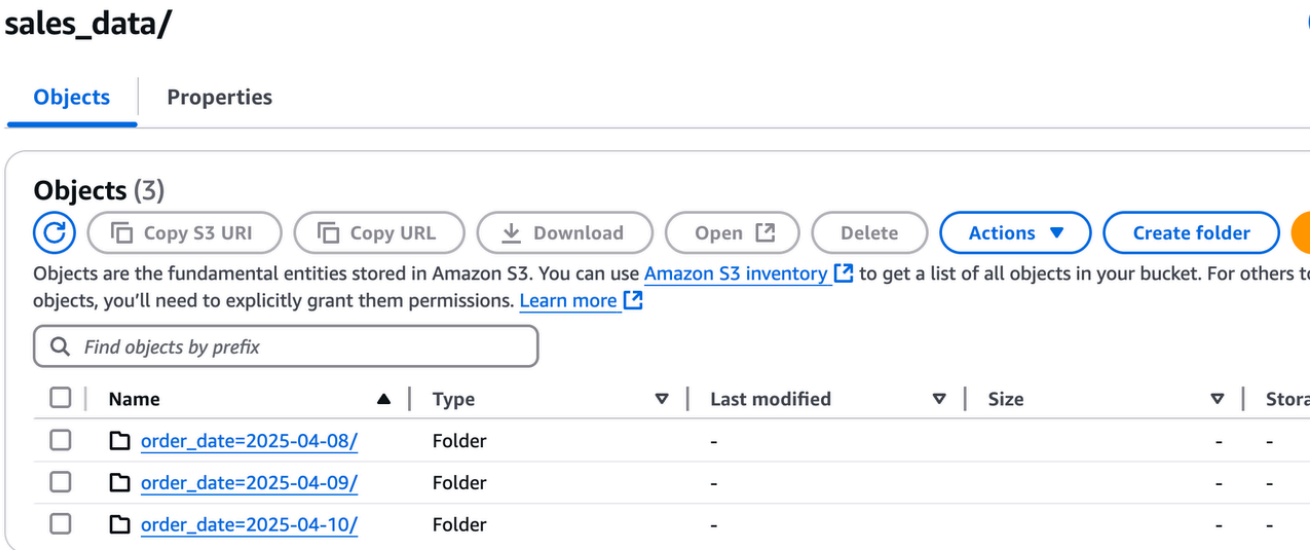
2.1 Export to S3 Parquet Files via UNLOAD

When exporting to S3, export according to the **Partition field of Doris**, as follows:

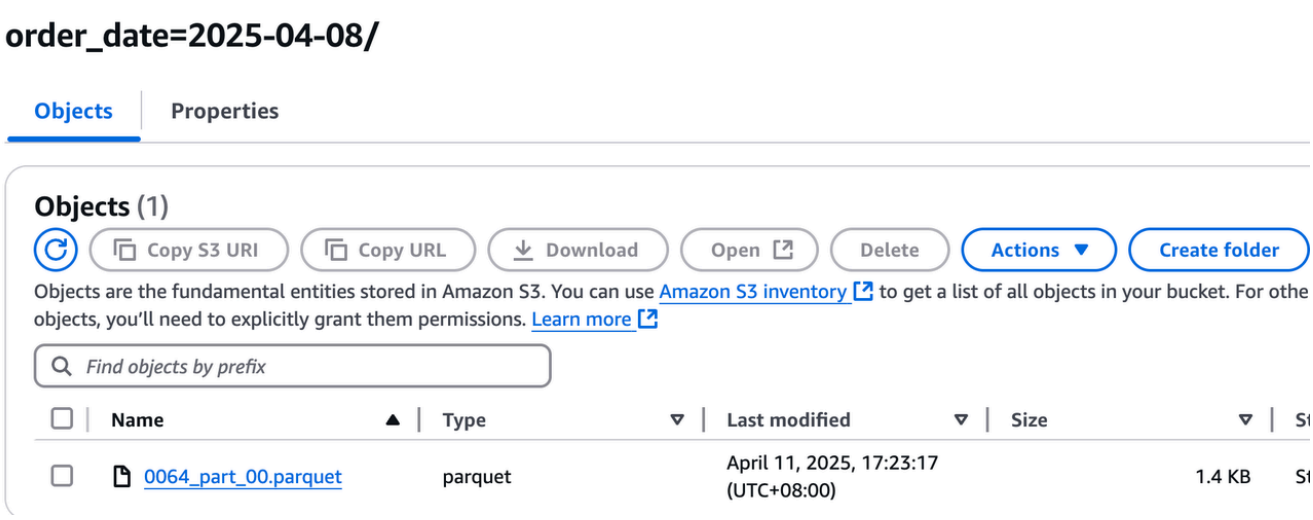
```
unload ('select * from sales_data')
to 's3://mybucket/redshift/sales_data/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
PARQUET
PARTITION BY (order_date) INCLUDE
```

2.2 Verify Exported Files on S3

Exported files are organized into **subdirectories by partition** on S3:



redshift_out



redshift_out2

3. Load Data to Doris

S3 Load is an asynchronous data load method. After execution, Doris actively pulls data from the data source. The data source supports object storage compatible with the S3 protocol, including (AWS S3，GCS，AZURE，etc)

This method is suitable for scenarios involving large volumes of data that require asynchronous processing in the background. For data imports that need to be handled synchronously, refer to TVF Load。

*Note: For **Parquet/ORC format files that contain complex types (Struct/Array/Map)**, TVF Load must be used.*

3.1 Load a Single Partition

```
LOAD LABEL sales_data_2025_04_08
(
    DATA INFILE("s3://mybucket/redshift/sales_data/order_date=2025-04-08/*")
    INTO TABLE sales_data
    FORMAT AS "parquet"
    (order_id, order_date, customer_name, amount, country)
```

```
)
WITH S3
(
    "provider" = "S3",
    "s3.endpoint" = "https://s3.ap-southeast-1.amazonaws.com",
    "s3.access_key" = "<ak>",
    "s3.secret_key"="<sk>",
    "s3.region" = "ap-southeast-1"
);
```

3.2 Check Load Status via SHOW LOAD

Since S3 Load import is submitted asynchronously, you can check the status of a specific label using SHOW LOAD:

```
mysql> show load where label = "sales_data_2025_04_08"\G
*************************** 1. row ***************************
         JobId: 17956078
         Label: sales_data_2025_04_08
         State: FINISHED
      Progress: 100.00% (1/1)
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=2
      TaskInfo: cluster:s3.ap-southeast-1.amazonaws.com; timeout(s):3600; max_filter_ratio:0.0; priority:NORMAL
      ErrorMsg: NULL
    CreateTime: 2025-04-10 17:50:53
  EtlStartTime: 2025-04-10 17:50:54
 EtlFinishTime: 2025-04-10 17:50:54
 LoadStartTime: 2025-04-10 17:50:54
LoadFinishTime: 2025-04-10 17:50:54
           URL: NULL
    JobDetails: {"Unfinished backends":{"5eec1be8612d4872-91040ff1e7208a4f":[]},"ScannedRows":2,"TaskNumber":1,"LoadBytes":91,"All
backends":{"5eec1be8612d4872-91040ff1e7208a4f":[10022]},"FileNumber":1,"FileSize":1620}
 TransactionId: 766228
  ErrorTablets: {}
          User: root
       Comment:
1 row in set (0.00 sec)
```

3.3 Handle Load Errors

When there are multiple load tasks, you can use the following statement to query the dates and reasons for data load failures.

```
mysql> show load where state='CANCELLED' and label like "label_test%"\G
*************************** 1. row ***************************
         JobId: 18312384
         Label: label_test123
         State: CANCELLED
      Progress: 100.00% (3/3)
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=4; dpp.norm.ALL=0
      TaskInfo: cluster:s3.ap-southeast-1.amazonaws.com; timeout(s):14400; max_filter_ratio:0.0; priority:NORMAL
      ErrorMsg: type:ETL_QUALITY_UNSATISFIED; msg:quality not good enough to cancel
    CreateTime: 2025-04-15 17:32:59
  EtlStartTime: 2025-04-15 17:33:02
 EtlFinishTime: 2025-04-15 17:33:02
 LoadStartTime: 2025-04-15 17:33:02
LoadFinishTime: 2025-04-15 17:33:02
           URL: http://10.16.10.6:28747/api/_load_error_log?file=__shard_2/error_log_insert_stmt_7602ccd7c3a4854-
```

```
95307efca7bfe342_7602ccd7c3a4854_95307efca7bfe342
    JobDetails: {"Unfinished backends":{"7602ccd7c3a4854-95307efca7bfe341":[]},"ScannedRows":4,"TaskNumber":1,"LoadBytes":188,"All
backends":{"7602ccd7c3a4854-95307efca7bfe341":[10022]},"FileNumber":3,"FileSize":4839}
 TransactionId: 769213
  ErrorTablets: {}
          User: root
       Comment:
```

As shown in the example above, the issue is a **data quality error**(ETL_QUALITY_UNSATISFIED). To view the detailed error, you need to visit the URL provided in the result. For example, the data exceeded the defined length of the country column in the table schema:

```
[root@VM-10-6-centos ~]$ curl "http://10.16.10.6:28747/api/_load_error_log?file=__shard_2/error_log_insert_stmt_7602ccd7c3a4854-
95307efca7bfe342_7602ccd7c3a4854_95307efca7bfe342"
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [USA] schema length: 1;
actual length: 3; . src line [];
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [Canada] schema length: 1;
actual length: 6; . src line [];
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [UK] schema length: 1;
actual length: 2; . src line [];
Reason: column_name[country], the length of input is too long than schema. first 32 bytes of input str: [Australia] schema length:
1; actual length: 9; . src line [];
```

For data quality errors, if you want to allow skipping erroneous records, you can set a fault tolerance rate in the Properties section of the S3 Load task. For details, refer to Import Configuration Parameters。

3.4 Load data for multiple partitions

When migrating a large volume of historical data, it is recommended to use a batch load strategy. Each batch corresponds to one or a few partitions in Doris. It is recommended to keep the data size under 100GB per batch to reduce system load and lower the cost of retries in case of load failures.

You can refer to the script s3_load_demo.sh, which can poll the partition directory on S3 and submit the S3 Load task to Doris to achieve batch load.

Migrating Data from Other OLAP

To migrate data from other OLAP systems to Doris, you have several options:

- For systems like Hive/Iceberg/Hudi, you can use Multi-Catalog to map them as external tables and then use “Insert Into” to load the data
- You can export data from the OLAP system into formats like CSV, and then load these data files into Doris
- You can use systems like Spark/Flink, utilizing the OLAP system’s Connector to read data, and then call the Doris Connector to write into Doris

> **NOTE**
>
> If you know of other migration tools that could be added to this list, please contact dev@doris.apache.org

Migrating Data from Other OLTP

There are various ways to migrate data from other TP systems, such as MySQL/SqlServer/Oracle, to Doris.

Multi-Catalog

Use the Catalog to map as an external table, and then use the INSERT INTO or CREATE-TABLE-AS-SELECT statements to complete the data load.

For example, with MySQL:

```
CREATE CATALOG mysql_catalog properties(
    'type' = 'jdbc',
    'user' = 'root',
    'password' = '123456',
    'jdbc_url' = 'jdbc:mysql://host:3306/mysql_db',
    'driver_url' = 'mysql-connector-java-8.0.25.jar',
    'driver_class' = 'com.mysql.cj.jdbc.Driver'
);

-- Load via INSERT
INSERT INTO internal.doris_db.tbl1
SELECT * FROM mysql_catalog.mysql_db.table1;

-- Load via CTAS
CREATE TABLE internal.doris_db.tbl1
PROPERTIES('replication_num' = '1')
AS
SELECT * FROM mysql_catalog.mysql_db.table1;
```

For more details, refer to Catalog Data Load。

Flink Doris Connector

You can leverage Flink to achieve offline and real-time synchronization for TP systems.

- Offline synchronization can be done using Flink's JDBC Source and Doris Sink to complete the data load. For example, using FlinkSQL:

  ```
  CREATE TABLE student_source (
      id INT,
      name STRING,
      age INT
    PRIMARY KEY (id) NOT ENFORCED
  ) WITH (
    'connector' = 'jdbc',
    'url' = 'jdbc:mysql://localhost:3306/mydatabase',
    'table-name' = 'students',
    'username' = 'username',
    'password' = 'password',
  );

  CREATE TABLE student_sink (
      id INT,
  ```

```
    name STRING,
    age INT
    )
    WITH (
      'connector' = 'doris',
      'fenodes' = '127.0.0.1:8030',
      'table.identifier' = 'test.students',
      'username' = 'root',
      'password' = 'password',
      'sink.label-prefix' = 'doris_label'
);

INSERT into student_sink select * from student_source;
```

For more details, refer to Flink JDBC。

- Real-time synchronization can be achieved using FlinkCDC to read both full and incremental data. For example, using FlinkSQL:

```
SET 'execution.checkpointing.interval' = '10s';

CREATE TABLE cdc_mysql_source (
  id int
  ,name VARCHAR
  ,PRIMARY KEY (id) NOT ENFORCED
) WITH (
'connector' = 'mysql-cdc',
'hostname' = '127.0.0.1',
'port' = '3306',
'username' = 'root',
'password' = 'password',
'database-name' = 'database',
'table-name' = 'table'
);

-- Supports synchronization of insert/update/delete events.
CREATE TABLE doris_sink (
id INT,
name STRING
)
WITH (
  'connector' = 'doris',
  'fenodes' = '127.0.0.1:8030',
  'table.identifier' = 'database.table',
  'username' = 'root',
  'password' = '',
  'sink.properties.format' = 'json',
  'sink.properties.read_json_by_line' = 'true',
  'sink.enable-delete' = 'true',  -- Synchronize delete events.
  'sink.label-prefix' = 'doris_label'
);

insert into doris_sink select id,name from cdc_mysql_source;
```

For synchronizing an entire database or multiple tables in a TP database, you can use the full-database synchronization feature provided by the Flink Doris Connector to complete the TP database write with a single click, as shown below:

```
<FLINK_HOME>bin/flink run \
    -Dexecution.checkpointing.interval=10s \
```

```
    -Dparallelism.default=1 \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    lib/flink-doris-connector-1.16-24.0.1.jar \
    mysql-sync-database \
    --database test_db \
    --mysql-conf hostname=127.0.0.1 \
    --mysql-conf port=3306 \
    --mysql-conf username=root \
    --mysql-conf password=123456 \
    --mysql-conf database-name=mysql_db \
    --including-tables "tbl1|test.*" \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password=123456 \
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

For more details, refer to Full Database Synchronization

Spark Connector

You can use the JDBC Source and Doris Sink of the Spark Connector to complete the data write.

```
val jdbcDF = spark.read
  .format("jdbc")
  .option("url", "jdbc:postgresql:dbserver")
  .option("dbtable", "schema.tablename")
  .option("user", "username")
  .option("password", "password")
  .load()

 jdbcDF.write.format("doris")
  .option("doris.table.identifier", "db.table")
  .option("doris.fenodes", "127.0.0.1:8030")
  .option("user", "root")
  .option("password", "")
  .save()
```

For more details, refer to JDBC To Other Databases， Spark Doris Connector

NineData / DataX / Seatunnel / CloudCanal and other third-party tools.

In addition, you can also use third-party synchronization tools for data synchronization. For more details, please refer to: - NineData - DataX - Seatunnel - CloudCanal

Stream Load

Stream Load supports importing local files or data streams into Doris through the HTTP protocol.

Stream Load is a synchronous import method that returns the import result after the import is executed, allowing you to determine the success of the import through the request response. Generally, users can use Stream Load to import files under 10GB. If the file is too large, it is recommended to split the file and then use Stream Load for importing. Stream Load can ensure the atomicity of a batch of import tasks, meaning that either all of them succeed or all of them fail.

> **In comparison to single-threaded load using `curl`, Doris Streamloader is a client tool designed for loading data into Apache Doris. It reduces the ingestion latency of large datasets by its concurrent loading capabilities. It comes with the following features:**
>
> - **Parallel loading**: multi-threaded load for the Stream Load method. You can set the parallelism level using the `workers` parameter.
> - **Multi-file load:** simultaneously loading of multiple files and directories with one shot. It supports recursive file fetching and allows you to specify file names with wildcard characters.
> - **Path traversal support:** supports path traversal when the source files are in directories
> - **Resilience and continuity:** in case of partial load failures, it can resume data loading from the point of failure.
> - **Automatic retry mechanism:** in case of loading failures, it can automatically retry a default number of times. If the loading remains unsuccessful, it will print the command for manual retry.
>
> See Doris Streamloader for detailed instructions and best practices.

User guide

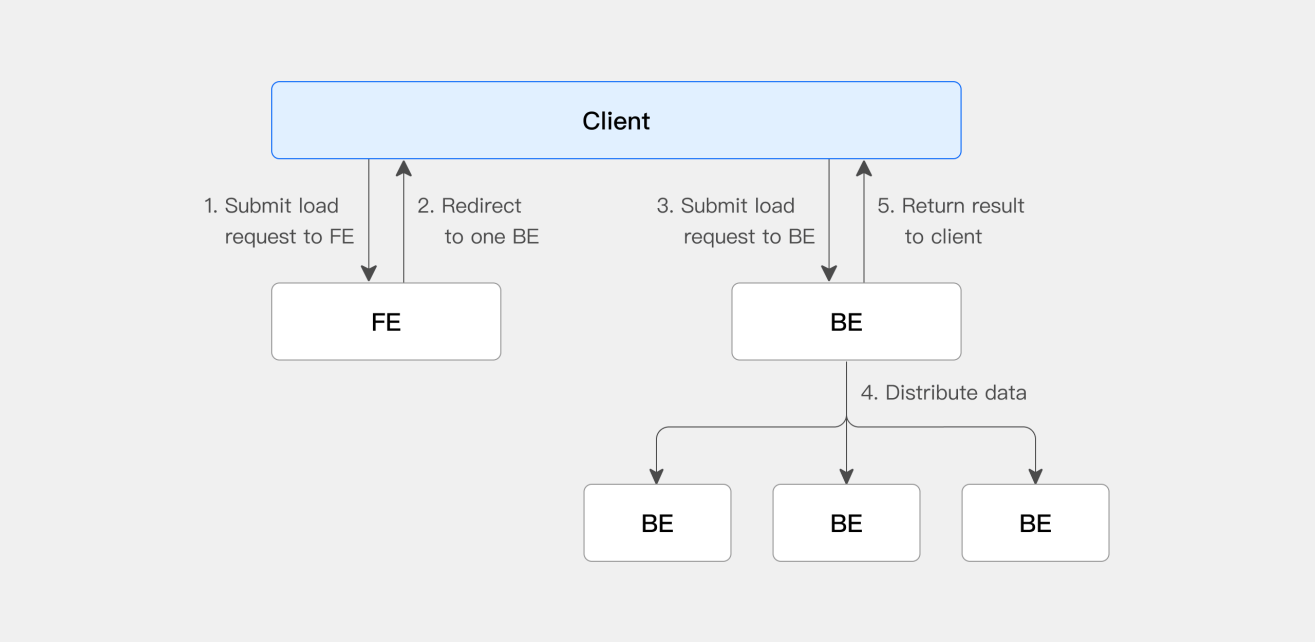
Stream Load supports importing CSV, JSON, Parquet, and ORC format data from local or remote sources via HTTP.

- Null values: Use `\N` to represent null. For example, `a,\N,b` indicates the middle column is null.
- Empty string: An empty string is represented when there are no characters between two delimiters. For example, in `a,,b`, there are no characters between the two commas, indicating that the middle column value is an empty string.

Basic principles

When using Stream Load, it is necessary to initiate an import job through the HTTP protocol to the FE (Frontend) node. The FE will redirect the request to a BE (Backend) node in a round-robin manner to achieve load balancing. It is also possible to send HTTP requests directly to a specific BE node. In Stream Load, Doris selects one node to serve as the Coordinator node. The Coordinator node is responsible for receiving data and distributing it to other nodes.

The following figure shows the main flow of Stream Load, omitting some import details.



Basic principles

1. The client submits a Stream Load imports job request to the FE (Frontend).
2. The FE selects a BE (Backend) as the Coordinator node in a round-robin manner, which is responsible for scheduling the import job, and then returns an HTTP redirect to the client.
3. The client connects to the Coordinator BE node and submits the import request.
4. The Coordinator BE distributes the data to the appropriate BE nodes and returns the import result to the client once the import is complete.
5. Alternatively, the client can directly specify a BE node as the Coordinator and distribute the import job directly.

Quick start

Stream Load import data through the HTTP protocol. The following example uses the curl tool to demonstrate submitting an import job through Stream Load.

Prerequisite check

Stream Load requires `INSERT` privileges on the target table. If there are no `INSERT` privileges, it can be granted to the user through the GRANT command.

Create load job

Loading CSV

1. Creating loading data

   Create a CSV file named `streamload_example.csv`. The specific content is as follows

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
```

```
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

2. Creating a table for loading

   Create the table that will be imported into, using the specific syntax as follows:

```
CREATE TABLE testdb.test_streamload(
    user_id            BIGINT       NOT NULL COMMENT "User ID",
    name               VARCHAR(20)           COMMENT "User name",
    age                INT                   COMMENT "User age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

3. Enable the load job

   The Stream Load job can be submitted using the `curl` command.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Stream Load is a synchronous method, where the result is directly returned to the user.

```
{
    "TxnId": 3,
    "Label": "123",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 10,
    "NumberLoadedRows": 10,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 118,
    "LoadTimeMs": 173,
    "BeginTxnTimeMs": 1,
    "StreamLoadPutTimeMs": 70,
    "ReadDataTimeMs": 2,
    "WriteDataTimeMs": 48,
    "CommitAndPublishTimeMs": 52
}
```

4. View data

```
mysql> select count(*) from testdb.test_streamload;
+----------+
| count(*) |
+----------+
```

```
|       10 |
+----------+
```

Loading JSON

1. Creating loading data

Create a JSON file named `streamload_example.json` . The specific content is as follows

```
[
{"userid":1,"username":"Emily","userage":25},
{"userid":2,"username":"Benjamin","userage":35},
{"userid":3,"username":"Olivia","userage":28},
{"userid":4,"username":"Alexander","userage":60},
{"userid":5,"username":"Ava","userage":17},
{"userid":6,"username":"William","userage":69},
{"userid":7,"username":"Sophia","userage":32},
{"userid":8,"username":"James","userage":64},
{"userid":9,"username":"Emma","userage":37},
{"userid":10,"username":"Liam","userage":64}
]
```

2. Creating a table for loading

   Create the table that will be imported into, using the specific syntax as follows:

```
CREATE TABLE testdb.test_streamload(
    user_id            BIGINT       NOT NULL COMMENT "User ID",
    name               VARCHAR(20)           COMMENT "User name",
    age                INT                   COMMENT "User age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

3. Enabling the load job

   The Stream Load job can be submitted using the `curl` command.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "label:124" \
    -H "Expect:100-continue" \
    -H "format:json" -H "strip_outer_array:true" \
    -H "jsonpaths:[\"$.userid\", \"$.username\", \"$.userage\"]" \
    -H "columns:user_id,name,age" \
    -T streamload_example.json \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

> **Note**
>
> If the JSON file is not a JSON array but each line is a JSON object, add the headers `-H "strip_outer_array:false"` and `-H "read_json_by_line:true"`.

> If you need to load the JSON object at the root node of a JSON file, the jsonpaths should be specified as $., e.g., `-H "jsonpaths:[\"$."]`"

Stream Load is a synchronous method, where the result is directly returned to the user.

```
{
    "TxnId": 7,
    "Label": "125",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 10,
    "NumberLoadedRows": 10,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 471,
    "LoadTimeMs": 52,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 11,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 23,
    "CommitAndPublishTimeMs": 16
}
```

View load job

By default, Stream Load synchronously returns results to the client, so the system does not record Stream Load historical jobs. If recording is required, add the configuration `enable_stream_load_record=true` in `be.conf`. Refer to the BE configuration options for specific details.

After configuring, you can use the `show stream load` command to view completed Stream Load jobs.

```
mysql> show stream load from testdb;
+-------+--------+-----------------+---------------+---------+---------+------+-----------+------------+--------------+-------------
---+-----------+-------------------------+-------------------------+------+---------+
| Label | Db     | Table           | ClientIp      | Status  | Message | Url  | TotalRows | LoadedRows | FilteredRows |
UnselectedRows | LoadBytes | StartTime               | FinishTime              | User | Comment |
+-------+--------+-----------------+---------------+---------+---------+------+-----------+------------+--------------+-------------
---+-----------+-------------------------+-------------------------+------+---------+
| 12356 | testdb | test_streamload | 192.168.88.31 | Success | OK      | N/A  | 10        | 10         | 0            | 0
| 118       | 2023-11-29 08:53:00.594 | 2023-11-29 08:53:00.650 | root |         |
+-------+--------+-----------------+---------------+---------+---------+------+-----------+------------+--------------+-------------
---+-----------+-------------------------+-------------------------+------+---------+
1 row in set (0.00 sec)
```

Cancel load job

Users cannot manually cancel a Stream Load operation. A Stream Load job will be automatically canceled by the system if it encounters a timeout (set to 0) or an import error.

Reference manual

Command

The syntax for Stream Load is as follows:

```
curl --location-trusted -u <doris_user>:<doris_password> \
  -H "Expect:100-continue" [-H ""...] \
  -T <file_path> \
  -XPUT http://fe_host:http_port/api/{db}/{table}/_stream_load
```

Stream Load operation supports both HTTP chunked and non-chunked import methods. For non-chunked imports, it is necessary to have a Content-Length header to indicate the length of the uploaded content, which ensures data integrity.

Load configuration parameters

FE configuration

1. `stream_load_default_timeout_second`
    - Default Value: 259200 (s)
    - Dynamic Configuration: Yes
    - FE Master-only Configuration: Yes

Parameter Description: The default timeout for Stream Load. The load job will be canceled by the system if it is not completed within the set timeout (in seconds). If the source file cannot be imported within the specified time, the user can set an individual timeout in the Stream Load request. Alternatively, adjust the `stream_load_default_timeout_second` parameter on the FE to set the global default timeout.

BE configuration

1. `streaming_load_max_mb`
    - Default value: 10240 (MB)
    - Dynamic configuration: Yes
    - Parameter description: The maximum import size for Stream Load. If the user's original file exceeds this value, the `streaming_load_max_mb` parameter on the BE needs to be adjusted.
2. Header parameters

    Load parameters can be passed through the HTTP Header section. See below for specific parameter descriptions.

Parameters	Parameters description
label	Used to specify a label for this Doris import. Data with the same label cannot be imported multiple times. If no label is specified, Doris will automatically generate one. Users can avoid duplicate imports of the same data by specifying a label. Doris retains import job labels for three days by default, but this duration can be adjusted using `label_keep_max_second`. For example, to specify the label for this import as 123, use the command `-H "label:123"`. The use of labels prevents users from importing the same data repeatedly. It is strongly recommended that users use the same label for the same batch of data. This ensures that duplicate requests for the same batch of data are only accepted once, guaranteeing At-Most-Once semantics. When the status of an import job corresponding to a label is CANCELLED, that label can be used again.
column_separator	Used to specify the column separator in the import file, which defaults to `\t`. If the separator is an invisible character, it needs to be prefixed with `\x` and represented in hexadecimal format. Multiple characters can be combined as a column separator. For example, to specify the separator as `\x01` for a Hive file, use the command `-H "column_separator:\x01"`.
line_delimiter	Used to specify the line delimiter in the import file, which defaults to `\n`. Multiple characters can be combined as a line delimiter. For example, to specify the line delimiter as `\n`, use the command `-H "line_delimiter:\n"`.
columns	Used to specify the correspondence between columns in the import file and columns in the table. If the columns in the source file exactly match the content of the table, this field does not need to be specified. If the schema of the source file does not match the table, this field is required for data transformation. There are two formats: direct column correspondence to fields in the import file, and derived columns represented by expressions. Refer to Data Transformation for detailed examples.
where	Used to filter out unnecessary data. If users need to exclude certain data, they can achieve this by setting this option. For example, to import only data where the k1 column is equal to 20180601, specify `-H "where: k1 = 20180601"` during the import.
max_filter_ratio	Used to specify the maximum tolerable ratio of filterable (irregular or otherwise problematic) data, which defaults to zero tolerance. The value range is 0 to 1. If the error rate of the imported data exceeds this value, the import will fail. Irregular data does not include rows filtered out by the where condition. For example, to maximize the import of all correct data (100% tolerance), specify the command `-H "max_filter_ratio:1"`.
partitions	Used to specify the partitions involved in this import. If users can determine the corresponding partitions for the data, it is recommended to specify this option. Data that does not meet these partition criteria will be filtered out. For example, to specify importing into partitions p1 and p2, use the command `-H "partitions: p1, p2"`.
timeout	Used to specify the timeout for the import in seconds. The default is 600 seconds, and the configurable range is from 1 second to 259200 seconds. For example, to specify an import timeout of 1200 seconds, use the command `-H "timeout:1200"`.
strict_mode	Used to specify whether to enable strict mode for this import, which is disabled by default. For example, to enable strict mode, use the command `-H "strict_mode:true"`.
timezone	Used to specify the timezone to be used for this import, which defaults to GMT+8. This parameter affects the results of all timezone-related functions involved in the import. For example, to specify the import timezone as Africa/Abidjan, use the command `-H "timezone:Africa/Abidjan"`.
exec_mem_limit	The memory limit for the import, which defaults to 2GB. The unit is bytes.

Parameters	Parameters description
format	Used to specify the format of the imported data, which defaults to CSV. Currently supported formats include: CSV, JSON, arrow, csv_with_names (supports filtering the first row of the csv file), csv_with_names_and_types (supports filtering the first two rows of the csv file), Parquet, and ORC. For example, to specify the imported data format as JSON, use the command `-H "format:json"`.
jsonpaths	There are two ways to import JSON data format: Simple Mode and Matching Mode. If no jsonpaths are specified, it is the simple mode that requires the JSON data to be of the object type.Matching mode used when the JSON data is relatively complex and requires matching the corresponding values through the jsonpaths parameter.In simple mode, the keys in JSON are required to correspond one-to-one with the column names in the table. For example, in the JSON data `{"k1":1, "k2":2, "k3":"hello"}`, k1, k2, and k3 correspond to the columns in the table respectively.
strip_outer_array	When `strip_outer_array` is set to true, it indicates that the JSON data starts with an array object and flattens the objects within the array. The default value is false. When the outermost layer of the JSON data is represented by `[]`, which denotes an array, `strip_outer_array` should be set to true. For example, with the following data, setting `strip_outer_array` to true will result in two rows of data being generated when imported into Doris: `[{"k1": 1, "v1": 2}, {"k1": 3, "v1": 4}]`.
json_root	`json_root` is a valid jsonpath string that specifies the root node of a JSON document, with a default value of “”.
merge_type	The merge type of data. Three types are supported: - APPEND (default): Indicates that all data in this batch will be appended to existing data - DELETE: Indicates deletion of all rows with Keys matching this batch of data - MERGE: Must be used in conjunction with DELETE conditions. Data meeting DELETE conditions will be processed according to DELETE semantics, while the rest will be processed according to APPEND semantics For example, to specify merge mode as MERGE: `-H "merge_type: MERGE" -H "delete: flag=1"`
delete	It is only meaningful under MERGE, representing the deletion conditions for data.
function_column.sequence_col	It is suitable only for the UNIQUE KEYS model. Within the same Key column, it ensures that the Value column is replaced according to the specified source_sequence column. The source_sequence can either be a column from the data source or an existing column in the table structure.
fuzzy_parse	It is a boolean type. If set to true, the JSON will be parsed with the first row as the schema. Enabling this option can improve the efficiency of JSON imports, but it requires that the order of the keys in all JSON objects be consistent with the first line. The default is false and it is only used for JSON format.
num_as_string	It is a boolean type. When set to true, indicates that numeric types will be converted to strings during JSON parsing to ensure no loss of precision during the import process.
read_json_by_line	It is a boolean type. When set to true, indicates support for reading one JSON object per line, defaulting to false.
send_batch_parallelism	Sets the parallelism for the parallelism for sending batch-processed data. If the parallelism value exceeds the `max_send_batch_parallelism_per_job` configured in BE, the coordinating BE will use the `max_send_batch_parallelism_per_job value`.
hidden_columns	Used to specify hidden columns in the imported data, which takes effect when the Header does not include Columns. Multiple hidden columns are separated by commas. The system will use the user-specified data for import. In the following example, the last column of data in the imported data is `__DORIS_SEQUENCE_COL__`. `hidden_columns: __DORIS_DELETE_SIGN__,__DORIS_SEQUENCE_COL__`.

Parameters	Parameters description
load_to_single_tablet	It is a boolean type. When set to true, indicates support for importing data only to a single Tablet corresponding to the partition, defaulting to false. This parameter is only allowed when importing to an OLAP table with random bucketing.
compress_type	Currently, only compression of CSV files is supported. Compression formats include gz, lzo, bz2, lz4, lzop, and deflate.
trim_double_quotes	It is a boolean type. When set to true, indicates trimming of the outermost double quotes for each field in the CSV file, defaulting to false.
skip_lines	It is an integer type. Used to specify the number of lines to skip at the beginning of the CSV file, defaulting to 0. When the `format` is set to `csv_with_names` or `csv_with_names_and_types`, this parameter will become invalid.
comment	It is a String type, with an empty string as the default value. Used to add additional information to the task.
enclose	Specify the enclosure character. When a CSV data field contains a row delimiter or column delimiter, to prevent unexpected truncation, you can specify a single-byte character as the enclosure for protection. For example, if the column delimiter is “,” and the enclosure is ““”, the data”a,’b,c’” will have “b,c” parsed as a single field. Note: When the enclosure is set to a double quote (“), make sure to set `trim_double_quotes` to true.
escape	Specify the escape character. It is used to escape characters that are the same as the enclosure character within a field. For example, if the data is “a,‘b,’c’”, and the enclosure is ““”, and you want”b,’c” to be parsed as a single field, you need to specify a single-byte escape character, such as ““, and modify the data to”a,’b’,‘c’”.
memtable_on_sink_node	Whether to enable MemTable on DataSink node when loading data, default is false.
unique_key_update_mode	The update modes on Unique tables, currently are only effective for Merge-On-Write Unique tables. Supporting three types: `UPSERT`, `UPDATE_FIXED_COLUMNS`, and `UPDATE_FLEXIBLE_COLUMNS`. `UPSERT`: Indicates that data is loaded with upsert semantics; `UPDATE_FIXED_COLUMNS`: Indicates that data is loaded through partial updates; `UPDATE_FLEXIBLE_COLUMNS`: Indicates that data is loaded through flexible partial updates.

Load return value

Stream Load is a synchronous import method, and the load result is directly provided to the user through the creation of an load return value, as shown below:


```
{
    "TxnId": 1003,
    "Label": "b6f3bc78-0d2c-45d9-9e4c-faa0a0149bee",
    "Status": "Success",
    "ExistingJobStatus": "FINISHED", // optional
    "Message": "OK",
    "NumberTotalRows": 1000000,
    "NumberLoadedRows": 1000000,
    "NumberFilteredRows": 1,
    "NumberUnselectedRows": 0,
    "LoadBytes": 40888898,
    "LoadTimeMs": 2144,
    "BeginTxnTimeMs": 1,
    "StreamLoadPutTimeMs": 2,
    "ReadDataTimeMs": 325,
    "WriteDataTimeMs": 1933,
    "CommitAndPublishTimeMs": 106,
    "ErrorURL": "http://192.168.1.1:8042/api/_load_error_log?file=__shard_0/error_log_insert_stmt_db18266d4d9b4ee5-
```

```
abb00ddd64bdf005_db18266d4d9b4ee5_abb00ddd64bdf005"
}
```

The return result parameters are explained in the following table:

Parameters	Parameters description
TxnId	Import transaction ID
Label	Label of load job，specified via `-H "label:<label_id>"`.
Status	Final load Status. **Success**: The load job was successful.**Publish Timeout**: The load job has been completed, but there may be a delay in data visibility. **Label Already Exists**: The label is duplicated, requiring a new label. **Fail**: The load job failed.
ExistingJobStatus	The status of the load job corresponding to the already existing label. This field is only displayed when the Status is **Label Already Exists**. Users can use this status to know the status of the import job corresponding to the existing label. **RUNNING** means the job is still executing, and **FINISHED** means the job was successful.
Message	Error information related to the load job.
NumberTotalRows	The total number of rows processed during the load job.
NumberLoadedRows	The number of rows that were successfully loaded.
NumberFilteredRows	The number of rows that did not meet the data quality standards.
NumberUnselectedRows	The number of rows that were filtered out based on the WHERE condition.
LoadBytes	The amount of data in bytes.
LoadTimeMs	The time taken for the load job to complete, measured in milliseconds.
BeginTxnTimeMs	The time taken to request the initiation of a transaction from the Frontend node (FE), measured in milliseconds.
StreamLoadPutTimeMs	The time taken to request the execution plan for the load job data from the FE, measured in milliseconds.
ReadDataTimeMs	The time spent reading the data during the load job, measured in milliseconds.
WriteDataTimeMs	The time taken to perform the data writing operations during the load job, measured in milliseconds.
CommitAndPublishTimeMs	The time taken to request the commit and publish the transaction from the FE, measured in milliseconds.
ErrorURL	If there are data quality issues, users can access this URL to view the specific rows with errors.

Users can access the ErrorURL to review data that failed to import due to issues with data quality. By executing the command `curl "<ErrorURL>"`, users can directly retrieve information about the erroneous data.

Load example

Setting load timeout and maximum size

The timeout for a load job is measured in seconds. If the load job is not completed within the specified timeout period, it will be cancelled by the system and marked as `CANCELLED`. You can adjust the timeout for a Stream Load job by specifying the `timeout` parameter or adding the `stream_load_default_timeout_second` parameter in the fe.conf file.

Before initiating the load, you need to calculate the timeout based on the file size. For example, for a 100GB file with an estimated load performance of 50MB/s:

```
Load time ≈ 100GB / 50MB/s ≈ 2048s
```

You can use the following command to specify a timeout of 3000 seconds for creating a Stream Load job:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "timeout:3000"
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Setting maximum error tolerance rate

Load job can tolerate a certain amount of data with formatting errors. The tolerance rate is configured using the `max_filter_ratio` parameter. By default, it is set to 0, meaning that if there is even a single erroneous data row, the entire load job will fail. If users wish to ignore some problematic data rows, they can set this parameter to a value between 0 and 1. Doris will automatically skip rows with incorrect data formats. For more information on calculating the tolerance rate, please refer to the Data Transformation documentation.

You can use the following command to specify a `max_filter_ratio` tolerance of 0.4 for creating a Stream Load job:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "max_filter_ratio:0.4" \
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Setting load filtering conditions

During the load job, you can use the WHERE parameter to apply conditional filtering to the imported data. The filtered data will not be included in the calculation of the filter ratio and will not affect the setting of `max_filter_ratio`. After the load job is complete, you can view the number of filtered rows by checking `num_rows_unselected`.

You can use the following command to specify WHERE filtering conditions for creating a Stream Load job:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "where:age>=35" \
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Loading data into specific partitions

Loading data from local files into partitions p1 and p2 of the table, allowing a 20% error rate.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "label:123" \
    -H "Expect:100-continue" \
    -H "max_filter_ratio:0.2" \
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -H "partitions: p1, p2" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Loading data into specific timezone

Since Doris currently does not have a built-in time zone time type, all `DATETIME` related types only represent absolute time points, do not contain time zone information, and will not change due to changes in the Doris system time zone. Therefore, for the import of data with time zones, our unified processing method is to convert it into data in a specific target time zone. In the Doris system, it is the time zone represented by the session variable `time_zone`.

In the import, our target time zone is specified through the parameter `timezone`. This variable will replace the session variable `time_zone` when time zone conversion occurs and time zone sensitive functions are calculated. Therefore, if there are no special circumstances, the `timezone` should be set in the import transaction to be consistent with the `time_zone` of the current Doris cluster. This means that all time data with a time zone will be converted to this time zone.

For example, the Doris system time zone is “+08:00”, and the time column in the imported data contains two pieces of data, namely “2012-01-01 01:00:00+00:00” and “2015-12-12 12 :12:12-08:00”, then after we specify the time zone of the imported transaction through `-H "timezone: +08:00"` when importing, both pieces of data will be converted to the time zone to obtain the result.” 2012-01-01 09:00:00” and “2015-12-13 04:12:12”.

For more information on time zone interpretation, please refer to the document Time Zone.

Streamingly import

Stream Load is based on the HTTP protocol for importing, which supports using programming languages such as Java, Go, or Python for streaming import. This is why it is named Stream Load.

The following example demonstrates this usage through a bash command pipeline. The imported data is generated streamingly by the program rather than from a local file.

```
seq 1 10 | awk '{OFS="\t"}{print $1, $1 * 10}' | curl --location-trusted -u root -T -
http://host:port/api/testDb/testTbl/_stream_load
```

Set CSV first row filtering

File data:

```
id,name,age
1,doris,20
2,flink,10
```

Filtering the first row during load by specifying `format=csv_with_names`

```
curl --location-trusted -u root -T test.csv  -H "label:1" -H "format:csv_with_names" -H "column_separator:,"
http://host:port/api/testDb/testTbl/_stream_load
```

Specifying merge_type for DELETE operations

In Stream Load, there are three import types: APPEND, DELETE, and MERGE. These can be adjusted by specifying the parameter `merge_type`. If you want to specify that all data with the same key as the imported data should be deleted, you can use the following command:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "merge_type: DELETE" \
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Before loading:

```
+--------+----------+----------+------+
| siteid | citycode | username | pv   |
+--------+----------+----------+------+
|      3 |        2 | tom      |    2 |
|      4 |        3 | bush     |    3 |
|      5 |        3 | helen    |    3 |
+--------+----------+----------+------+
```

The imported data is:

```
3,2,tom,0
```

After importing, the original table data will be deleted, resulting in the following result:

```
+--------+----------+----------+------+
| siteid | citycode | username | pv   |
+--------+----------+----------+------+
|      4 |        3 | bush     |    3 |
|      5 |        3 | helen    |    3 |
+--------+----------+----------+------+
```

Specifying merge_type for MERGE operation

By specifying `merge_type` as MERGE, the imported data can be merged into the table. The MERGE semantics need to be used in combination with the DELETE condition, which means that data satisfying the DELETE condition is processed according to the DELETE semantics, and the rest is added to the table according to the

APPEND semantics. The following operation represents deleting the row with `siteid` of 1, and adding the rest of the data to the table:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "merge_type: MERGE" \
    -H "delete: siteid=1" \
    -H "column_separator:," \
    -H "columns:user_id,name,age" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Before loading:

```
+--------+----------+----------+------+
| siteid | citycode | username | pv   |
+--------+----------+----------+------+
|      4 |        3 | bush     |    3 |
|      5 |        3 | helen    |    3 |
|      1 |        1 | jim      |    2 |
+--------+----------+----------+------+
```

The imported data is:

```
2,1,grace,2
3,2,tom,2
1,1,jim,2
```

After loading, the row with `siteid = 1` will be deleted according to the condition, and the rows with `siteid` of 2 and 3 will be added to the table:

```
+--------+----------+----------+------+
| siteid | citycode | username | pv   |
+--------+----------+----------+------+
|      4 |        3 | bush     |    3 |
|      2 |        1 | grace    |    2 |
|      3 |        2 | tom      |    2 |
|      5 |        3 | helen    |    3 |
+--------+----------+----------+------+
```

Specifying sequence column for merge

When a table with a Unique Key has a Sequence column, the value of the Sequence column serves as the basis for the replacement order in the REPLACE aggregation function under the same Key column. A larger value can replace a smaller one. When marking deletions based on `DORIS_DELETE_SIGN` for such a table, it is necessary to ensure that the Key is the same and that the Sequence column value is greater than or equal to the current value. By specifying the `function_column.sequence_col` parameter, deletion operations can be performed in combination with `merge_type: DELETE`.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "merge_type: DELETE" \
    -H "function_column.sequence_col: age" \
```

```
    -H "column_separator:," \
    -H "columns: name, gender, age"
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Given the following table schema:

```
SET show_hidden_columns=true;
Query OK, 0 rows affected (0.00 sec)

DESC table1;
+------------------------+--------------+------+-------+---------+---------+
| Field                  | Type         | Null | Key   | Default | Extra   |
+------------------------+--------------+------+-------+---------+---------+
| name                   | VARCHAR(100) | No   | true  | NULL    |         |
| gender                 | VARCHAR(10)  | Yes  | false | NULL    | REPLACE |
| age                    | INT          | Yes  | false | NULL    | REPLACE |
| __DORIS_DELETE_SIGN__  | TINYINT      | No   | false | 0       | REPLACE |
| __DORIS_SEQUENCE_COL__ | INT          | Yes  | false | NULL    | REPLACE |
+------------------------+--------------+------+-------+---------+---------+
4 rows in set (0.00 sec)
```

The original table data is:

```
+-------+--------+------+
| name  | gender | age  |
+-------+--------+------+
| li    | male   |   10 |
| wang  | male   |   14 |
| zhang | male   |   12 |
+-------+--------+------+
```

1. Sequence parameter takes Eeffect, loading sequence column value is larger than or equal to the existing data in the table.

   loading data as:

```
li,male,10
```

Since `function_column.sequence_col` is specified as `age`, and the `age` value is larger than or equal to the existing column in the table, the original table data is deleted. The table data becomes:

```
+-------+--------+------+
| name  | gender | age  |
+-------+--------+------+
| wang  | male   |   14 |
| zhang | male   |   12 |
+-------+--------+------+
```

2. Sequence parameter do not take effect, loading sequence column value is less than or equal to the existing data in the table:

   loading data as:

```
li,male,9
```

Since `function_column.sequence_col` is specified as `age`, but the `age` value is less than the existing column in the table, the delete operation does not take effect. The table data remains unchanged, and the row with the primary key of `li` is still visible:

```
+-------+--------+------+
| name  | gender | age  |
+-------+--------+------+
| li    | male   |   10 |
| wang  | male   |   14 |
| zhang | male   |   12 |
+-------+--------+------+
```

It is not deleted because that, at the underlying dependency level, it first checks for rows with the same key. It displays the row data with the larger sequence column value. Then, it checks the `DORIS_DELETE_SIGN` value for that row. If it is 1, it is not displayed externally. If it is 0, it is still read and displayed.

Loading data with enclosing characters

When the data in a CSV file contains delimiters or separators, single-byte characters can be specified as enclosing characters to protect the data from being truncated.

For example, in the following data where a comma is used as the separator but also exists within a field:

```
zhangsan,30,'Shanghai, HuangPu District, Dagu Road'
```

By specifying an enclosing character such as a single quotation mark ', the entire `Shanghai, HuangPu District, Dagu Road` can be treated as a single field.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "enclose:'" \
    -H "columns:username,age,address" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

If the enclosing character also appears within a field, such as wanting to treat `Shanghai City, Huangpu District, \'Dagu Road` as a single field, it is necessary to first perform string escaping within the column:

```
Zhang San,30,'Shanghai, Huangpu District, \'Dagu Road'
```

An escape character, which is a single-byte character, can be specified using the escape parameter. In the example, the backslash \ is used as the escape character.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "enclose:'" \
    -H "escape:\\" \
```

```
    -H "columns:username,age,address" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Loading fields containing default CURRENT_TIMESTAMP type

Here's an example of loading data into a table that contains a field with the DEFAULT CURRENT_TIMESTAMP type:

Table schema:

```
`id` bigint(30) NOT NULL,
`order_code` varchar(30) DEFAULT NULL COMMENT '',
`create_time` datetimev2(3) DEFAULT CURRENT_TIMESTAMP
```

JSON data type:

```
{"id":1,"order_Code":"avc"}
```

Command:

```
curl --location-trusted -u root -T test.json -H "label:1" -H "format:json" -H 'columns: id, order_code,
create_time=CURRENT_TIMESTAMP()' http://host:port/api/testDb/testTbl/_stream_load
```

Simple mode for loading JSON format data

When the JSON fields correspond one-to-one with the column names in the table, you can import JSON data format into the table by specifying the parameters "strip_outer_array:true" and "format:json".

For example, if the table is defined as follows:

```
CREATE TABLE testdb.test_streamload(
    user_id            BIGINT       NOT NULL COMMENT "User ID",
    name               VARCHAR(20)           COMMENT "User name",
    age                INT                   COMMENT "User age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

And the data field names correspond one-to-one with the column names in the table:

```
[
{"user_id":1,"name":"Emily","age":25},
{"user_id":2,"name":"Benjamin","age":35},
{"user_id":3,"name":"Olivia","age":28},
{"user_id":4,"name":"Alexander","age":60},
{"user_id":5,"name":"Ava","age":17},
{"user_id":6,"name":"William","age":69},
{"user_id":7,"name":"Sophia","age":32},
{"user_id":8,"name":"James","age":64},
{"user_id":9,"name":"Emma","age":37},
```

```
{"user_id":10,"name":"Liam","age":64}
]
```

You can use the following command to load JSON data into the table:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "format:json" \
    -H "strip_outer_array:true" \
    -T streamload_example.json \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Matching mode for loading complex JSON format data

When the JSON data is more complex and cannot correspond one-to-one with the column names in the table, or there are extra columns, you can use the jsonpaths parameter to complete the column name mapping and perform data matching import. For example, with the following data:

```
[
{"userid":1,"hudi":"lala","username":"Emily","userage":25,"userhp":101},
{"userid":2,"hudi":"kpkp","username":"Benjamin","userage":35,"userhp":102},
{"userid":3,"hudi":"ji","username":"Olivia","userage":28,"userhp":103},
{"userid":4,"hudi":"popo","username":"Alexander","userage":60,"userhp":103},
{"userid":5,"hudi":"uio","username":"Ava","userage":17,"userhp":104},
{"userid":6,"hudi":"lkj","username":"William","userage":69,"userhp":105},
{"userid":7,"hudi":"komf","username":"Sophia","userage":32,"userhp":106},
{"userid":8,"hudi":"mki","username":"James","userage":64,"userhp":107},
{"userid":9,"hudi":"hjk","username":"Emma","userage":37,"userhp":108},
{"userid":10,"hudi":"hua","username":"Liam","userage":64,"userhp":109}
]
```

You can specify the jsonpaths parameter to match the specified columns:

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "format:json" \
    -H "strip_outer_array:true" \
    -H "jsonpaths:[\"$.userid\", \"$.username\", \"$.userage\"]" \
    -H "columns:user_id,name,age" \
    -T streamload_example.json \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Specifying JSON root node for data load

If the JSON data contains nested JSON fields, you need to specify the root node of the imported JSON. The default value is “”.

For example, with the following data, if you want to import the data in the comment column into the table:

```
[
    {"user":1,"comment":{"userid":101,"username":"Emily","userage":25}},
    {"user":2,"comment":{"userid":102,"username":"Benjamin","userage":35}},
    {"user":3,"comment":{"userid":103,"username":"Olivia","userage":28}},
    {"user":4,"comment":{"userid":104,"username":"Alexander","userage":60}},
```

```
    {"user":5,"comment":{"userid":105,"username":"Ava","userage":17}},
    {"user":6,"comment":{"userid":106,"username":"William","userage":69}},
    {"user":7,"comment":{"userid":107,"username":"Sophia","userage":32}},
    {"user":8,"comment":{"userid":108,"username":"James","userage":64}},
    {"user":9,"comment":{"userid":109,"username":"Emma","userage":37}},
    {"user":10,"comment":{"userid":110,"username":"Liam","userage":64}}
    ]
```

First, you need to specify the root node as comment using the json_root parameter, and then complete the column name mapping according to the jsonpaths parameter.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "format:json" \
    -H "strip_outer_array:true" \
    -H "json_root: $.comment" \
    -H "jsonpaths:[\"$.userid\", \"$.username\", \"$.userage\"]" \
    -H "columns:user_id,name,age" \
    -T streamload_example.json \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Loading array data type

For example, if the following data contains an array type:

```
1|Emily|[1,2,3,4]
2|Benjamin|[22,45,90,12]
3|Olivia|[23,16,19,16]
4|Alexander|[123,234,456]
5|Ava|[12,15,789]
6|William|[57,68,97]
7|Sophia|[46,47,49]
8|James|[110,127,128]
9|Emma|[19,18,123,446]
10|Liam|[89,87,96,12]
```

Load data into the following table structure:

```
CREATE TABLE testdb.test_streamload(
    typ_id     BIGINT          NOT NULL COMMENT "ID",
    name       VARCHAR(20)     NULL     COMMENT "Name",
    arr        ARRAY<int(10)>  NULL     COMMENT "Array"
)
DUPLICATE KEY(typ_id)
DISTRIBUTED BY HASH(typ_id) BUCKETS 10;
```

You can directly load the ARRAY type from a text file into the table using a Stream Load job.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "column_separator:|" \
    -H "columns:typ_id,name,arr" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Loading map data type

When the imported data contains a map type, as in the following example:

```
[
{"user_id":1,"namemap":{"Emily":101,"age":25}},
{"user_id":2,"namemap":{"Benjamin":102,"age":35}},
{"user_id":3,"namemap":{"Olivia":103,"age":28}},
{"user_id":4,"namemap":{"Alexander":104,"age":60}},
{"user_id":5,"namemap":{"Ava":105,"age":17}},
{"user_id":6,"namemap":{"William":106,"age":69}},
{"user_id":7,"namemap":{"Sophia":107,"age":32}},
{"user_id":8,"namemap":{"James":108,"age":64}},
{"user_id":9,"namemap":{"Emma":109,"age":37}},
{"user_id":10,"namemap":{"Liam":110,"age":64}}
]
```

Load data into the following table structure:

```
CREATE TABLE testdb.test_streamload(
    user_id            BIGINT       NOT NULL COMMENT "ID",
    namemap            Map<STRING, INT>  NULL     COMMENT "Name"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

You can directly load the map type from a text file into the table using a Stream Load task.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "format: json" \
    -H "strip_outer_array:true" \
    -T streamload_example.json \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Loading bitmap data type

During the import process, when encountering Bitmap type data, you can use to_bitmap to convert the data into Bitmap, or use the bitmap_empty function to fill the Bitmap.

For example, with the following data:

```
1|koga|17723
2|nijg|146285
3|lojn|347890
4|lofn|489871
5|jfin|545679
6|kon|676724
7|nhga|767689
8|nfubg|879878
9|huang|969798
10|buag|97997
```

Load data into the following table containing the Bitmap type:

```
CREATE TABLE testdb.test_streamload(
    typ_id     BIGINT               NULL   COMMENT "ID",
    hou        VARCHAR(10)          NULL   COMMENT "one",
    arr        BITMAP  BITMAP_UNION  NOT NULL   COMMENT "two"
)
AGGREGATE KEY(typ_id,hou)
DISTRIBUTED BY HASH(typ_id,hou) BUCKETS 10;
```

And use to_bitmap to convert the data into the Bitmap type.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "Expect:100-continue" \
    -H "columns:typ_id,hou,arr,arr=to_bitmap(arr)" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Loading HyperLogLog data type

You can use the hll_hash function to convert data into the hll type, as in the following example:

```
1001|koga
1002|nijg
1003|lojn
1004|lofn
1005|jfin
1006|kon
1007|nhga
1008|nfubg
1009|huang
1010|buag
```

Load data into the following table:

```
CREATE TABLE testdb.test_streamload(
    typ_id           BIGINT          NULL   COMMENT "ID",
    typ_name         VARCHAR(10)     NULL   COMMENT "NAME",
    pv               hll hll_union   NOT NULL   COMMENT "hll"
)
AGGREGATE KEY(typ_id,typ_name)
DISTRIBUTED BY HASH(typ_id) BUCKETS 10;
```

And use the hll_hash command for import.

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "column_separator:|" \
    -H "columns:typ_id,typ_name,pv=hll_hash(typ_id)" \
    -T streamload_example.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_streamload/_stream_load
```

Column mapping, derived columns, and filtering

Doris supports a very rich set of column transformations and filtering operations in load statements. Supports most built-in functions. For how to use this feature correctly, please refer to the Data Transformation

documentation.

Enable strict mode import

The strict_mode attribute is used to set whether the import task runs in strict mode. This attribute affects the results of column mapping, transformation, and filtering, and it also controls the behavior of partial column updates. For specific instructions on strict mode, please refer to the Handling Messy Data documentation.

Perform partial column updates/flexible partial update during import

For how to express partial column updates during import, please refer to the Data Manipulation/Data Update documentation.

Broker Load

Broker Load is initiated from the MySQL API. Doris will actively pull the data from the source based on the information in the LOAD statement. Broker Load is an asynchronous import method. The progress and result of Broker Load tasks can be viewed by the SHOW LOAD statement.

Broker Load is suitable for scenarios where the source data is stored in remote storage systems, such as HDFS, and the data volume is relatively large.

Direct reads from HDFS or S3 can also be imported through HDFS TVF or S3 TVF in the Lakehouse/TVF. The current “Insert Into” based on TVF is a synchronous import, while Broker Load is an asynchronous import method.

In early versions of Doris, both S3 Load and HDFS Load were implemented by connecting to specific Broker processes using `WITH BROKER`. In newer versions, S3 Load and HDFS Load have been optimized as the most commonly used import methods, and they no longer depend on an additional Broker process, though they still use syntax similar to Broker Load. Due to historical reasons and the similarity in syntax, S3 Load, HDFS Load, and Broker Load are collectively referred to as Broker Load.

Limitations

Supported data sources:

- S3 protocol
- HDFS protocol
- Custom protocol (require broker process)

Supported data types:

- CSV
- JSON
- PARQUET
- ORC

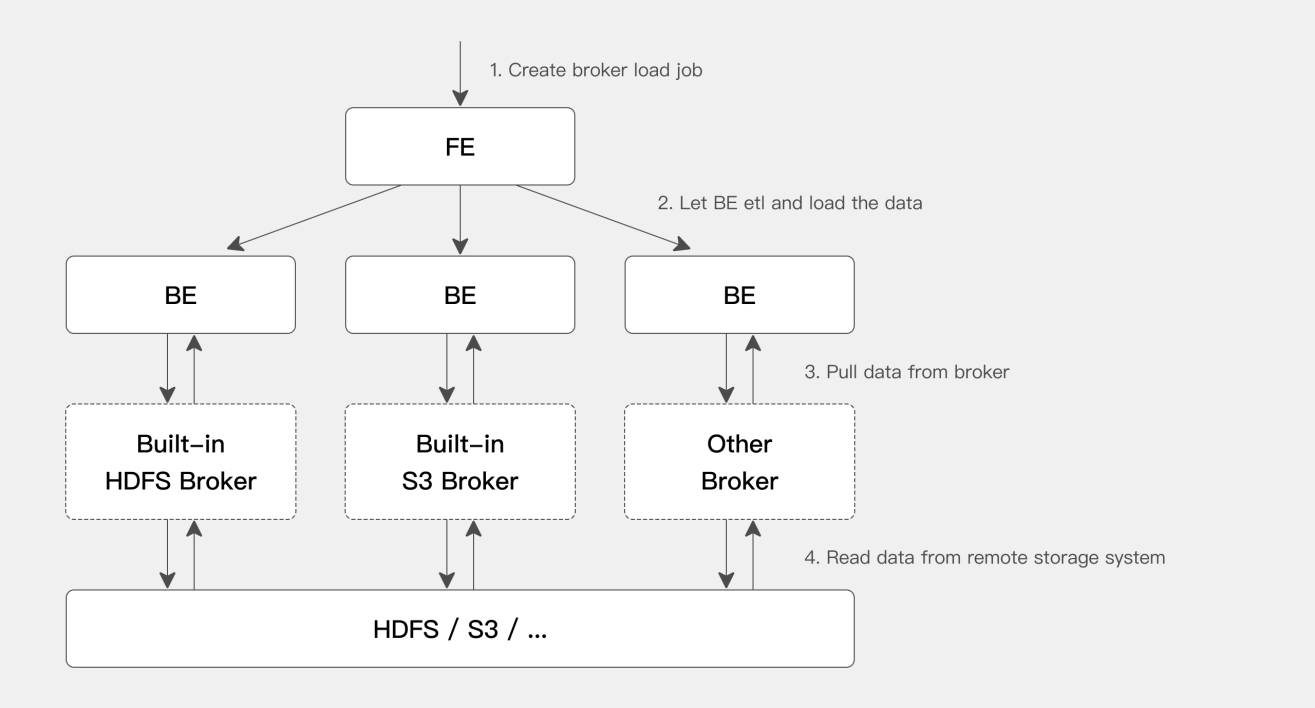
Supported compress types:

- PLAIN
- GZ
- LZO
- BZ2
- LZ4FRAME
- DEFLATE
- LZOP
- LZ4BLOCK
- SNAPPYBLOCK
- ZLIB
- ZSTD

Basic Principles

After a user submits an import task, the Frontend (FE) generates a corresponding plan. Based on the current number of Backend (BE) nodes and the size of the file, the plan is distributed to multiple BE nodes for execution, with each BE node handling a portion of the import data.

During execution, the BE nodes pull data from the Broker, perform necessary transformations, and then import the data into the system. Once all BE nodes have completed the import, the FE makes the final determination on whether the import was successful.



Broker Load

As seen in the diagram, BE nodes rely on Broker processes to read data from corresponding remote storage systems. The introduction of Broker processes primarily aims to accommodate different remote storage systems. Users can develop their own Broker processes according to established standards. These Broker processes, which can be developed using Java, offer better compatibility with various storage systems in the big data ecosystem. The separation of Broker processes from BE nodes ensures error isolation between the two, enhancing the stability of the BE.

Currently, BE nodes have built-in support for HDFS and S3 Brokers. Therefore, when importing data from HDFS or S3, there is no need to additionally start a Broker process. However, if a customized Broker implementation is required, the corresponding Broker process needs to be deployed.

Quick start

This section shows a demo for S3 Load. For the specific syntax for usage, please refer to BROKER LOAD in the SQL manual.

Prerequisite check

1. Grant privileges on the table

Broker Load requires `INSERT` privileges on the target table. If there are no `INSERT` privileges, it can be granted to the user through the GRANT command.

2. S3 authentication and connection info

Here, we mainly introduce how to import data stored in AWS S3. For importing data from other object storage systems that support the S3 protocol, you can refer to the steps for AWS S3.

- AK and SK: First, you need to find or regenerate your AWS `Access Keys`. You can find instructions on how to generate them in the AWS console under `My Security Credentials`.
- REGION and ENDPOINT: The REGION can be selected when creating a bucket or viewed in the bucket list. The S3 ENDPOINT for each REGION can be found in the AWS documentation.

Create load job

1. Create a CSV file brokerload_example.csv. The file is stored on S3 and its content is as follows:

```
1,Emily,25
2,Benjamin,35
3,Olivia,28
4,Alexander,60
5,Ava,17
6,William,69
7,Sophia,32
8,James,64
9,Emma,37
10,Liam,64
```

2. Create Doris table for the load

Create the imported table in Doris. The SQL statement is as follows:

```
CREATE TABLE testdb.test_brokerload(
    user_id            BIGINT       NOT NULL COMMENT "user id",
    name               VARCHAR(20)           COMMENT "name",
    age                INT                   COMMENT "age"
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

3. Use Broker Load to import data from S3. The bucket name and S3 authentication information should be filled in according to the actual situation:

```
    LOAD LABEL broker_load_2022_04_01
    (
        DATA INFILE("s3://your_bucket_name/brokerload_example.csv")
        INTO TABLE test_brokerload
        COLUMNS TERMINATED BY ","
        FORMAT AS "CSV"
        (user_id, name, age)
    )
    WITH S3
    (
        "provider" = "S3",
        "AWS_ENDPOINT" = "https://s3.us-west-2.amazonaws.com",
```

```
        "AWS_ACCESS_KEY" = "<your-ak>",
        "AWS_SECRET_KEY"="<your-sk>",
        "AWS_REGION" = "us-west-2",
        "compress_type" = "PLAIN"
    )
    PROPERTIES
    (
        "timeout" = "3600"
    );
```

The `provider` specifies the vendor of the S3 Service. Supported S3 Provider list:

- “S3” (AWS, Amazon Web Services)
- “AZURE” (Microsoft Azure)
- “GCP” (GCP, Google Cloud Platform)
- “OSS” (Alibaba Cloud)
- “COS” (Tencent Cloud)
- “OBS” (Huawei Cloud)
- “BOS” (Baidu Cloud)

If your service is not in the list (such as MinIO), you can try using “S3” (AWS compatible mode)

Checking import status

Broker Load is an asynchronous import method, and the specific import results can be viewed through the SHOW LOAD command.

```
mysql> show load order by createtime desc limit 1\G;
*************************** 1. row ***************************
         JobId: 41326624
         Label: broker_load_2022_04_01
         State: FINISHED
      Progress: ETL:100%; LOAD:100%
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=27
      TaskInfo: cluster:N/A; timeout(s):1200; max_filter_ratio:0.1
      ErrorMsg: NULL
    CreateTime: 2022-04-01 18:59:06
  EtlStartTime: 2022-04-01 18:59:11
 EtlFinishTime: 2022-04-01 18:59:11
 LoadStartTime: 2022-04-01 18:59:11
LoadFinishTime: 2022-04-01 18:59:11
           URL: NULL
    JobDetails: {"Unfinished backends":{"5072bde59b74b65-8d2c0ee5b029adc0":[]},"ScannedRows":27,"TaskNumber":1,"All backends":
{"5072bde59b74b65-8d2c0ee5b029adc0":[36728051]},"FileNumber":1,"FileSize":5540}
1 row in set (0.01 sec)
```

Cancelling an Import

When the status of a Broker Load job is not CANCELLED or FINISHED, it can be manually cancelled by the user. To cancel, the user needs to specify the label of the import task to be cancelled. The syntax for the cancel import command can be viewed by executing CANCEL LOAD.

For example: To cancel the import job with the label “broker_load_2022_04_01” on the DEMO database.

```
CANCEL LOAD FROM demo WHERE LABEL = "broker_load_2022_04_01";
```

Reference Manual

SQL syntax for broker load

```
LOAD LABEL load_label
(
data_desc1[, data_desc2, ...]
[format_properties]
)
WITH [S3|HDFS|BROKER broker_name]
[broker_properties]
[load_properties]
[COMMENT "comments"];
```

The WITH clause specifies how to access the storage system, and `broker_properties` is the configuration parameter for the access method

- `S3`: Storage system using the S3 protocol
- `HDFS`: Storage system using the HDFS protocol
- `BROKER broker_name`: Storage system using other protocols. You can view the currently available broker_name list through `SHOW BROKER`. For more information, see “Other Broker Import” in the Common Issues section.

Related Configurations

Load Properties

Property Name	Type	Default Value	Description
“timeout”	Long	14400	Used to specify the timeout for the import in seconds. The configurable range is from 1 second to 259200 seconds.
“max_filter_ratio”	Float	0.0	Used to specify the maximum tolerable ratio of filterable (irregular or otherwise problematic) data, which defaults to zero tolerance. The value range is 0 to 1. If the error rate of the imported data exceeds this value, the import will fail. Irregular data does not include rows filtered out by the where condition.
“strict_mode”	Boolean	false	Used to specify whether to enable strict mode for this import.

Property Name	Type	Default Value	Description
“partial_columns”	Boolean	false	Used to specify whether to enable partial column update, the default value is false, this parameter is only available for Unique Key + Merge on Write tables.
“timezone”	String	“Asia/Shanghai”	Used to specify the timezone to be used for this import. This parameter affects the results of all timezone-related functions involved in the import.
“load_parallelism”	Integer	8	Limits the maximum parallel instances on each backend.
“send_batch_parallelism”	Integer	1	The parallelism for sink node to send data, when memtable_on_sink_node is disabled.
“load_to_single_tablet”	Boolean	“false”	Used to specify whether to load data only to a single tablet corresponding to the partition. This parameter is only available when loading to an OLAP table with random bucketing.
“priority”	oneof “HIGH”, “NORMAL”, “LOW”	“NORMAL”	The priority of the task.

Format Properties

Property Name	Type	Default Value	Description
`skip_lines`	Integer	`0`	Number of lines to skip at the start of a CSV file. Ignored if using `csv_with_names` or `csv_with_names_and_types`.
`trim_double_quotes`	Boolean	`false`	If `true`, trims outermost double quotes from each field.
`enclose`	String	`""`	Enclosure character for fields containing delimiters or newlines. E.g., if delimiter is `,` and encloser is `'`, then `'b,c'` is parsed as one field.

Property Name	Type	Default Value	Description
escape	String	""	Escape character to include enclosure characters in field content. E.g., 'b,\'c' keeps 'b,'c' as one field when ' is the enclosure and \ is the escape.

Note: Format properties define how to parse the source file (e.g., delimiters, quote handling) and must be set inside the LOAD clause. Load properties control the execution behavior (e.g., timeout, retries) and must be set outside, in the outer PROPERTIES block.

```
LOAD LABEL s3_load_example (
    DATA INFILE("s3://bucket/path/file.csv")
    INTO TABLE users
    COLUMNS TERMINATED BY ","
    FORMAT AS "CSV"
    (user_id, name, age)
    PROPERTIES (
        "trim_double_quotes" = "true"  -- format property
    )
)
WITH S3 (
    ...
)
PROPERTIES (
    "timeout" = "3600"  -- load property
);
```

fe.conf

The following configurations belong to the system-level settings for Broker load, which affect all Broker load import tasks. These configurations can be adjusted by modifying the `fe.conf`file.

Session Variable	Type	Default Value	Description
min_bytes_per_broker_scanner	Long	67108864 (64 MB)	The minimum amount of data processed by a single BE in a Broker Load job, in bytes.
max_bytes_per_broker_scanner	Long	536870912000 (500 GB)	The maximum amount of data processed by a single BE in a Broker Load job, in bytes. Usually, the maximum amount of data supported by an import job is `max_bytes_per_broker_scanner * number of BE nodes`. If you need to import a larger amount of data, you need to adjust the size of the `max_bytes_per_broker_scanner` parameter appropriately.
max_broker_concurrency	Integer	10	Limits the maximum number of concurrent imports for a job.

Session Variable	Type	Default Value	Description
default_load_parallelism	Integer	8	Maximum number of concurrent instances per BE node
broker_load_default_timeout_second	14400	Default timeout for Broker Load import, in seconds.	

Note: The `min_bytes_per_broker_scanner`, the `max_broker_concurrency`, the size of the source file and the number of BEs in the current cluster jointly determine the number of concurrent execution instances for this load.

```
Import Concurrency = Math.min(Source File Size / min_bytes_per_broker_scanner, max_broker_concurrency, Current Number of BE Nodes * load_parallelism)
Processing Volume per BE for this Import = Source File Size / Import Concurrency
```

session variables

Session Variable	Type	Default	Description
time_zone	String	“Asia/Shanghai”	Default time zone, which will affect the results of time zone related functions in import.
send_batch_parallelism	Integer	1	The concurrency of the sink node sending data, which takes effect only when `enable_memtable_on_sink_node` is set to false.

Common Issues

Common Errors

1. Import Error: `Scan bytes per broker scanner exceed limit:xxx`

Please refer to the best practices section in the documentation and modify the FE configuration items `max_bytes_per_broker_scanner` and `max_broker_concurrency`.

2. Import Error: : `failed to send batch` **or** `TabletWriter add batch with unknown id`

Appropriately adjust the `query_timeout` and `streaming_load_rpc_max_alive_time_sec` settings.

3. Import Error: `LOAD_RUN_FAIL; msg:Invalid Column Name:xxx`

For PARQUET or ORC format data, the column names in the file header must match the column names in the Doris table. For example:

```
(tmp_c1,tmp_c2)
SET
(
    id=tmp_c2,
    name=tmp_c1
)
```

This represents fetching columns named (tmp_c1, tmp_c2) in the parquet or orc file and mapping them to the (id, name) columns in the Doris table. If no set is specified, the columns in the file header will be used for mapping.

> **Note**
>
> If ORC files are generated directly using certain Hive versions, the column headers in the ORC file may not be the Hive metadata, but (_col0, _col1, _col2, …), which may lead to the Invalid Column Name error. In this case, mapping using SET is necessary.

5. Import Error: `Failed to get S3 FileSystem for bucket is null/empty`

The bucket information is incorrect or does not exist. Or the bucket format is not supported. When creating a bucket name with an underscore using GCS, such as `s3://gs_bucket/load_tbl`, the S3 Client may report an error when accessing GCS. It is recommended not to use underscores when creating buckets.

6. Import Timeout

The default timeout for imports is 4 hours. If a timeout occurs, it is not recommended to directly increase the maximum import timeout to solve the problem. If the single import time exceeds the default import timeout of 4 hours, it is best to split the file to be imported and perform multiple imports to solve the problem. Setting an excessively long timeout time can lead to high costs for retrying failed imports.

You can calculate the expected maximum import file data volume for the Doris cluster using the following formula:

Expected Maximum Import File Data Volume = 14400s * 10M/s * Number of BEs

For example, if the cluster has 10 BEs:

Expected Maximum Import File Data Volume = 14400s * 10M/s * 10 = 1440000M ≈ 1440G

> **Note**
>
> In general, user environments may not reach speeds of 10M/s, so it is recommended to split files exceeding 500G before importing.

S3 Load URL style

- The S3 SDK defaults to using the virtual-hosted style method for accessing objects. However, some object storage systems may not have enabled or supported the virtual-hosted style access. In such cases, we can add the `use_path_style` parameter to force the use of the path style method:

```
WITH S3
(
      "AWS_ENDPOINT" = "AWS_ENDPOINT",
      "AWS_ACCESS_KEY" = "AWS_ACCESS_KEY",
      "AWS_SECRET_KEY"="AWS_SECRET_KEY",
      "AWS_REGION" = "AWS_REGION",
```

```
      "use_path_style" = "true"
)
```

S3 Load temporary credentials

- Support for accessing all object storage systems that support the S3 protocol using temporary credentials (TOKEN) is available. The usage is as follows:

```
WITH S3
(
      "AWS_ENDPOINT" = "AWS_ENDPOINT",
      "AWS_ACCESS_KEY" = "AWS_TEMP_ACCESS_KEY",
      "AWS_SECRET_KEY" = "AWS_TEMP_SECRET_KEY",
      "AWS_TOKEN" = "AWS_TEMP_TOKEN",
      "AWS_REGION" = "AWS_REGION"
)
```

HDFS Simple Authentication

Simple authentication refers to the configuration of Hadoop where hadoop.security.authentication is set to “simple”.

```
(
    "username" = "user",
    "password" = ""
);
```

The username should be configured as the user to be accessed, and the password can be left blank.

HDFS Kerberos Authentication

This authentication method requires the following information:

- **hadoop.security.authentication:** Specifies the authentication method as Kerberos.
- **hadoop.kerberos.principal:** Specifies the Kerberos principal.
- **hadoop.kerberos.keytab:** Specifies the file path of the Kerberos keytab. The file must be an absolute path on the server where the Broker process is located and must be accessible by the Broker process.
- **kerberos_keytab_content:** Specifies the content of the Kerberos keytab file after being encoded in base64. This can be used as an alternative to the kerberos_keytab configuration.

Example configuration:

```
(
    "hadoop.security.authentication" = "kerberos",
    "hadoop.kerberos.principal" = "doris@YOUR.COM",
    "hadoop.kerberos.keytab" = "/home/doris/my.keytab"
)
(
    "hadoop.security.authentication" = "kerberos",
    "hadoop.kerberos.principal" = "doris@YOUR.COM",
```

```
    "kerberos_keytab_content" = "ASDOWHDLAWIDJHWLDKSALDJSDIWALD"
)
```

To use Kerberos authentication, the krb5.conf (opens new window) file is required. The krb5.conf file contains Kerberos configuration information. Typically, the krb5.conf file should be installed in the /etc directory. You can override the default location by setting the KRB5_CONFIG environment variable. An example of the krb5.conf file content is as follows:

```
[libdefaults]
    default_realm = DORIS.HADOOP
    default_tkt_enctypes = des3-hmac-sha1 des-cbc-crc
    default_tgs_enctypes = des3-hmac-sha1 des-cbc-crc
    dns_lookup_kdc = true
    dns_lookup_realm = false

[realms]
    DORIS.HADOOP = {
        kdc = kerberos-doris.hadoop.service:7005
    }
```

HDFS HA Mode

This configuration is used to access HDFS clusters deployed in HA (High Availability) mode.

- **dfs.nameservices:** Specifies the name of the HDFS service, which can be customized. For example: “dfs.nameservices” = “my_ha”.
- **dfs.ha.namenodes.xxx:** Customizes the names of the namenodes, with multiple names separated by commas. Here, xxx represents the custom name specified in dfs.nameservices. For example: “dfs.ha.namenodes.my_ha” = “my_nn”.
- **dfs.namenode.rpc-address.xxx.nn:** Specifies the RPC address information for the namenode. In this context, nn represents the namenode name configured in dfs.ha.namenodes.xxx. For example: “dfs.namenode.rpc-address.my_ha.my_nn” = “host:port”.
- **dfs.client.failover.proxy.provider.[nameservice ID]:** Specifies the provider for client connections to the namenode. The default is org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider.

An example configuration is as follows:

```
(
    "fs.defaultFS" = "hdfs://my_ha",
    "dfs.nameservices" = "my_ha",
    "dfs.ha.namenodes.my_ha" = "my_namenode1, my_namenode2",
    "dfs.namenode.rpc-address.my_ha.my_namenode1" = "nn1_host:rpc_port",
    "dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
    "dfs.client.failover.proxy.provider.my_ha" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
)
```

HA mode can be combined with the previous two authentication methods for cluster access. For example, accessing HA HDFS through simple authentication:

```
(
    "username"="user",
    "password"="passwd",
    "fs.defaultFS" = "hdfs://my_ha",
    "dfs.nameservices" = "my_ha",
    "dfs.ha.namenodes.my_ha" = "my_namenode1, my_namenode2",
    "dfs.namenode.rpc-address.my_ha.my_namenode1" = "nn1_host:rpc_port",
    "dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
    "dfs.client.failover.proxy.provider.my_ha" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
)
```

Load with other brokers

The Broker for other remote storage systems is an optional process in the Doris cluster, primarily used to support Doris in reading and writing files and directories on remote storage. Currently, Doris provides Broker implementations for various remote storage systems. In earlier versions, different object storage Brokers were also available, but now it is recommended to use the `WITH S3` method to import data from object storage, and the `WITH BROKER` method is no longer recommended.

- Tencent Cloud CHDFS
- Tencent Cloud GFS
- JuiceFS

The Broker provides services through an RPC service port and operates as a stateless Java process. Its primary responsibility is to encapsulate POSIX-like file operations for remote storage, such as open, pread, pwrite, and more. Additionally, the Broker does not keep track of any other information, which means that all the connection details, file information, and permission details related to the remote storage must be passed to the Broker process through parameters during RPC calls. This ensures that the Broker can correctly read and write files.

The Broker serves solely as a data pathway and does not involve any computational tasks, thus requiring minimal memory usage. Typically, a Doris system would deploy one or more Broker processes. Furthermore, Brokers of the same type are grouped together and assigned a unique name (Broker name).

This section primarily focuses on the parameters required by the Broker when accessing different remote storage systems, such as connection information, authentication details, and more. Understanding and correctly configuring these parameters is crucial for successful and secure data exchange between Doris and the remote storage systems.

Broker Information

The information of the Broker consists of two parts: the name (Broker name) and the authentication information. The usual syntax format is as follows:

```
WITH BROKER "broker_name"
(
    "username" = "xxx",
    "password" = "yyy",
    "other_prop" = "prop_value",
    ...
);
```

Broker Name

Typically, users need to specify an existing Broker Name through the `WITH BROKER "broker_name"` clause in the operation command. The Broker Name is a name designated by the user when adding a Broker process through the `ALTER SYSTEM ADD BROKER` command. One name usually corresponds to one or more Broker processes. Doris will select an available Broker process based on the name. Users can view the Brokers that currently exist in the cluster through the `SHOW BROKER` command.

> **Note**
>
> The Broker Name is merely a user-defined name and does not represent the type of Broker.

Authentication Information Different Broker types and access methods require different authentication information. The authentication information is usually provided in the Property Map in a Key-Value format after `WITH BROKER "broker_name"`.

Broker Load examples

Importing TXT Files from HDFS

```
LOAD LABEL demo.label_20220402
(
    DATA INFILE("hdfs://host:port/tmp/test_hdfs.txt")
    INTO TABLE `load_hdfs_file_test`
    COLUMNS TERMINATED BY "\t"
    (id,age,name)
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username" = "user"
)
PROPERTIES
(
    "timeout"="1200",
    "max_filter_ratio"="0.1"
);
```

HDFS requires the configuration of NameNode HA (High Availability)

```
LOAD LABEL demo.label_20220402
(
    DATA INFILE("hdfs://hafs/tmp/test_hdfs.txt")
    INTO TABLE `load_hdfs_file_test`
    COLUMNS TERMINATED BY "\t"
    (id,age,name)
)
with HDFS
(
    "hadoop.username" = "user",
    "fs.defaultFS"="hdfs://hafs",
    "dfs.nameservices" = "hafs",
    "dfs.ha.namenodes.hafs" = "my_namenode1, my_namenode2",
```

```
    "dfs.namenode.rpc-address.hafs.my_namenode1" = "nn1_host:rpc_port",
    "dfs.namenode.rpc-address.hafs.my_namenode2" = "nn2_host:rpc_port",
    "dfs.client.failover.proxy.provider.hafs" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
)
PROPERTIES
(
    "timeout"="1200",
    "max_filter_ratio"="0.1"
);
```

Importing data from HDFS using wildcards to match two batches of files and importing them into two separate tables

```
LOAD LABEL example_db.label2
(
    DATA INFILE("hdfs://host:port/input/file-10*")
    INTO TABLE `my_table1`
    PARTITION (p1)
    COLUMNS TERMINATED BY ","
    (k1, tmp_k2, tmp_k3)
    SET (
        k2 = tmp_k2 + 1,
        k3 = tmp_k3 + 1
    ),
    DATA INFILE("hdfs://host:port/input/file-20*")
    INTO TABLE `my_table2`
    COLUMNS TERMINATED BY ","
    (k1, k2, k3)
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username" = "user"
);
```

To import two batches of files matching the wildcards `file-10*` and `file-20*` from HDFS and load them into two separate tables `my_table1` and `my_table2`. In this case, my_table1 specifies that the data should be imported into partition p1, and the values in the second and third columns of the source files should be incremented by 1 before being imported.

Import a batch of data from HDFS using wildcards

```
LOAD LABEL example_db.label3
(
    DATA INFILE("hdfs://host:port/user/doris/data/*/*")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY "\\x01"
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username" = "user"
);
```

To specify the delimiter as the commonly used default delimiter for Hive, which is , and to use the wildcard character * to refer to all files in all directories under the data directory.

Import Parquet format data and specify the FORMAT as `parquet`

```
LOAD LABEL example_db.label4
(
    DATA INFILE("hdfs://host:port/input/file")
    INTO TABLE `my_table`
    FORMAT AS "parquet"
    (k1, k2, k3)
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username" = "user"
);
```

The default method is to determine by file extension.

Import the data and extract the partition field from the file path

```
LOAD LABEL example_db.label5
(
    DATA INFILE("hdfs://host:port/input/city=beijing/*/*")
    INTO TABLE `my_table`
    FORMAT AS "csv"
    (k1, k2, k3)
    COLUMNS FROM PATH AS (city, utc_date)
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username" = "user"
);
```

The columns in the `my_table` are `k1`, `k2`, `k3`, `city`, and `utc_date`.

The directory `hdfs://hdfs_host:hdfs_port/user/doris/data/input/dir/city=beijing` contains the following files:

```
hdfs://hdfs_host:hdfs_port/input/city=beijing/utc_date=2020-10-01/0000.csv
hdfs://hdfs_host:hdfs_port/input/city=beijing/utc_date=2020-10-02/0000.csv
hdfs://hdfs_host:hdfs_port/input/city=tianji/utc_date=2020-10-03/0000.csv
hdfs://hdfs_host:hdfs_port/input/city=tianji/utc_date=2020-10-04/0000.csv
```

The file only contains three columns of data:`k1`,`k2`, and `k3`. The other two columns,`city` and `utc_date`, will be extracted from the file path.

Filter the imported data

```
LOAD LABEL example_db.label6
(
    DATA INFILE("hdfs://host:port/input/file")
    INTO TABLE `my_table`
    (k1, k2, k3)
    SET (
        k2 = k2 + 1
    )
    PRECEDING FILTER k1 = 1
```

```
    WHERE k1 > k2
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username" = "user"
);
```

Only the rows where k1 = 1 in the original data and k1 > k2 after transformation will be imported.

Import data and extract the time partition field from the file path.

```
LOAD LABEL example_db.label7
(
    DATA INFILE("hdfs://host:port/user/data/*/test.txt")
    INTO TABLE `tbl12`
    COLUMNS TERMINATED BY ","
    (k2,k3)
    COLUMNS FROM PATH AS (data_time)
    SET (
        data_time=str_to_date(data_time, '%Y-%m-%d %H%%3A%i%%3A%s')
    )
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username" = "user"
);
```

> **Tip**
>
> The time contains “%3A”. In HDFS paths, colons “:” are not allowed, so all colons are replaced with “%3A”.

There are the following files under the path:

```
/user/data/data_time=2020-02-17 00%3A00%3A00/test.txt
/user/data/data_time=2020-02-18 00%3A00%3A00/test.txt
```

The table structure is as follows:

```
CREATE TABLE IF NOT EXISTS tbl12 (
    data_time DATETIME,
    k2        INT,
    k3        INT
) DISTRIBUTED BY HASH(data_time) BUCKETS 10
PROPERTIES (
    "replication_num" = "3"
);
```

Use Merge mode for import

```
LOAD LABEL example_db.label8
(
    MERGE DATA INFILE("hdfs://host:port/input/file")
    INTO TABLE `my_table`
    (k1, k2, k3, v2, v1)
    DELETE ON v2 > 100
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username"="user"
)
PROPERTIES
(
    "timeout" = "3600",
    "max_filter_ratio" = "0.1"
);
```

To use Merge mode for import, the “my_table” must be a Unique Key table. When the value of the “v2” column in the imported data is greater than 100, that row will be considered a deletion row. The timeout for the import task is 3600 seconds, and an error rate of up to 10% is allowed.

Specify the “source_sequence” column during import to ensure the order of replacements.

```
LOAD LABEL example_db.label9
(
    DATA INFILE("hdfs://host:port/input/file")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY ","
    (k1,k2,source_sequence,v1,v2)
    ORDER BY source_sequence
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username"="user"
);
The "my_table" must be a Unique Key model table and have a specified Sequence column. The data will maintain its order based on the values in the "source_sequence" column in the source data.
```

Import the specified file format as `json`, and specify the `json_root` and jsonpaths accordingly.

```
LOAD LABEL example_db.label10
(
    DATA INFILE("hdfs://host:port/input/file.json")
    INTO TABLE `my_table`
    FORMAT AS "json"
    PROPERTIES(
      "json_root" = "$.item",
      "jsonpaths" = "[\"$.id\", \"$.city\", \"$.code\"]"
    )
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
```

```
  "hadoop.username"="user"
);
```

The `jsonpaths` can also be used in conjunction with the column list and `SET (column_mapping)` :

```
LOAD LABEL example_db.label10
(
    DATA INFILE("hdfs://host:port/input/file.json")
    INTO TABLE `my_table`
    FORMAT AS "json"
    (id, code, city)
    SET (id = id * 10)
    PROPERTIES(
      "json_root" = "$.item",
      "jsonpaths" = "[\"$.id\", \"$.city\", \"$.code\"]"
    )
)
with HDFS
(
  "fs.defaultFS"="hdfs://host:port",
  "hadoop.username"="user"
);
```

> **Note**
>
> If you need to load the JSON object at the root node of a JSON file, the jsonpaths should be specified as $., e.g., `PROPERTIES("jsonpaths"="$.")`”

Load from other brokers

- Alibaba Cloud OSS

  ```
  (
      "fs.oss.accessKeyId" = "",
      "fs.oss.accessKeySecret" = "",
      "fs.oss.endpoint" = ""
  )
  ```

- JuiceFS

  ```
  (
      "fs.defaultFS" = "jfs://xxx/",
      "fs.jfs.impl" = "io.juicefs.JuiceFileSystem",
      "fs.AbstractFileSystem.jfs.impl" = "io.juicefs.JuiceFS",
      "juicefs.meta" = "xxx",
      "juicefs.access-log" = "xxx"
  )
  ```

- GCS

  When using a Broker to access GCS, the Project ID is required, while other parameters are optional. Please refer to the GCS Config for all parameter configurations.

```
(
    "fs.gs.project.id" = "Your Project ID",
    "fs.AbstractFileSystem.gs.impl" = "com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS",
    "fs.gs.impl" = "com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem",
)
```

More Help

For more detailed syntax and best practices for using Broker Load , please refer to the Broker Load command manual. You can also enter HELP BROKER LOAD in the MySQL client command line to obtain more help information.

Routine Load

Doris can continuously consume data from Kafka Topic through the Routine Load method. After submitting a Routine Load job, Doris will continuously run the load job, generating real-time loading tasks to constantly consume messages from the specified Topic in the Kafka cluster.

Routine Load is a streaming load job that supports Exactly-Once semantics, ensuring that data is neither lost nor duplicated.

Usage Scenarios

Supported Data Sources

Routine Load supports consuming data from Kafka clusters.

Supported Data File Formats

Routine Load supports consuming data in CSV and JSON formats.

When loading CSV format, it is necessary to clearly distinguish between null values and empty strings:

- Null values need to be represented with `\n`. For example, `a,\n,b` indicates that the middle column is a null value.
- Empty strings can be represented by leaving the data field empty. For example, `a,,b` indicates that the middle column is an empty string.

Usage Limitations

When using Routine Load to consume data from Kafka, there are the following limitations:

- The supported message formats are CSV and JSON text formats. Each message in CSV should be on a separate line, and the line should not end with a newline character.
- By default, it supports Kafka versions 0.10.0.0 and above. If you need to use a Kafka version below 0.10.0.0 (such as 0.9.0, 0.8.2, 0.8.1, 0.8.0), you need to modify the BE configuration by setting the value of `kafka_broker_version_fallback` to the compatible older version, or directly set the value of `property.broker.version.fallback` when creating the Routine Load. However, using an older version may mean that some new features of Routine Load, such as setting the offset of Kafka partitions based on time, may not be available.

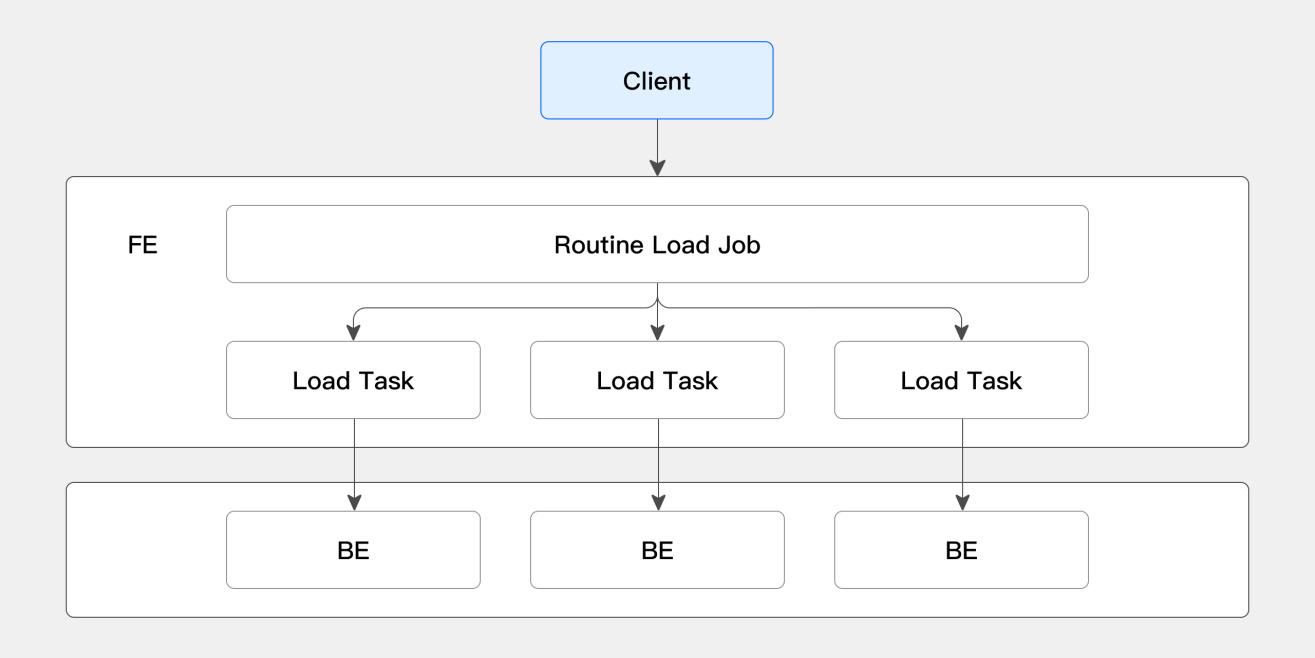
Basic Principles

Routine Load continuously consumes data from Kafka Topics and writes it into Doris.

When a Routine Load job is created in Doris, it generates a resident import job that consists of several import tasks:

- Load Job: A Routine Load Job is a resident import job that continuously consumes data from the data source.
- Load Task: An import job is broken down into several import tasks for actual consumption, with each task being an independent transaction.

The specific import process of Routine Load is shown in the following diagram:



Routine Load

1. The Client submits a request to create a Routine Load job to the FE, and the FE generates a resident import job (Routine Load Job) through the Routine Load Manager.
2. The FE splits the Routine Load Job into several Routine Load Tasks through the Job Scheduler, which are then scheduled by the Task Scheduler and distributed to BE nodes.
3. On the BE, after a Routine Load Task is completed, it submits the transaction to the FE and updates the Job's metadata.
4. After a Routine Load Task is submitted, it continues to generate new Tasks or retries timed-out Tasks.
5. The newly generated Routine Load Tasks continue to be scheduled by the Task Scheduler in a continuous cycle.

Auto Resume

To ensure high availability of jobs, an auto-resume mechanism has been introduced. In the event of an unexpected pause, the Routine Load Scheduler thread will attempt to auto-resume the job. For unexpected Kafka outages or other scenarios where the system is unable to function, the auto-resume mechanism ensures that once Kafka is restored, the routine load job can continue running normally without manual intervention.

Situations where auto-resume will not occur:

- The user manually executes the PAUSE ROUTINE LOAD command.

- There are issues with data quality.
- Situations where resuming is not possible, such as when a database table is deleted.

Apart from these three situations, other paused jobs will attempt to resume automatically.

FAQ

Auto-resume may encounter some issues during cluster restarts or upgrades. Before version 2.1.7, there was a high probability that tasks would not automatically resume after being paused due to cluster restarts or upgrades. Since version 2.1.7, the likelihood of tasks not resuming automatically after such events has decreased.

Quick Start

Create Job

In Doris, you can create persistent Routine Load tasks using the `CREATE ROUTINE LOAD` command. For detailed syntax, please refer to CREATE ROUTINE LOAD. Routine Load supports consuming data in CSV and JSON formats.

Loading CSV Data

1. Loading data sample

   In Kafka, there is the following sample data:

   ```
   kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-routine-load-csv --from-beginning
   1,Emily,25
   2,Benjamin,35
   3,Olivia,28
   4,Alexander,60
   5,Ava,17
   6,William,69
   7,Sophia,32
   8,James,64
   9,Emma,37
   10,Liam,64
   ```

2. Creating table

   In Doris, create the table for loading with the following syntax:

   ```
   CREATE TABLE testdb.test_routineload_tbl(
       user_id            BIGINT       NOT NULL COMMENT "User ID",
       name               VARCHAR(20)           COMMENT "User Name",
       age                INT                   COMMENT "User Age"
   )
   DUPLICATE KEY(user_id)
   DISTRIBUTED BY HASH(user_id) BUCKETS 10;
   ```

3. Creating the Routine Load job

   In Doris, use the `CREATE ROUTINE LOAD` command to create the load job:

```
CREATE ROUTINE LOAD testdb.example_routine_load_csv ON test_routineload_tbl
COLUMNS TERMINATED BY ",",
COLUMNS(user_id, name, age)
FROM KAFKA(
    "kafka_broker_list" = "192.168.88.62:9092",
    "kafka_topic" = "test-routine-load-csv",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

Loading JSON Data

1. Loading sample data

   In Kafka, there is the following sample data:

   ```
   kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-routine-load-json --from-beginning
   ```

2. Creating table

   In Doris, create the table for loading with the following syntax:

   ```
   CREATE TABLE testdb.test_routineload_tbl(
       user_id            BIGINT       NOT NULL COMMENT "User ID",
       name               VARCHAR(20)           COMMENT "User Name",
       age                INT                   COMMENT "User Age"
   )
   DUPLICATE KEY(user_id)
   DISTRIBUTED BY HASH(user_id) BUCKETS 10;
   ```

3. Creating the Routine Load job

   In Doris, use the `CREATE ROUTINE LOAD` command to create the job:

   ```
   CREATE ROUTINE LOAD testdb.example_routine_load_json ON test_routineload_tbl
   COLUMNS(user_id, name, age)
   PROPERTIES(
       "format"="json",
       "jsonpaths"="[\"$.user_id\",\"$.name\",\"$.age\"]"
   )
   FROM KAFKA(
       "kafka_broker_list" = "192.168.88.62:9092"
   );
   ```

> **Note**
>
> If you need to load the JSON object at the root node of a JSON file, the jsonpaths should be specified as $., e.g., `PROPERTIES("jsonpaths"="$.")`”

Viewing Status

In Doris, you can check the status of Routine Load jobs and tasks using the following methods:

- Load Jobs: Used to view information about load tasks, such as the target table, number of subtasks, load delay status, load configuration, and load results.
- Load Tasks: Used to view the status of individual load tasks, including task ID, transaction status, task status, execution start time, and BE (Backend) node assignment.

01 Viewing Running Jobs

You can use the `SHOW ROUTINE LOAD` command to check the status of jobs. The `SHOW ROUTINE LOAD` command provides information about the current job, including the target table, load delay status, load configuration, and error messages.

For example, to view the status of the `testdb.example_routine_load_csv` job, you can run the following command:

```
mysql> SHOW ROUTINE LOAD FOR testdb.example_routine_load\G
*************************** 1. row ***************************
                  Id: 12025
                Name: example_routine_load
          CreateTime: 2024-01-15 08:12:42
           PauseTime: NULL
             EndTime: NULL
              DbName: default_cluster:testdb
           TableName: test_routineload_tbl
        IsMultiTable: false
               State: RUNNING
      DataSourceType: KAFKA
      CurrentTaskNum: 1
       JobProperties:
{"max_batch_rows":"200000","timezone":"America/New_York","send_batch_parallelism":"1","load_to_single_tablet":"false","column_separa
tor":"','","line_delimiter":"\n","current_concurrent_number":"1","delete":"*","partial_columns":"false","merge_type":"APPEND","exec_
mem_limit":"2147483648","strict_mode":"false","jsonpaths":"","max_batch_interval":"10","max_batch_size":"104857600","fuzzy_parse":"f
alse","partitions":"*","columnToColumnExpr":"user_id,name,age","whereExpr":"*","desired_concurrent_number":"5","precedingFilter":"*"
,"format":"csv","max_error_number":"0","max_filter_ratio":"1.0","json_root":"","strip_outer_array":"false","num_as_string":"false"}
DataSourceProperties: {"topic":"test-topic","currentKafkaPartitions":"0","brokerList":"192.168.88.62:9092"}
    CustomProperties: {"kafka_default_offsets":"OFFSET_BEGINNING","group.id":"example_routine_load_73daf600-884e-46c0-a02b-
4e49fdf3b4dc"}
           Statistic: {"receivedBytes":28,"runningTxns":
[],"errorRows":0,"committedTaskNum":3,"loadedRows":3,"loadRowsRate":0,"abortedTaskNum":0,"errorRowsAfterResumed":0,"totalRows":3,"un
selectedRows":0,"receivedBytesRate":0,"taskExecuteTimeMs":30069}
            Progress: {"0":"2"}
                 Lag: {"0":0}
ReasonOfStateChanged:
        ErrorLogUrls:
            OtherMsg:
                User: root
             Comment:
1 row in set (0.00 sec)
```

02 Viewing Running Tasks

You can use the `SHOW ROUTINE LOAD TASK` command to check the status of load tasks. The `SHOW ROUTINE LOAD TASK` command provides information about the individual tasks under a specific load job, including task ID, transaction status, task status, execution start time, and BE ID.

For example, to view the task status of the `example_routine_load_csv` job, you can run the following command:

```
mysql> SHOW ROUTINE LOAD TASK WHERE jobname = 'example_routine_load_csv';
+----------------------------------+-------+----------+-------+--------------------+--------------------+---------+-------+-----
```

```
-----------------+
| TaskId                           | TxnId | TxnStatus | JobId | CreateTime          | ExecuteStartTime    | Timeout | BeId  |
DataSourceProperties |
+----------------------------------+-------+-----------+-------+---------------------+---------------------+---------+-------+-----
-----------------+
| 8cf47e6a68ed4da3-8f45b431db50e466 | 195   | PREPARE   | 12177 | 2024-01-15 12:20:41 | 2024-01-15 12:21:01 | 20      | 10429 |
{"4":1231,"9":2603}  |
| f2d4525c54074aa2-b6478cf8daaeb393 | 196   | PREPARE   | 12177 | 2024-01-15 12:20:41 | 2024-01-15 12:21:01 | 20      | 12109 |
{"1":1225,"6":1216}  |
| cb870f1553864250-975279875a25fab6 | -1    | NULL      | 12177 | 2024-01-15 12:20:52 | NULL                | 20      | -1    |
{"2":7234,"7":4865}  |
| 68771fd8a1824637-90a9dac2a7a0075e | -1    | NULL      | 12177 | 2024-01-15 12:20:52 | NULL                | 20      | -1    |
{"3":1769,"8":2982}  |
| 77112dfea5e54b0a-a10eab3d5b19e565 | 197   | PREPARE   | 12177 | 2024-01-15 12:21:02 | 2024-01-15 12:21:02 | 20      | 12098 |
{"0":3000,"5":2622}  |
+----------------------------------+-------+-----------+-------+---------------------+---------------------+---------+-------+-----
-----------------+
```

Pausing Jobs

You can pause an load job using the PAUSE ROUTINE LOAD command. When a job is paused, it enters the PAUSED state, but the load job is not terminated and can be resumed using the RESUME ROUTINE LOAD command.

To pause the `testdb.example_routine_load_csv` load job, you can use the following command:

```
PAUSE ROUTINE LOAD FOR testdb.example_routine_load_csv;
```

Resuming Jobs

You can resume a paused load job using the RESUME ROUTINE LOAD command.

To resume the `testdb.example_routine_load_csv` job, you can use the following command:

```
RESUME ROUTINE LOAD FOR testdb.example_routine_load_csv;
```

Modifying Jobs

You can modify a created loading job using the ALTER ROUTINE LOAD command. Before modifying the job, you need to pause it using the`PAUSE ROUTINE LOAD` command, and after making the modifications, you can resume it using the `RESUME ROUTINE LOAD` command.

To modify the `desired_concurrent_number` parameter for the job and update the Kafka topic information, you can use the following command:

```
ALTER ROUTINE LOAD FOR testdb.example_routine_load_csv
PROPERTIES(
    "desired_concurrent_number" = "3"
)
FROM KAFKA(
    "kafka_broker_list" = "192.168.88.60:9092",
    "kafka_topic" = "test-topic"
);
```

Canceling Jobs

You can stop and delete a Routine Load job using the STOP ROUTINE LOAD command. Once deleted, the load job cannot be recovered and cannot be viewed using the `SHOW ROUTINE LOAD` command.

To stop and delete the `testdb.example_routine_load_csv` load job, you can use the following command:

```
STOP ROUTINE LOAD FOR testdb.example_routine_load_csv;
```

Reference Manual

Load Commands

The syntax for creating a Routine Load persistent load job is as follows:

```
CREATE ROUTINE LOAD [<db_name>.]<job_name> [ON <tbl_name>]
[merge_type]
[load_properties]
[job_properties]
FROM KAFKA [data_source_properties]
[COMMENT "<comment>"]
```

The modules for creating a loading job are explained as follows:

Module	Description
db_name	Specifies the name of the database for creating the loading task.
job_name	Specifies the name of the created loading job. The job name must be unique within the same database.
tbl_name	Specifies the name of the table to be loaded. This parameter is optional. If not specified, the dynamic table mode will be used, where Kafka data should contain the table name information.
merge_type	Specifies the data merge type. The default value is APPEND. Possible merge_type options are:
load_properties	Describes the load properties, including:
job_properties	Specifies the general load parameters for Routine Load.
data_source_properties	Describes the properties of Kafka data source.
comment	Describes any additional comments for the loading job.

Load Parameter Description

01 FE Configuration Parameters

Parameter Name	Default Value	Dynamic Configuration	FE Master Exclusive Configuration	Description
max_routine_load_task_concurrent_num	256	Yes	Yes	Limits the maximum number of concurrent subtasks for Routine Load jobs. It is recommended to maintain the default value. If set too high, it may lead to excessive concurrent tasks, consuming cluster resources.
max_routine_load_task_num_per_be	1024	Yes	Yes	The maximum number of concurrent Routine Load tasks allowed per BE. `max_routine_load_task_num_per_be` should be less than `routine_load_thread_pool_size`.
max_routine_load_job_num	100	Yes	Yes	Limits the maximum number of Routine Load jobs, including NEED_SCHEDULED, RUNNING, and PAUSE.
max_tolerable_backend_down_num	0	Yes	Yes	If any BE is down, Routine Load cannot automatically recover. Under certain conditions, Doris can reschedule PAUSED tasks to RUNNING state. A value of 0 means that rescheduling is only allowed when all BE nodes are alive.
period_of_auto_resume_min	5 (minutes)	Yes	Yes	The period for automatically resuming Routine Load.

02 BE Configuration Parameters

Parameter Name	Default Value	Dynamic Configuration	Description
max_consumer_num_per_group	3	Yes	The maximum number of consumers that can be generated for a subtask to consume data. For Kafka data sources, a consumer may consume one or more Kafka partitions. If a task needs to consume 6 Kafka partitions, it will generate 3 consumers, each consuming 2 partitions. If there are only 2 partitions, it will generate only 2 consumers, each consuming 1 partition.

Load Configuration Parameters

When creating a Routine Load job, you can specify the load configuration parameters for different modules using the `CREATE ROUTINE LOAD` command.

tbl_name Clause

Specifies the name of the table to be loaded. This parameter is optional.

If not specified, the dynamic table mode is used, which requires the data in Kafka to contain the table name information. Currently, only extracting the table name from the Value field of Kafka is supported. The format should be as follows, using JSON as an example: `table_name|{"col1": "val1", "col2": "val2"}`, where `tbl_name` is the table name and | is used as the separator between the table name and the table data. The same format applies to CSV data, such as `table_name|val1,val2,val3`. Note that the `table_name` here must be consistent with the table name in Doris, otherwise the load will fail. Note that dynamic tables do not support the column_mapping configuration described later.

merge_type Clause

The merge_type module specifies the type of data merging. There are three options for merge_type:

- APPEND: Append load mode.
- MERGE: Merge load mode. Only applicable to Unique Key models. It needs to be used together with the [DELETE ON] module to mark the Delete Flag column.
- DELETE: All loaded data is data that needs to be deleted.

load_properties Clause

The load_properties module describes the properties of the loaded data using the following syntax:

```
[COLUMNS TERMINATED BY <column_separator>,]
[COLUMNS (<column1_name>[, <column2_name>, <column_mapping>, ...]),]
[WHERE <where_expr>,]
[PARTITION(<partition1_name>, [<partition2_name>, <partition3_name>, ...]),]
[DELETE ON <delete_expr>,]
[ORDER BY <order_by_column1>[, <order_by_column2>, <order_by_column3>, ...]]
```

The specific parameters for each module are as follows:

Submodule	Parameter	Description
COLUMNS TERMINATED BY	`<column_separator>`	Specifies the column delimiter, defaulting to `\t`. For example, to specify a comma as the delimiter, use `COLUMNS TERMINATED BY ","`. When handling empty values, note the following:
COLUMNS	`<column_name>`	Specifies the corresponding column names. For example, to specify the load columns as `(k1, k2, k3)`, use `COLUMNS(k1, k2, k3)`. The COLUMNS clause can be omitted in the following cases:
	`<column_mapping>`	During the load process, column mapping can be used to filter and transform columns. For example, if the target column needs to perform a derived calculation based on a column in the data source (e.g., the target column k4 is calculated as k3 + 1 based on the k3 column), you can use `COLUMNS(k1, k2, k3, k4 = k3 + 1)`. For more details, refer to the Data Conversion documentation.
WHERE	`<where_expr>`	Specifies the condition to filter the loaded data source. For example, to load only data where age > 30, use `WHERE age > 30`.
PARTITION	`<partition_name>`	Specifies which partitions in the target table to load. If not specified, it will automatically load into the corresponding partitions. For example, to load partitions p1 and p2 of the target table, use `PARTITION(p1, p2)`.
DELETE ON	`<delete_expr>`	In the MERGE load mode, using delete_expr to mark which columns need to be deleted. For example, to delete columns where age > 30 during the MERGE process, use `DELETE ON age > 30`.
ORDER BY	`<order_by_column>`	Only effective for Unique Key models. Specifies the Sequence Column in the loaded data to ensure the order of the data. For example, when loading into a Unique Key table and specifying create_time as the Sequence Column, use `ORDER BY create_time`. For more information on Sequence Columns in Unique Key models, refer to the Data Update/Sequence Columns

job_properties Clause

The job_properties clause is used to specify the properties of a Routine Load job when creating it. The syntax is as follows:

```
PROPERTIES ("<key1>" = "<value1>"[, "<key2>" = "<value2>" ...])
```

Here are the available parameters for the job_properties clause:

Parameter	Description
desired_concurrent_number	
max_batch_interval	The maximum running time for each subtask, in seconds. Must be greater than 0, with a default value of 60s. max_batch_interval/max_batch_rows/max_batch_size together form the execution threshold for subtasks. If any of these parameters reaches the threshold, the load subtask ends and a new one is generated.
max_batch_rows	The maximum number of rows read by each subtask. Must be greater than or equal to 200,000. The default value is 20,000,000. max_batch_interval/max_batch_rows/max_batch_size together form the execution threshold for subtasks. If any of these parameters reaches the threshold, the load subtask ends and a new one is generated.
max_batch_size	The maximum number of bytes read by each subtask. The unit is bytes, and the range is from 100MB to 10GB. The default value is 1G. max_batch_interval/max_batch_rows/max_batch_size together form the execution threshold for subtasks. If any of these parameters reaches the threshold, the load subtask ends and a new one is generated.
max_error_number	The maximum number of error rows allowed within a sampling window. Must be greater than or equal to 0. The default value is 0, which means no error rows are allowed. The sampling window is `max_batch_rows * 10`. If the number of error rows within the sampling window exceeds `max_error_number`, the regular job will be paused and manual intervention is required to check for data quality issues using the SHOW ROUTINE LOAD command and `ErrorLogUrls`. Rows filtered out by the WHERE condition are not counted as error rows.
strict_mode	Whether to enable strict mode. The default value is disabled. Strict mode applies strict filtering to type conversions during the load process. If enabled, non-null original data that results in a NULL after type conversion will be filtered out. The filtering rules in strict mode are as follows:
timezone	Specifies the time zone used by the load job. The default is to use the session's timezone parameter. This parameter affects the results of all timezone-related functions involved in the load.
format	Specifies the data format for the load. The default is CSV, and JSON format is supported.
jsonpaths	When the data format is JSON, jsonpaths can be used to specify the JSON paths to extract data from nested structures. It is a JSON array of strings, where each string represents a JSON path.
json_root	When importing JSON format data, you can specify the root node of the JSON data through json_root. Doris will extract and parse elements from the root node. Default is empty. For example, specify the JSON root node with: `"json_root" = "$.RECORDS"`
strip_outer_array	When importing JSON format data, if strip_outer_array is true, it indicates that the JSON data is presented as an array, and each element in the data will be treated as a row. Default value is false. Typically, JSON data in Kafka might be represented as an array with square brackets `[]` in the outermost layer. In this case, you can specify `"strip_outer_array" = "true"` to consume Topic data in array mode. For example, the following data will be parsed into two rows: `[{"user_id":1,"name":"Emily","age":25},{"user_id":2,"name":"Benjamin","age":35}]`

Parameter	Description
send_batch_parallelism	Used to set the parallelism of sending batch data. If the parallelism value exceeds the `max_send_batch_parallelism_per_job` in BE configuration, the coordinating BE will use the value of `max_send_batch_parallelism_per_job`.
load_to_single_tablet	Supports importing data to only one tablet in the corresponding partition per task. Default value is false. This parameter can only be set when importing data to OLAP tables with random bucketing.
partial_columns	Specifies whether to enable partial column update feature. Default value is false. This parameter can only be set when the table model is Unique and uses Merge on Write. Multi-table streaming does not support this parameter. For details, refer to Partial Column Update
max_filter_ratio	The maximum allowed filter ratio within the sampling window. Must be between 0 and 1 inclusive. Default value is 1.0, indicating any error rows can be tolerated. The sampling window is `max_batch_rows * 10`. If the ratio of error rows to total rows within the sampling window exceeds `max_filter_ratio`, the routine job will be suspended and require manual intervention to check data quality issues. Rows filtered by WHERE conditions are not counted as error rows.
enclose	Specifies the enclosing character. When CSV data fields contain line or column separators, a single-byte character can be specified as an enclosing character for protection to prevent accidental truncation. For example, if the column separator is “,” and the enclosing character is ““, the data”a,’b,c’” will have “b,c” parsed as one field.
escape	Specifies the escape character. Used to escape characters in fields that are identical to the enclosing character. For example, if the data is “a,‘b,’c’”, the enclosing character is ““, and you want”b,’c” to be parsed as one field, you need to specify a single-byte escape character, such as “", and modify the data to”a,’b,'c’”.

These parameters can be used to customize the behavior of a Routine Load job according to your specific requirements.

04 data_source_properties Clause

When creating a Routine Load job, you can specify the data_source_properties clause to specify properties of the Kafka data source. The syntax is as follows:

```
FROM KAFKA ("<key1>" = "<value1>"[, "<key2>" = "<value2>" ...])
```

The available options for the data_source_properties clause are as follows:

Parameter	Description
kafka_broker_list	Specifies the connection information for Kafka brokers. The format is `<kafka_broker_ip>:<kafka_port>`. Multiple brokers are separated by commas. For example, to specify a Broker List with the default port 9092, you can use the following command: `"kafka_broker_list" = "<broker1_ip>:9092,<broker2_ip>:9092"`
kafka_topic	Specifies the Kafka topic to subscribe to. A load job can only consume one Kafka topic.
kafka_partitions	Specifies the Kafka partitions to subscribe to. If not specified, all partitions are consumed by default.
kafka_offsets	Specifies the starting consumption offset for Kafka partitions. If a timestamp is specified, consumption starts from the nearest offset equal to or greater than that timestamp. The offset can be a specific offset greater than or equal to 0, or it can use the following formats:

Parameter	Description
property	Specifies custom Kafka parameters. This is equivalent to the “–property” parameter in the Kafka shell. When the value of a parameter is a file, the keyword “FILE:” needs to be added before the value. For creating a file, you can refer to the CREATE FILE command documentation. For more supported custom parameters, you can refer to the client-side configuration options in the official CONFIGURATION documentation of librdkafka. For example: `"property.client.id" = "12345"`, `"property.group.id" = "group_id_0"`, `"property.ssl.ca.location" = "FILE:ca.pem"`

By configuring the Kafka property parameter in the `data_source_properties`, you can set up security access options. Currently, Doris supports various Kafka security protocols such as plaintext (default), SSL, PLAIN, and Kerberos.

Load Status

You can check the status of a load job using the `SHOW ROUTINE LOAD` command. The syntax for the command is as follows:

```
SHOW [ALL] ROUTINE LOAD [FOR jobName];
```

For example, executing `SHOW ROUTINE LOAD` will return a result set similar to the following:

```
mysql> SHOW ROUTINE LOAD FOR testdb.example_routine_load\G
*************************** 1. row ***************************
                  Id: 12025
                Name: example_routine_load
          CreateTime: 2024-01-15 08:12:42
           PauseTime: NULL
             EndTime: NULL
              DbName: default_cluster:testdb
           TableName: test_routineload_tbl
        IsMultiTable: false
               State: RUNNING
      DataSourceType: KAFKA
      CurrentTaskNum: 1
       JobProperties:
{"max_batch_rows":"200000","timezone":"America/New_York","send_batch_parallelism":"1","load_to_single_tablet":"false","column_separa
tor":"','","line_delimiter":"\n","current_concurrent_number":"1","delete":"*","partial_columns":"false","merge_type":"APPEND","exec_
mem_limit":"2147483648","strict_mode":"false","jsonpaths":"","max_batch_interval":"10","max_batch_size":"104857600","fuzzy_parse":"f
alse","partitions":"*","columnToColumnExpr":"user_id,name,age","whereExpr":"*","desired_concurrent_number":"5","precedingFilter":"*"
,"format":"csv","max_error_number":"0","max_filter_ratio":"1.0","json_root":"","strip_outer_array":"false","num_as_string":"false"}
DataSourceProperties: {"topic":"test-topic","currentKafkaPartitions":"0","brokerList":"192.168.88.62:9092"}
    CustomProperties: {"kafka_default_offsets":"OFFSET_BEGINNING","group.id":"example_routine_load_73daf600-884e-46c0-a02b-
4e49fdf3b4dc"}
           Statistic: {"receivedBytes":28,"runningTxns":
[],"errorRows":0,"committedTaskNum":3,"loadedRows":3,"loadRowsRate":0,"abortedTaskNum":0,"errorRowsAfterResumed":0,"totalRows":3,"un
selectedRows":0,"receivedBytesRate":0,"taskExecuteTimeMs":30069}
            Progress: {"0":"2"}
                 Lag: {"0":0}
ReasonOfStateChanged:
        ErrorLogUrls:
            OtherMsg:
                User: root
             Comment:
1 row in set (0.00 sec)
```

The columns in the result set provide the following information:

Column Name	Description
Id	The ID of the load job, automatically generated by Doris.
Name	The name of the load job.
CreateTime	The time when the job was created.
PauseTime	The time when the job was last paused.
EndTime	The time when the job ended.
DbName	The name of the associated database.
TableName	The name of the associated table. For multi-table scenarios, it is displayed as “multi-table”.
IsMultiTbl	Indicates whether it is a multi-table load.
State	The running state of the job, which can have five values:
DataSourceType	The type of data source, which is KAFKA, in this example.
CurrentTaskNum	The current number of subtasks.
JobProperties	Details of the job configuration.
DataSourceProperties	Details of the data source configuration.
CustomProperties	Custom configuration properties.
Statistic	Statistics of the job’s running status.
Progress	The job’s progress. For Kafka data sources, it shows the offset consumed for each partition. For example, `{"0":"2"}` indicates that partition 0 has consumed 2 offsets.
Lag	The lag of the job. For Kafka data sources, it shows the consumption lag for each partition. For example, `{"0":10}` indicates a consumption lag of 10 for partition 0.
ReasonOfStateChanged	The reason for the state change of jobs.
ErrorLogUrls	The URL(s) to view the filtered low-quality data.
OtherMsg	Other error messages.

Load example

Setting the Maximum Error Tolerance

1. Load sample data:

```
1,Benjamin,18
2,Emily,20
3,Alexander,dirty_data
```

2. Create table:

```
CREATE TABLE demo.routine_test01 (
    id       INT             NOT NULL   COMMENT "User ID",
    name     VARCHAR(30)     NOT NULL   COMMENT "Name",
    age      INT                        COMMENT "Age"
)
```

```
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job01 ON routine_test01
        COLUMNS TERMINATED BY ","
        PROPERTIES
        (
            "max_filter_ratio"="0.5",
            "max_error_number" = "100",
            "strict_mode" = "true"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad01",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test01;
+------+------------+------+
| id   | name       | age  |
+------+------------+------+
|    1 | Benjamin   |   18 |
|    2 | Emily      |   20 |
+------+------------+------+
2 rows in set (0.01 sec)
```

Consuming Data from a Specified Offset

1. Load sample data:

```
1,Benjamin,18
2,Emily,20
3,Alexander,22
4,Sophia,24
5,William,26
6,Charlotte,28
```

2. Create table:

```
CREATE TABLE demo.routine_test02 (
    id       INT             NOT NULL   COMMENT "User ID",
    name     VARCHAR(30)     NOT NULL   COMMENT "Name",
    age      INT                        COMMENT "Age"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job02 ON routine_test02
        COLUMNS TERMINATED BY ","
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad02",
            "kafka_partitions" = "0",
            "kafka_offsets" = "3"
        );
```

4. Load result:

```
mysql> select * from routine_test02;
+------+--------------+------+
| id   | name         | age  |
+------+--------------+------+
|    4 | Sophia       |   24 |
|    5 | William      |   26 |
|    6 | Charlotte    |   28 |
+------+--------------+------+
3 rows in set (0.01 sec)
```

Specifying the Consumer Group's group.id and client.id

1. Load sample data:

```
1,Benjamin,18
2,Emily,20
3,Alexander,22
```

2. Create table:

```
CREATE TABLE demo.routine_test03 (
    id       INT             NOT NULL   COMMENT "User ID",
    name     VARCHAR(30)     NOT NULL   COMMENT "Name",
    age      INT                        COMMENT "Age"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job03 ON routine_test03
        COLUMNS TERMINATED BY ","
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad01",
            "property.group.id" = "kafka_job03",
            "property.client.id" = "kafka_client_03",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test03;
+------+------------+------+
| id   | name       | age  |
+------+------------+------+
|    1 | Benjamin   |   18 |
|    2 | Emily      |   20 |
|    3 | Alexander  |   22 |
+------+------------+------+
3 rows in set (0.01 sec)
```

Setting load filtering conditions

1. Load sample data:

```
1,Benjamin,18
2,Emily,20
3,Alexander,22
4,Sophia,24
5,William,26
6,Charlotte,28
```

2. Create table:

```
CREATE TABLE demo.routine_test04 (
    id       INT             NOT NULL   COMMENT "User ID",
    name     VARCHAR(30)     NOT NULL   COMMENT "Name",
    age      INT                        COMMENT "Age"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job04 ON routine_test04
        COLUMNS TERMINATED BY ",",
        WHERE id >= 3
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad04",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test04;
+------+--------------+------+
| id   | name         | age  |
+------+--------------+------+
|    4 | Sophia       |   24 |
|    5 | William      |   26 |
|    6 | Charlotte    |   28 |
+------+--------------+------+
3 rows in set (0.01 sec)
```

Loading specified partition data

1. Load sample data:

```
1,Benjamin,18,2024-02-04 10:00:00
2,Emily,20,2024-02-05 11:00:00
3,Alexander,22,2024-02-06 12:00:00
```

2. Create table:

```
CREATE TABLE demo.routine_test05 (
    id      INT            NOT NULL  COMMENT "ID",
    name    VARCHAR(30)    NOT NULL  COMMENT "Name",
    age     INT                      COMMENT "Age",
    date    DATETIME                 COMMENT "Date"
)
DUPLICATE KEY(`id`)
PARTITION BY RANGE(`id`)
(PARTITION partition_a VALUES [("0"), ("1")),
PARTITION partition_b VALUES [("1"), ("2")),
PARTITION partition_c VALUES [("2"), ("3")))
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job05 ON routine_test05
        COLUMNS TERMINATED BY ",",
        PARTITION(partition_b)
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad05",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test05;
+------+----------+------+---------------------+
| id   | name     | age  | date                |
+------+----------+------+---------------------+
|    1 | Benjamin |   18 | 2024-02-04 10:00:00 |
+------+----------+------+---------------------+
1 rows in set (0.01 sec)
```

Setting Time Zone for load

1. Load sample data:

```
1,Benjamin,18,2024-02-04 10:00:00
2,Emily,20,2024-02-05 11:00:00
3,Alexander,22,2024-02-06 12:00:00
```

2. Create table:

```
CREATE TABLE demo.routine_test06 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age",
    date    DATETIME                 COMMENT "date"
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job06 ON routine_test06
        COLUMNS TERMINATED BY ","
        PROPERTIES
        (
            "timezone" = "Asia/Shanghai"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad06",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test06;
+------+-------------+------+---------------------+
| id   | name        | age  | date                |
+------+-------------+------+---------------------+
|    1 | Benjamin    |   18 | 2024-02-04 10:00:00 |
|    2 | Emily       |   20 | 2024-02-05 11:00:00 |
|    3 | Alexander   |   22 | 2024-02-06 12:00:00 |
+------+-------------+------+---------------------+
3 rows in set (0.00 sec)
```

Setting merge_type

Specify merge_type for delete operation

1. Load sample data:

```
3,Alexander,22
5,William,26
```

Table data before load:

```
mysql> SELECT * FROM routine_test07;
+------+----------------+------+
| id   | name           | age  |
+------+----------------+------+
|    1 | Benjamin       |   18 |
|    2 | Emily          |   20 |
|    3 | Alexander      |   22 |
|    4 | Sophia         |   24 |
|    5 | William        |   26 |
```

```
|    6 | Charlotte      |   28 |
+------+----------------+------+
```

2. Create table:

```
CREATE TABLE demo.routine_test07 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age"
)
UNIQUE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job07 ON routine_test07
        WITH DELETE
        COLUMNS TERMINATED BY ","
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad07",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> SELECT * FROM routine_test07;
+------+----------------+------+
| id   | name           | age  |
+------+----------------+------+
|    1 | Benjamin       |   18 |
|    2 | Emily          |   20 |
|    4 | Sophia         |   24 |
|    6 | Charlotte      |   28 |
+------+----------------+------+
```

Specify merge_type for merge operation

1. Load sample data:

```
1,xiaoxiaoli,28
2,xiaoxiaowang,30
3,xiaoxiaoliu,32
4,dadali,34
5,dadawang,36
6,dadaliu,38
```

Table data before load:

```
mysql> SELECT * FROM routine_test08;
+------+----------------+------+
| id   | name           | age  |
+------+----------------+------+
|    1 | Benjamin       |   18 |
|    2 | Emily          |   20 |
```

```
|    3 | Alexander      |   22 |
|    4 | Sophia         |   24 |
|    5 | William        |   26 |
|    6 | Charlotte      |   28 |
+------+----------------+------+
6 rows in set (0.01 sec)
```

2. Create table:

```
CREATE TABLE demo.routine_test08 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age"
)
UNIQUE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job08 ON routine_test08
        WITH MERGE
        COLUMNS TERMINATED BY ",",
        DELETE ON id = 2
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad08",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> SELECT * FROM routine_test08;
+------+-------------+------+
| id   | name        | age  |
+------+-------------+------+
|    1 | xiaoxiaoli  |   28 |
|    3 | xiaoxiaoliu |   32 |
|    4 | dadali      |   34 |
|    5 | dadawang    |   36 |
|    6 | dadaliu     |   38 |
+------+-------------+------+
5 rows in set (0.00 sec)
```

Specifying the sequence column to be merged

1. Load sample data:

```
1,xiaoxiaoli,28
2,xiaoxiaowang,30
3,xiaoxiaoliu,32
4,dadali,34
5,dadawang,36
6,dadaliu,38
```

Data in the table before loading:

```
mysql> SELECT * FROM routine_test09;
+------+----------------+------+
| id   | name           | age  |
+------+----------------+------+
|    1 | Benjamin       |   18 |
|    2 | Emily          |   20 |
|    3 | Alexander      |   22 |
|    4 | Sophia         |   24 |
|    5 | William        |   26 |
|    6 | Charlotte      |   28 |
+------+----------------+------+
6 rows in set (0.01 sec)
```

2. Create table

```
CREATE TABLE demo.routine_test08 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age",
)
UNIQUE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "function_column.sequence_col" = "age"
);
```

3. Load Command

```
CREATE ROUTINE LOAD demo.kafka_job09 ON routine_test09
        WITH MERGE
        COLUMNS TERMINATED BY ",",
        COLUMNS(id, name, age),
        DELETE ON id = 2,
        ORDER BY age
        PROPERTIES
        (
            "desired_concurrent_number"="1",
            "strict_mode" = "false"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad09",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> SELECT * FROM routine_test09;
+------+-------------+------+
| id   | name        | age  |
+------+-------------+------+
|    1 | xiaoxiaoli  |   28 |
|    3 | xiaoxiaoliu |   32 |
|    4 | dadali      |   34 |
|    5 | dadawang    |   36 |
|    6 | dadaliu     |   38 |
```

```
+------+-------------+------+
5 rows in set (0.00 sec)
```

Load with column mapping and derived column calculation

1. Load sample data:

```
1,Benjamin,18
2,Emily,20
3,Alexander,22
```

2. Create table:

```
CREATE TABLE demo.routine_test10 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age",
    num     INT                      COMMENT "number"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job10 ON routine_test10
        COLUMNS TERMINATED BY ",",
        COLUMNS(id, name, age, num=age*10)
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad10",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> SELECT * FROM routine_test10;
+------+----------------+------+------+
| id   | name           | age  | num  |
+------+----------------+------+------+
|    1 | Benjamin       |   18 |  180 |
|    2 | Emily          |   20 |  200 |
|    3 | Alexander      |   22 |  220 |
+------+----------------+------+------+
3 rows in set (0.01 sec)
```

Load with enclosed data

1. Load sample data:

```
1,"Benjamin",18
2,"Emily",20
3,"Alexander",22
```

2. Create table:

```
CREATE TABLE demo.routine_test11 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age",
    num     INT                      COMMENT "number"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job11 ON routine_test11
        COLUMNS TERMINATED BY ","
        PROPERTIES
        (
            "desired_concurrent_number"="1",
            "enclose" = "\""
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad12",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> SELECT * FROM routine_test11;
+------+----------------+------+------+
| id   | name           | age  | num  |
+------+----------------+------+------+
|    1 | Benjamin       |   18 |  180 |
|    2 | Emily          |   20 |  200 |
|    3 | Alexander      |   22 |  220 |
+------+----------------+------+------+
3 rows in set (0.02 sec)
```

JSON Format Load

Load JSON format data in simple mode

1. Load sample data:

```
{ "id" : 1, "name" : "Benjamin", "age":18 }
{ "id" : 2, "name" : "Emily", "age":20 }
{ "id" : 3, "name" : "Alexander", "age":22 }
```

2. Create table:

```
CREATE TABLE demo.routine_test12 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age"
)
```

```
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job12 ON routine_test12
        PROPERTIES
        (
            "format" = "json"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad12",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test12;
+------+----------------+------+
| id   | name           | age  |
+------+----------------+------+
|    1 | Benjamin       |   18 |
|    2 | Emily          |   20 |
|    3 | Alexander      |   22 |
+------+----------------+------+
3 rows in set (0.02 sec)
```

Load complex JSON format data in match mode

1. Load sample data

```
{ "name" : "Benjamin", "id" : 1, "num":180 , "age":18 }
{ "name" : "Emily", "id" : 2, "num":200 , "age":20 }
{ "name" : "Alexander", "id" : 3, "num":220 , "age":22 }
```

2. Create table:

```
CREATE TABLE demo.routine_test13 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age",
    num     INT                      COMMENT "num"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job13 ON routine_test13
        COLUMNS(name, id, num, age)
        PROPERTIES
        (
            "format" = "json",
            "jsonpaths" = "[\"$.name\",\"$.id\",\"$.num\",\"$.age\"]"
```

```
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad13",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test13;
+------+----------------+------+------+
| id   | name           | age  | num  |
+------+----------------+------+------+
|    1 | Benjamin       |   18 |  180 |
|    2 | Emily          |   20 |  200 |
|    3 | Alexander      |   22 |  220 |
+------+----------------+------+------+
3 rows in set (0.01 sec)
```

Loading data with specified JSON root node

1. Load sample data

```
{"id": 1231, "source" :{ "id" : 1, "name" : "Benjamin", "age":18 }}
{"id": 1232, "source" :{ "id" : 2, "name" : "Emily", "age":20 }}
{"id": 1233, "source" :{ "id" : 3, "name" : "Alexander", "age":22 }}
```

2. Create table:

```
CREATE TABLE demo.routine_test14 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command

```
CREATE ROUTINE LOAD demo.kafka_job14 ON routine_test14
        PROPERTIES
        (
            "format" = "json",
            "json_root" = "$.source"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad14",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result

```
mysql> select * from routine_test14;
+------+----------------+------+
| id   | name           | age  |
+------+----------------+------+
|    1 | Benjamin       |   18 |
|    2 | Emily          |   20 |
|    3 | Alexander      |   22 |
+------+----------------+------+
3 rows in set (0.01 sec)
```

Loading data with column mapping and derived column calculation

1. Load sample data:

```
{ "id" : 1, "name" : "Benjamin", "age":18 }
{ "id" : 2, "name" : "Emily", "age":20 }
{ "id" : 3, "name" : "Alexander", "age":22 }
```

2. Create table:

```
CREATE TABLE demo.routine_test15 (
    id      INT            NOT NULL  COMMENT "id",
    name    VARCHAR(30)    NOT NULL  COMMENT "name",
    age     INT                      COMMENT "age",
    num     INT                      COMMENT "num"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job15 ON routine_test15
        COLUMNS(id, name, age, num=age*10)
        PROPERTIES
        (
            "format" = "json",
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad15",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test15;
+------+----------------+------+------+
| id   | name           | age  | num  |
+------+----------------+------+------+
|    1 | Benjamin       |   18 |  180 |
|    2 | Emily          |   20 |  200 |
|    3 | Alexander      |   22 |  220 |
+------+----------------+------+------+
3 rows in set (0.01 sec)
```

Loading Complex Data Types

Load Array Data Type

1. Load sample data:

```
{ "id" : 1, "name" : "Benjamin", "age":18, "array":[1,2,3,4,5]}
{ "id" : 2, "name" : "Emily", "age":20, "array":[6,7,8,9,10]}
{ "id" : 3, "name" : "Alexander", "age":22, "array":[11,12,13,14,15]}
```

2. Create table:

```
CREATE TABLE demo.routine_test16
(
    id      INT             NOT NULL  COMMENT "id",
    name    VARCHAR(30)     NOT NULL  COMMENT "name",
    age     INT                       COMMENT "age",
    array   ARRAY<int(11)>  NULL      COMMENT "test array column"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job16 ON routine_test16
        PROPERTIES
        (
            "format" = "json"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad16",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test16;
+------+----------------+------+----------------------+
| id   | name           | age  | array                |
+------+----------------+------+----------------------+
|    1 | Benjamin       |   18 | [1, 2, 3, 4, 5]      |
|    2 | Emily          |   20 | [6, 7, 8, 9, 10]     |
|    3 | Alexander      |   22 | [11, 12, 13, 14, 15] |
+------+----------------+------+----------------------+
3 rows in set (0.00 sec)
```

Loading Map Data Type

1. Load sample data:

```
{ "id" : 1, "name" : "Benjamin", "age":18, "map":{"a": 100, "b": 200}}
{ "id" : 2, "name" : "Emily", "age":20, "map":{"c": 300, "d": 400}}
{ "id" : 3, "name" : "Alexander", "age":22, "map":{"e": 500, "f": 600}}
```

2. Create table:

```
CREATE TABLE demo.routine_test17 (
    id      INT                 NOT NULL  COMMENT "id",
    name    VARCHAR(30)         NOT NULL  COMMENT "name",
    age     INT                           COMMENT "age",
    map     Map<STRING, INT>    NULL      COMMENT "test column"
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job17 ON routine_test17
    PROPERTIES
        (
            "format" = "json"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad17",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test17;
+------+----------------+------+--------------------+
| id   | name           | age  | map                |
+------+----------------+------+--------------------+
|    1 | Benjamin       |   18 | {"a":100, "b":200} |
|    2 | Emily          |   20 | {"c":300, "d":400} |
|    3 | Alexander      |   22 | {"e":500, "f":600} |
+------+----------------+------+--------------------+
3 rows in set (0.01 sec)
```

Loading Bitmap Data Type

1. Load sample data

```
{ "id" : 1, "name" : "Benjamin", "age":18, "bitmap_id":243}
{ "id" : 2, "name" : "Emily", "age":20, "bitmap_id":28574}
{ "id" : 3, "name" : "Alexander", "age":22, "bitmap_id":8573}
```

2. Create table:

```
CREATE TABLE demo.routine_test18 (
    id        INT            NOT NULL      COMMENT "id",
    name      VARCHAR(30)    NOT NULL      COMMENT "name",
    age       INT                          COMMENT "age",
    bitmap_id INT                          COMMENT "test",
    device_id BITMAP         BITMAP_UNION  COMMENT "test column"
)
AGGREGATE KEY (`id`,`name`,`age`,`bitmap_id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job18 ON routine_test18
        COLUMNS(id, name, age, bitmap_id, device_id=to_bitmap(bitmap_id))
        PROPERTIES
        (
            "format" = "json"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad18",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select id, BITMAP_UNION_COUNT(pv) over(order by id) uv from(
    ->    select id, BITMAP_UNION(device_id) as pv
    ->    from routine_test18
    -> group by id
    -> ) final;
+------+------+
| id   | uv   |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    3 |    3 |
+------+------+
3 rows in set (0.00 sec)
```

Loading HLL Data Type

1. Loading sample data:

```
2022-05-05,10001,Test01,Beijing,windows
2022-05-05,10002,Test01,Beijing,linux
2022-05-05,10003,Test01,Beijing,macos
2022-05-05,10004,Test01,Hebei,windows
2022-05-06,10001,Test01,Shanghai,windows
2022-05-06,10002,Test01,Shanghai,linux
2022-05-06,10003,Test01,Jiangsu,macos
2022-05-06,10004,Test01,Shaanxi,windows
```

2. Create table:

```
create table demo.routine_test19 (
    dt        DATE,
    id        INT,
    name      VARCHAR(10),
    province  VARCHAR(10),
    os        VARCHAR(10),
    pv        hll hll_union
)
Aggregate KEY (dt,id,name,province,os)
distributed by hash(id) buckets 10;
```

3. Load command:

```
CREATE ROUTINE LOAD demo.kafka_job19 ON routine_test19
        COLUMNS TERMINATED BY ",",
        COLUMNS(dt, id, name, province, os, pv=hll_hash(id))
        FROM KAFKA
        (
            "kafka_broker_list" = "10.16.10.6:9092",
            "kafka_topic" = "routineLoad19",
            "property.kafka_default_offsets" = "OFFSET_BEGINNING"
        );
```

4. Load result:

```
mysql> select * from routine_test19;
+------------+-------+----------+----------+---------+------+
| dt         | id    | name     | province | os      | pv   |
+------------+-------+----------+----------+---------+------+
| 2022-05-05 | 10001 | Test01   | Beijing     | windows | NULL |
| 2022-05-06 | 10001 | Test01   | Shanghai    | windows | NULL |
| 2022-05-05 | 10002 | Test01   | Beijing     | linux   | NULL |
| 2022-05-06 | 10002 | Test01   | Shanghai    | linux   | NULL |
| 2022-05-05 | 10004 | Test01   | Hebei       | windows | NULL |
| 2022-05-06 | 10004 | Test01   | Shaanxi      | windows | NULL |
| 2022-05-05 | 10003 | Test01   | Beijing     | macos   | NULL |
| 2022-05-06 | 10003 | Test01   | Jiangsu     | macos   | NULL |
+------------+-------+----------+----------+---------+------+
8 rows in set (0.01 sec)

mysql> SELECT HLL_UNION_AGG(pv) FROM routine_test19;
+------------------+
| hll_union_agg(pv) |
+------------------+
|                4 |
+------------------+
1 row in set (0.01 sec)
```

Kafka Security Authentication

Loading Kafka Data with SSL Authentication

Example load command:

```
CREATE ROUTINE LOAD demo.kafka_job20 ON routine_test20
        PROPERTIES
        (
            "format" = "json"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "192.168.100.129:9092",
            "kafka_topic" = "routineLoad21",
            "property.security.protocol" = "ssl",
            "property.ssl.ca.location" = "FILE:ca.pem",
            "property.ssl.certificate.location" = "FILE:client.pem",
            "property.ssl.key.location" = "FILE:client.key",
            "property.ssl.key.password" = "ssl_passwd"
        );
```

Parameter descriptions:

Parameter	Description
property.security.protocol	The security protocol used, in this example it is SSL
property.ssl.ca.location	The location of the CA (Certificate Authority) certificate
property.ssl.certificate.location	The location of the Client's public key (required if client authentication is enabled on the Kafka server)
property.ssl.key.location	The location of the Client's private key (required if client authentication is enabled on the Kafka server)
property.ssl.key.password	The password for the Client's private key (required if client authentication is enabled on the Kafka server)

Loading Kafka Data with Kerberos Authentication

Example load command:

```
CREATE ROUTINE LOAD demo.kafka_job21 ON routine_test21
        PROPERTIES
        (
            "format" = "json"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "192.168.100.129:9092",
            "kafka_topic" = "routineLoad21",
            "property.security.protocol" = "SASL_PLAINTEXT",
            "property.sasl.kerberos.service.name" = "kafka",
            "property.sasl.kerberos.keytab"="/opt/third/kafka/kerberos/kafka_client.keytab",
            "property.sasl.kerberos.principal" = "clients/stream.dt.local@EXAMPLE.COM"
        );
```

Parameter descriptions:

Parameter	Description
property.security.protocol	The security protocol used, in this example it is SASL_PLAINTEXT
property.sasl.kerberos.service.name	Specifies the broker service name, default is Kafka
property.sasl.kerberos.keytab	The location of the keytab file
property.sasl.kerberos.principal	Specifies the Kerberos principal

Loading Kafka Cluster with PLAIN Authentication

1. Example load command:

```
CREATE ROUTINE LOAD demo.kafka_job22 ON routine_test22
        PROPERTIES
        (
            "format" = "json"
        )
        FROM KAFKA
        (
            "kafka_broker_list" = "192.168.100.129:9092",
```

```
            "kafka_topic" = "routineLoad22",
            "property.security.protocol"="SASL_PLAINTEXT",
            "property.sasl.mechanism"="PLAIN",
            "property.sasl.username"="admin",
            "property.sasl.password"="admin"
        );
```

Parameter descriptions:

Parameter	Description
property.security.protocol	The security protocol used, in this example it is SASL_PLAINTEXT
property.sasl.mechanism	Specifies the SASL authentication mechanism as PLAIN
property.sasl.username	The username for SASL
property.sasl.password	The password for SASL

Single-task Loading to Multiple Tables

Create a Kafka routine dynamic table load task named “test1” for the “example_db”. Specify the column delimiter, group.id, and client.id. Automatically consume all partitions and start subscribing from the available data position (OFFSET_BEGINNING).

Assuming we need to load data from Kafka into tables “tbl1” and “tbl2” in the “example_db”, we create a Routine Load task named “test1”. This task will load data from Kafka’s topic `my_topic` into both “tbl1” and “tbl2” simultaneously. This way, we can load data from Kafka into two tables using a single routine load task.

```
CREATE ROUTINE LOAD example_db.test1
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

Currently, only extracting the table name from the Value field of Kafka is supported. The format should be as follows, using JSON as an example: `table_name|{"col1": "val1", "col2": "val2"}`, where `tbl_name` is the table name and `|` is used as the separator between the table name and the table data. The same format applies to CSV data, such as `table_name|val1,val2,val3`. Note that the `table_name` here must be consistent with the table name in Doris, otherwise the load will fail. Note that dynamic tables do not support the column_mapping configuration described later.

Strict Mode Load

Create a Kafka routine load task named “test1” for the “example_db” and “example_tbl”. The load task is set to strict mode.

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100),
PRECEDING FILTER k1 = 1,
WHERE k1 < 100 and k2 like "%doris%"
PROPERTIES
```

```
(
    "strict_mode" = "true"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic"
);
```

Connect to the SASL Kafka service

Here we take accessing the StreamNative message service as an example:

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(user_id, name, age)
FROM KAFKA (
"kafka_broker_list" = "pc-xxxx.aws-mec1-test-xwiqv.aws.snio.cloud:9093",
"kafka_topic" = "my_topic",
"property.security.protocol" = "SASL_SSL",
"property.sasl.mechanism" = "PLAIN",
"property.sasl.username" = "user",
"property.sasl.password" = "token:eyJhbxxx",
"property.group.id" = "my_group_id_1",
"property.client.id" = "my_client_id_1",
"property.enable.ssl.certificate.verification" = "false"
);
```

Note that if the trusted CA certificate path is not configured on the BE side, you need to set `"property.enable.ssl.certificate.verification" = "false"` to not verify whether the server certificate is credible.

Otherwise, you need to configure the trusted CA certificate path: `"property.ssl.ca.location" = "/path/to/ca-cert.pem"`.

More Details

Refer to the SQL manual on Routine Load. You can also enter `HELP ROUTINE LOAD` in the client command line for more help.

Insert Into Select

The INSERT INTO statement supports importing the results of a Doris query into another table. INSERT INTO is a synchronous import method, where the import result is returned after the import is executed. Whether the import is successful can be determined based on the returned result. INSERT INTO ensures the atomicity of the import task, meaning that either all the data is imported successfully or none of it is imported.

Applicable scenarios

2. If a user wants to perform ETL on existing data in a Doris table and then import it into a new Doris table, the INSERT INTO SELECT syntax is applicable.
3. In conjunction with the Multi-Catalog external table mechanism, tables from MySQL or Hive systems can be mapped via Multi-Catalog. Then, data from external tables can be imported into Doris tables using the INSERT INTO SELECT syntax.
4. Utilizing the Table Value Functions (TVFs), users can directly query data stored in object storage or files on HDFS as tables, with automatic column type inference. Then, data from external tables can be imported into Doris tables using the INSERT INTO SELECT syntax.

Implementation

When using INSERT INTO, the import job needs to be initiated and submitted to the FE node using the MySQL protocol. The FE generates an execution plan, which includes query-related operators, with the last operator being the OlapTableSink. The OlapTableSink operator is responsible for writing the query result to the target table. The execution plan is then sent to the BE nodes for execution. Doris designates one BE node as the Coordinator, which receives and distributes the data to other BE nodes.

Get started

An INSERT INTO job is submitted and transmitted using the MySQL protocol. The following example demonstrates submitting an import job using INSERT INTO through the MySQL command-line interface.

Detailed syntax can be found in the INSERT INTO documentation.

Preparation

INSERT INTO requires INSERT permissions on the target table. You can grant permissions to user accounts using the GRANT command.

Create an INSERT INTO job

1. Create a source table

```
CREATE TABLE testdb.test_table(
    user_id            BIGINT       NOT NULL COMMENT "User ID",
    name               VARCHAR(20)           COMMENT "User name",
    age                INT                   COMMENT "User age"
```

```
)
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

2. Import data into the source table using any load method. (Here we use `INSERT INTO VALUES` for example).

```
INSERT INTO testdb.test_table (user_id, name, age)
VALUES (1, "Emily", 25),
       (2, "Benjamin", 35),
       (3, "Olivia", 28),
       (4, "Alexander", 60),
       (5, "Ava", 17);
```

3. Building upon the above operations, create a new table as the target table (with the same schema as the source table).

```
CREATE TABLE testdb.test_table2 LIKE testdb.test_table;
```

4. Ingest data into the new table using `INSERT INTO SELECT`.

```
INSERT INTO testdb.test_table2
SELECT * FROM testdb.test_table WHERE age < 30;
Query OK, 3 rows affected (0.544 sec)
{'label':'label_9c2bae970023407d_b2c5b78b368e78a7', 'status':'VISIBLE', 'txnId':'9084'}
```

5. View imported data.

```
MySQL> SELECT * FROM testdb.test_table2 ORDER BY age;
+---------+--------+------+
| user_id | name   | age  |
+---------+--------+------+
|       5 | Ava    |   17 |
|       1 | Emily  |   25 |
|       3 | Olivia |   28 |
+---------+--------+------+
3 rows in set (0.02 sec)
```

6. You can use JOB make the INSERT operation execute asynchronously.

7. Sources can be tvf or tables in a catalog.

View INSERT INTO jobs

You can use the `SHOW LOAD` command to view the completed INSERT INTO tasks.

```
MySQL> SHOW LOAD FROM testdb;
+--------+---------------------------------------+----------+--------------------+--------+---------+-----------------------------
-----------------------------------------+----------+--------------------+--------------------+--------------------+-------------
--------+--------------------+------+-----------------------------------------------------------------------------------------------------------
------------------------+--------------+-------------+------+---------+
| JobId  | Label                                 | State    | Progress           | Type   | EtlInfo | TaskInfo
| ErrorMsg | CreateTime          | EtlStartTime        | EtlFinishTime       | LoadStartTime       | LoadFinishTime      | URL  |
JobDetails                                                                                                                  |
TransactionId | ErrorTablets | User | Comment |
```

```
+--------+----------------------------------------+----------+--------------------+--------+---------+-----------------------------
-----------------------------------------+----------+--------------------+--------------------+--------------------+-------------
--------+---------------------+------+-----------------------------------------------------------------------------------------------------------------
------------------------+--------------+-------------+------+---------+
| 376416 | label_3e52da787aab4222_9126d2fce8f6d1e5 | FINISHED | Unknown id: 376416 | INSERT | NULL    | cluster:N/A;
timeout(s):26200; max_filter_ratio:0.0; priority:NORMAL | NULL     | 2024-02-27 01:22:17 | 2024-02-27 01:22:17 | 2024-02-27 01:22:17
| 2024-02-27 01:22:17 | 2024-02-27 01:22:18 |      | {"Unfinished backends":{},"ScannedRows":0,"TaskNumber":0,"LoadBytes":0,"All
backends":{},"FileNumber":0,"FileSize":0} | 9081          | {}           | root |         |
| 376664 | label_9c2bae970023407d_b2c5b78b368e78a7 | FINISHED | Unknown id: 376664 | INSERT | NULL    | cluster:N/A;
timeout(s):26200; max_filter_ratio:0.0; priority:NORMAL | NULL     | 2024-02-27 01:39:37 | 2024-02-27 01:39:37 | 2024-02-27 01:39:37
| 2024-02-27 01:39:37 | 2024-02-27 01:39:38 |      | {"Unfinished backends":{},"ScannedRows":0,"TaskNumber":0,"LoadBytes":0,"All
backends":{},"FileNumber":0,"FileSize":0} | 9084          | {}           | root |         |
+--------+----------------------------------------+----------+--------------------+--------+---------+-----------------------------
-----------------------------------------+----------+--------------------+--------------------+--------------------+-------------
--------+---------------------+------+-----------------------------------------------------------------------------------------------------------------
------------------------+--------------+-------------+------+---------+
```

Cancel INSERT INTO jobs

You can cancel the currently executing INSERT INTO job via Ctrl-C.

Manual

Syntax

The syntax of INSERT INTO is as follows:

1. INSERT INTO SELECT

INSERT INTO SELECT is used to write the query results to the target table.

```
INSERT INTO target_table SELECT ... FROM source_table;
```

The SELECT statement above is similar to a regular SELECT query, allowing operations such as WHERE and JOIN.

Parameter configuration

FE Config

Name	Default Value	Description
insert_load_default_timeout_second	14400s (4 hours)	Timeout for import tasks, in seconds. If the import task does not complete within this timeout period, it will be canceled by the system and marked as `CANCELLED`.

Session Variable

Name	Default Value	Description
insert_timeout	14400s (4 hours)	Timeout for INSERT INTO as an SQL statement, in seconds.

Name	Default Value	Description
enable_insert_strict	true	If this is set to true, INSERT INTO will fail when the task involves invalid data. If set to false, INSERT INTO will ignore invalid rows, and the import will be considered successful as long as at least one row is imported successfully. Until version 2.1.4. INSERT INTO cannot control the error rate, so this parameter is used to either strictly check data quality or completely ignore invalid data. Common reasons for data invalidity include: source data column length exceeding destination column length, column type mismatch, partition mismatch, and column order mismatch.
insert_max_filter_ratio	1.0	Since version 2.1.5. Only effective when `enable_insert_strict` is false. Used to control the error tolerance when using `INSERT INTO FROM S3/HDFS/LOCAL()`. The default value is 1.0, which means all errors are tolerated. It can be a decimal between 0 and 1. It means that when the number of error rows exceeds this ratio, the INSERT task will fail.

Return values

INSERT INTO an SQL statement, and it returns different results based on different query outcomes:

Empty result set

If the query result set of the SELECT statement in INSERT INTO is empty, the returned value will be as follows:

```
mysql> INSERT INTO tbl1 SELECT * FROM empty_tbl;
Query OK, 0 rows affected (0.02 sec)
```

`Query OK` indicates successful execution. `0 rows affected` means no data was imported.

Non-empty result set and successful INSERT

```
mysql> INSERT INTO tbl1 SELECT * FROM tbl2;
Query OK, 4 rows affected (0.38 sec)
{'label':'INSERT_8510c568-9eda-4173-9e36-6adc7d35291c', 'status':'visible', 'txnId':'4005'}

mysql> INSERT INTO tbl1 WITH LABEL my_label1 SELECT * FROM tbl2;
Query OK, 4 rows affected (0.38 sec)
{'label':'my_label1', 'status':'visible', 'txnId':'4005'}

mysql> INSERT INTO tbl1 SELECT * FROM tbl2;
Query OK, 2 rows affected, 2 warnings (0.31 sec)
{'label':'INSERT_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'visible', 'txnId':'4005'}

mysql> INSERT INTO tbl1 SELECT * FROM tbl2;
Query OK, 2 rows affected, 2 warnings (0.31 sec)
{'label':'INSERT_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
```

`Query OK` indicates successful execution. `4 rows affected` indicates that a total of 4 rows of data have been imported. `2 warnings` indicates the number of rows that were filtered out.

Additionally, a JSON string is returned:

```
{'label':'my_label1', 'status':'visible', 'txnId':'4005'}
{'label':'INSERT_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
{'label':'my_label1', 'status':'visible', 'txnId':'4005', 'err':'some other error'}
```

Parameter description:

Parameter	Description
TxnId	ID of the import transaction
Label	Label of the import job: can be specified using “INSERT INTO tbl WITH LABEL label…”
Status	Visibility of the imported data: If it is visible, it will be displayed as “visible.” If not, it will be displayed as “committed.” In the “committed” state, the import is completed, but the data may be delayed in becoming visible. There is no need to retry in this case.`visible`: The import is successful and the data is visible.`committed`: The import is completed, but the data may be delayed in becoming visible. There is no need to retry in this case.Label Already Exists: The specified label already exists and needs to be changed to a different one.Fail: The import fails.
Err	Error message

You can use the SHOW LOAD statement to view the filtered rows.

```
SHOW LOAD WHERE label="xxx";
```

The result of this statement will include a URL that can be used to query the error data. For more details, refer to the “View error rows” section below.

The invisible state of data is temporary, and the data will eventually become visible.

You can check the visibility status of a batch of data using the SHOW TRANSACTION statement.

```
SHOW TRANSACTION WHERE id=4005;
```

If the `TransactionStatus` column in the result is `visible`, it indicates that the data is visible.

```
{'label':'my_label1', 'status':'visible', 'txnId':'4005'}
{'label':'INSERT_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
{'label':'my_label1', 'status':'visible', 'txnId':'4005', 'err':'some other error'}
```

Non-empty result set but failed INSERT

Failed execution means that no data was successfully imported. An error message will be returned:

```
mysql> INSERT INTO tbl1 SELECT * FROM tbl2 WHERE k1 = "a";
ERROR 1064 (HY000): all partitions have no load data. url: http://10.74.167.16:8042/api/_load_error_log?
file=_shard_2/error_loginsert_stmt_ba8bb9e158e4879-ae8de8507c0bf8a2_ba8bb9e158e4879_ae8de8507c0bf8a2
```

`ERROR 1064 (HY000): all partitions have no load data` indicates the root cause for the failure. The URL provided in the error message can be used to locate the error data. For more details, refer to the “View error rows” section below.

Best practice

Data size

INSERT INTO imposes no limitations on data volume and can support large-scale data imports. However, if you are importing a large amount of data, it is recommended to adjust the system’s INSERT INTO timeout settings to ensure that `import timeout >= data volume ``/`` estimated import speed`.

1. FE configuration parameter `insert_load_default_timeout_second`
2. Environment parameter `insert_timeout`

View error rows

When the INSERT INTO result includes a URL field, you can use the following command to view the error rows:

```
SHOW LOAD WARNINGS ON "url";
```

Example:

```
SHOW LOAD WARNINGS ON "http://ip:port/api/_load_error_log?file=_shard_13/error_loginsert_stmt_d2cac0a0a16d482d-9041c949a4b71605_d2cac0a0a16d482d_9041c949a4b71605";
```

Common reasons for errors include: source data column length exceeding destination column length, column type mismatch, partition mismatch, and column order mismatch.

You can control whether INSERT INTO ignores error rows by configuring the environment variable `enable_insert_strict`.

Ingest external data via Multi-Catalog

Doris supports the creation of external tables. Once created, data from external tables can be imported into Doris internal tables using `INSERT INTO SELECT`, or queried directly using SELECT statements.

With its Multi-Catalog feature, Doris supports connections to various mainstream data lakes and databases including Apache Hive, Apache Iceberg, Apache Hudi, Apache Paimon (Incubating), Elasticsearch, MySQL, Oracle, and SQL Server.

For more information on Multi-Catalog, please refer to Lakehouse overview.

The followings illustrate importing data from a Hive external table into a Doris internal table.

Create Hive Catalog

```
CREATE CATALOG hive PROPERTIES (
    'type'='hms',
```

```
    'hive.metastore.uris' = 'thrift://172.0.0.1:9083',
    'hadoop.username' = 'hive',
    'dfs.nameservices'='your-nameservice',
    'dfs.ha.namenodes.your-nameservice'='nn1,nn2',
    'dfs.namenode.rpc-address.your-nameservice.nn1'='172.21.0.2:8088',
    'dfs.namenode.rpc-address.your-nameservice.nn2'='172.21.0.3:8088',
    'dfs.client.failover.proxy.provider.your-
nameservice'='org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider'
);
```

Ingest data

1. Create a target table for the data import in Doris.

```
CREATE TABLE `target_tbl` (
  `k1` decimal(9, 3) NOT NULL COMMENT "",
  `k2` char(10) NOT NULL COMMENT "",
  `k3` datetime NOT NULL COMMENT "",
  `k5` varchar(20) NOT NULL COMMENT "",
  `k6` double NOT NULL COMMENT ""
)
COMMENT "Doris Table"
DISTRIBUTED BY HASH(k1) BUCKETS 2
PROPERTIES (
    "replication_num" = "1"
);
```

2. For detailed instructions on creating Doris tables, please refer to CREATE TABLE.

3. Importing data (from the `hive.db1.source_tbl` table into the `target_tbl` table).

```
INSERT INTO target_tbl SELECT k1,k2,k3 FROM  hive.db1.source_tbl limit 100;
```

The INSERT command is a synchronous command. If it returns a result, that indicates successful import.

Notes

- Ensure that the external data source and the Doris cluster can communicate, including mutual network accessibility between BE nodes and external data sources.

Ingest data by TVF

Doris can directly query and analyze files stored in object storage or HDFS as tables through the Table Value Functions (TVFs), which supports automatic column type inference. For detailed information, please refer to the Lakehouse/TVF documentation.

Automatic column type inference

```
DESC FUNCTION s3 (
    "URI" = "http://127.0.0.1:9312/test2/test.snappy.parquet",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "format" = "parquet",
```

```
    "use_path_style"="true"
);
+---------------+--------------+------+-------+---------+-------+
| Field         | Type         | Null | Key   | Default | Extra |
+---------------+--------------+------+-------+---------+-------+
| p_partkey     | INT          | Yes  | false | NULL    | NONE  |
| p_name        | TEXT         | Yes  | false | NULL    | NONE  |
| p_mfgr        | TEXT         | Yes  | false | NULL    | NONE  |
| p_brand       | TEXT         | Yes  | false | NULL    | NONE  |
| p_type        | TEXT         | Yes  | false | NULL    | NONE  |
| p_size        | INT          | Yes  | false | NULL    | NONE  |
| p_container   | TEXT         | Yes  | false | NULL    | NONE  |
| p_retailprice | DECIMAL(9,0) | Yes  | false | NULL    | NONE  |
| p_comment     | TEXT         | Yes  | false | NULL    | NONE  |
+---------------+--------------+------+-------+---------+-------+
```

In this example of an S3 TVF, the file path, connection information, and authentication information are specified.

You can use the `DESC FUNCTION` syntax to view the schema of this file.

It can be seen that for Parquet files, Doris automatically infers column types based on the metadata within the file.

Currently, Doris supports analysis and column type inference for Parquet, ORC, CSV, and JSON formats.

It can be used in combination with the `INSERT INTO SELECT` syntax to quickly import files into Doris tables for faster analysis.

```
// 1. Create Doris internal table
CREATE TABLE IF NOT EXISTS test_table
(
    id int,
    name varchar(50),
    age int
)
DISTRIBUTED BY HASH(id) BUCKETS 4
PROPERTIES("replication_num" = "1");

// 2. Insert data by S3 Table Value Function
INSERT INTO test_table (id,name,age)
SELECT cast(id as INT) as id, name, cast (age as INT) as age
FROM s3(
    "uri" = "http://127.0.0.1:9312/test2/test.snappy.parquet",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "format" = "parquet",
    "use_path_style" = "true");
```

Notes

- If the URI specified in the `S3 / hdfs` TVF does not match any files, or if all matched files are empty, the `S3 / hdfs` TVF will return an empty result set. In such cases, if you use `DESC FUNCTION` to view the schema of the file, you will see a dummy column `__dummy_col`, which can be ignored.
- If the format specified for the TVF is CSV and the file being read is not empty but the first line of the file is empty, an error will be prompted: `The first line is empty, can not parse column numbers`. This is because the schema cannot be parsed from the first line of the file.

More help

For more detailed syntax on INSERT INTO, refer to the INSERT INTO command manual. You can also type `HELP INSERT` at the MySQL client command line for further information.

Insert Into Values

The INSERT INTO VALUES statement supports importing values from SQL into a Doris table. INSERT INTO VALUES is a synchronous import method, where the import result is returned after the import is executed. Whether the import is successful can be determined based on the returned result. INSERT INTO VALUES ensures the atomicity of the import task, meaning that either all the data is imported successfully or none of it is imported.

Applicable scenarios

1. If a user wants to import only a few test data records to verify the functionality of the Doris system, the INSERT INTO VALUES syntax is applicable. It is similar to the MySQL syntax. However, it is not recommended to use INSERT INTO VALUES in a production environment.
2. The performance of concurrent INSERT INTO VALUES jobs will be bottlenecked by commit stage. When loading large quantity of data, you can enable group commit to achieve high performance.

Implementation

When using INSERT INTO VALUES, the import job needs to be initiated and submitted to the FE node using the MySQL protocol. The FE generates an execution plan, which includes query-related operators, with the last operator being the OlapTableSink. The OlapTableSink operator is responsible for writing the query result to the target table. The execution plan is then sent to the BE nodes for execution. Doris designates one BE node as the Coordinator, which receives and distributes the data to other BE nodes.

Get started

An INSERT INTO VALUES job is submitted and transmitted using the MySQL protocol. The following example demonstrates submitting an import job using INSERT INTO VALUES through the MySQL command-line interface.

Preparation

INSERT INTO VALUES requires INSERT permissions on the target table. You can grant permissions to user accounts using the GRANT command.

Create an INSERT INTO VALUES job

INSERT INTO VALUES

1. Create a source table

```
CREATE TABLE testdb.test_table(
    user_id            BIGINT       NOT NULL COMMENT "User ID",
    name               VARCHAR(20)           COMMENT "User name",
    age                INT                   COMMENT "User age"
)
```

```
DUPLICATE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 10;
```

2. Import data into the source table using `INSERT INTO VALUES` (not recommended for production environments).

```
INSERT INTO testdb.test_table (user_id, name, age)
VALUES (1, "Emily", 25),
       (2, "Benjamin", 35),
       (3, "Olivia", 28),
       (4, "Alexander", 60),
       (5, "Ava", 17);
```

INSERT INTO VALUES is a synchronous import method, where the import result is directly returned to the user.

```
Query OK, 5 rows affected (0.308 sec)
{'label':'label_26eebc33411f441c_b2b286730d495e2c', 'status':'VISIBLE', 'txnId':'61071'}
```

3. View imported data.

```
MySQL> SELECT COUNT(*) FROM testdb.test_table;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.179 sec)
```

View INSERT INTO VALUES jobs

You can use the `SHOW LOAD` command to view the completed INSERT INTO VALUES tasks.

```
mysql> SHOW LOAD FROM testdb\G
*************************** 1. row ***************************
         JobId: 77172
         Label: label_26eebc33411f441c_b2b286730d495e2c
         State: FINISHED
      Progress: Unknown id: 77172
          Type: INSERT
       EtlInfo: NULL
      TaskInfo: cluster:N/A; timeout(s):14400; max_filter_ratio:0.0
      ErrorMsg: NULL
    CreateTime: 2024-11-20 16:44:08
  EtlStartTime: 2024-11-20 16:44:08
 EtlFinishTime: 2024-11-20 16:44:08
 LoadStartTime: 2024-11-20 16:44:08
LoadFinishTime: 2024-11-20 16:44:08
           URL:
    JobDetails: {"Unfinished backends":{},"ScannedRows":0,"TaskNumber":0,"LoadBytes":0,"All backends":
{},"FileNumber":0,"FileSize":0}
 TransactionId: 61071
  ErrorTablets: {}
          User: root
       Comment:
1 row in set (0.00 sec)
```

Cancel INSERT INTO VALUES jobs

You can cancel the currently executing INSERT INTO VALUES job via Ctrl-C.

Manual

Syntax

INSERT INTO VALUES is typically used for testing purposes. It is not recommended for production environments.

```
INSERT INTO target_table (col1, col2, ...)
VALUES (val1, val2, ...), (val3, val4, ...), ...;
```

Parameter configuration

FE Config

Name	Default Value	Description
insert_load_default_timeout_second	14400s (4 hours)	Timeout for import tasks, in seconds. If the import task does not complete within this timeout period, it will be canceled by the system and marked as `CANCELLED`.

Session Variable

Name	Default Value	Description
insert_timeout	14400s (4 hours)	Timeout for INSERT INTO as an SQL statement, in seconds.
enable_insert_strict	true	If this is set to true, INSERT INTO will fail when the task involves invalid data. If set to false, INSERT INTO will ignore invalid rows, and the import will be considered successful as long as at least one row is imported successfully. Until version 2.1.4. INSERT INTO cannot control the error rate, so this parameter is used to either strictly check data quality or completely ignore invalid data. Common reasons for data invalidity include: source data column length exceeding destination column length, column type mismatch, partition mismatch, and column order mismatch.

Name	Default Value	Description
insert_max_filter_ratio	1.0	Since version 2.1.5. Only effective when `enable_insert_strict` is false. Used to control the error tolerance when using `INSERT INTO FROM S3/HDFS/LOCAL()`. The default value is 1.0, which means all errors are tolerated. It can be a decimal between 0 and 1. It means that when the number of error rows exceeds this ratio, the INSERT task will fail.

Return values

INSERT INTO VALUES is a SQL statement, and it returns a JSON string in its results.

Parameters in the JSON string:

Parameter	Description
Label	Label of the import job: can be specified using “INSERT INTO tbl WITH LABEL label…”
Status	Visibility of the imported data: If it is visible, it will be displayed as “visible.” If not, it will be displayed as “committed.” In the “committed” state, the import is completed, but the data may be delayed in becoming visible. There is no need to retry in this case.`visible`: The import is successful and the data is visible.`committed`: The import is completed, but the data may be delayed in becoming visible. There is no need to retry in this case.Label Already Exists: The specified label already exists and needs to be changed to a different one.Fail: The import fails.
Err	Error message
TxnId	ID of the import transaction

Successful INSERT

```
mysql> INSERT INTO test_table (user_id, name, age) VALUES (1, "Emily", 25), (2, "Benjamin", 35), (3, "Olivia", 28), (NULL,
"Alexander", 60), (5, "Ava", 17);
Query OK, 5 rows affected (0.05 sec)
{'label':'label_26eebc33411f441c_b2b286730d495e2c', 'status':'VISIBLE', 'txnId':'61071'}
```

`Query OK` indicates successful execution. `5 rows affected` indicates that a total of 5 rows of data have been imported.

Successful INSERT with warnings

```
mysql> INSERT INTO test_table (user_id, name, age) VALUES (1, "Emily", 25), (2, "Benjamin", 35), (3, "Olivia", 28), (NULL,
"Alexander", 60), (5, "Ava", 17);
Query OK, 4 rows affected, 1 warning (0.04 sec)
{'label':'label_a8d99ae931194d2b_93357aac59981a18', 'status':'VISIBLE', 'txnId':'61068'}
```

`Query OK` indicates successful execution. `4 rows affected` indicates that a total of 4 rows of data have been imported. `1 warnings` indicates the number of rows that were filtered out.

You can use the SHOW LOAD statement to view the filtered rows.

The result of this statement will include a URL that can be used to query the error data. For more details, refer to the “View error rows” section below.

```
mysql> SHOW LOAD WHERE label="label_a8d99ae931194d2b_93357aac59981a18"\G
*************************** 1. row ***************************
         JobId: 77158
         Label: label_a8d99ae931194d2b_93357aac59981a18
         State: FINISHED
      Progress: Unknown id: 77158
          Type: INSERT
       EtlInfo: NULL
      TaskInfo: cluster:N/A; timeout(s):14400; max_filter_ratio:0.0
      ErrorMsg: NULL
    CreateTime: 2024-11-20 16:35:40
  EtlStartTime: 2024-11-20 16:35:40
 EtlFinishTime: 2024-11-20 16:35:40
 LoadStartTime: 2024-11-20 16:35:40
LoadFinishTime: 2024-11-20 16:35:40
           URL: http://10.16.10.7:8743/api/_load_error_log?file=__shard_18/error_log_insert_stmt_a8d99ae931194d2b-
93357aac59981a19_a8d99ae931194d2b_93357aac59981a19
    JobDetails: {"Unfinished backends":{},"ScannedRows":0,"TaskNumber":0,"LoadBytes":0,"All backends":
{},"FileNumber":0,"FileSize":0}
 TransactionId: 61068
  ErrorTablets: {}
          User: root
       Comment:
1 row in set (0.00 sec)
```

Successful INSERT with committed status

```
mysql> INSERT INTO test_table (user_id, name, age) VALUES (1, "Emily", 25), (2, "Benjamin", 35), (3, "Olivia", 28), (4, "Alexander",
60), (5, "Ava", 17);
Query OK, 5 rows affected (0.04 sec)
{'label':'label_78bf5396d9594d4d_a8d9a914af40f73d', 'status':'COMMITTED', 'txnId':'61074'}
```

The invisible state of data is temporary, and the data will eventually become visible.

You can check the visibility status of a batch of data using the SHOW TRANSACTION statement.

If the `TransactionStatus` column in the result is `visible`, it indicates that the data is visible.

```
mysql> SHOW TRANSACTION WHERE id=61074\G
*************************** 1. row ***************************
     TransactionId: 61074
             Label: label_78bf5396d9594d4d_a8d9a914af40f73d
       Coordinator: FE: 10.16.10.7
 TransactionStatus: VISIBLE
 LoadJobSourceType: INSERT_STREAMING
       PrepareTime: 2024-11-20 16:51:54
     PreCommitTime: NULL
        CommitTime: 2024-11-20 16:51:54
       PublishTime: 2024-11-20 16:51:54
        FinishTime: 2024-11-20 16:51:54
            Reason:
ErrorReplicasCount: 0
        ListenerId: -1
         TimeoutMs: 14400000
            ErrMsg:
1 row in set (0.00 sec)
```

Non-empty result set but failed INSERT

Failed execution means that no data was successfully imported. An error message will be returned:

```
mysql> INSERT INTO test_table (user_id, name, age) VALUES (1, "Emily", 25), (2, "Benjamin", 35), (3, "Olivia", 28), (NULL,
"Alexander", 60), (5, "Ava", 17);
ERROR 1105 (HY000): errCode = 2, detailMessage = Insert has too many filtered data 1/5 insert_max_filter_ratio is 0.100000. url:
http://10.16.10.7:8747/api/_load_error_log?file=__shard_22/error_log_insert_stmt_5fafe6663e1a45e0-
a666c1722ffc8c55_5fafe6663e1a45e0_a666c1722ffc8c55
```

`ERROR 1105 (HY000): errCode = 2, detailMessage = Insert has too many filtered data 1/5 insert_max_filter_ratio is 0.100000.` indicates the root cause for the failure. The URL provided in the error message can be used to locate the error data. For more details, refer to the “View error rows” section below.

Best practice

Data size

INSERT INTO VALUES is usually used for test and demo, it is not recommended to load large quantity data with INSERT INTO VALUES.

View error rows

When the INSERT INTO result includes a URL field, you can use the following command to view the error rows:

```
SHOW LOAD WARNINGS ON "url";
```

Example:

```
mysql> SHOW LOAD WARNINGS ON "http://10.16.10.7:8743/api/_load_error_log?file=__shard_18/error_log_insert_stmt_a8d99ae931194d2b-
93357aac59981a19_a8d99ae931194d2b_93357aac59981a19"\G
*************************** 1. row ***************************
         JobId: -1
         Label: NULL
ErrorMsgDetail: Reason: column_name[user_id], null value for not null column, type=BIGINT. src line [];
1 row in set (0.00 sec)
```

Common reasons for errors include: source data column length exceeding destination column length, column type mismatch, partition mismatch, and column order mismatch.

You can control whether INSERT INTO ignores error rows by configuring the environment variable `enable_insert_strict`.

More help

For more detailed syntax on INSERT INTO, refer to the INSERT INTO command manual. You can also type `HELP INSERT` at the MySQL client command line for further information.

MySQL Load

Apache Doris is compatible with the MySQL protocol and supports the use of the standard MySQL LOAD DATA syntax to import local files. MySQL Load is a synchronous import method where the import result is returned upon completion. That means users can tell whether the import suceeds from the returned result. Generally, the MySQL Load method can be used to ingest files smaller than 10GB in size. For files larger than 10GB, it is recommended to split them into smaller ones. MySQL Load ensures the atomicity of a batch of import tasks, meaning that either all imports succeed or all imports fail.

Applicable scenarios

Supported format

MySQL Load is primarily designed for importing CSV files from the client's local machine or importing data from data streams through programs.

Restrictions

When importing CSV files, it is important to differentiate between null values and empty strings (''):

- Null values are represented by the escape sequence . For example, in `a,\N,b`, the middle column represents a null value.
- Empty strings are represented directly as empty, such as `a, ,b`, where the middle column represents an empty string.

Implementation

MySQL Load is similar to Stream Load in terms of functionality. They both involve importing local files into the Doris cluster. Therefore, the implementation of MySQL Load reuses the basic import capabilities of Stream Load.

The main processes of MySQL Load include:

1. The user submits a LOAD DATA request to the frontend (FE), which performs the parsing and encapsulates the request into a Stream Load task.
2. The FE selects a backend (BE) node and sends the Stream Load request to it.
3. Meanwhile, the FE reads the local file data from the MySQL client in an asynchronous and streaming manner and sends it in real time to the HTTP request of the Stream Load.
4. Once the data transfer from the MySQL client is complete, the FE waits for the Stream Load to finish and displays the import result (success or failure) to the client.

Get started

Preparations

MySQL Load requires INSERT permission on the target table. You can grant permissions to user account using the GRANT command.

Create a MySQL Load job

1. Prepare the test data

Create a data file `client_local.csv` containing the following sample data:

```
1,10
2,20
3,30
4,40
5,50
6,60
```

2. Connect to MySQL client

Connect to the MySQL client before executing the LOAD DATA command:

```
mysql --local-infile  -h <fe_ip> -P <fe_query_port> -u root -D testdb
```

> **Specific parameter options need to be used during the connection:**
>
> 1. When connecting to the MySQL client, `--local-infile` must be included, otherwise an error might be thrown.
> 2. When connecting via JDBC, `allowLoadLocalInfile=true` must be specified in the URL.

3. Create the test table

Create a table as follows in Doris:

```
CREATE TABLE testdb.t1 (
    pk     INT,
    v1     INT SUM
) AGGREGATE KEY (pk)
DISTRIBUTED BY hash (pk);
```

4. Run the LOAD DATA command

After connecting to the MySQL client, create a Load job. The command is as follows:

```
LOAD DATA LOCAL
INFILE 'client_local.csv'
INTO TABLE testdb.t1
COLUMNS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

View results

MySQL Load is a synchronous import method, and the results of the import are returned to the user in the command-line interface. If the import execution fails, it will display specific error messages.

Below is an example of a successful import result, which returns the number of imported rows:

```
Query OK, 6 rows affected (0.17 sec)
Records: 6  Deleted: 0  Skipped: 0  Warnings: 0
```

When there are exceptions during the import, the corresponding error will be displayed on the client:

```
ERROR 1105 (HY000): errCode = 2, detailMessage = [DATA_QUALITY_ERROR]too many filtered rows with load id b612907c-ccf4-4ac2-82fe-107ece655f0f
```

The `loadId` is included in the error message, based on which you can view the detailed information via the `show load warnings` command:

```
show load warnings where label='b612907c-ccf4-4ac2-82fe-107ece655f0f';
```

Cancel a MySQL Load job

Doris does not allow manual cancellation of MySQL Load jobs. In the event of a timeout or import error, the corresponding MySQL Load job will be automatically cancelled by the system.

Manual

Syntax

The syntax for LOAD DATA is as follows:

```
LOAD DATA LOCAL
INFILE '<load_data_file>'
INTO TABLE [<db_name>.]<table_name>
[PARTITION (partition_name [, partition_name] ...)]
[COLUMNS TERMINATED BY '<column_terminated_operator>']
[LINES TERMINATED BY '<line_terminated_operator>']
[IGNORE <ignore_lines> LINES]
[(col_name_or_user_var[, col_name_or_user_var] ...)]
[SET col_name={expr | DEFAULT}[, col_name={expr | DEFAULT}] ...]
[PROPERTIES (key1 = value1 [, key2=value2]) ]
```

Descriptions of modules in the Load job:

Module	Description
INFILE	This specifies the local file path, which can be either a relative path or an absolute path.Currently, load_data_file only supports a single file.
INTO TABLE	This specifies the database and table, and the database name can be omitted.
PARTITION	This specifies the target partition. If the user can determine the partition corresponding to the data, it is recommended to specify this. Data that does not fit into the specified partitions will be filtered out.
COLUMNS TERMINATED BY	This specifies the column delimiter.
LINE TERMINATED BY	This specifies the row delimiter.

Module	Description
IGNORE num LINES	This specifies the number of rows to skip in the CSV import, typically specified as 1 to skip the header.
col_name_or_user_var	This specifies the column mapping syntax. For more information, refer to Column Mapping.
PROPERTIES	Parameters for the Load.

Parameters

By the `PROPERTIES (key1 = value1 [, key2=value2])` syntax, you can configure the parameters for the Load.

Parameter	Description
max_filter_ratio	The maximum filtering rate allowed. Must be between 0 and 1, inclusive. The default value is 0, indicating no tolerance for any error rows.
timeout	The import timeout, measured in seconds. The default value is 600. The range allowed is from 1s to 259200s.
strict_mode	Whether to enable strict mode for this import. Disabled by default.
timezone	The time zone for this import. The default time zone is UTC+8. This parameter will affect the results of any time zone-related functions involved in the import.
exec_mem_limit	Memory limit for the import, defaults to 2GB, measured in bytes.
trim_double_quotes	Boolean, defaults to false. If this is set to true, the outermost double quotes will be trimmed from each field in the import file.
enclose	This specifies the enclosure character. When a CSV data field contains line breaks or column delimiters, you can specify a single-byte character as the enclosure character to prevent accidental truncation.For example, if the column delimiter is “,”, and the enclosure character is ““, in data”a,’b,c’”, “b,c” will be parsed as one field.
escape	This specifies the escape character. This is used when the data contains the same character as the enclosure character, which needs to be treated as part of the field.For example, if the data is “a,‘b,’c’”, the enclosure character is ““, and you want”b,’c” to be parsed as one field, you need to specify a single-byte escape character, such as “”, to modify the data to”a,’b,'c’”.

Example

Specify load timeout

You can adjust the import timeout by specifying `timeout` in PROPERTIES. For example, set it to 100s:

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("timeout"="100");
```

Specify allowable error rate

You can adjust the allowable error rate by specifying `max_filter_ratio` in PROPERTIES. For example, set it to 20%:

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("max_filter_ratio"="0.2");
```

Import column mapping

The following example adjusts the order of columns in the CSV file.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
(k2, k1, v1);
```

Specify column and row delimiters

You can specify the column and row delimiters using the `COLUMNS TERMINATED BY` and `LINES TERMINATED BY` clauses. In the following example, (,) and () are used as the column and row delimiters, respectively.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
COLUMNS TERMINATED BY ','
LINES TERMINATED BY '\n';
```

Specify target partition

You can specify the target partition for the import using the `PARTITION` clause. In the following example, data will be loaded into the specified partitions ‘p1’ and ‘p2’, and any data that does not belong to these two partitions will be filtered out:

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PARTITION (p1, p2);
```

Specify time zone

You can specify the `timezone` in PROPERTIES. In the following example, the timezone is set to Africa/Abidjan:

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("timezone"="Africa/Abidjan");
```

Specify the memory limit for the import

You can specify the memory limit for the import by the `exec_mem_limit` parameter in PROPERTIES. In the following example, the memory limit is set to 10G:

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("exec_mem_limit"="10737418240");
```

More help

For more detailed syntax and best practices related to MySQL Load, refer to the MySQL Load command manual.

Log Storage and Analysis | Doris Docs

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

Logs record key events in the system and contain crucial information such as the events' subject, time, location, and content. To meet the diverse needs of observability in operations, network security monitoring, and business analysis, enterprises might need to collect scattered logs for centralized storage, querying, and analysis to extract valuable content from the log data further.

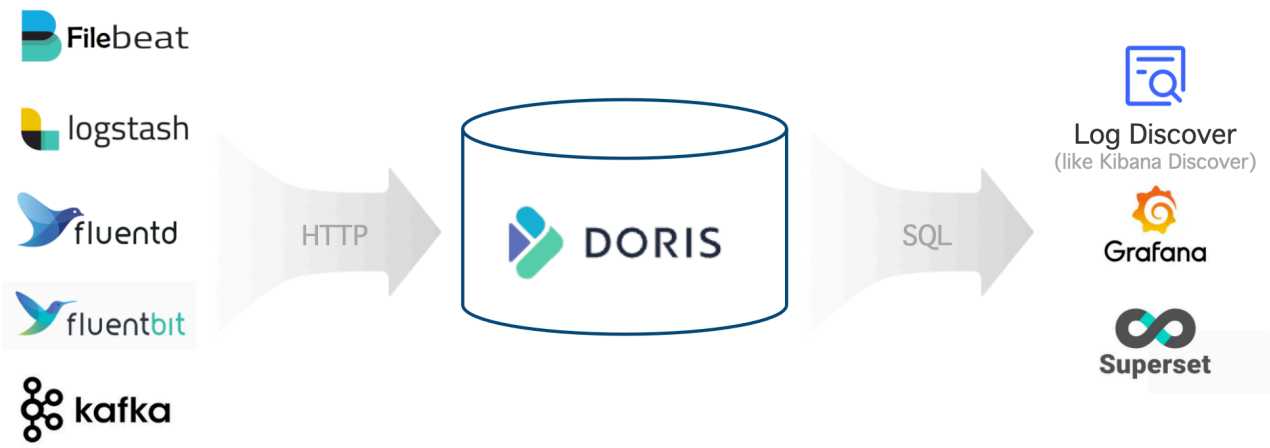
In this scenario, Apache Doris provides a corresponding solution. With the characteristics of log scenarios in mind, Apache Doris added inverted-index and ultra-fast full-text search capabilities, optimizing write performance and storage space to the extreme. This allows users to build an open, high-performance, cost-effective, and unified log storage and analysis platform based on Apache Doris.

Focused on this solution, this chapter contains the following 3 sections:

- **Overall architecture**: This section explains the core components and architecture of the log storage and analysis platform built on Apache Doris.
- **Features and advantages**: This section explains the features and advantages of the log storage and analysis platform built on Apache Doris.
- **Operational guide**: This section explains how to build a log storage and analysis platform based on Apache Doris.

Overall architecture

The following figure illustrates the architecture of the log storage and analysis platform built on Apache Doris:



Overall architecture

The architecture contains the following 3 parts:

- **Log collection and preprocessing**: Various log collection tools can write log data into Apache Doris through HTTP APIs.
- **Log storage and analysis engine**: Apache Doris provides high-performance, low-cost unified log storage, offering rich retrieval and analysis capabilities through SQL interfaces.
- **Log analysis and alert interface**: Various log retrieval and analysis tools can query Apache Doris through standard SQL interfaces, providing users with a simple and user-friendly interface.

Features and advantages

The following figure illustrates the architecture of the log storage and analysis platform built on Apache Doris::

- **High throughput, low latency log writing**: Supports stable writing of log data at the level of hundreds of TBs and GB/s per day, while maintaining latency within 1 second.
- **Cost-effective storage of massive log data**: Supports petabyte-scale storage, saving 60% to 80% in storage costs compared to Elasticsearch, and further reducing storage costs by 50% by storing cold data in S3/HDFS.
- **High-performance log full-text search and analysis**: Supports inverted indexes and full-text search, providing second-level response times for common log queries (keyword searches, trend analysis, etc.).
- **Open and user-friendly upstream and downstream ecosystem**: Upstream integration with common log collection systems and data sources such as Logstash, Filebeat, Fluentbit, and Kafka through Stream Load's universal HTTP APIs, and downstream integration with various visual analytics UIs using standard MySQL protocol and syntax, such as observability platform Grafana, BI analytics Superset, and log retrieval Doris WebUI similar to Kibana.

Cost-effective performance

After Benchmark testing and production validation, the log storage and analysis platform built on Apache Doris has a 5 to 10 times cost-performance advantage over Elasticsearch. Apache Doris's performance benefits are mainly due to its globally leading high-performance storage and query engine, as well as specialized optimizations for log scenarios:

- **Improved write throughput**: Elasticsearch's write performance bottleneck lies in CPU consumption for parsing data and building inverted indexes. In comparison, Apache Doris has optimized writes in two aspects: using SIMD and other CPU vector instructions to improve JSON data parsing speed and index-building performance and simplifying the inverted index structure for log scenarios by removing unnecessary data structures like forward indexes, effectively reducing index build complexity. With the same resources, Apache Doris's write performance is 3 to 5 times higher than Elasticsearch.
- **Reduced Storage Costs**: The storage bottleneck in Elasticsearch lies in the multiple storage formats for forward indexes, inverted indexes, and Docvalue columns, as well as the relatively low compression ratio of its general compression algorithms. In contrast, Apache Doris has made the following optimizations in storage: it removes the forward index, reducing the index data size by 30%; it uses columnar storage and the Zstandard compression algorithm, achieving a compression ratio of 5 to 10

times, which is significantly higher than Elasticsearch's 1.5 times; in log data, cold data is accessed very infrequently, and Apache Doris's hot and cold data tiering feature can automatically store logs that exceed a defined time period into lower-cost object storage, reducing the storage cost of cold data by more than 70%. For the same raw data, Doris's storage costs are only about 20% of those of Elasticsearch.

Strong analytical capabilities

Apache Doris supports standard SQL and is compatible with MySQL protocol and syntax. Therefore, log systems built on Apache Doris can use SQL for log analysis, giving the following advantages to log systems:

- **Easy to use**: Engineers and data analysts are very familiar with SQL, their expertise can be reused, no need to learn new technology stacks to quickly get started.
- **Rich ecosystem**: The MySQL ecosystem is the most widely used language in the database field, seamlessly integrating with and applying to the MySQL ecosystem. Doris can leverage MySQL command line and various GUI tools, BI tools, and other big data ecosystem tools for more complex and diverse data processing and analysis needs.
- **Strong analytical capabilities**: SQL has become the de facto standard for database and big data analysis, with powerful expressive capabilities and functions supporting retrieval, aggregation, multi-table JOIN, subqueries, UDFs, logical views, materialized views, and various data analysis capabilities.

Flexible Schema

Here is a typical example of a semi-structured log in JSON format. The top-level fields are some fixed fields, such as timestamp, source, node, component, level, clientRequestID, message, and properties, which are present in every log entry. The nested fields of the properties , such as properties.size and properties.format are more dynamic, and the fields of each log may vary.

```
{
  "timestamp": "2014-03-08T00:50:03.8432810Z",
  "source": "ADOPTIONCUSTOMERS81",
  "node": "Engine000000000405",
  "level": "Information",
  "component": "DOWNLOADER",
  "clientRequestId": "671db15d-abad-94f6-dd93-b3a2e6000672",
  "message": "Downloading file path: benchmark/2014/ADOPTIONCUSTOMERS81_94_0.parquet.gz",
  "properties": {
    "size": 1495636750,
    "format": "parquet",
    "rowCount": 855138,
    "downloadDuration": "00:01:58.3520561"
  }
}
```

Apache Doris provides several aspects of support for Flexible Schema log data:

- For changes to top-level fields, Light Schema Change can be used to add or remove columns and to add or remove indexes, enabling schema changes to be completed in seconds. When planning a log platform, users only need to consider which fields need to be indexed.

- For extension fields similar to properties, the native semi-structured data type `VARIANT` is provided, which can write any JSON data, automatically recognize field names and types in JSON, and automatically split frequently occurring fields for columnar storage for subsequent analysis. Additionally, `VARIANT` can create inverted indexes to accelerate internal field queries and retrievals.

Compared to Elasticsearch's Dynamic Mapping, Apache Doris's Flexible Schema has the following advantages:

- Allows a field to have multiple types, `VARIANT` automatically handles conflicts and type promotion for fields, better adapting to iterative changes in log data.
- `VARIANT` automatically merges infrequently occurring fields into a column store to avoid performance issues caused by excessive fields, metadata, or columns.
- Not only can columns be dynamically added, but they can also be dynamically deleted, and indexes can be dynamically added or removed, eliminating the need to index all fields at the beginning like Elasticsearch, reducing unnecessary costs.

Operational guide

Step 1: Estimate resources

Before deploying the cluster, you need to estimate the hardware resources required for the servers. Follow the steps below:

1. Estimate the resources for data writing by the following calculation formulas:

- `Average write throughput = Daily data increment / 86400 s`
- `Peak write throughput = Average write throughput \* Ratio of the peak write throughput to the average write throughput`
- `Number of CPU cores for the peak write throughput = Peak write throughput / Write throughput of a single-core CPU`

1. Estimate the resources for data storage by the calculation formula: `Storage space = Daily data increment / Data compression ratio * Number of data copies * Data storage duration`.
2. Estimate the resources for data querying. The resources for data querying depend on the query volume and complexity. It is recommended to reserve 50% of CPU resources for data query initially and then adjust according to the actual test results.
3. Integrate the calculation results as follows:
    1. Divide the number of CPU cores calculated in Step 1 and Step 3 by the number of CPU cores of a BE server, and you can get the number of BE servers.
    2. Based on the number of BE servers and the calculation result of Step 2, estimate the storage space required for each BE server.
    3. Allocate the storage space required for each BE server to 4 to 12 data disks, and you can get the storage capacity required for a single data disk.

For example, suppose that the daily data increment is 100 TB, the data compression ratio is 5, the number of data copies is 1, the storage duration of hot data is 3 days, the storage duration of cold data is 30 days, the ratio of the peak write throughput to the average write throughput is 200%, the write throughput of a single-core CUP is 10 MB/s, and 50% of CPU resources are reserved for data querying, one can estimate that:

- 3 FE servers are required, each configured with a 16-core CPU, 64 GB memory, and an 1 100 GB SSD disk.
- 15 BE servers are required, each configured with a 32-core CPU, 256 GB memory, and 10 600 GB SSD disks.
- S3 object storage space 600 TB

Refer to the following table to learn about the values of indicators in the example above and how they are calculated.

Indicator (Unit)	Value	Description
Daily data increment (TB)	100	Specify the value according to your actual needs.
Data compression ratio	5	Specify the value according to your actual needs, which is often between 3 to 10. Note that the data contains index data.
Number of data copies	1	Specify the value according to your actual needs, which can be 1, 2, or 3. The default value is 1.
Storage duration of hot data (day)	3	Specify the value according to your actual needs.
Storage duration of cold data (day)	30	Specify the value according to your actual needs.
Data storage duration	33	Calculation formula: `Storage duration of hot data + Storage duration of cold data`
Estimated storage space for hot data (TB)	60	Calculation formula: `Daily data increment / Data compression ratios * Number of data copies * Storage duration of hot data`
Estimated storage space for cold data (TB)	600	Calculation formula: `Daily data increment / Data compression ratios * Number of data copies * Storage duration of cold data`
Ratio of the peak write throughput to the average write throughput	200%	Specify the value according to your actual needs. The default value is 200%.
Number of CPU cores of a BE server	32	Specify the value according to your actual needs. The default value is 32.
Average write throughput (MB/s)	1214	Calculation formula: `Daily data increment / 86400 s`
Peak write throughput (MB/s)	2427	Calculation formula: `Average write throughput * Ratio of the peak write throughput to the average write throughput`

Indicator (Unit)	Value	Description
Number of CPU cores for the peak write throughput	242.7	Calculation formula: `Peak write throughput / Write throughput of a single-core CPU`
Percent of CPU resources reserved for data querying	50%	Specify the value according to your actual needs. The default value is 50%.
Estimated number of BE servers	15.2	Calculation formula: `Number of CPU cores for the peak write throughput / Number of CPU cores of a BE server /(1 - Percent of CPU resources reserved for data querying)`
Rounded number of BE servers	15	Calculation formula: `MAX (Number of data copies, Estimated number of BE servers)`
Estimated data storage space for each BE server (TB)	5.7	Calculation formula: `Estimated storage space for hot data / Estimated number of BE servers /(1 - 30%)`, where 30% represents the percent of reserved storage space. It is recommended to mount 4 to 12 data disks on each BE server to enhance I/O capabilities.

Step 2: Deploy the cluster

After estimating the resources, you need to deploy the cluster. It is recommended to deploy in both physical and virtual environments manually. For manual deployment, refer to Manual Deployment.

Step 3: Optimize FE and BE configurations

After completing the cluster deployment, it is necessary to optimize the configuration parameters for both the front-end and back-end separately, so as to better suit the scenario of log storage and analysis.

Optimize FE configurations

You can find FE configuration fields in `fe/conf/fe.conf`. Refer to the following table to optimize FE configurations.

Configuration fields to be optimized	Description
`max_running_txn_num_per_db = 10000`	Increase the parameter value to adapt to high-concurrency import transactions.
`streaming_label_keep_max_second = 3600 label_keep_max_second = 7200`	Increase the retention time to handle high-frequency import transactions with high memory usage.
`enable_round_robin_create_tablet = true`	When creating Tablets, use a Round Robin strategy to distribute evenly.
`tablet_rebalancer_type = partition`	When balancing Tablets, use a strategy to evenly distribute within each partition.
`autobucket_min_buckets = 10`	Increase the minimum number of automatically bucketed buckets from 1 to 10 to avoid insufficient buckets when the log volume increases.

Configuration fields to be optimized	Description
`max_backend_heartbeat_failure_tolerance_count = 10`	In log scenarios, the BE server may experience high pressure, leading to short-term timeouts, so increase the tolerance count from 1 to 10.

For more information, refer to FE Configuration.

Optimize BE configurations

You can find BE configuration fields in `be/conf/be.conf`. Refer to the following table to optimize BE configurations.

Module	Configuration fields to be optimized	Description
Storage	`storage_root_path = /path/to/dir1;/path/to/dir2;...;/path/to/dir12`	Configure the storage path for hot data on disk directories.
-	`enable_file_cache = true`	Enable file caching.
-	`file_cache_path = [{"path": "/mnt/datadisk0/file_cache", "total_size":53687091200, "query_limit": "10737418240"}, {"path": "/mnt/datadisk1/file_cache", "total_size":53687091200,"query_limit": "10737418240"}]`	Configure the cache path and related settings for cold data with the following specific configurations: `path`: cache path `total_size`: total size of the cache path in bytes, where 53687091200 bytes equals 50 GB `query_limit`: maximum amount of data that can be queried from the cache path in one query in bytes, where 10737418240 bytes equals 10 GB
Write	`write_buffer_size = 1073741824`	Increase the file size of the write buffer to reduce small files and random I/O operations, improving performance.
-	`max_tablet_version_num = 20000`	In coordination with the time_series compaction strategy for table creation, allow more versions to remain temporarily unmerged. No longer required after version 2.1.11, as there is a time_series_max_tablet_version_num configuration
Compaction	`max_cumu_compaction_threads = 8`	Set to CPU core count / 4, indicating that 1/4 of CPU resources are used for writing, 1/4 for background compaction, and 2/1 for queries and other operations.
-	`inverted_index_compaction_enable = true`	Enable inverted index compaction to reduce CPU consumption during compaction.
-	`enable_segcompaction = false enable_ordered_data_compaction = false`	Disable two compaction features that are unnecessary for log scenarios.
-	`enable_compaction_priority_scheduling = false`	Low-priority compaction is limited to 2 tasks on a single disk, which can affect the speed of compaction.
-	`total_permits_for_compaction_score = 200000`	The parameter is used to control memory, under the memory time series strategy, the parameter itself can control memory.
Cache	`disable_storage_page_cache = true inverted_index_searcher_cache_limit = 30%`	Due to the large volume of log data and limited caching effect, switch from data caching to index caching.
-	`inverted_index_cache_stale_sweep_time_sec = 3600 index_cache_entry_stay_time_after_lookup_s = 3600`	Maintain index caching in memory for up to 1 hour.

Module	Configuration fields to be optimized	Description
-	`enable_inverted_index_cache_on_cooldown = true` `enable_write_index_searcher_cache = false`	Enable automatic caching of cold data storage during index uploading.
-	`tablet_schema_cache_recycle_interval = 3600` `segment_cache_capacity = 20000`	Reduce memory usage by other caches.
-	`inverted_index_ram_dir_enable = true`	Reduce the IO overhead caused by writing to index files temporarily.
Thread	`pipeline_executor_size = 24` `doris_scanner_thread_pool_thread_num = 48`	Configure computing threads and I/O threads for a 32-core CPU in proportion to core count.
-	`scan_thread_nice_value = 5`	Lower the priority of query I/O threads to ensure writing performance and timeliness.
Other	`string_type_length_soft_limit_bytes = 10485760`	Increase the length limit of string-type data to 10 MB.
-	`trash_file_expire_time_sec = 300` `path_gc_check_interval_second = 900 path_scan_interval_second = 900`	Accelerate the recycling of trash files.

For more information, refer to BE Configuration.

Step 4: Create tables

Due to the distinct characteristics of both writing and querying log data, it is recommended to configure tables with targeted settings to enhance performance.

Configure data partitioning and bucketing

- For data partitioning:
    - Enable range partitioning (`PARTITION BY RANGE(ts)`) with dynamic partitions (`"dynamic_partition.enable" = "true"`) managed automatically by day.
    - Use a field in the DATETIME type as the key (`DUPLICATE KEY(ts)`) for accelerated retrieval of the latest N log entries.
- For data bucketing:
    - Configure the number of buckets to be roughly three times the total number of disks in the cluster, with each bucket containing approximately 5GB of data after compression.
    - Use the Random strategy (`DISTRIBUTED BY RANDOM BUCKETS 60`) to optimize batch writing efficiency when paired with single tablet imports.

For more information, refer to Data Partitioning.

Configure compression parameters

Use the zstd compression algorithm (“compression” = “zstd”) to improve data compression efficiency.

Configure compaction parameters

Configure compaction fields as follows:

- Use the time_series strategy (`"compaction_policy" = "time_series"`) to reduce write amplification, which is crucial for high-throughput log writes.

Configure index parameters

Configuring index fields as follows:

- Create indexes for fields that are frequently queried (`USING INVERTED`).
- For fields that require full-text search, specify the parser field as unicode, which satisfies most requirements. If there is a need to support phrase queries, set the support_phrase field to true; if not needed, set it to false to reduce storage space.

Configure storage parameters

Configure storage policies as follows:

- For storage of hot data, if using cloud storage, configure the number of data copies as 1; if using physical disks, configure the number of data copies as at least 2 (`"replication_num" = "2"`).
- Configure the storage location for log_s3 (`CREATE RESOURCE "log_s3"`) and set the log_policy_3day policy (`CREATE STORAGE POLICY log_policy_3day`), where the data is cooled and moved to the specified storage location of log_s3 after 3 days. Refer to the code below.

```
CREATE DATABASE log_db;
USE log_db;

CREATE RESOURCE "log_s3"
PROPERTIES
(
    "type" = "s3",
    "s3.endpoint" = "your_endpoint_url",
    "s3.region" = "your_region",
    "s3.bucket" = "your_bucket",
    "s3.root.path" = "your_path",
    "s3.access_key" = "your_ak",
    "s3.secret_key" = "your_sk"
);

CREATE STORAGE POLICY log_policy_3day
PROPERTIES(
    "storage_resource" = "log_s3",
    "cooldown_ttl" = "259200"
);

CREATE TABLE log_table
(
  `ts` DATETIME,
  `host` TEXT,
  `path` TEXT,
  `message` TEXT,
  INDEX idx_host (`host`) USING INVERTED,
  INDEX idx_path (`path`) USING INVERTED,
  INDEX idx_message (`message`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true")
)
ENGINE = OLAP
DUPLICATE KEY(`ts`)
```

```
PARTITION BY RANGE(`ts`) ()
DISTRIBUTED BY RANDOM BUCKETS 60
PROPERTIES (
  "compression" = "zstd",
  "compaction_policy" = "time_series",
  "dynamic_partition.enable" = "true",
  "dynamic_partition.create_history_partition" = "true",
  "dynamic_partition.time_unit" = "DAY",
  "dynamic_partition.start" = "-30",
  "dynamic_partition.end" = "1",
  "dynamic_partition.prefix" = "p",
  "dynamic_partition.buckets" = "60",
  "dynamic_partition.replication_num" = "2", -- unneccessary for the compute-storage coupled mode
  "replication_num" = "2", -- unneccessary for the compute-storage coupled mode
  "storage_policy" = "log_policy_3day" -- unneccessary for the compute-storage coupled mode
);
```

Step 5: Collect logs

After completing table creation, you can proceed with log collection.

Apache Doris provides open and versatile Stream HTTP APIs, through which you can connect with popular log collectors such as Logstash, Filebeat, Kafka, and others to carry out log collection work. This section explains how to integrate these log collectors using the Stream HTTP APIs.

Integrating Logstash

Follow these steps:

1. Download and install the Logstash Doris Output plugin. You can choose one of the following two methods:
    - Click to download and install.
    - Compile from the source code and run the following command to install:

```
./bin/logstash-plugin install logstash-output-doris-1.2.0.gem
```

2. Configure Logstash. Specify the following fields:

- `logstash.yml`: Used to configure Logstash batch processing log sizes and timings for improved data writing performance.

```
Plain Text pipeline.batch.size: 1000000 pipeline.batch.delay: 10000
```

- `logstash_demo.conf`: Used to configure the specific input path of the collected logs and the settings for output to Apache Doris.

```
input {
    file {
    path => "/path/to/your/log"
  }
}

output {
  doris {
```

```
    http_hosts => [ "<http://fehost1:http_port>", "<http://fehost2:http_port>", "<http://fehost3:http_port">]
    user => "your_username"
    password => "your_password"
    db => "your_db"
    table => "your_table"

    # doris stream load http headers
    headers => {
    "format" => "json"
    "read_json_by_line" => "true"
    "load_to_single_tablet" => "true"
    }

    # field mapping: doris fileld name => logstash field name
    # %{} to get a logstash field, [] for nested field such as [host][name] for host.name
    mapping => {
    "ts" => "%{@timestamp}"
    "host" => "%{[host][name]}"
    "path" => "%{[log][file][path]}"
    "message" => "%{message}"
    }
    log_request => true
    log_speed_interval => 10
  }
}
    ```

3. Run Logstash according to the command below, collect logs, and output to Apache Doris.

```shell
./bin/logstash -f logstash_demo.conf
```
```

For more information about the Logstash Doris Output plugin, see Logstash Doris Output Plugin.

**Integrating Filebeat**

Follow these steps:

1. Obtain the Filebeat binary file that supports output to Apache Doris. You can click to download or compile it from the Apache Doris source code.

2. Configure Filebeat. Specify the filebeat_demo.yml field that is used to configure the specific input path of the collected logs and the settings for output to Apache Doris.

```
input
filebeat.inputs:
- type: log
enabled: true
paths:
    - /path/to/your/log
multiline:
    type: pattern
    pattern: '^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}'
    negate: true
    match: after
    skip_newline: true

processors:
- script:
    lang: javascript
    source: >
```
```

```
        function process(event) {
            var msg = event.Get("message");
            msg = msg.replace(/\t/g, "  ");
            event.Put("message", msg);
        }
- dissect:
    # 2024-06-08 18:26:25,481 INFO (report-thread|199) [ReportHandler.cpuReport():617] begin to handle
    tokenizer: "%{day} %{time} %{log_level} (%{thread}) [%{position}] %{content}"
    target_prefix: ""
    ignore_failure: true
    overwrite_keys: true

# queue and batch
queue.mem:
events: 1000000
flush.min_events: 100000
flush.timeout: 10s

# output
output.doris:
fenodes: [ "http://fehost1:http_port", "http://fehost2:http_port", "http://fehost3:http_port" ]
user: "your_username"
password: "your_password"
database: "your_db"
table: "your_table"
# output string format
codec_format_string: '{"ts": "%{[day]} %{[time]}", "host": "%{[agent][hostname]}", "path": "%{[log][file][path]}", "message": "%
{[message]}"}'
headers:
    format: "json"
    read_json_by_line: "true"
    load_to_single_tablet: "true"
```

3. Run Filebeat according to the command below, collect logs, and output to Apache Doris.

```
chmod +x filebeat-doris-2.1.1
./filebeat-doris-2.1.1 -c filebeat_demo.yml
```

For more information about Filebeat, refer to Beats Doris Output Plugin.

Integrating Kafka

Write JSON formatted logs to Kafka's message queue, create a Kafka Routine Load, and allow Apache Doris to actively pull data from Kafka.

You can refer to the example below, where `property.*` represents Librdkafka client-related configurations and needs to be adjusted according to the actual Kafka cluster situation.

```
CREATE ROUTINE LOAD load_log_kafka ON log_db.log_table
COLUMNS(ts, clientip, request, status, size)
PROPERTIES (
"max_batch_interval" = "60",
"max_batch_rows" = "20000000",
"max_batch_size" = "1073741824",
"load_to_single_tablet" = "true",
"format" = "json"
)
FROM KAFKA (
"kafka_broker_list" = "host:port",
"kafka_topic" = "log__topic_",
```

```
"property.group.id" = "your_group_id",
"property.security.protocol"="SASL_PLAINTEXT",
"property.sasl.mechanism"="GSSAPI",
"property.sasl.kerberos.service.name"="kafka",
"property.sasl.kerberos.keytab"="/path/to/xxx.keytab",
"property.sasl.kerberos.principal"="<xxx@yyy.com>"
);
<br />SHOW ROUTINE LOAD;
```

For more information about Kafka, see Routine Load.

Using customized programs to collect logs

In addition to integrating common log collectors, you can also customize programs to import log data into Apache Doris using the Stream Load HTTP API. Refer to the following code:

```
curl
--location-trusted
-u username:password
-H "format:json"
-H "read_json_by_line:true"
-H "load_to_single_tablet:true"
-H "timeout:600"
-T logfile.json
http://fe_host:fe_http_port/api/log_db/log_table/_stream_load
```

When using custom programs, pay attention to the following key points:

- Use Basic Auth for HTTP authentication, calculate using the command echo -n 'username:password' | base64.
- Set HTTP header "format:json" to specify the data format as JSON.
- Set HTTP header "read_json_by_line:true" to specify one JSON per line.
- Set HTTP header "load_to_single_tablet:true" to import data into one bucket at a time to reduce small file imports.
- It is recommended to write batches whose sizes are between 100MB to 1GB on the client side. For Apache Doris version 2.1 and higher, you need to reduce batch sizes on the client side through the Group Commit function.

Step 6: Query and analyze logs

Query logs

Apache Doris supports standard SQL, so you can connect to the cluster through MySQL client or JDBC to execute SQL for log queries.

```
Plain Text mysql -h fe_host -P fe_mysql_port -u your_username -Dyour_db_name
```

Here are 5 common SQL query commands for reference:

- View the latest 10 log entries

```
SELECT * FROM your_table_name ORDER BY ts DESC LIMIT 10;
```

- Query the latest 10 log entries with the host as 8.8.8.8

```
SELECT * FROM your_table_name WHERE host = '8.8.8.8' ORDER BY ts DESC LIMIT 10;
```

- Retrieve the latest 10 log entries with error or 404 in the request field. In the command below, MATCH_ANY is a full-text search SQL syntax used by Apache Doris for matching any keyword in the fields.

```
SELECT * FROM your_table_name WHERE message **MATCH_ANY** 'error 404'
ORDER BY ts DESC LIMIT 10;
```

- Retrieve the latest 10 log entries with image and faq in the request field. In the command below, MATCH_ALL is a full-text search SQL syntax used by Apache Doris for matching all keywords in the fields.

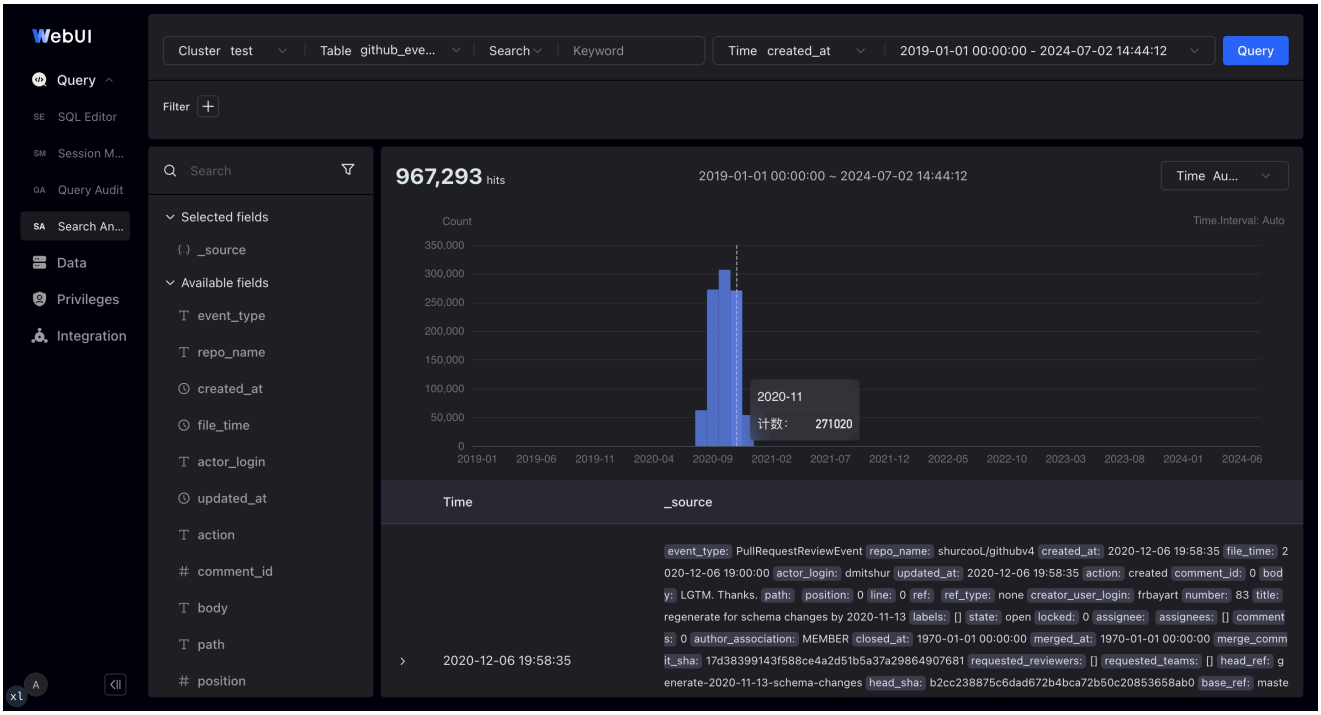
```
SELECT * FROM your_table_name WHERE message **MATCH_ALL** 'image faq'
ORDER BY ts DESC LIMIT 10;
```

- Retrieve the latest 10 entries with image and faq in the request field. In the following command, MATCH_PHRASE is a full-text search SQL syntax used by Apache Doris for matching all keywords in the fields and requiring consistent order. In the example below, a image faq b can match, but a faq image b cannot match because the order of image and faq does not match the syntax.

```
SELECT * FROM your_table_name WHERE message **MATCH_PHRASE** 'image faq'
ORDER BY ts DESC LIMIT 10;
```

Analyze logs visually

Some third-party vendors offer visual log analysis development platforms based on Apache Doris, which include a log search and analysis interface similar to Kibana Discover. These platforms provide an intuitive and user-friendly exploratory log analysis interaction.



WebUI

- Support for full-text search and SQL modes
- Support for selecting query log timeframes with time boxes and histograms
- Display of detailed log information, expandable into JSON or tables
- Interactive clicking to add and remove filter conditions in the log data context
- Display of top field values in search results for finding anomalies and further drilling down for analysis

Please contact dev@doris.apache.org to find more.

CSV

This document explains how to load CSV format data files into Doris. Doris supports flexible CSV format configuration, including custom delimiters, field enclosures, and provides various loading methods to meet data loading requirements in different scenarios.

Loading Methods

Doris supports the following methods to load CSV format data:

- Stream Load
- Broker Load
- Routine Load
- MySQL Load
- INSERT INTO FROM S3 TVF
- INSERT INTO FROM HDFS TVF

Parameter Configuration

Parameter Support

The following table lists the CSV format parameters supported by various loading methods:

Parameter	Default Value	Stream Load	Broker Load	Routine Load	MySQL Load	TVF
Line Delimiter	`\n`	line_delimiter	LINES TERMINATED BY	Not supported	LINES TERMINATED BY	line_delimiter
Column Delimiter	`\t`	column_separator	COLUMNS TERMINATED BY	COLUMNS TERMINATED BY	COLUMNS TERMINATED BY	column_separator
Enclosure	None	enclose	PROPERTIES.enclose	PROPERTIES.enclose	PROPERTIES.enclose	enclose
Escape Character	\	escape	PROPERTIES.escape	PROPERTIES.escape	PROPERTIES.escape	escape
Skip Lines	0	skip_lines	PROPERTIES.skip_lines	Not supported	IGNORE LINES	skip_lines
Trim Double Quotes	false	trim_double_quotes	Not supported	PROPERTIES.trim_double_quotes	Not supported	trim_double_quotes
Compression Format	plain	compress_type	PROPERTIES.compress_type	Not supported	Not supported	compress_type

Note

1. Stream Load: Parameters are specified directly through HTTP Headers, e.g., `-H "line_delimiter:\n"`
2. Broker Load: Parameters are specified in SQL statements, where:
    - Delimiters are specified through `COLUMNS TERMINATED BY`, `LINES TERMINATED BY`
    - Other parameters are specified through `PROPERTIES`, e.g., `PROPERTIES("compress_type"="gz")`
3. Routine Load: Parameters are specified in SQL statements, where:
    - Delimiters are specified through `COLUMNS TERMINATED BY`
    - Other parameters are specified through `PROPERTIES`, e.g., `PROPERTIES("enclose"="\"")`
4. MySQL Load: Parameters are specified in SQL statements, where:
    - Delimiters are specified through `LINES TERMINATED BY`, `COLUMNS TERMINATED BY`
    - Other parameters are specified through `PROPERTIES`, e.g., `PROPERTIES("escape"="\\")`
5. TVF: Parameters are specified in TVF statements, e.g., `S3("line_delimiter"="\n")`

Parameter Description

Line Delimiter

- Purpose: Specifies the line break character in the data file
- Default Value: `\n`
- Features: Supports multiple characters combination as line break
- Use Cases and Examples:
    - Linux/Unix System Files:

```
Data File:
John,25\n
Mary,30\n

Parameter Setting:
line_delimiter: \n (default value, can be omitted)
```

    - Windows System Files:

```
Data File:
John,25\r\n
Mary,30\r\n

Parameter Setting:
line_delimiter: \r\n
```

    - Special Program Generated Files:

```
Data File:
John,25\r
Mary,30\r
```

```
Parameter Setting:
line_delimiter: \r
```

- Custom Multi-character Delimiter:

```
Data File:
John,25||
Mary,30||

Parameter Setting:
line_delimiter: ||
```

Column Delimiter

- Purpose: Specifies the column delimiter in the data file
- Default Value: `\t` (tab)
- Features:
  - Supports visible and invisible characters
  - Supports multiple character combinations
  - Invisible characters need to use hex representation with `\x` prefix
- Special Handling for MySQL Protocol:
  - Invisible characters need an extra backslash
  - For example, Hive's `\x01` needs to be written as `\\x01` in Broker Load
- Use Cases and Examples:
  - Common Visible Characters:

```
Data File:
John,25,New York
Mary,30,Los Angeles

Parameter Setting:
column_separator: ,
```

  - Tab (Default):

```
Data File:
John    25    New York
Mary    30    Los Angeles

Parameter Setting:
column_separator: \t (default value, can be omitted)
```

  - Hive Files (Stream Load):

```
Data File:
John\x0125\x01New York
Mary\x0130\x01Los Angeles

Parameter Setting:
column_separator: \x01
```

- Hive Files (Broker Load):

```
Data File:
John\x0125\x01New York
Mary\x0130\x01Los Angeles

Parameter Setting:
PROPERTIES("column_separator"="\\x01")
```

- Multi-character Delimiter:

```
Data File:
John||25||New York
Mary||30||Los Angeles

Parameter Setting:
column_separator: ||
```

Enclosure

- Purpose: Protects fields containing special characters from being incorrectly parsed
- Limitation: Only supports single-byte characters
- Common Characters:
  - Single quote: '
  - Double quote: "
- Use Cases and Examples:
  - Field Contains Column Delimiter:

```
Data: a,'b,c',d
Column Delimiter: ,
Enclosure: '
Parsing Result: 3 fields [a] [b,c] [d]
```

  - Field Contains Line Delimiter:

```
Data: a,'b\nc',d
Column Delimiter: ,
Enclosure: '
Parsing Result: 3 fields [a] [b\nc] [d]
```

  - Field Contains Both Column and Line Delimiters:

```
Data: a,'b,c\nd,e',f
Column Delimiter: ,
Enclosure: '
Parsing Result: 3 fields [a] [b,c\nd,e] [f]
```

Escape Character

- Purpose: Escapes characters in fields that are the same as the enclosure character

- Limitation: Only supports single-byte characters, default is \
- Use Cases and Examples:
    - Field Contains Enclosure Character:

```
Data: a,'b,\'c',d
Column Delimiter: ,
Enclosure: '
Escape Character: \
Parsing Result: 3 fields [a] [b,'c] [d]
```

    - Field Contains Multiple Enclosure Characters:

```
Data: a,"b,\"c\"d",e
Column Delimiter: ,
Enclosure: "
Escape Character: \
Parsing Result: 3 fields [a] [b,"c"d] [e]
```

    - Field Contains Escape Character Itself:

```
Data: a,'b\\c',d
Column Delimiter: ,
Enclosure: '
Escape Character: \
Parsing Result: 3 fields [a] [b\c] [d]
```

Skip Lines

- Purpose: Skips the first few lines of the CSV file
- Type: Integer
- Default Value: 0
- Special Note:
    - When format is `csv_with_names`, the system automatically skips the first line (column names), ignoring the `skip_lines` parameter
    - When format is `csv_with_names_and_types`, the system automatically skips the first two lines (column names and types), ignoring the `skip_lines` parameter
- Use Cases and Examples:
    - Skip Title Line:

```
Data File:
Name,Age,City
John,25,New York
Mary,30,Los Angeles

Parameter Setting:
skip_lines: 1
Result: Skip title line, load subsequent data
```

    - Skip Comment Lines:

```
Data File:
# User Information Table
# Created Time: 2024-01-01
John,25,New York
Mary,30,Los Angeles

Parameter Setting:
skip_lines: 2
Result: Skip comment lines, load subsequent data
```

- Use csv_with_names Format:

```
Data File:
name,age,city
John,25,New York
Mary,30,Los Angeles

Parameter Setting:
format: csv_with_names
Result: System automatically skips the first line of column names
```

- Use csv_with_names_and_types Format:

```
Data File:
name,age,city
string,int,string
John,25,New York
Mary,30,Los Angeles

Parameter Setting:
format: csv_with_names_and_types
Result: System automatically skips the first two lines of column names and types
```

Trim Double Quotes

- Purpose: Trims the outermost double quotes from each field in the CSV file
- Type: Boolean
- Default Value: false
- Use Cases and Examples:

  - Trim Double Quotes:

```
Data File:
"John","25","New York"
"Mary","30","Los Angeles"

Parameter Setting:
trim_double_quotes: true
Result:
John,25,New York
Mary,30,Los Angeles
```

Compression Format

- Purpose: Specifies the compression format of the data file
- Type: String, ignoring case
- Default Value: plain
- Supported Compression Formats:
    - plain: No compression (default)
    - bz2: BZIP2 compression
    - deflate: DEFLATE compression
    - gz: GZIP compression
    - lz4: LZ4 Frame format compression
    - lz4_block: LZ4 Block format compression
    - lzo: LZO compression
    - lzop: LZOP compression
    - snappy_block: SNAPPY Block format compression
- Note:
    - tar is a file packaging format, not a compression format, so it is not supported
    - If you need to use tar packaged files, please unpack them first before loading

Usage Examples

This section demonstrates the usage of CSV format in different loading methods.

Stream Load

```
# Specify delimiter
curl --location-trusted -u root: \
    -H "column_separator:," \
    -H "line_delimiter:\n" \
    -T example.csv \
    http://<fe_host>:<fe_http_port>/api/test_db/test_table/_stream_load

# Handle quoted data
curl --location-trusted -u root: \
    -H "column_separator:," \
    -H "enclose:\"" \
    -T example.csv \
    http://<fe_host>:<fe_http_port>/api/test_db/test_table/_stream_load

# Load compressed file
curl --location-trusted -u root: \
    -H "compress_type:gz" \
    -T example.csv.gz \
    http://<fe_host>:<fe_http_port>/api/test_db/test_table/_stream_load
```

Broker Load

```
-- Specify delimiter
LOAD LABEL test_db.test_label
(
    DATA INFILE("s3://bucket/example.csv")
    INTO TABLE test_table
```

```
    COLUMNS TERMINATED BY ","
    LINES TERMINATED BY "\n"
    ...
);

-- Handle quoted data
LOAD LABEL test_db.test_label
(
    DATA INFILE("s3://bucket/example.csv")
    INTO TABLE test_table
    COLUMNS TERMINATED BY ","
    LINES TERMINATED BY "\n"
    PROPERTIES
    (
        "enclose" = "\"",
        "escape" = "\\"
    )
    ...
);

-- Load compressed file
LOAD LABEL test_db.test_label
(
    DATA INFILE("s3://bucket/example.csv.gz")
    INTO TABLE test_table
    COLUMNS TERMINATED BY ","
    LINES TERMINATED BY "\n"
    PROPERTIES
    (
        "compress_type" = "gz"
    )
    ...
);
```

Routine Load

```
-- Specify delimiter
CREATE ROUTINE LOAD test_db.test_job ON test_table
COLUMNS TERMINATED BY ","
FROM KAFKA
(
     ...
);

-- Handle quoted data
CREATE ROUTINE LOAD test_db.test_job ON test_table
COLUMNS TERMINATED BY ","
PROPERTIES
(
    "enclose" = "\"",
    "escape" = "\\"
)
FROM KAFKA
(
     ...
);
```

MySQL Load

```
-- Specify delimiter
LOAD DATA LOCAL INFILE 'example.csv'
INTO TABLE test_table
COLUMNS TERMINATED BY ','
LINES TERMINATED BY '\n';

-- Handle quoted data
LOAD DATA LOCAL INFILE 'example.csv'
INTO TABLE test_table
COLUMNS TERMINATED BY ','
LINES TERMINATED BY '\n'
PROPERTIES
(
    "enclose" = "\"",
    "escape" = "\\"
);

-- Skip table header
LOAD DATA LOCAL INFILE 'example.csv'
INTO TABLE test_table
COLUMNS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

TVF Load

```sql – Specify delimiter INSERT INTO test_table SELECT * FROM S3 ( “path” = “s3://bucket/example.csv”, “column_separator” = “,”, “line_delimiter” = “”, … );

– Handle quoted data INSERT INTO test_table SELECT * FROM S3 ( “path” = “s3://bucket/example.csv”, “column_separator” = “,”, “enclose” = “"”, “escape” = “\”, … );

– Load compressed file INSERT INTO test_table SELECT * FROM S3 ( “path” = “s3://bucket/example.csv.gz”, “column_separator” = “,”, “line_delimiter” = “”, “compress_type” = “gz”, … );
```

JSON | File Format

This document explains how to load JSON format data files into Doris. Doris supports loading standard JSON format data and can flexibly handle different JSON data structures through parameter configuration, supporting field extraction from JSON data and handling nested structures.

Loading Methods

The following loading methods support JSON format data:

- Stream Load
- Broker Load
- Routine Load
- INSERT INTO FROM S3 TVF
- INSERT INTO FROM HDFS TVF

Supported JSON Formats

Doris supports the following three JSON formats:

Multiple Rows Represented as Array

Suitable for batch loading multiple rows of data, requirements: - Root node must be an array - Each element in the array is an object representing a row of data - Must set `strip_outer_array=true`

Example data:

```
[
    {"id": 123, "city": "beijing"},
    {"id": 456, "city": "shanghai"}
]

// Supports nested structures
[
    {"id": 123, "city": {"name": "beijing", "region": "haidian"}},
    {"id": 456, "city": {"name": "beijing", "region": "chaoyang"}}
]
```

Single Row Represented as Object

Suitable for loading single row data, requirements: - Root node must be an object - The entire object represents one row of data

Example data:

```
{"id": 123, "city": "beijing"}
```

```
// Supports nested structures
{"id": 123, "city": {"name": "beijing", "region": "haidian"}}
```

> **Note**
>
> Typically used with Routine Load method, such as single messages in Kafka.

Multiple Object Rows Separated by Delimiter

Suitable for batch loading multiple rows of data, requirements: - Each line is a complete JSON object - Must set `read_json_by_line=true` - Line delimiter can be specified using `line_delimiter` parameter, default is `\n`

Example data:

```
{"id": 123, "city": "beijing"}
{"id": 456, "city": "shanghai"}
```

Parameter Configuration

Parameter Support

The following table lists the JSON format parameters supported by various loading methods:

Parameter	Default Value	Stream Load	Broker Load	Routine Load	TVF
json paths	None	jsonpaths	properties.jsonpaths	properties.jsonpaths	jsonpaths
json root	None	json_root	properties.json_root	properties.json_root	json_root
strip outer array	false	strip_outer_array	properties.strip_outer_array	properties.strip_outer_array	strip_outer_array
read json by line	false	read_json_by_line	Always true	Not supported	read_json_by_line, default true
fuzzy parse	false	fuzzy_parse	properties.fuzzy_parse	Not supported	fuzzy_parse
num as string	false	num_as_string	properties.num_as_string	properties.num_as_string	num_as_string
compression format	plain	Not Supported	PROPERTIES.compress_type	Not supported	compress_type

Note

1. Stream Load: Parameters are specified directly through HTTP Headers, e.g., `-H "jsonpaths: $.data"`
2. Broker Load: Parameters are specified through `PROPERTIES`, e.g., `PROPERTIES("jsonpaths"="$.data")`
3. Routine Load: Parameters are specified through `PROPERTIES`, e.g., `PROPERTIES("jsonpaths"="$.data")`
4. TVF: Parameters are specified in TVF statements, e.g., `S3("jsonpaths"="$.data")`
5. If you need to load the JSON object at the root node of a JSON file, the jsonpaths should be specified as $., e.g., PROPERTIES("jsonpaths"="$.")

Parameter Description

JSON Path

- Purpose: Specifies how to extract fields from JSON data
- Type: String array
- Default Value: None, defaults to matching column names
- Usage Examples:

```
-- Basic usage
["$.id", "$.city"]

-- Nested structures
["$.id", "$.info.city", "$.data[0].name"]
```

JSON Root

- Purpose: Specifies the parsing starting point for JSON data
- Type: String
- Default Value: None, defaults to parsing from root node
- Usage Example:

```
-- Original data
{
  "data": {
    "id": 123,
    "city": "beijing"
  }
}

-- Set json_root
json_root = $.data
```

Strip Outer Array

- Purpose: Specifies whether to remove the outermost array structure
- Type: Boolean
- Default Value: false
- Usage Example:

```
-- Original data
[
  {"id": 1, "city": "beijing"},
  {"id": 2, "city": "shanghai"}
]

-- Set strip_outer_array=true
```

Read JSON By Line

- Purpose: Specifies whether to read JSON data line by line
- Type: Boolean
- Default Value: false
- Usage Example:

```
-- Original data (one complete JSON object per line)
{"id": 1, "city": "beijing"}
{"id": 2, "city": "shanghai"}

-- Set read_json_by_line=true
```

Fuzzy Parse

- Purpose: Accelerates JSON data loading efficiency
- Type: Boolean
- Default Value: false
- Limitations:
  - Field order in each row of the Array must be identical
  - Usually used with strip_outer_array
- Performance: Can improve loading efficiency by 3-5 times

Num As String

- Purpose: Specifies whether to parse JSON numeric types as strings
- Type: Boolean
- Default Value: false

- Use Cases:
    - Handling numbers outside numeric range
    - Avoiding numeric precision loss
- Usage Example:

```
-- Original data
{
  "id": "12345678901234567890",
  "price": "99999999.999999"
}
-- Set num_as_string=true, price field will be parsed as string
```

Relationship between JSON Path and Columns

During data loading, JSON Path and Columns serve different responsibilities:

JSON Path: Defines data extraction rules - Extracts fields from JSON data according to specified paths - Extracted fields are reordered according to the order defined in JSON Path

Columns: Defines data mapping rules - Maps extracted fields to target table columns - Can perform column reordering and transformation

These two parameters are processed serially: first, JSON Path extracts fields from source data and forms an ordered dataset, then Columns maps these data to table columns. If Columns is not specified, extracted fields will be mapped directly according to table column order.

Usage Examples

Using JSON Path Only

Table structure and data:

```
-- Table structure
CREATE TABLE example_table (
    k2 int,
    k1 int
);

-- JSON data
{"k1": 1, "k2": 2}
```

Load command:

```
curl -v ... -H "format: json" \
    -H "jsonpaths: [\"$.k2\", \"$.k1\"]" \
    -T example.json \
    http://<fe_host>:<fe_http_port>/api/db_name/table_name/_stream_load
```

Load result:

```
+------+------+
| k1   | k2   |
+------+------+
|    2 |    1 |
+------+------+
```

Using JSON Path + Columns

Using the same table structure and data, adding columns parameter:

Load command:

```
curl -v ... -H "format: json" \
    -H "jsonpaths: [\"$.k2\", \"$.k1\"]" \
    -H "columns: k2, k1" \
    -T example.json \
    http://<fe_host>:<fe_http_port>/api/db_name/table_name/_stream_load
```

Load result:

```
+------+------+
| k1   | k2   |
+------+------+
|    1 |    2 |
+------+------+
```

Field Reuse

Table structure and data:

```
-- Table structure
CREATE TABLE example_table (
    k2 int,
    k1 int,
    k1_copy int
);

-- JSON data
{"k1": 1, "k2": 2}
```

Load command:

```
curl -v ... -H "format: json" \
    -H "jsonpaths: [\"$.k2\", \"$.k1\", \"$.k1\"]" \
    -H "columns: k2, k1, k1_copy" \
    -T example.json \
    http://<fe_host>:<fe_http_port>/api/db_name/table_name/_stream_load
```

Load result:

```
+------+------+---------+
| k2   | k1   | k1_copy |
+------+------+---------+
|    2 |    1 |       1 |
+------+------+---------+
```

Nested Field Mapping

Table structure and data:

```
-- Table structure
CREATE TABLE example_table (
    k2 int,
    k1 int,
    k1_nested1 int,
    k1_nested2 int
);

-- JSON data
{
    "k1": 1,
    "k2": 2,
    "k3": {
        "k1": 31,
        "k1_nested": {
            "k1": 32
        }
    }
}
```

Load command:

```
curl -v ... -H "format: json" \
    -H "jsonpaths: [\"$.k2\", \"$.k1\", \"$.k3.k1\", \"$.k3.k1_nested.k1\"]" \
    -H "columns: k2, k1, k1_nested1, k1_nested2" \
    -T example.json \
    http://<fe_host>:<fe_http_port>/api/db_name/table_name/_stream_load
```

Load result:

```
+------+------+------------+------------+
| k2   | k1   | k1_nested1 | k1_nested2 |
+------+------+------------+------------+
|    2 |    1 |         31 |         32 |
+------+------+------------+------------+
```

Usage Examples

This section demonstrates the usage of JSON format in different loading methods.

Stream Load

```
# Using JSON Path
curl --location-trusted -u <user>:<passwd> \
    -H "format: json" \
    -H "jsonpaths: [\"$.id\", \"$.city\"]" \
    -T example.json \
    http://<fe_host>:<fe_http_port>/api/example_db/example_table/_stream_load

# Specifying JSON root
curl --location-trusted -u <user>:<passwd> \
    -H "format: json" \
    -H "json_root: $.events" \
    -T example.json \
    http://<fe_host>:<fe_http_port>/api/example_db/example_table/_stream_load

# Reading JSON by line
curl --location-trusted -u <user>:<passwd> \
    -H "format: json" \
    -H "read_json_by_line: true" \
    -T example.json \
    http://<fe_host>:<fe_http_port>/api/example_db/example_table/_stream_load
```

Broker Load

```
-- Using JSON Path
LOAD LABEL example_db.example_label
(
    DATA INFILE("s3://bucket/path/example.json")
    INTO TABLE example_table
    FORMAT AS "json"
    PROPERTIES
    (
        "jsonpaths" = "[\"$.id\", \"$.city\"]"
    )
)
WITH S3
(
    ...
);

-- Specifying JSON root
LOAD LABEL example_db.example_label
(
    DATA INFILE("s3://bucket/path/example.json")
    INTO TABLE example_table
    FORMAT AS "json"
    PROPERTIES
    (
        "json_root" = "$.events"
    )
)
WITH S3
(
    ...
);
```

Routine Load

```
-- Using JSON Path
CREATE ROUTINE LOAD example_db.example_job ON example_table
PROPERTIES
(
    "format" = "json",
    "jsonpaths" = "[\"$.id\", \"$.city\"]"
)
FROM KAFKA
(
    ...
);
```

TVF Load

```sql – Using JSON Path INSERT INTO example_table SELECT * FROM S3 ( “path” = “s3://bucket/example.json”, “format” = “json”, “jsonpaths” = “["$.id\", \"$.city"]”, … );

– Specifying JSON root INSERT INTO example_table SELECT * FROM S3 ( “path” = “s3://bucket/example.json”, “format” = “json”, “json_root” = “$.events”, … );
```

Parquet | File Format

Parquet

This document explains how to load Parquet format data files in Doris.

Supported Loading Methods

The following loading methods support Parquet format data:

- Stream Load
- Broker Load
- INSERT INTO FROM S3 TVF
- INSERT INTO FROM HDFS TVF

Usage Examples

This section demonstrates the usage of Parquet format in different loading methods.

Stream Load

```
curl --location-trusted -u <user>:<passwd> \
    -H "format: parquet" \
    -T example.parquet \
    http://<fe_host>:<fe_http_port>/api/example_db/example_table/_stream_load
```

Broker Load

```
LOAD LABEL example_db.example_label
(
    DATA INFILE("s3://bucket/example.parquet")
    INTO TABLE example_table
    FORMAT AS "parquet"
)
WITH S3
(
    ...
);
```

TVF Load

```sql INSERT INTO example_table SELECT * FROM S3 ( “path” = “s3://bucket/example.parquet”, “format” = “parquet”, ... );
```

ORC | File Format

ORC

This document explains how to load ORC format data files in Doris.

Supported Loading Methods

The following loading methods support ORC format data:

- Stream Load
- Broker Load
- INSERT INTO FROM S3 TVF
- INSERT INTO FROM HDFS TVF

Usage Examples

This section demonstrates the usage of ORC format in different loading methods.

Stream Load

```
curl --location-trusted -u <user>:<passwd> \
    -H "format: orc" \
    -T example.orc \
    http://<fe_host>:<fe_http_port>/api/example_db/example_table/_stream_load
```

Broker Load

```
LOAD LABEL example_db.example_label
(
    DATA INFILE("s3://bucket/example.orc")
    INTO TABLE example_table
    FORMAT AS "orc"
)
WITH S3
(
    ...
);
```

TVF Load

```sql INSERT INTO example_table SELECT * FROM S3 ( “path” = “s3://bucket/example.orc”, “format” = “orc”, … );
```

ARRAY | Complex Types

ARRAY

ARRAY<T> An array of T-type items, it cannot be used as a key column.

- Before version 2.0, it was only supported in the Duplicate model table.
- Starting from version 2.0, it is supported in the non-key columns of the Unique model table.

T-type could be any of:

```
BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, DATE,
DATEV2, DATETIME, DATETIMEV2, CHAR, VARCHAR, STRING
```

CSV format import

Step 1: Prepare the data

Create the following csv file: `test_array.csv` The separator is | instead of comma to distinguish it from the comma in array.

```
1|[1,2,3,4,5]
2|[6,7,8]
3|[]
4|null
```

Step 2: Create a table in the database

```
CREATE TABLE `array_test` (
    `id`         INT           NOT NULL,
    `c_array`    ARRAY<INT>    NULL
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "column_separator:|" \
        -H "columns: id, c_array" \
        -T "test_array.csv" \
        http://localhost:8040/api/testdb/array_test/_stream_load
```

Step 4: Check the imported data

```
mysql> SELECT * FROM array_test;
+------+-----------------+
| id   | c_array         |
+------+-----------------+
|    1 | [1, 2, 3, 4, 5] |
|    2 | [6, 7, 8]       |
|    3 | []              |
|    4 | NULL            |
+------+-----------------+
4 rows in set (0.01 sec)
```

JSON format import

Step 1: Prepare the data

Create the following JSON file, `test_array.json`

```
[
    {"id":1, "c_array":[1,2,3,4,5]},
    {"id":2, "c_array":[6,7,8]},
    {"id":3, "c_array":[]},
    {"id":4, "c_array":null}
]
```

Step 2: Create a table in the database

```
CREATE TABLE `array_test` (
    `id`         INT           NOT NULL,
    `c_array`    ARRAY<INT>    NULL
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "format:json" \
        -H "columns: id, c_array" \
        -H "strip_outer_array:true" \
        -T "test_array.json" \
        http://localhost:8040/api/testdb/array_test/_stream_load
```

Step 4: Check the imported data

```
mysql> SELECT * FROM array_test;
+------+-----------------+
```

```
| id   | c_array         |
+------+-----------------+
|    1 | [1, 2, 3, 4, 5] |
|    2 | [6, 7, 8]       |
|    3 | []              |
|    4 | NULL            |
+------+-----------------+
4 rows in set (0.01 sec)
```

MAP | Complex Types

MAP

`MAP<K, V>` A Map of K, V items, it cannot be used as a key column. Now MAP can only be used in Duplicate and Unique Model Tables.

K,V could be any of:

```
BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, DECIMALV3, DATE,
DATEV2, DATETIME, DATETIMEV2, CHAR, VARCHAR, STRING
```

CSV format import

Step 1: Prepare the data

Create the following csv file: `test_map.csv` The separator is `|` instead of comma to distinguish it from the comma in map.

```
1|{"Emily":101,"age":25}
2|{"Benjamin":102}
3|{}
4|null
```

Step 2: Create a table in the database

```
CREATE TABLE map_test (
    id       INT                 NOT NULL,
    c_map    MAP<STRING, INT>    NULL
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "column_separator:|" \
        -H "columns: id, c_map" \
        -T "test_map.csv" \
        http://localhost:8040/api/testdb/map_test/_stream_load
```

Step 4: Check the imported data

```
mysql> SELECT * FROM map_test;
+------+-------------------------+
| id   | c_map                   |
+------+-------------------------+
|    1 | {"Emily":101, "age":25} |
|    2 | {"Benjamin":102}        |
|    3 | {}                      |
|    4 | NULL                    |
+------+-------------------------+
4 rows in set (0.01 sec)
```

JSON format import

Step 1: Prepare the data

Create the following JSON file, `test_map.json`

```
[
    {"id":1, "c_map":{"Emily":101, "age":25}},
    {"id":2, "c_map":{"Benjamin":102}},
    {"id":3, "c_map":{}},
    {"id":4, "c_map":null}
]
```

Step 2: Create a table in the database

```
CREATE TABLE map_test (
    id       INT                NOT NULL,
    c_map    MAP<STRING, INT>    NULL
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "format:json" \
        -H "columns: id, c_map" \
        -H "strip_outer_array:true" \
        -T "test_map.json" \
        http://localhost:8040/api/testdb/map_test/_stream_load
```

Step 4: Check the imported data

```
mysql> SELECT * FROM map_test;
+------+-------------------------+
```

```
| id   | c_map                  |
+------+------------------------+
|    1 | {"Emily":101, "age":25} |
|    2 | {"Benjamin":102}       |
|    3 | {}                     |
|    4 | NULL                   |
+------+------------------------+
4 rows in set (0.01 sec)
```

STRUCT | Complex Types

STRUCT

`STRUCT<field_name:field_type [COMMENT 'comment_string'], ... >` Represents value with structure described by multiple fields, which can be viewed as a collection of multiple columns.

- It cannot be used as a Key column. Now STRUCT can only be used in Duplicate Model Tables.
- The names and number of Fields in a Struct are fixed and always Nullable, and a Field typically consists of the following parts.
    - field_name: Identifier naming the field, non repeatable.
    - field_type: A data type.
    - COMMENT: An optional string describing the field. (currently not supported)

The currently supported types are:

```
BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, DECIMALV3, DATE,
DATEV2, DATETIME, DATETIMEV2, CHAR, VARCHAR, STRING
```

CSV format import

Step 1: Prepare the data

Create the following csv file: `test_struct.csv` The separator is `|` instead of comma to distinguish it from the comma in struct.

```
1|{10, 3.14, "Emily"}
2|{4, 1.5, null}
3|{7, null, "Benjamin"}
4|{}
5|null
```

Step 2: Create a table in the database

```
CREATE TABLE struct_test (
    id          INT                                 NOT NULL,
    c_struct    STRUCT<f1:INT,f2:FLOAT,f3:STRING>   NULL
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "column_separator:|" \
        -H "columns: id, c_struct" \
        -T "test_struct.csv" \
        http://localhost:8040/api/testdb/struct_test/_stream_load
```

Step 4: Check the imported data

```
mysql> SELECT * FROM struct_test;
+------+-------------------------------------+
| id   | c_struct                            |
+------+-------------------------------------+
|    1 | {"f1":10, "f2":3.14, "f3":"Emily"}  |
|    2 | {"f1":4, "f2":1.5, "f3":null}       |
|    3 | {"f1":7, "f2":null, "f3":"Benjamin"} |
|    4 | {"f1":null, "f2":null, "f3":null}   |
|    5 | NULL                                |
+------+-------------------------------------+
5 rows in set (0.01 sec)
```

JSON format import

Step 1: Prepare the data

Create the following JSON file, `test_struct.json`

```
[
    {"id":1, "c_struct":{"f1":10, "f2":3.14, "f3":"Emily"}},
    {"id":2, "c_struct":{"f1":4, "f2":1.5, "f3":null}},
    {"id":3, "c_struct":{"f1":7, "f2":null, "f3":"Benjamin"}},
    {"id":4, "c_struct":{}},
    {"id":5, "c_struct":null}
]
```

Step 2: Create a table in the database

```
CREATE TABLE struct_test (
    id          INT                                 NOT NULL,
    c_struct    STRUCT<f1:INT,f2:FLOAT,f3:STRING>   NULL
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "format:json" \
        -H "columns: id, c_struct" \
        -H "strip_outer_array:true" \
        -T "test_struct.json" \
        http://localhost:8040/api/testdb/struct_test/_stream_load
```

Step 4: Check the imported data

```
mysql> SELECT * FROM struct_test;
+------+--------------------------------------+
| id   | c_struct                             |
+------+--------------------------------------+
|    1 | {"f1":10, "f2":3.14, "f3":"Emily"}   |
|    2 | {"f1":4, "f2":1.5, "f3":null}        |
|    3 | {"f1":7, "f2":null, "f3":"Benjamin"} |
|    4 | {"f1":null, "f2":null, "f3":null}    |
|    5 | NULL                                 |
+------+--------------------------------------+
5 rows in set (0.00 sec)
```

JSON | Complex Types

JSON

The JSON data type stores JSON data efficiently in a binary format and allows access to its internal fields through JSON functions.

By default, it supports up to 1048576 bytes (1MB), and can be increased up to 2147483643 bytes (2GB). This can be adjusted via the string_type_length_soft_limit_bytes configuration.

Compared to storing JSON strings in a regular STRING type, the JSON type has two main advantages:

JSON format validation during data insertion. More efficient binary storage format, enabling faster access to JSON internal fields using functions like json_extract, compared to get_json_xx functions. Note: In version 1.2.x, the JSON type was named JSONB. To maintain compatibility with MySQL, it was renamed to JSON starting from version 2.0.0. Older tables can still use the previous name.

CSV format import

Step 1: Prepare the data

Create the following csv file: `test_json.csv` The separator is | instead of comma to distinguish it from the comma in json.

```
1|{"name": "tom", "age": 35}
2|{"name": null, "age": 28}
3|{"name": "michael", "age": null}
4|{"name": null, "age": null}
5|null
```

Step 2: Create a table in the database

```
CREATE TABLE json_test (
    id          INT     NOT NULL,
    c_json      JSON    NULL
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "column_separator:|" \
        -H "columns: id, c_json" \
```

```
        -T "test_json.csv" \
        http://localhost:8040/api/testdb/json_test/_stream_load
```

Step 4: Check the imported data

```
SELECT * FROM json_test;
+------+------------------------------+
| id   | c_json                       |
+------+------------------------------+
|    1 | {"name":"tom","age":35}      |
|    2 | {"name":null,"age":28}       |
|    3 | {"name":"michael","age":null} |
|    4 | {"name":null,"age":null}     |
|    5 | null                         |
+------+------------------------------+
5 rows in set (0.01 sec)
```

JSON format import

Step 1: Prepare the data

Create the following JSON file, `test_json.json`

```
[
    {"id": 1, "c_json": {"name": "tom", "age": 35}},
    {"id": 2, "c_json": {"name": null, "age": 28}},
    {"id": 3, "c_json": {"name": "michael", "age": null}},
    {"id": 4, "c_json": {"name": null, "age": null}},
    {"id": 5, "c_json": null}
]
```

Step 2: Create a table in the database

```
CREATE TABLE json_test (
    id          INT     NOT NULL,
    c_json      JSON    NULL
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

Step 3: Load data

```
curl --location-trusted \
        -u "root":"" \
        -H "format:json" \
        -H "columns: id, c_json" \
        -H "strip_outer_array:true" \
        -T "test_json.json" \
        http://localhost:8040/api/testdb/json_test/_stream_load
```

Step 4: Check the imported data

```
mysql> SELECT * FROM json_test;
+------+------------------------------+
| id   | c_json                       |
+------+------------------------------+
|    1 | {"name":"tom","age":35}      |
|    2 | {"name":null,"age":28}       |
|    3 | {"name":"michael","age":null} |
|    4 | {"name":null,"age":null}     |
|    5 | NULL                         |
+------+------------------------------+
5 rows in set (0.01 sec)
```

Bitmap

The BITMAP type can be used in Duplicate tables, Unique tables, and Aggregate tables, and can only be used as a Value type, not as a Key type. When using the BITMAP type in an Aggregate table, the table must be created with the aggregate type BITMAP_UNION. Users do not need to specify length and default values. The length is controlled by the system based on the degree of data aggregation. For more documentation, refer to Bitmap.

Usage Example

Step 1: Prepare Data

Create the following CSV file: test_bitmap.csv

```
1|koga|17723
2|nijg|146285
3|lojn|347890
4|lofn|489871
5|jfin|545679
6|kon|676724
7|nhga|767689
8|nfubg|879878
9|huang|969798
10|buag|97997
```

Step 2: Create Table in Database

```
CREATE TABLE testdb.test_bitmap(
    typ_id     BIGINT                NULL   COMMENT "ID",
    hou        VARCHAR(10)           NULL   COMMENT "one",
    arr        BITMAP  BITMAP_UNION  NOT NULL   COMMENT "two"
)
AGGREGATE KEY(typ_id,hou)
DISTRIBUTED BY HASH(typ_id,hou) BUCKETS 10;
```

Step 3: Load Data

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "column_separator:|" \
    -H "columns:typ_id,hou,arr,arr=to_bitmap(arr)" \
    -T test_bitmap.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_bitmap/_stream_load
```

Step 4: Check Loaded Data

```
mysql> select typ_id,hou,bitmap_to_string(arr) from testdb.test_bitmap;
+--------+-------+-----------------------+
| typ_id | hou   | bitmap_to_string(arr) |
```

```
+--------+-------+----------------------+
|      4 | lofn  | 489871               |
|      6 | kon   | 676724               |
|      9 | huang | 969798               |
|      3 | lojn  | 347890               |
|      8 | nfubg | 879878               |
|      7 | nhga  | 767689               |
|      1 | koga  | 17723                |
|      2 | nijg  | 146285               |
|      5 | jfin  | 545679               |
|     10 | buag  | 97997                |
+--------+-------+----------------------+
10 rows in set (0.07 sec)
```

Importing a Bitmap Containing Multiple Elements

The following demonstrates two methods for importing a bitmap column containing multiple elements using stream load. Users can choose the appropriate method based on their source file format.

bitmap_from_string

When using `bitmap_from_string`, square brackets are not allowed in the arr column of the source file. Otherwise, it will be considered a data quality error.

```
1|koga|17,723
2|nijg|146,285
3|lojn|347,890
4|lofn|489,871
5|jfin|545,679
6|kon|676,724
7|nhga|767,689
8|nfubg|879,878
9|huang|969,798
10|buag|97,997
```

Command for stream load

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "column_separator:|" \
    -H "columns:typ_id,hou,arr,arr=bitmap_from_string(arr)" \
    -T test_bitmap.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_bitmap/_stream_load
```

bitmap_from_array

When using `bitmap_from_array`, the source file can contain square brackets in the arr column. However, in stream load, the string type must first be cast to an array type before use. If the cast conversion is not applied, an error will occur due to the function signature not being found: `[ANALYSIS_ERROR]TStatus: errCode = 2, detailMessage = Does not support non-builtin functions, or function does not exist: bitmap_from_array(<slot 8>)`

```
1|koga|[17,723]
2|nijg|[146,285]
3|lojn|[347,890]
```

```
4|lofn|[489,871]
5|jfin|[545,679]
6|kon|[676,724]
7|nhga|[767,689]
8|nfubg|[879,878]
9|huang|[969,798]
10|buag|[97,997]
```

Command for stream load

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "column_separator:|" \
    -H "columns:typ_id,hou,arr_str,arr=bitmap_from_array(cast(arr_str as array<int>))" \
    -T test_bitmap.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_bitmap/_stream_load
```

HLL

HLL is used for approximate deduplication and performs better than Count Distinct when dealing with large data volumes. The load of HLL needs to be combined with functions like hll_hash. For more documentation, refer to HLL.

Usage Example

Step 1: Prepare Data

Create the following CSV file: test_hll.csv

```
1001|koga
1002|nijg
1003|lojn
1004|lofn
1005|jfin
1006|kon
1007|nhga
1008|nfubg
1009|huang
1010|buag
```

Step 2: Create Table in Database

```
CREATE TABLE testdb.test_hll(
    typ_id           BIGINT          NULL   COMMENT "ID",
    typ_name         VARCHAR(10)     NULL   COMMENT "NAME",
    pv               hll hll_union   NOT NULL   COMMENT "hll"
)
AGGREGATE KEY(typ_id,typ_name)
DISTRIBUTED BY HASH(typ_id) BUCKETS 10;
```

Step 3: Load Data

```
curl --location-trusted -u <doris_user>:<doris_password> \
    -H "column_separator:|" \
    -H "columns:typ_id,typ_name,pv=hll_hash(typ_id)" \
    -T test_hll.csv \
    -XPUT http://<fe_ip>:<fe_http_port>/api/testdb/test_hll/_stream_load
```

Step 4: Check Loaded Data

Use hll_cardinality to query:

```sql mysql> select typ_id,typ_name,hll_cardinality(pv) from testdb.test_hll; +———+———-+——————+ | typ_id | typ_name | hll_cardinality(pv) | +———+———-+——————+ | 1010 | buag | 1 | | 1002 | nijg | 1 | |
```

1001 | koga | 1 | | 1008 | nfubg | 1 | | 1005 | jfin | 1 | | 1009 | huang | 1 | | 1004 | lofn | 1 | | 1007 | nhga | 1 | | 1003 | lojn | 1 | | 1006 | kon | 1 | +——+———-+————————+ 10 rows in set (0.06 sec)

Variant

The VARIANT type can store semi-structured JSON data, allowing for the storage of complex data structures that contain different data types (such as integers, strings, booleans, etc.) without the need to predefine specific columns in the table structure. This type is particularly suitable for handling complex nested structures that may change at any time. During the writing process, the VARIANT type can automatically infer the structure and type of the columns, dynamically merging the written schema, and storing the JSON keys and their corresponding values as columns and dynamic sub-columns. For more documentation, please refer to VARIANT.

Usage Limitations

Supports CSV and JSON formats.

Loading CSV Format

Step 1: Prepare Data

Create a CSV file named `test_variant.csv` with the following content:

```
14186154924|PushEvent|
{"avatar_url":"https://avatars.githubusercontent.com/u/282080?","display_login":"brianchandotcom","gravatar_id":"","id":282080,"logi
n":"brianchandotcom","url":"https://api.github.com/users/brianchandotcom"}|{"id":1920851,"name":"brianchandotcom/liferay-
portal","url":"https://api.github.com/repos/brianchandotcom/liferay-portal"}|
{"before":"abb58cc0db673a0bd5190000d2ff9c53bb51d04d","commits":
[""],"distinct_size":4,"head":"91edd3c8c98c214155191feb852831ec535580ba","push_id":6027092734,"ref":"refs/heads/master","size":4}|1|
2020-11-14 02:00:00
```

Step 2: Create Table in Database

Execute the following SQL statement to create the table:

```
CREATE TABLE IF NOT EXISTS testdb.test_variant (
    id BIGINT NOT NULL,
    type VARCHAR(30) NULL,
    actor VARIANT NULL,
    repo VARIANT NULL,
    payload VARIANT NULL,
    public BOOLEAN NULL,
    created_at DATETIME NULL,
    INDEX idx_payload (`payload`) USING INVERTED PROPERTIES("parser" = "english") COMMENT 'inverted index for payload'
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(id) BUCKETS 10
properties("replication_num" = "1");
```

Step 3: Load Data

Using stream load as an example, use the following command to load data:

```
curl --location-trusted -u root:  -T test_variant.csv -H "column_separator:|"
http://127.0.0.1:8030/api/testdb/test_variant/_stream_load
```

Example of load results:

```
{
    "TxnId": 12,
    "Label": "96cd6250-9c78-4a9f-b8b3-2b7cef0dd606",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 1,
    "NumberLoadedRows": 1,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 660,
    "LoadTimeMs": 213,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 6,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 183,
    "ReceiveDataTimeMs": 14,
    "CommitAndPublishTimeMs": 20
}
```

Step 4: Check Loaded Data

Use the following SQL query to check the loaded data:

```
mysql> select * from testdb.test_variant\G
*************************** 1. row ***************************
        id: 14186154924
      type: PushEvent
     actor:
{"avatar_url":"https://avatars.githubusercontent.com/u/282080?","display_login":"brianchandotcom","gravatar_id":"","id":282080,"logi
n":"brianchandotcom","url":"https://api.github.com/users/brianchandotcom"}
      repo: {"id":1920851,"name":"brianchandotcom/liferay-portal","url":"https://api.github.com/repos/brianchandotcom/liferay-
portal"}
   payload: {"before":"abb58cc0db673a0bd5190000d2ff9c53bb51d04d","commits":
[""],"distinct_size":4,"head":"91edd3c8c98c214155191feb852831ec535580ba","push_id":6027092734,"ref":"refs/heads/master","size":4}
    public: 1
created_at: 2020-11-14 02:00:00
```

Loading JSON Format

Step 1: Prepare Data

Create a JSON file named `test_variant.json` with the following content:

```
{"id": "14186154924","type": "PushEvent","actor": {"id": 282080,"login":"brianchandotcom","display_login":
"brianchandotcom","gravatar_id": "","url": "https://api.github.com/users/brianchandotcom","avatar_url":
"https://avatars.githubusercontent.com/u/282080?"},"repo": {"id": 1920851,"name": "brianchandotcom/liferay-portal","url":
"https://api.github.com/repos/brianchandotcom/liferay-portal"},"payload": {"push_id": 6027092734,"size": 4,"distinct_size": 4,"ref":
```

"refs/heads/master","head": "91edd3c8c98c214155191feb852831ec535580ba","before":
"abb58cc0db673a0bd5190000d2ff9c53bb51d04d","commits": [""]},"public": true,"created_at": "2020-11-13T18:00:00Z"}

Step 2: Create Table in Database

Execute the following SQL statement to create the table:

```
CREATE TABLE IF NOT EXISTS testdb.test_variant (
    id BIGINT NOT NULL,
    type VARCHAR(30) NULL,
    actor VARIANT NULL,
    repo VARIANT NULL,
    payload VARIANT NULL,
    public BOOLEAN NULL,
    created_at DATETIME NULL,
    INDEX idx_payload (`payload`) USING INVERTED PROPERTIES("parser" = "english") COMMENT 'inverted index for payload'
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(id) BUCKETS 10;
```

Step 3: Load Data

Using stream load as an example, use the following command to load data:

```
curl --location-trusted -u root:  -T test_variant.json -H "format:json"  http://127.0.0.1:8030/api/testdb/test_variant/_stream_load
```

Example of load results:


```
{
    "TxnId": 12,
    "Label": "96cd6250-9c78-4a9f-b8b3-2b7cef0dd606",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 1,
    "NumberLoadedRows": 1,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 660,
    "LoadTimeMs": 213,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 6,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 183,
    "ReceiveDataTimeMs": 14,
    "CommitAndPublishTimeMs": 20
}
```


Step 4: Check Loaded Data

Use the following SQL query to check the loaded data:

```
mysql> select * from testdb.test_variant\G
*************************** 1. row ***************************
```

```
        id: 14186154924
      type: PushEvent
     actor:
{"avatar_url":"https://avatars.githubusercontent.com/u/282080?","display_login":"brianchandotcom","gravatar_id":"","id":282080,"logi
n":"brianchandotcom","url":"https://api.github.com/users/brianchandotcom"}
      repo: {"id":1920851,"name":"brianchandotcom/liferay-portal","url":"https://api.github.com/repos/brianchandotcom/liferay-
portal"}
   payload: {"before":"abb58cc0db673a0bd5190000d2ff9c53bb51d04d","commits":
[""],"distinct_size":4,"head":"91edd3c8c98c214155191feb852831ec535580ba","push_id":6027092734,"ref":"refs/heads/master","size":4}
    public: 1
created_at: 2020-11-14 02:00:00
```

Step 5: Check type inference

Running desc command to view schema information, sub-columns will automatically expand at the storage layer and undergo type inference.

```
mysql> desc test_variant;
+-------------------------------------------------------------+------------+------+-------+---------+-------+
| Field                                                       | Type       | Null | Key   | Default | Extra |
+-------------------------------------------------------------+------------+------+-------+---------+-------+
| id                                                          | BIGINT     | No   | true  | NULL    |       |
| type                                                        | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| actor                                                       | VARIANT    | Yes  | false | NULL    | NONE  |
| created_at                                                  | DATETIME   | Yes  | false | NULL    | NONE  |
| payload                                                     | VARIANT    | Yes  | false | NULL    | NONE  |
| public                                                      | BOOLEAN    | Yes  | false | NULL    | NONE  |
+-------------------------------------------------------------+------------+------+-------+---------+-------+
6 rows in set (0.07 sec)

mysql> set describe_extend_variant_column = true;
Query OK, 0 rows affected (0.01 sec)

mysql> desc test_variant;
+-------------------------------------------------------------+------------+------+-------+---------+-------+
| Field                                                       | Type       | Null | Key   | Default | Extra |
+-------------------------------------------------------------+------------+------+-------+---------+-------+
| id                                                          | BIGINT     | No   | true  | NULL    |       |
| type                                                        | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| actor                                                       | VARIANT    | Yes  | false | NULL    | NONE  |
| actor.avatar_url                                            | TEXT       | Yes  | false | NULL    | NONE  |
| actor.display_login                                         | TEXT       | Yes  | false | NULL    | NONE  |
| actor.id                                                    | INT        | Yes  | false | NULL    | NONE  |
| actor.login                                                 | TEXT       | Yes  | false | NULL    | NONE  |
| actor.url                                                   | TEXT       | Yes  | false | NULL    | NONE  |
| created_at                                                  | DATETIME   | Yes  | false | NULL    | NONE  |
| payload                                                     | VARIANT    | Yes  | false | NULL    | NONE  |
| payload.action                                              | TEXT       | Yes  | false | NULL    | NONE  |
| payload.before                                              | TEXT       | Yes  | false | NULL    | NONE  |
| payload.comment.author_association                          | TEXT       | Yes  | false | NULL    | NONE  |
| payload.comment.body                                        | TEXT       | Yes  | false | NULL    | NONE  |
....
+-------------------------------------------------------------+------------+------+-------+---------+-------+
406 rows in set (0.07 sec)
```

DESC can be used to specify partition and view the schema of a particular partition. The syntax is as follows:

```
DESCRIBE ${table_name} PARTITION ($partition_name);
```

Handling Data Issues

When loading data, sometimes the types of data in the source and target columns don't match. The system tries to fix these mismatches, but problems like wrong types, too long fields, or wrong precision can cause errors.

To deal with these problems, Doris has two key settings:

- Strict Mode (strict_mode): Decide if rows with errors should be removed.
- Max Filter Ratio (max_filter_ratio): Sets the highest allowed percentage of data that can be removed during loading.

This makes it easier to handle data loading problems and keeps data management strong and simple.

Strict Mode

Strict mode serves two primary purposes: 1. Filtering out data rows where column type conversion fails during load 2. Restricting updates to existing columns only in partial column update scenarios

Filtering Strategy for Column Type Conversion Failures

The system employs different strategies based on the strict mode setting:

- When strict mode is OFF: Fields that fail conversion are set to NULL, and rows containing these NULL values are loaded along with the correct data rows.
- When strict mode is ON: The system filters out rows with conversion failures and only loads correct data rows. Here, "conversion failure" specifically refers to cases where the original data is non-NULL but becomes NULL after column type conversion. Note that NULL values resulting from function calculations are not included in this type of conversion.
- Handling NULL values: Both correct and abnormal data rows may contain NULL values. If the target column is defined as NOT NULL, all rows containing NULL values will be filtered out.

1. Example with TinyInt column type:

Original Data Type	Original Data Example	Value After TinyInt Conversion	Strict Mode	Result
NULL		NULL	ON/OFF	NULL
Non-NULL	"abc" or 2000	NULL	ON	Invalid (Filtered)
Non-NULL	"abc"	NULL	OFF	NULL
Non-NULL	1	1	ON/OFF	Loaded Successfully

> **1. The column in the table allows NULL values**
>
> 2. Both `abc` and `2000` become NULL after conversion to TinyInt due to type or precision issues. When strict mode is ON, such data will be filtered out. When OFF, NULL will be loaded.

2. Example with Decimal(1,0) type:

Original Data Type	Original Data Example	Value After Decimal Conversion	Strict Mode	Result
NULL		NULL	ON/OFF	NULL
Non-NULL	aaa	NULL	ON	Invalid (Filtered)
Non-NULL	aaa	NULL	OFF	NULL
Non-NULL	1 or 10	1 or 10	ON/OFF	Loaded Successfully

> **1. The column in the table allows NULL values**
>
> 2. `abc` becomes NULL after conversion to Decimal due to type issues. When strict mode is ON, such data will be filtered out. When OFF, NULL will be loaded.
> 3. Although `10` exceeds the range, since its type meets decimal requirements, strict mode does not affect it.

Restricting Partial Column Updates to Existing Columns Only

In strict mode, each row in a partial column update must have its Key already exist in the table. In non-strict mode, partial column updates can both update existing rows (where Key exists) and insert new rows (where Key doesn't exist).

For example, given a table structure as follows:

```
CREATE TABLE user_profile
(
    id               INT,
    name             VARCHAR(10),
    age              INT,
    city             VARCHAR(10),
    balance          DECIMAL(9, 0),
    last_access_time DATETIME
) ENGINE=OLAP
UNIQUE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
    "enable_unique_key_merge_on_write" = "true"
);
```

The table contains one record as follows:

```
mysql> select * from user_profile;
+------+-------+------+----------+---------+---------------------+
| id   | name  | age  | city     | balance | last_access_time    |
+------+-------+------+----------+---------+---------------------+
|    1 | kevin |   18 | shenzhen |     400 | 2023-07-01 12:00:00|
+------+-------+------+----------+---------+---------------------+
```

When using Insert Into with strict mode to perform partial column updates, the insertion will fail because the second and third rows with keys `(3)` and `(18)` do not exist in the original table:

```
SET enable_unique_key_partial_update=true;
SET enable_insert_strict = true;
INSERT INTO user_profile (id, balance, last_access_time) VALUES
(1, 500, '2023-07-03 12:00:01'),
(3, 23, '2023-07-03 12:00:02'),
(18, 9999999, '2023-07-03 12:00:03');
ERROR 1105 (HY000): errCode = 2, detailMessage = Insert has filtered data in strict mode
```

When using Insert Into with non-strict mode to perform partial column updates:

```
SET enable_unique_key_partial_update=true;
SET enable_insert_strict = false;
INSERT INTO user_profile (id, balance, last_access_time) VALUES
(1, 500, '2023-07-03 12:00:01'),
(3, 23, '2023-07-03 12:00:02'),
(18, 9999999, '2023-07-03 12:00:03');
```

The existing record will be updated, and two new records will be inserted. For columns not specified in the insert statement, if a default value is defined, it will be used; if the column allows NULL values, NULL will be used; otherwise, the insertion will fail.

The query result is as follows:

```
mysql> select * from user_profile;
+------+-------+------+----------+---------+---------------------+
| id   | name  | age  | city     | balance | last_access_time    |
+------+-------+------+----------+---------+---------------------+
|    1 | kevin |   18 | shenzhen |     500 | 2023-07-03 12:00:01 |
|    3 | NULL  | NULL | NULL     |      23 | 2023-07-03 12:00:02 |
|   18 | NULL  | NULL | NULL     | 9999999 | 2023-07-03 12:00:03 |
+------+-------+------+----------+---------+---------------------+
```

Enable Strict Mode

Strict mode (strict_mode) defaults to False. Here's how to set it for different load methods:

Stream Load

```
curl --location-trusted -u user:passwd \
-H "strict_mode: true" \
```

```
-T data.txt \
http://host:port/api/example_db/test_table/_stream_load
```

Broker Load

```
LOAD LABEL example_db.label_1
(
    DATA INFILE("s3://bucket/data.txt")
    INTO TABLE test_table
)
WITH S3 (...)
PROPERTIES
(
    "strict_mode" = "true"
);
```

Routine Load

```
CREATE ROUTINE LOAD example_db.job1 ON test_table
PROPERTIES
(
    "strict_mode" = "true"
)
FROM KAFKA (...);
```

MySQL Load

```
LOAD DATA LOCAL INFILE 'data.txt'
INTO TABLE test_table
PROPERTIES
(
    "strict_mode" = "true"
);
```

Insert Into

```
SET enable_insert_strict = true;
INSERT INTO test_table ...;
```

Maximum Filter Ratio

Maximum Filter Ratio (max_filter_ratio) is a crucial load control parameter that defines the maximum allowable ratio of filtered data to total data during load. If the actual filter ratio is below the set maximum, the load task will continue and filtered data will be ignored; if it exceeds this ratio, the load task fails.

Filter Ratio Calculation Method

- Filtered Rows: Data filtered out due to quality issues, including type errors, precision errors, string length exceeding limits, file column count mismatches, and rows filtered due to missing corresponding partitions.
- Unselected Rows: Data rows filtered out due to Pre-filtering or Post-filtering conditions.

- Loaded Rows: Data rows successfully loaded.

The filter ratio is calculated as:

```
#Filtered Rows / (#Filtered Rows + #Loaded Rows)
```

Note that `Unselected Rows` are not included in the filter ratio calculation.

Configuring the Maximum Filter Ratio

The maximum filter ratio (max_filter_ratio) defaults to 0, meaning no filtered data is allowed. Here's how to set it for different load methods:

Stream Load

```
curl --location-trusted -u user:passwd \
-H "max_filter_ratio: 0.1" \
-T data.txt \
http://host:port/api/example_db/test_table/_stream_load
```

Broker Load

```
LOAD LABEL example_db.label_1
(
    DATA INFILE("s3://bucket/data.txt")
    INTO TABLE test_table
)
WITH S3 (...)
PROPERTIES
(
    "max_filter_ratio" = "0.1"
);
```

Routine Load

```
CREATE ROUTINE LOAD example_db.job1 ON test_table
PROPERTIES
(
    "max_filter_ratio" = "0.1"
)
FROM KAFKA (...);
```

MySQL Load

```
LOAD DATA LOCAL INFILE 'data.txt'
INTO TABLE test_table
PROPERTIES
(
    "max_filter_ratio" = "0.1"
);
```

Insert Into `sql SET insert_max_filter_ratio = 0.1; INSERT INTO test_table FROM S3/HDFS/LOCAL();`

> **For Insert Into statements,** `insert_max_filter_ratio` **only takes effect when** `enable_insert_strict = false`**, and only applies to** `INSERT INTO FROM S3/HDFS/LOCAL()` **syntax. The default value is 1.0, which means that all abnormal data are allowed to be filtered.**

Transforming Data During Load

Doris provides powerful data transformation capabilities during data loading, which can simplify data processing workflows and reduce dependency on additional ETL tools. It mainly supports four types of transformations:

- **Column Mapping**: Map source data columns to different columns in the target table.
- **Column Transformation**: Transform source data in real-time using functions and expressions.
- **Pre-filtering**: Filter out unwanted raw data before column mapping and transformation.
- **Post-filtering**: Filter the final results after column mapping and transformation.

Through these built-in data transformation functions, you can improve loading efficiency and ensure consistency in data processing logic.

Load Syntax

Stream Load

Configure data transformation by setting the following parameters in HTTP headers:

Parameter	Description
`columns`	Specify column mapping and transformation
`where`	Specify post-filtering

> **Note**: Stream Load does not support pre-filtering.

Example:

```
curl --location-trusted -u user:passwd \
    -H "columns: k1, k2, tmp_k3, k3 = tmp_k3 + 1" \
    -H "where: k1 > 1" \
    -T data.csv \
    http://<fe_ip>:<fe_http_port>/api/example_db/example_table/_stream_load
```

Broker Load

Implement data transformation in SQL statements using the following clauses:

Clause	Description
`column list`	Specify column mapping, format: `(k1, k2, tmp_k3)`
`SET`	Specify column transformation

Clause	Description
PRECEDING FILTER	Specify pre-filtering
WHERE	Specify post-filtering

Example:

```
LOAD LABEL test_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE `test_tbl`
    (k1, k2, tmp_k3)
    PRECEDING FILTER k1 = 1
    SET (
        k3 = tmp_k3 + 1
    )
    WHERE k1 > 1
)
WITH S3 (...);
```

Routine Load

Implement data transformation in SQL statements using the following clauses:

Clause	Description
COLUMNS	Specify column mapping and transformation
PRECEDING FILTER	Specify pre-filtering
WHERE	Specify post-filtering

Example:

```
CREATE ROUTINE LOAD test_db.label1 ON test_tbl
    COLUMNS(k1, k2, tmp_k3, k3 = tmp_k3 + 1),
    PRECEDING FILTER k1 = 1,
    WHERE k1 > 1
    ...
```

Insert Into

Insert Into can directly perform data transformation in the `SELECT` statement, using the `WHERE` clause for data filtering.

Column Mapping

Column mapping is used to define the correspondence between source data columns and target table columns. It can handle the following scenarios: - The order of source data columns and target table columns is inconsistent - The number of source data columns and target table columns is inconsistent

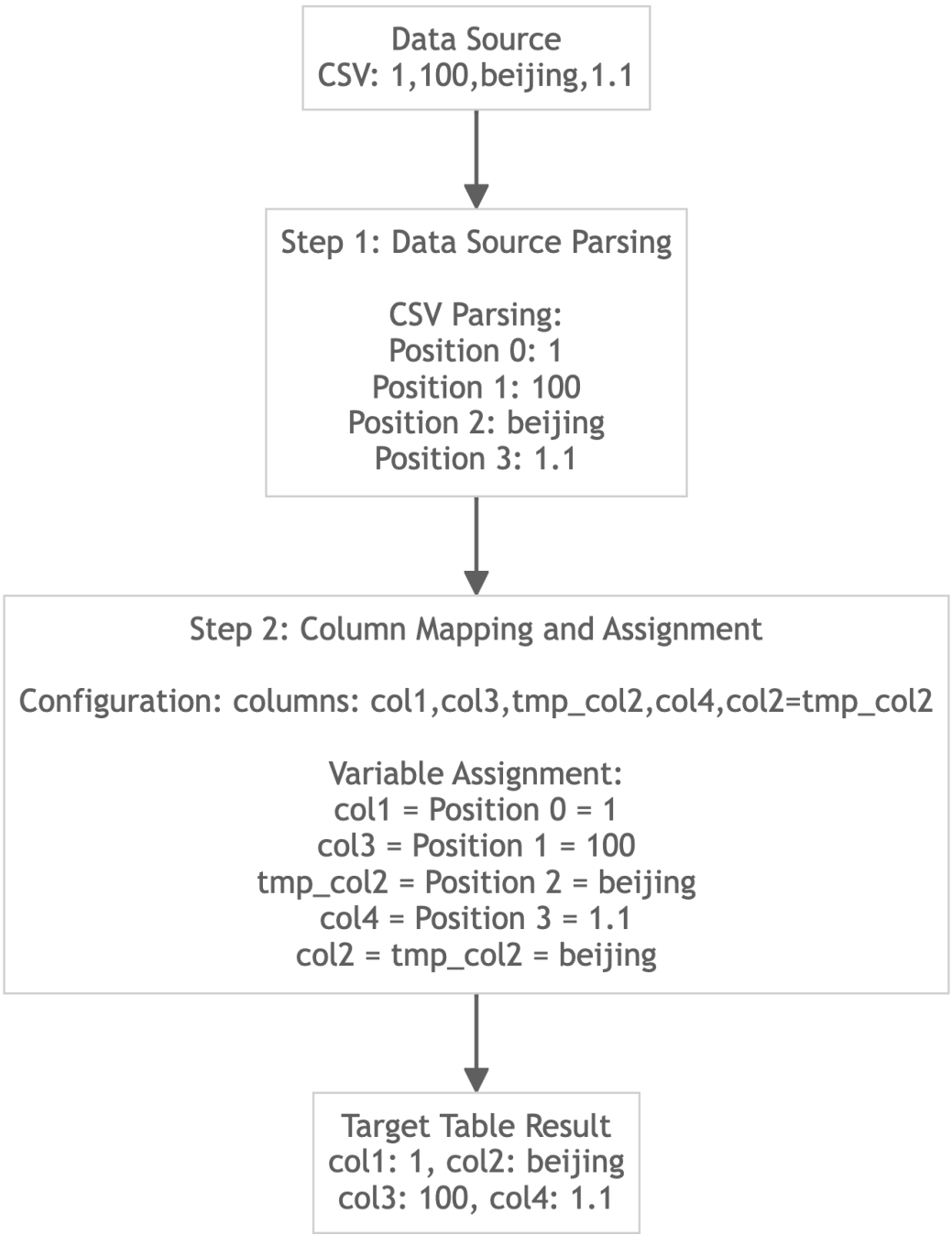
Implementation Principle

Column mapping implementation can be divided into two steps:

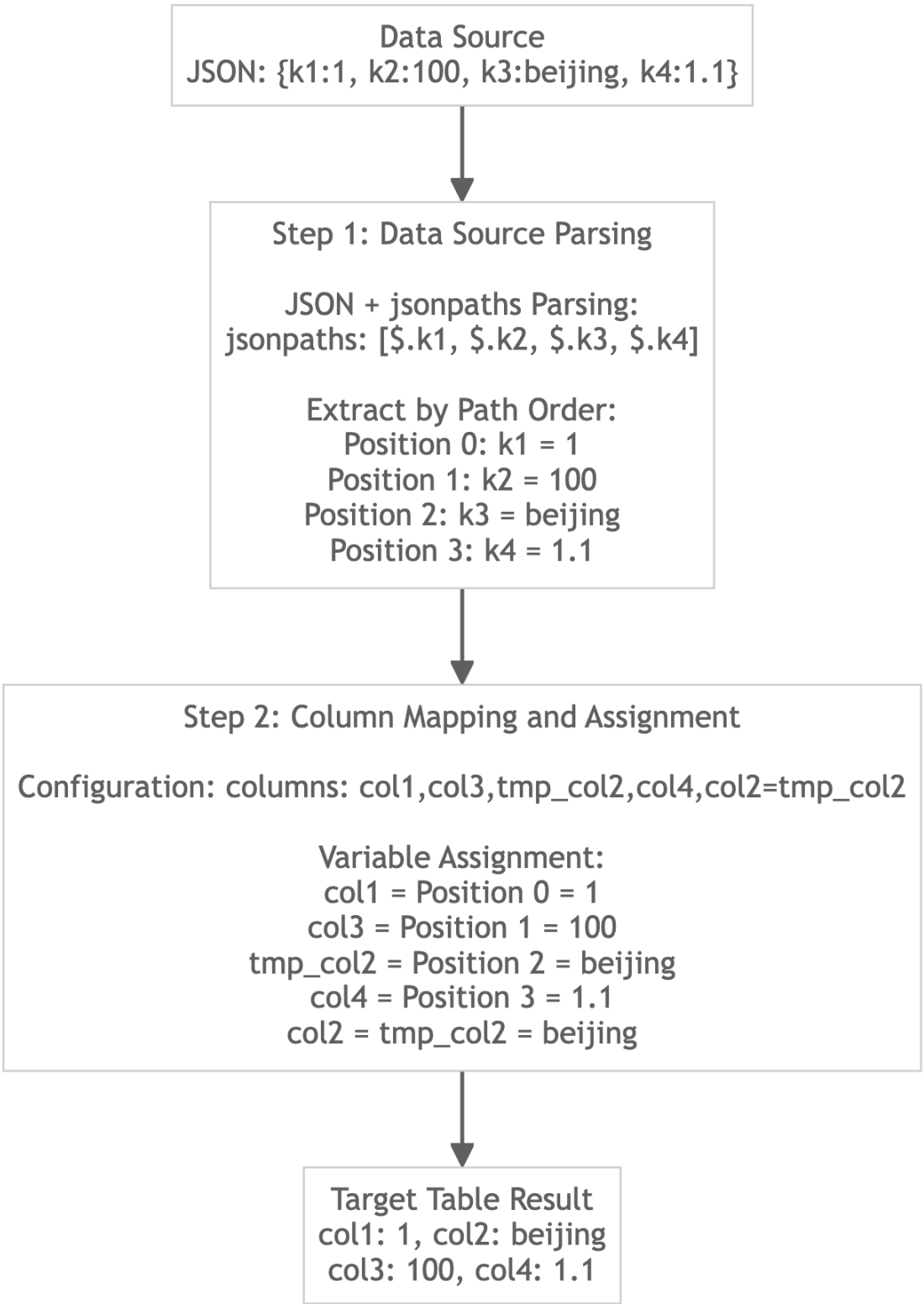
- **Step 1: Data Source Parsing** - Parse raw data into intermediate variables based on data format
- **Step 2: Column Mapping and Assignment** - Map intermediate variables to target table fields by column name

The following are processing flows for three different data formats:

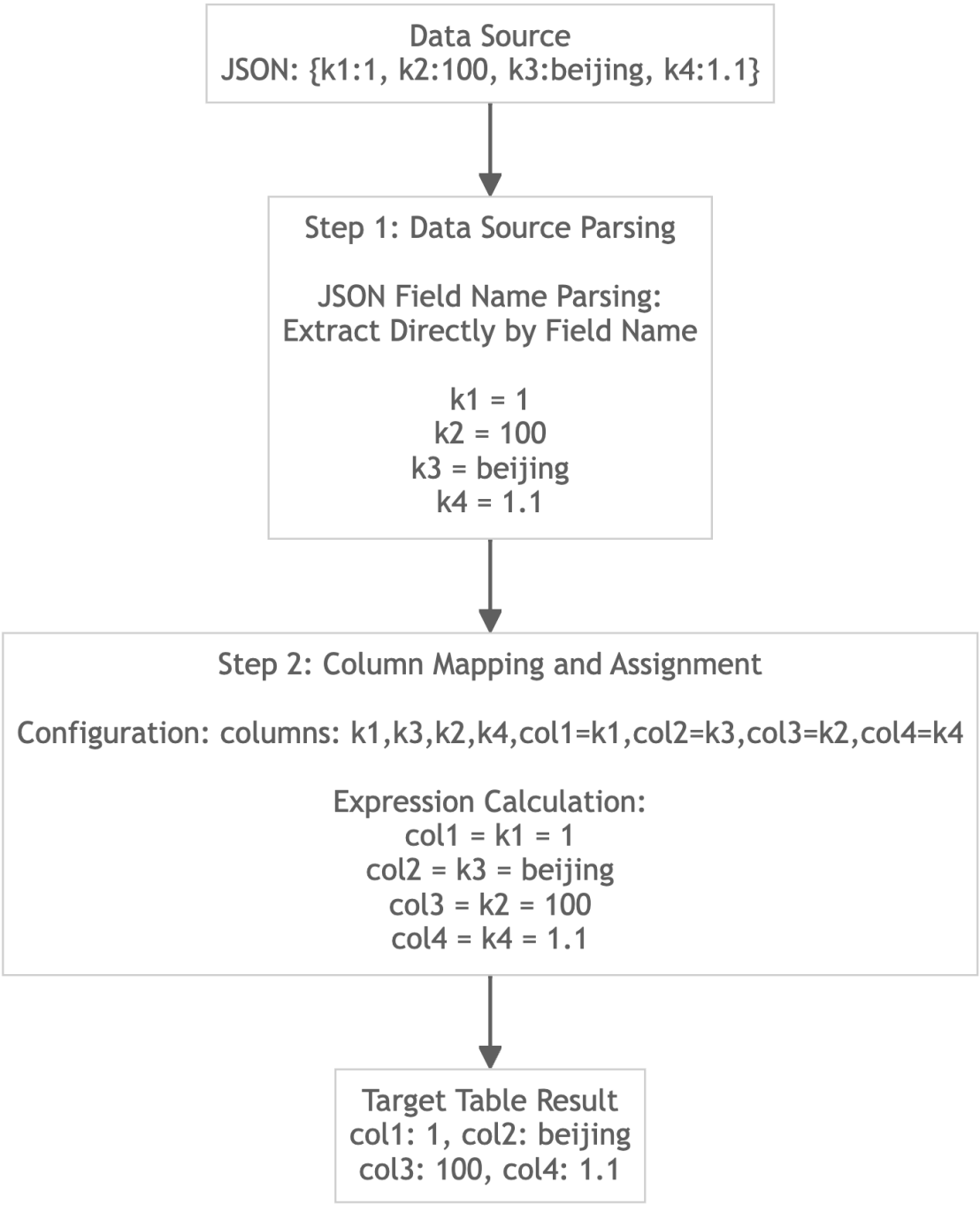
Load CSV Format Data



Load JSON Format Data with Specified jsonpaths



Load JSON Format Data without Specified jsonpaths



Load JSON Data with Specified jsonpaths

Assume the following source data (column headers are for illustration only, no actual headers exist):

```
{"k1":1,"k2":"100","k3":"beijing","k4":1.1}
{"k1":2,"k2":"200","k3":"shanghai","k4":1.2}
{"k1":3,"k2":"300","k3":"guangzhou","k4":1.3}
{"k1":4,"k2":"\\N","k3":"chongqing","k4":1.4}
```

Create Target Table

```
CREATE TABLE example_table
(
    col1 INT,
    col2 STRING,
    col3 INT,
    col4 DOUBLE
) ENGINE = OLAP
DUPLICATE KEY(col1)
DISTRIBUTED BY HASH(col1) BUCKETS 1;
```

Load Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "columns:col1, col3, col2, col4" \
    -H "jsonpaths:[\"$.k1\", \"$.k2\", \"$.k3\", \"$.k4\"]" \
    -H "format:json" \
    -H "read_json_by_line:true" \
    -T data.json \
    -X PUT \
    http://<fe_ip>:<fe_http_port>/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label_broker
(
    DATA INFILE("s3://bucket_name/data.json")
    INTO TABLE example_table
    FORMAT AS "json"
    (col1, col3, col2, col4)
    PROPERTIES
    (
        "jsonpaths" = "[\"$.k1\", \"$.k2\", \"$.k3\", \"$.k4\"]"
    )
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(col1, col3, col2, col4)
PROPERTIES
(
    "format" = "json",
    "jsonpaths" = "[\"$.k1\", \"$.k2\", \"$.k3\", \"$.k4\"]",
    "read_json_by_line" = "true"
)
FROM KAFKA (...);
```

Query Results

```
mysql> SELECT * FROM example_table;
+------+-----------+------+------+
| col1 | col2      | col3 | col4 |
+------+-----------+------+------+
|    1 | beijing   |  100 |  1.1 |
|    2 | shanghai  |  200 |  1.2 |
|    3 | guangzhou |  300 |  1.3 |
|    4 | chongqing | NULL |  1.4 |
+------+-----------+------+------+
```

Load JSON Data without Specified jsonpaths

Assume the following source data (column headers are for illustration only, no actual headers exist):

```
{"k1":1,"k2":"100","k3":"beijing","k4":1.1}
{"k1":2,"k2":"200","k3":"shanghai","k4":1.2}
{"k1":3,"k2":"300","k3":"guangzhou","k4":1.3}
{"k1":4,"k2":"\\N","k3":"chongqing","k4":1.4}
```

Create Target Table

```
CREATE TABLE example_table
(
    col1 INT,
    col2 STRING,
    col3 INT,
    col4 DOUBLE
) ENGINE = OLAP
DUPLICATE KEY(col1)
DISTRIBUTED BY HASH(col1) BUCKETS 1;
```

Load Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "columns:k1, k3, k2, k4,col1 = k1, col2 = k3, col3 = k2, col4 = k4" \
    -H "format:json" \
    -H "read_json_by_line:true" \
    -T data.json \
    -X PUT \
    http://<fe_ip>:<fe_http_port>/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label_broker
(
    DATA INFILE("s3://bucket_name/data.json")
    INTO TABLE example_table
```

```
    FORMAT AS "json"
    (k1, k3, k2, k4)
    SET (
        col1 = k1,
        col2 = k3,
        col3 = k2,
        col4 = k4
    )
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k3, k2, k4, col1 = k1, col2 = k3, col3 = k2, col4 = k4),
PROPERTIES
(
    "format" = "json",
    "read_json_by_line" = "true"
)
FROM KAFKA (...);
```

Query Results

```
mysql> SELECT * FROM example_table;
+------+-----------+------+------+
| col1 | col2      | col3 | col4 |
+------+-----------+------+------+
|    1 | beijing   |  100 |  1.1 |
|    2 | shanghai  |  200 |  1.2 |
|    3 | guangzhou |  300 |  1.3 |
|    4 | chongqing | NULL |  1.4 |
+------+-----------+------+------+
```

Adjusting Column Order

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has four columns: k1, k2, k3, and k4. We want to map the columns as follows:

```
column1 -> k1
column2 -> k3
column3 -> k2
column4 -> k4
```

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 STRING,
    k3 INT,
    k4 DOUBLE
) ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1,k3,k2,k4" \
    -T data.csv \
    -X PUT \
    http://<fe_ip>:<fe_http_port>/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label_broker
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, k3, k2, k4)
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k3, k2, k4),
COLUMNS TERMINATED BY ","
FROM KAFKA (...);
```

Query Results

```
mysql> select * from example_table;
+------+-----------+------+------+
| k1   | k2        | k3   | k4   |
+------+-----------+------+------+
|    2 | shanghai  |  200 |  1.2 |
|    4 | chongqing | NULL |  1.4 |
|    3 | guangzhou |  300 |  1.3 |
|    1 | beijing   |  100 |  1.1 |
+------+-----------+------+------+
```

Source File Columns Exceed Table Columns

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has three columns: k1, k2, and k3. We only need the first, second, and fourth columns from the source file, with the following mapping relationship:

```
column1 -> k1
column2 -> k2
column4 -> k3
```

To skip certain columns in the source file, you can use any column name that does not exist in the target table during column mapping. These column names can be customized and are not restricted. The data in these columns will be automatically ignored during loading.

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 STRING,
    k3 DOUBLE
) ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u usr:passwd \
    -H "column_separator:," \
    -H "columns: k1,k2,tmp_skip,k3" \
    -T data.csv \
    http://<fe_ip>:<fe_http_port>/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label_broker
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (tmp_k1, tmp_k2, tmp_skip, tmp_k3)
    SET (
```

```
        k1 = tmp_k1,
        k2 = tmp_k2,
        k3 = tmp_k3
    )
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k2, tmp_skip, k3),
PROPERTIES
(
    "format" = "csv",
    "column_separator" = ","
)
FROM KAFKA (...);
```

> Note: The `tmp_skip` in the example can be replaced with any name, as long as it is not in the column definition of the target table.

Query Results

```
mysql> select * from example_table;
+------+------+------+
| k1   | k2   | k3   |
+------+------+------+
|    1 | 100  |  1.1 |
|    2 | 200  |  1.2 |
|    3 | 300  |  1.3 |
|    4 | NULL |  1.4 |
+------+------+------+
```

Source File Columns Less Than Table Columns

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has five columns: k1, k2, k3, k4, and k5. We only need the first, second, third, and fourth columns from the source file, with the following mapping relationship:

```
column1 -> k1
column2 -> k3
column3 -> k2
```

```
column4 -> k4
k5 uses the default value
```

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 STRING,
    k3 INT,
    k4 DOUBLE,
    k5 INT DEFAULT 2
) ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1,k3,k2,k4" \
    -T data.csv \
    http://<fe_ip>:<fe_http_port>/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label_broker
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (tmp_k1, tmp_k3, tmp_k2, tmp_k4)
    SET (
        k1 = tmp_k1,
        k3 = tmp_k3,
        k2 = tmp_k2,
        k4 = tmp_k4
    )
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k3, k2, k4),
COLUMNS TERMINATED BY ","
FROM KAFKA (...);
```

Note: - If k5 has a default value, it will be filled with the default value - If k5 is a nullable column but has no default value, it will be filled with NULL - If k5 is a non-nullable column and has no default value, the load will fail

Query Results

```
mysql> select * from example_table;
+------+-----------+------+------+------+
| k1   | k2        | k3   | k4   | k5   |
+------+-----------+------+------+------+
|    1 | beijing   |  100 |  1.1 |    2 |
|    2 | shanghai  |  200 |  1.2 |    2 |
|    3 | guangzhou |  300 |  1.3 |    2 |
|    4 | chongqing | NULL |  1.4 |    2 |
+------+-----------+------+------+------+
```

Column Transformation

Column transformation allows users to transform column values in the source file, supporting the use of most built-in functions. Column transformation is usually defined together with column mapping, i.e., first map the columns and then transform them.

Transforming Source File Column Values Before Loading

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has four columns: k1, k2, k3, and k4. We want to transform the column values as follows:

```
column1 -> k1
column2 * 100 -> k3
column3 -> k2
column4 -> k4
```

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 STRING,
    k3 INT,
    k4 DOUBLE
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1, tmp_k3, k2, k4, k3 = tmp_k3 * 100" \
    -T data.csv \
    http://host:port/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, tmp_k3, k2, k4)
    SET (
        k3 = tmp_k3 * 100
    )
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, tmp_k3, k2, k4, k3 = tmp_k3 * 100),
COLUMNS TERMINATED BY ","
FROM KAFKA (...);
```

Query Results

```
mysql> select * from example_table;
+------+------+-------+------+
| k1   | k2   | k3    | k4   |
+------+------+-------+------+
|    1 | beijing   | 10000 |  1.1 |
|    2 | shanghai  | 20000 |  1.2 |
|    3 | guangzhou | 30000 |  1.3 |
|    4 | chongqing |  NULL |  1.4 |
+------+------+-------+------+
```

Using Case When Function for Conditional Column Transformation

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has four columns: k1, k2, k3, and k4. We want to transform the column values as follows:

```
column1 -> k1
column2 -> k2
column3 -> k3 (transformed to area id)
column4 -> k4
```

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 INT,
    k3 INT,
    k4 DOUBLE
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1, k2, tmp_k3, k4, k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN 'guangzhou' THEN 3 WHEN
'chongqing' THEN 4 ELSE NULL END" \
    -T data.csv \
    http://host:port/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, k2, tmp_k3, k4)
    SET (
        k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN 'guangzhou' THEN 3 WHEN 'chongqing' THEN 4 ELSE NULL END
    )
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k2, tmp_k3, k4, k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN 'guangzhou' THEN 3 WHEN 'chongqing'
THEN 4 ELSE NULL END),
COLUMNS TERMINATED BY ","
FROM KAFKA (...);
```

Query Results

```
mysql> select * from example_table;
+------+------+------+------+
| k1   | k2   | k3   | k4   |
+------+------+------+------+
|    1 |  100 |    1 |  1.1 |
|    2 |  200 |    2 |  1.2 |
|    3 |  300 |    3 |  1.3 |
|    4 | NULL |    4 |  1.4 |
+------+------+------+------+
```

Handling NULL Values in Source Files

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has four columns: k1, k2, k3, and k4. We want to transform the column values as follows:

```
column1 -> k1 (transform NULL to 0)
column2 -> k2
column3 -> k3
column4 -> k4
```

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 INT,
    k3 INT,
    k4 DOUBLE
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1, tmp_k2, tmp_k3, k4, k2 = ifnull(tmp_k2, 0), k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN
'guangzhou' THEN 3 WHEN 'chongqing' THEN 4 ELSE NULL END" \
```

```
    -T data.csv \
    http://host:port/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, tmp_k2, tmp_k3, k4)
    SET (
        k2 = ifnull(tmp_k2, 0),
        k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN 'guangzhou' THEN 3 WHEN 'chongqing' THEN 4 ELSE NULL END
    )
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, tmp_k2, tmp_k3, k4, k2 = ifnull(tmp_k2, 0), k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN
'guangzhou' THEN 3 WHEN 'chongqing' THEN 4 ELSE NULL END),
COLUMNS TERMINATED BY ","
FROM KAFKA (...);
```

Query Results

```
mysql> select * from example_table;
+------+------+------+------+
| k1   | k2   | k3   | k4   |
+------+------+------+------+
|    1 |  100 |    1 |  1.1 |
|    2 |  200 |    2 |  1.2 |
|    3 |  300 |    3 |  1.3 |
|    4 |    0 |    4 |  1.4 |
+------+------+------+------+
```

Pre-filtering

Pre-filtering is the process of filtering out unwanted raw data before column mapping and transformation. This feature is only supported in Broker Load and Routine Load.

Pre-filtering has the following application scenarios:

- Filtering before transformation

Scenarios where filtering is needed before column mapping and transformation, allowing for the removal of unwanted data before processing.

- Filtering columns that do not exist in the table, only as filtering indicators

For example, source data contains multiple tables' data (or multiple tables' data is written to the same Kafka message queue). Each row of data has a column indicating which table the data belongs to. Users can use

pre-filtering conditions to filter out the corresponding table data for loading.

Pre-filtering has the following limitations: - Column filtering restrictions

Pre-filtering can only filter independent simple columns in the column list and cannot filter columns with expressions. For example: when the column mapping is (a, tmp, b = tmp + 1), column b cannot be used as a filter condition.

- Data processing restrictions

Pre-filtering occurs before data transformation, using raw data values for comparison, and raw data is treated as string type. For example: for data like `\N`, it will be compared directly as the `\N` string, rather than being converted to NULL before comparison.

Example 1: Using Numeric Conditions for Pre-filtering

This example demonstrates how to filter source data using simple numeric comparison conditions. By setting the filter condition k1 > 1, we can filter out unwanted records before data transformation.

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

Pre-filtering condition:

```
column1 > 1, i.e., only load data where column1 > 1, and filter out other data.
```

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 INT,
    k3 STRING,
    k4 DOUBLE
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Broker Load

```
LOAD LABEL example_db.label1
(
```

```
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, k2, k3, k4)
    PRECEDING FILTER k1 > 1
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k2, k3, k4),
COLUMNS TERMINATED BY ","
PRECEDING FILTER k1 > 1
FROM KAFKA (...)
```

Query Results

```
mysql> select * from example_table;
+------+------+-----------+------+
| k1   | k2   | k3        | k4   |
+------+------+-----------+------+
|    2 |  200 | shanghai  |  1.2 |
|    3 |  300 | guangzhou |  1.3 |
|    4 | NULL | chongqing |  1.4 |
+------+------+-----------+------+
```

Example 2: Using Intermediate Columns to Filter Invalid Data

This example demonstrates how to handle data import scenarios containing invalid data.

Source data: `plain text 1,1 2,abc 3,3`

Table Creation

```
CREATE TABLE example_table
(
    k1 INT,
    k2 INT NOT NULL
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

For column k2, which is of type INT, `abc` is invalid dirty data. To filter this data, we can introduce an intermediate column for filtering.

Load Statements

- Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, tmp, k2 = tmp)
    PRECEDING FILTER tmp != "abc"
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, tmp, k2 = tmp),
COLUMNS TERMINATED BY ","
PRECEDING FILTER tmp != "abc"
FROM KAFKA (...);
```

Load Results

```
mysql> select * from example_table;
+------+----+
| k1   | k2 |
+------+----+
|    1 |  1 |
|    3 |  3 |
+------+----+
```

Post-filtering

Post-filtering is the process of filtering the final results after column mapping and transformation.

Filtering Without Column Mapping and Transformation

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has four columns: k1, k2, k3, and k4. We want to load only the data where the fourth column is greater than 1.2.

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
```

```
    k2 INT,
    k3 STRING,
    k4 DOUBLE
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1, k2, k3, k4" \
    -H "where: k4 > 1.2" \
    -T data.csv \
    http://host:port/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, k2, k3, k4)
    where k4 > 1.2
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k2, k3, k4),
COLUMNS TERMINATED BY ","
WHERE k4 > 1.2;
FROM KAFKA (...)
```

Query Results

```
mysql> select * from example_table;
+------+------+-----------+------+
| k1   | k2   | k3        | k4   |
+------+------+-----------+------+
|    3 |  300 | guangzhou |  1.3 |
|    4 | NULL | chongqing |  1.4 |
+------+------+-----------+------+
```

Filtering Transformed Data

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has four columns: k1, k2, k3, and k4. We want to transform the column values as follows:

```
column1 -> k1
column2 -> k2
column3 -> k3 (transformed to area id)
column4 -> k4
```

We want to filter out the data where the transformed k3 value is 3.

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 INT,
    k3 INT,
    k4 DOUBLE
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1, k2, tmp_k3, k4, k3 = case tmp_k3 when 'beijing' then 1 when 'shanghai' then 2 when 'guangzhou' then 3 when
'chongqing' then 4 else null end" \
    -H "where: k3 != 3" \
    -T data.csv \
    http://host:port/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, k2, tmp_k3, k4)
    SET (
        k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN 'guangzhou' THEN 3 WHEN 'chongqing' THEN 4 ELSE NULL END
    )
    WHERE k3 != 3
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k2, tmp_k3, k4),
COLUMNS TERMINATED BY ","
SET (
    k3 = CASE tmp_k3 WHEN 'beijing' THEN 1 WHEN 'shanghai' THEN 2 WHEN 'guangzhou' THEN 3 WHEN 'chongqing' THEN 4 ELSE NULL END
)
WHERE k3 != 3;
FROM KAFKA (...)
```

Query Results

```
mysql> select * from example_table;
+------+------+------+------+
| k1   | k2   | k3   | k4   |
+------+------+------+------+
|    1 |  100 |    1 |  1.1 |
|    2 |  200 |    2 |  1.2 |
|    4 | NULL |    4 |  1.4 |
+------+------+------+------+
```

Multiple Conditions Filtering

Suppose we have the following source data (column names are for illustration purposes only, and there is no actual header):

```
column1,column2,column3,column4
1,100,beijing,1.1
2,200,shanghai,1.2
3,300,guangzhou,1.3
4,\N,chongqing,1.4
```

The target table has four columns: k1, k2, k3, and k4. We want to filter out the data where k1 is NULL and k4 is less than 1.2.

Creating the Target Table

```
CREATE TABLE example_table
(
    k1 INT,
    k2 INT,
    k3 STRING,
    k4 DOUBLE
)
ENGINE = OLAP
DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS 1;
```

Loading Data

- Stream Load

```
curl --location-trusted -u user:passwd \
    -H "column_separator:," \
    -H "columns: k1, k2, k3, k4" \
    -H "where: k1 is not null and k4 > 1.2" \
    -T data.csv \
    http://host:port/api/example_db/example_table/_stream_load
```

- Broker Load

```
LOAD LABEL example_db.label1
(
    DATA INFILE("s3://bucket_name/data.csv")
    INTO TABLE example_table
    COLUMNS TERMINATED BY ","
    (k1, k2, k3, k4)
    where k1 is not null and k4 > 1.2
)
WITH s3 (...);
```

- Routine Load

```
CREATE ROUTINE LOAD example_db.example_routine_load ON example_table
COLUMNS(k1, k2, k3, k4),
COLUMNS TERMINATED BY ","
WHERE k1 is not null and k4 > 1.2
FROM KAFKA (...);
```

Query Results

```
mysql> select * from example_table;
+------+------+-----------+------+
| k1   | k2   | k3        | k4   |
+------+------+-----------+------+
|    3 |  300 | guangzhou |  1.3 |
|    4 | NULL | chongqing |  1.4 |
+------+------+-----------+------+
```

Load High Availability

Load High Availability

Overview

Doris provides various mechanisms to ensure high availability during data import. This article will detail Doris's default import behavior and additional options for improving import availability, especially the minimum write replica number feature.

Majority Write

By default, Doris adopts a majority write strategy to ensure data reliability and consistency:

- An import is considered successful when the number of successfully written replicas exceeds half of the total number of replicas.
- For example, for a table with three replicas, at least two replicas must be successfully written for the import to be considered successful.

How It Works

1. Data Distribution: The import task first distributes data to all relevant BE nodes.
2. Parallel Writing: Each BE node processes data writing operations in parallel.
3. Write Confirmation: After completing the data write, each BE node sends a confirmation to the FE.
4. Majority Judgment: The FE counts the number of successfully written replicas, and considers the import successful when a majority is reached.
5. Transaction Commit: The FE commits the import transaction, making the data visible externally.
6. Asynchronous Replication: For replicas that were not successfully written, the system will asynchronously replicate data in the background to ensure eventual consistency across all replicas.

The majority write strategy is Doris's balance between data reliability and system availability. For scenarios with special requirements, Doris provides other options such as the minimum write replica number to further enhance system flexibility.

Minimum Write Replica Number

While the majority write strategy ensures data reliability, it may affect system availability in certain scenarios. For example, in a two-replica situation, both replicas must be successfully written to complete the import, meaning no replica is allowed to be unavailable during the import process.

To address this issue and improve import availability, Doris provides the Min Load Replica Num option.

Feature Description

The minimum write replica number allows users to specify the minimum number of replicas that need to be successfully written during data import. The import is considered successful when the number of successfully written replicas is greater than or equal to this value.

Use Cases

- When some nodes are unavailable, but data import still needs to be guaranteed.
- When there are high requirements for data import speed, and users are willing to sacrifice some reliability for higher availability.

Configuration Methods

1. Single Table Configuration

a. Set when creating a table:

```
CREATE TABLE example_table
(
id INT,
name STRING
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 10
PROPERTIES
(
'replication_num' = '3',
'min_load_replica_num' = '2'
);
```

b. Modify an existing table:

```
ALTER TABLE example_table
SET ( 'min_load_replica_num' = '2' );
```

2. Global Configuration

Set through the FE configuration item `min_load_replica_num`.

- Valid values: greater than 0
- Default value: -1 (indicating that the global minimum write replica number is not enabled)

Priority: Table property > Global configuration > Default majority rule

If the table property is not set or invalid, and the global configuration is valid, the minimum write replica number for the table is: `min(FE configured min_load_replica_num, table's replica number/2 + 1)`

For viewing and modifying FE configuration items, please refer to the FE Configuration Document.

Other High Availability Mechanisms

In addition to the minimum write replica number option, Doris also adopts the following mechanisms to improve import availability:

1. Import Retry: Automatically retry failed import tasks caused by temporary failures.
2. Load Balancing: Distribute import tasks to different BE nodes to avoid excessive pressure on a single point.
3. Transaction Mechanism: Ensure data consistency, automatically rollback in case of failure.

High Concurrency LOAD Optimization(Group Commit)

In high-frequency small batch write scenarios, traditional loading methods have the following issues:

- Each load creates an independent transaction, requiring FE to parse SQL and generate execution plans, affecting overall performance
- Each load generates a new version, causing rapid version growth and increasing background compaction pressure

To solve these problems, Doris introduced the Group Commit mechanism. Group Commit is not a new loading method, but rather an optimization extension of existing loading methods, mainly targeting:

- `INSERT INTO tbl VALUES(...)` statements
- Stream Load

By merging multiple small batch loads into one large transaction commit in the background, it significantly improves high-concurrency small batch write performance. Additionally, using Group Commit with PreparedStatement can achieve even higher performance improvements.

Group Commit Modes

Group Commit has three modes:

- Off Mode (`off_mode`)

  Group Commit is disabled.

- Synchronous Mode (`sync_mode`)

  Doris commits multiple loads in one transaction based on load and table's `group_commit_interval` property, returning after transaction commit. This is suitable for high-concurrency write scenarios requiring immediate data visibility after loading.

- Asynchronous Mode (`async_mode`)

  Doris first writes data to WAL (Write Ahead Log), then returns immediately. Doris asynchronously commits data based on load and table's `group_commit_interval` property, making data visible after commit. To prevent WAL from occupying too much disk space, it automatically switches to `sync_mode` for large single loads. This is suitable for write latency-sensitive and high-frequency write scenarios.

  WAL count can be viewed through FE http interface as shown here, or by searching for `wal` keyword in BE metrics.

How to Use Group Commit

Assuming the table structure is:

```
CREATE TABLE `dt` (
    `id` int(11) NOT NULL,
    `name` varchar(50) NULL,
    `score` int(11) NULL
) ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
    "replication_num" = "1"
);
```

Using JDBC

When users write using JDBC's `insert into values` method, to reduce SQL parsing and planning overhead, we support MySQL protocol's `PreparedStatement` feature on the FE side. When using `PreparedStatement`, SQL and its load plan are cached in session-level memory cache, and subsequent loads directly use the cached objects, reducing FE CPU pressure. Here's an example of using `PreparedStatement` in JDBC:

1. Set JDBC URL and enable Prepared Statement on Server side

```
url = jdbc:mysql://127.0.0.1:9030/db?
useServerPrepStmts=true&useLocalSessionState=true&rewriteBatchedStatements=true&cachePrepStmts=true&prepStmtCacheSqlLimit=99999&prep
StmtCacheSize=500
```

2. Configure `group_commit` session variable in one of two ways:

- Through JDBC url by adding `sessionVariables=group_commit=async_mode`

```
url = jdbc:mysql://127.0.0.1:9030/db?
useServerPrepStmts=true&useLocalSessionState=true&rewriteBatchedStatements=true&cachePrepStmts=true&prepStmtCacheSqlLimit=99999&prep
StmtCacheSize=500&sessionVariables=group_commit=async_mode
```

- Through SQL execution

```
try (Statement statement = conn.createStatement()) {
    statement.execute("SET group_commit = async_mode;");
}
```

3. Use `PreparedStatement`

```
private static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
private static final String URL_PATTERN = "jdbc:mysql://%s:%d/%s?
useServerPrepStmts=true&useLocalSessionState=true&rewriteBatchedStatements=true&cachePrepStmts=true&prepStmtCacheSqlLimit=99999&prep
StmtCacheSize=50$sessionVariables=group_commit=async_mode";
private static final String HOST = "127.0.0.1";
private static final int PORT = 9030;
private static final String DB = "db";
private static final String TBL = "dt";
private static final String USER = "root";
private static final String PASSWD = "";
private static final int INSERT_BATCH_SIZE = 10;

private static void groupCommitInsertBatch() throws Exception {
    Class.forName(JDBC_DRIVER);
    // add rewriteBatchedStatements=true and cachePrepStmts=true in JDBC url
```

```
    // set session variables by sessionVariables=group_commit=async_mode in JDBC url
    try (Connection conn = DriverManager.getConnection(
            String.format(URL_PATTERN, HOST, PORT, DB), USER, PASSWD)) {

        String query = "insert into " + TBL + " values(?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(query)) {
            for (int j = 0; j < 5; j++) {
                // 10 rows per insert
                for (int i = 0; i < INSERT_BATCH_SIZE; i++) {
                    stmt.setInt(1, i);
                    stmt.setString(2, "name" + i);
                    stmt.setInt(3, i + 10);
                    stmt.addBatch();
                }
                int[] result = stmt.executeBatch();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Note: Since high-frequency insert into statements will print large amounts of audit logs affecting final performance, printing prepared statement audit logs is disabled by default. You can control whether to print prepared statement audit logs through session variable settings.

```
# Configure session variable to enable printing prepared statement audit log, default is false
set enable_prepared_stmt_audit_log=true;
```

For more about **JDBC** usage, refer to Using Insert Method to Synchronize Data.

Using Golang for Group Commit

Since Golang has limited support for prepared statements, we can improve Group Commit performance through manual client-side batching. Here's a sample program:

```
package main

import (
    "database/sql"
    "fmt"
    "math/rand"
    "strings"
    "sync"
    "sync/atomic"
    "time"

    _ "github.com/go-sql-driver/mysql"
)

const (
    host     = "127.0.0.1"
    port     = 9030
    db       = "test"
    user     = "root"
    password = ""
    table    = "async_lineitem"
)
```

```
var (
    threadCount = 20
    batchSize   = 100
)

var totalInsertedRows int64
var rowsInsertedLastSecond int64

func main() {
    dbDSN := fmt.Sprintf("%s:%s@tcp(%s:%d)/%s?parseTime=true", user, password, host, port, db)
    db, err := sql.Open("mysql", dbDSN)
    if err != nil {
        fmt.Printf("Error opening database: %s\n", err)
        return
    }
    defer db.Close()

    var wg sync.WaitGroup
    for i := 0; i < threadCount; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            groupCommitInsertBatch(db)
        }()
    }

    go logInsertStatistics()

    wg.Wait()
}

func groupCommitInsertBatch(db *sql.DB) {
    for {
        valueStrings := make([]string, 0, batchSize)
        valueArgs := make([]interface{}, 0, batchSize*16)
        for i := 0; i < batchSize; i++ {
            valueStrings = append(valueStrings, "(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)")
            valueArgs = append(valueArgs, rand.Intn(1000))
            valueArgs = append(valueArgs, rand.Intn(1000))
            valueArgs = append(valueArgs, rand.Intn(1000))
            valueArgs = append(valueArgs, rand.Intn(1000))
            valueArgs = append(valueArgs, sql.NullFloat64{Float64: 1.0, Valid: true})
            valueArgs = append(valueArgs, sql.NullFloat64{Float64: 1.0, Valid: true})
            valueArgs = append(valueArgs, sql.NullFloat64{Float64: 1.0, Valid: true})
            valueArgs = append(valueArgs, sql.NullFloat64{Float64: 1.0, Valid: true})
            valueArgs = append(valueArgs, "N")
            valueArgs = append(valueArgs, "O")
            valueArgs = append(valueArgs, time.Now())
            valueArgs = append(valueArgs, time.Now())
            valueArgs = append(valueArgs, time.Now())
            valueArgs = append(valueArgs, "DELIVER IN PERSON")
            valueArgs = append(valueArgs, "SHIP")
            valueArgs = append(valueArgs, "N/A")
        }
        stmt := fmt.Sprintf("INSERT INTO %s VALUES %s",
            table, strings.Join(valueStrings, ","))
        _, err := db.Exec(stmt, valueArgs...)
        if err != nil {
            fmt.Printf("Error executing batch: %s\n", err)
            return
        }
        atomic.AddInt64(&rowsInsertedLastSecond, int64(batchSize))
        atomic.AddInt64(&totalInsertedRows, int64(batchSize))
    }
}
```

```
func logInsertStatistics() {
    for {
        time.Sleep(1 * time.Second)
        fmt.Printf("Total inserted rows: %d\n", totalInsertedRows)
        fmt.Printf("Rows inserted in the last second: %d\n", rowsInsertedLastSecond)
        rowsInsertedLastSecond = 0
    }
}
```

INSERT INTO VALUES

- Asynchronous Mode

```
# Configure session variable to enable group commit (default is off_mode), enable asynchronous mode
mysql> set group_commit = async_mode;

# The returned label is prefixed with group_commit, indicating whether group commit is used
mysql> insert into dt values(1, 'Bob', 90), (2, 'Alice', 99);
Query OK, 2 rows affected (0.05 sec)
{'label':'group_commit_a145ce07f1c972fc-bd2c54597052a9ad', 'status':'PREPARE', 'txnId':'181508'}

# The label, txn_id, and previous one are the same, indicating that they are accumulated into the same import task
mysql> insert into dt(id, name) values(3, 'John');
Query OK, 1 row affected (0.01 sec)
{'label':'group_commit_a145ce07f1c972fc-bd2c54597052a9ad', 'status':'PREPARE', 'txnId':'181508'}

# Cannot query immediately
mysql> select * from dt;
Empty set (0.01 sec)

# 10 seconds later, data can be queried, and data visibility delay can be controlled by table attribute group_commit_interval.
mysql> select * from dt;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|    1 | Bob   |    90 |
|    2 | Alice |    99 |
|    3 | John  |  NULL |
+------+-------+-------+
3 rows in set (0.02 sec)
```

- Synchronous Mode

```
# Configure session variable to enable group commit (default is off_mode), enable synchronous mode
mysql> set group_commit = sync_mode;

# The returned label is prefixed with group_commit, indicating whether group commit is used, and import time is at least table
attribute group_commit_interval.
mysql> insert into dt values(4, 'Bob', 90), (5, 'Alice', 99);
Query OK, 2 rows affected (10.06 sec)
{'label':'group_commit_d84ab96c09b60587_ec455a33cb0e9e87', 'status':'PREPARE', 'txnId':'3007', 'query_id':'fc6b94085d704a94-
a69bfc9a202e66e2'}

# Data can be read immediately
mysql> select * from dt;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|    1 | Bob   |    90 |
```

```
|    2 | Alice |    99 |
|    3 | John  |  NULL |
|    4 | Bob   |    90 |
|    5 | Alice |    99 |
+------+-------+-------+
5 rows in set (0.03 sec)
```

- Off Mode

```
mysql> set group_commit = off_mode;
```

Stream Load

Assuming `data.csv` contains:

```
6,Amy,60
7,Ross,98
```

- Asynchronous Mode

```
# Import with "group_commit:async_mode" configuration in header

curl --location-trusted -u {user}:{passwd} -T data.csv -H "group_commit:async_mode"  -H "column_separator:,"  http://{fe_host}:
{http_port}/api/db/dt/_stream_load
{
    "TxnId": 7009,
    "Label": "group_commit_c84d2099208436ab_96e33fda01eddba8",
    "Comment": "",
    "GroupCommit": true,
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 2,
    "NumberLoadedRows": 2,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 19,
    "LoadTimeMs": 35,
    "StreamLoadPutTimeMs": 5,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 26
}

# The returned GroupCommit is true, indicating that the group commit process is entered
# The returned Label is prefixed with group_commit, indicating the label associated with the import that truly consumes data
```

- Synchronous Mode

```
# Import with "group_commit:sync_mode" configuration in header

curl --location-trusted -u {user}:{passwd} -T data.csv -H "group_commit:sync_mode"  -H "column_separator:,"  http://{fe_host}:
{http_port}/api/db/dt/_stream_load
{
    "TxnId": 3009,
    "Label": "group_commit_d941bf17f6efcc80_ccf4afdde9881293",
    "Comment": "",
    "GroupCommit": true,
    "Status": "Success",
```

```
    "Message": "OK",
    "NumberTotalRows": 2,
    "NumberLoadedRows": 2,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 19,
    "LoadTimeMs": 10044,
    "StreamLoadPutTimeMs": 4,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 10038
}

# The returned GroupCommit is true, indicating that the group commit process is entered
# The returned Label is prefixed with group_commit, indicating the label associated with the import that truly consumes data
```

About Stream Load usage, please refer to Stream Load.

Data is automatically committed when either time interval (default 10 seconds) or data volume (default 64 MB) condition is met. These parameters should be used together and tuned based on actual scenarios.

Modifying Commit Interval

Default commit interval is 10 seconds, users can adjust through table configuration:

```
# Modify commit interval to 2 seconds
ALTER TABLE dt SET ("group_commit_interval_ms" = "2000");
```

Parameter Adjustment Recommendations: - Shorter intervals (e.g., 2 seconds): - Pros: Lower data visibility latency, suitable for scenarios requiring high real-time performance - Cons: More commits, faster version growth, higher background compaction pressure

- Longer intervals (e.g., 30 seconds):
    - Pros: Larger commit batches, slower version growth, lower system overhead
    - Cons: Higher data visibility latency

Recommend setting based on business tolerance for data visibility delay. If system pressure is high, consider increasing the interval.

Modifying Commit Data Volume

Group Commit's default commit data volume is 64 MB, users can adjust through table configuration:

```
# Modify commit data volume to 128MB
ALTER TABLE dt SET ("group_commit_data_bytes" = "134217728");
```

Parameter Adjustment Recommendations: - Smaller threshold (e.g., 32MB): - Pros: Less memory usage, suitable for resource-constrained environments - Cons: Smaller commit batches, potentially limited throughput

- Larger threshold (e.g., 256MB):
    - Pros: Higher batch commit efficiency, greater system throughput
    - Cons: Uses more memory

Recommend balancing based on system memory resources and data reliability requirements. If memory is sufficient and higher throughput is desired, consider increasing to 128MB or more.

BE Configuration

1. `group_commit_wal_path`
    - Description: Directory for storing group commit WAL files
    - Default: Creates a `wal` directory under each configured `storage_root_path`. Configuration example:

```
group_commit_wal_path=/data1/storage/wal;/data2/storage/wal;/data3/storage/wal
```

Usage Limitations

- **Group Commit Limitations**
    - `INSERT INTO VALUES` statements degrade to non-Group Commit mode in these cases:
        - Transaction writes (`Begin; INSERT INTO VALUES; COMMIT`)
        - Specified Label (`INSERT INTO dt WITH LABEL {label} VALUES`)
        - VALUES containing expressions (`INSERT INTO dt VALUES (1 + 100)`)
        - Column update writes
        - Table doesn't support lightweight mode changes
    - `Stream Load` degrades to non-Group Commit mode in these cases:
        - Using two-phase commit
        - Specified Label (`-H "label:my_label"`)
        - Column update writes
        - Table doesn't support lightweight mode changes
- **Unique Model**
    - Group Commit doesn't guarantee commit order, recommend using Sequence column to ensure data consistency.
- **WAL Limitations**
    - `async_mode` writes data to WAL, deletes after success, recovers through WAL on failure.
    - WAL files are stored with a single replica on one BE, disk damage or accidental file deletion may cause data loss.
    - When offlining BE nodes, use `DECOMMISSION` command to prevent data loss.
    - `async_mode` switches to `sync_mode` in these cases:
        - Load data volume too large (exceeds 80% of WAL single directory space)
        - Unknown data volume chunked stream load
        - Insufficient disk space
    - During heavyweight Schema Change, Group Commit writes are rejected, client needs to retry.

Performance

We have separately tested the write performance of group commit in high-concurrency scenarios with small data volumes using `Stream Load` and `JDBC` (in `async mode`).

Stream Load

Environment

- 1 Front End (FE) server: Alibaba Cloud with 8-core CPU, 16GB RAM, and one 100GB ESSD PL1 SSD.
- 3 Backend (BE) servers: Alibaba Cloud with 16-core CPU, 64GB RAM, and one 1TB ESSD PL1 SSD.
- 1 Test Client: Alibaba Cloud with 16-core CPU, 64GB RAM, and one 100GB ESSD PL1 SSD.
- The version for testing is Doris-2.1.5.

DataSet

- `httplogs`, 31 GB, 247249096 (247 million) rows

Test Tool

- doris-streamloader

Test Method

- Testing with different data sizes per request and concurrency levels between `non group_commit` and `group_commit=async mode` modes.

Test Results

Load Way	Single-concurrency Data Size	Concurrency	Cost Seconds	Rows / Seconds	MB / Seconds
group_commit	10 KB	10	3306	74,787	9.8
group_commit	10 KB	30	3264	75,750	10.0
group_commit	100 KB	10	424	582,447	76.7
group_commit	100 KB	30	366	675,543	89.0
group_commit	500 KB	10	187	1,318,661	173.7
group_commit	500 KB	30	183	1,351,087	178.0
group_commit	1 MB	10	178	1,385,148	182.5
group_commit	1 MB	30	178	1,385,148	182.5
group_commit	10 MB	10	177	1,396,887	184.0

Load Way	Single-concurrency Data Size	Concurrency	Cost Seconds	Rows / Seconds	MB / Seconds
non group_commit	1 MB	10	2824	87,536	11.5
non group_commit	10 MB	10	450	549,442	68.9
non group_commit	10 MB	30	177	1,396,887	184.0

In the above test, the CPU usage of BE fluctuates between 10-40%.

The `group_commit` effectively enhances load performance while reducing the number of versions, thereby alleviating the pressure on compaction.

JDBC

Environment

1 Front End (FE) server: Alibaba Cloud with an 8-core CPU, 16GB RAM, and one 100GB ESSD PL1 SSD.

1 Backend (BE) server: Alibaba Cloud with a 16-core CPU, 64GB RAM, and one 500GB ESSD PL1 SSD.

1 Test Client: Alibaba Cloud with a 16-core CPU, 64GB RAM, and one 100GB ESSD PL1 SSD.

The testing version is Doris-2.1.5.

Disable the printing of prepared statement audit logs to enhance performance.

DataSet

- The data of tpch sf10 `lineitem` table, 20 files, 14 GB, 120 million rows

Test Method

- DataX

Test Method

- Use `txtfilereader` wtite data to `mysqlwriter`, config different concurrennncy and rows for per `INSERT` sql.

Test Results

Rows per insert	Concurrency	Rows / Second	MB / Second
100	10	107,172	11.47
100	20	140,317	14.79
100	30	142,882	15.28

In the above test, the CPU usage of BE fluctuates between 10-20%, FE fluctuates between 60-70%.

Insert into Sync Mode Small Batch Data

Machine Configuration

- 1 Front-End (FE): Alibaba Cloud, 16-core CPU, 64GB RAM, 1 x 500GB ESSD PL1 cloud disk
- 5 Back-End (BE) nodes: Alibaba Cloud, 16-core CPU, 64GB RAM, 1 x 1TB ESSD PL1 cloud disk.
- 1 Testing Client: Alibaba Cloud, 16-core CPU, 64GB RAM, 1 x 100GB ESSD PL1 cloud disk
- Test version: Doris-2.1.5

Dataset

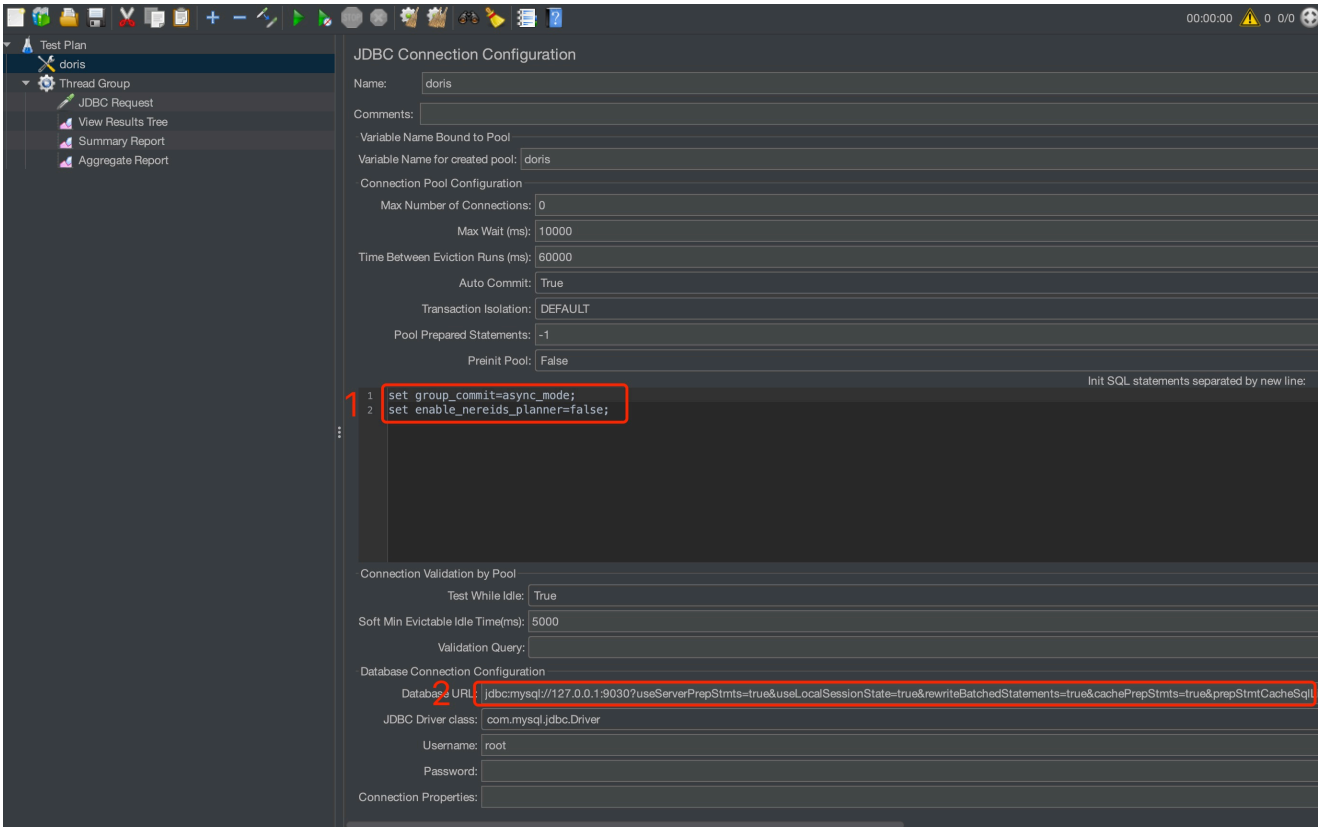
- The data of tpch sf10 `lineitem` table.
- The create table statement is

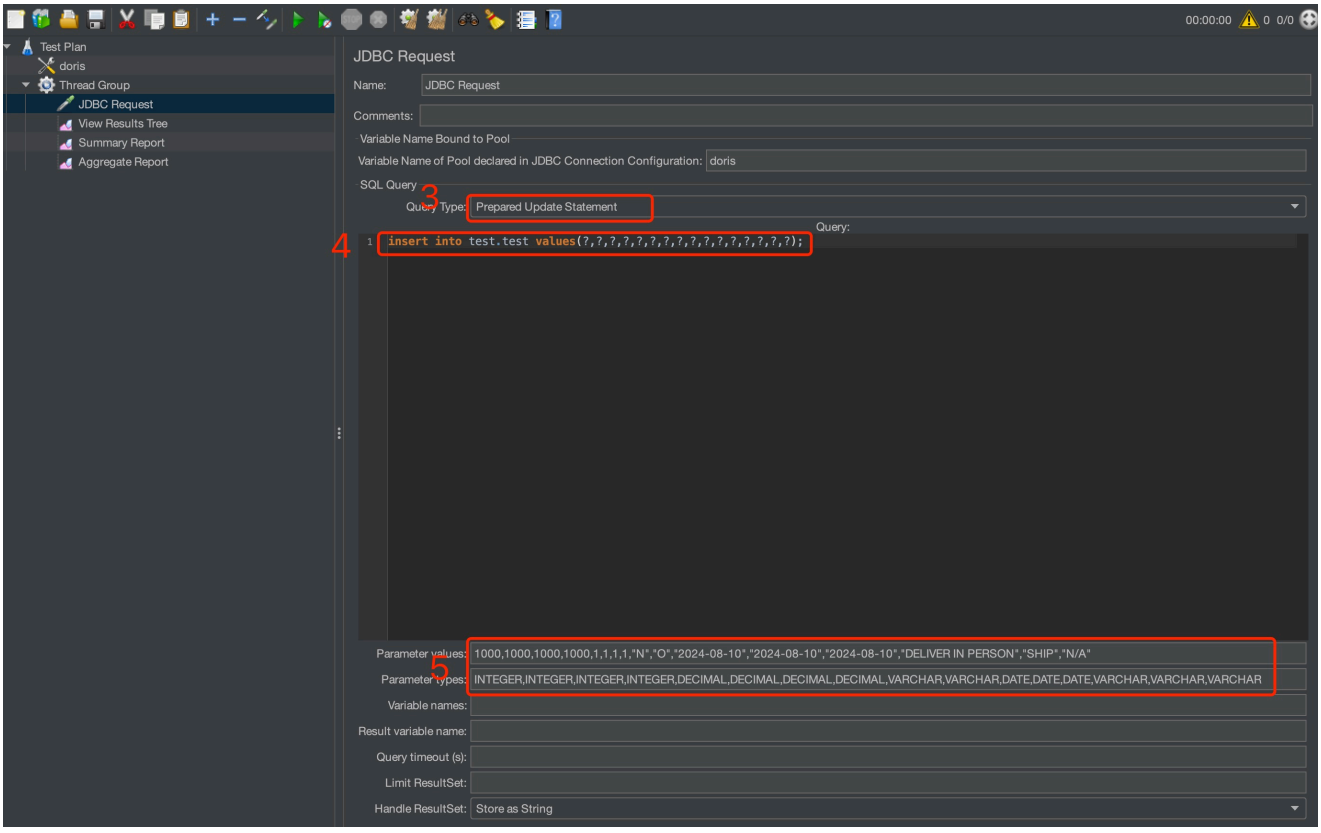
```
CREATE TABLE IF NOT EXISTS lineitem (
  L_ORDERKEY    INTEGER NOT NULL,
  L_PARTKEY     INTEGER NOT NULL,
  L_SUPPKEY     INTEGER NOT NULL,
  L_LINENUMBER  INTEGER NOT NULL,
  L_QUANTITY    DECIMAL(15,2) NOT NULL,
  L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
  L_DISCOUNT    DECIMAL(15,2) NOT NULL,
  L_TAX         DECIMAL(15,2) NOT NULL,
  L_RETURNFLAG  CHAR(1) NOT NULL,
  L_LINESTATUS  CHAR(1) NOT NULL,
  L_SHIPDATE    DATE NOT NULL,
  L_COMMITDATE  DATE NOT NULL,
  L_RECEIPTDATE DATE NOT NULL,
  L_SHIPINSTRUCT CHAR(25) NOT NULL,
  L_SHIPMODE     CHAR(10) NOT NULL,
  L_COMMENT      VARCHAR(44) NOT NULL
)
DUPLICATE KEY(L_ORDERKEY, L_PARTKEY, L_SUPPKEY, L_LINENUMBER)
DISTRIBUTED BY HASH(L_ORDERKEY) BUCKETS 32
PROPERTIES (
  "replication_num" = "3"
);
```

Testing Tool

- Jmeter

JMeter Parameter Settings as Shown in the Images





1. Set the Init Statement Before Testing: set group_commit=async_mode and set enable_nereids_planner=false.

2. Enable JDBC Prepared Statement: Complete URL: jdbc:mysql://127.0.0.1:9030?useServerPrepStmts=true&useLocalSessionState=true&rewriteBatchedStatements=true&cachePrepSt

mts=true&prepStmtCacheSqlLimit=99999&prepStmtCacheSize=50&sessionVariables=group_commit=async_mode,enable_nereids_planner=false.

3. Set the Import Type to Prepared Update Statement.
4. Set the Import Statement.
5. Set the Values to Be Imported: Ensure that the imported values match the data types one by one.

Testing Methodology

- Use JMeter to write data into Doris. Each thread writes 1 row of data per execution using the insert into statement.

Test Results

- Data unit: rows per second.
- The following tests are divided into 30, 100, and 500 concurrency.

Performance Test with 30 Concurrent Users in Sync Mode, 5 BEs, and 3 Replicas

Group Commit Interval	10ms	20ms	50ms	100ms
	321.5	307.3	285.8	224.3

Performance Test with 100 Concurrent Users in Sync Mode, 5 BEs, and 3 Replicas

Group Commit Interval	10ms	20ms	50ms	100ms
	1175.2	1108.7	1016.3	704.5

Performance Test with 500 Concurrent Users in Sync Mode, 5 BEs, and 3 Replicas

Group Commit Interval	10ms	20ms	50ms	100ms
	3289.8	3686.7	3280.7	2609.2

Insert into Sync Mode Large Batch Data

Machine Configuration

- 1 Front-End (FE): Alibaba Cloud, 16-core CPU, 64GB RAM, 1 x 500GB ESSD PL1 cloud disk
- 5 Back-End (BE) nodes: Alibaba Cloud, 16-core CPU, 64GB RAM, 1 x 1TB ESSD PL1 cloud disk.
- 1 Testing Client: Alibaba Cloud, 16-core CPU, 64GB RAM, 1 x 100GB ESSD PL1 cloud disk
- Test version: Doris-2.1.5

Dataset

- Insert into statement for 1000 rows: `insert into tbl values(1,1)...` (1000 rows omitted)

Testing Tool

- Jmeter

Testing Methodology

- Use JMeter to write data into Doris. Each thread writes 1000 row of data per execution using the insert into statement.

Test Results

- Data unit: rows per second.
- The following tests are divided into 30, 100, and 500 concurrency.

Performance Test with 30 Concurrent Users in Sync Mode, 5 BEs, and 3 Replicas

Group commit interval	10ms	20ms	50ms	100ms
	92.2K	85.9K	84K	83.2K

Performance Test with 100 Concurrent Users in Sync Mode, 5 BEs, and 3 Replicas

Group commit interval	10ms	20ms	50ms	100ms
	70.4K	70.5K	73.2K	69.4K

Performance Test with 500 Concurrent Users in Sync Mode, 5 BEs, and 3 Replicas

Group commit interval	10ms	20ms	50ms	100ms
	46.3K	47.7K	47.4K	46.5K

Load Best Practices

Table Model Selection

It is recommended to prioritize using the Duplicate Key model, which offers advantages in both data loading and query performance compared to other models. For more information, please refer to: Data Model

Partition and Bucket Configuration

It is recommended to keep the size of a tablet between 1-10GB. Tablets that are too small may lead to poor aggregation performance and increase metadata management overhead; tablets that are too large may hinder replica migration and repair. For details, please refer to: Data Distribution.

Random Bucketing

When using Random bucketing, you can enable single-tablet loading mode by setting load_to_single_tablet to true. This mode can improve data loading concurrency and throughput while reducing write amplification during large-scale data loading. For details, refer to: Random Bucketing

Batch Loading

Client-side batching: It is recommended to batch data (from several MB to GB in size) on the client side before loading. High-frequency small loads will cause frequent compaction, leading to severe write amplification issues. Server-side batching: For high-concurrency small data volume loading, it is recommended to enable Group Commit to implement batching on the server side.

Partition Loading

It is recommended to load data from only a few partitions at a time. Loading from too many partitions simultaneously will increase memory usage and may cause performance issues. Each tablet in Doris has an active Memtable in memory, which is flushed to disk when it reaches a certain size. To prevent process OOM, when the active Memtable's memory usage is too high, it will trigger early flushing, resulting in many small files and affecting loading performance.

Large-scale Data Batch Loading

When dealing with a large number of files or large data volumes, it is recommended to load in batches to avoid high retry costs in case of loading failures and to reduce system resource impact. For Broker Load, it is recommended not to exceed 100GB per batch. For large local data files, you can use Doris's streamloader tool, which automatically performs batch loading.

Broker Load Concurrency

Compressed files/Parquet/ORC files: It is recommended to split files into multiple smaller files for loading to achieve higher concurrency.

Uncompressed CSV and JSON files: Doris will automatically split files and load them concurrently.

For concurrency strategies, please refer to: Broker Load Configuration Parameters

Stream Load Concurrency

It is recommended to keep Stream load concurrency per BE under 128 (controlled by BE's webserver_num_workers parameter). High concurrency may cause webserver thread exhaustion and affect loading performance. Particularly when a single BE's concurrency exceeds 512 (doris_max_remote_scanner_thread_pool_thread_num parameter), it may cause the BE process to hang.

Data Update Overview

In today's data-driven decision-making landscape, data "freshness" has become a core competitive advantage for enterprises to stand out in fierce market competition. Traditional T+1 data processing models, due to their inherent latency, can no longer meet the stringent real-time requirements of modern business. Whether it's achieving millisecond-level synchronization between business databases and data warehouses, dynamically adjusting operational strategies, or correcting erroneous data within seconds to ensure decision accuracy, robust real-time data update capabilities are crucial.

Apache Doris, as a modern real-time analytical database, has one of its core design goals to provide ultimate data freshness. Through its powerful data models and flexible update mechanisms, it successfully compresses data analysis latency from day-level and hour-level to second-level, providing a solid foundation for users to build real-time, agile business decision-making loops.

This document serves as an official guide that systematically explains Apache Doris's data update capabilities, covering its core principles, diverse update and deletion methods, typical application scenarios, and performance best practices under different deployment modes, aiming to help you comprehensively master and efficiently utilize Doris's data update functionality.

1. Core Concepts: Table Models and Update Mechanisms

In Doris, the **Data Model** of a data table determines its data organization and update behavior. To support different business scenarios, Doris provides three table models: Unique Key Model, Aggregate Key Model, and Duplicate Key Model. Among these, **the Unique Key Model is the core for implementing complex, high-frequency data updates**.

1.1. Table Model Overview

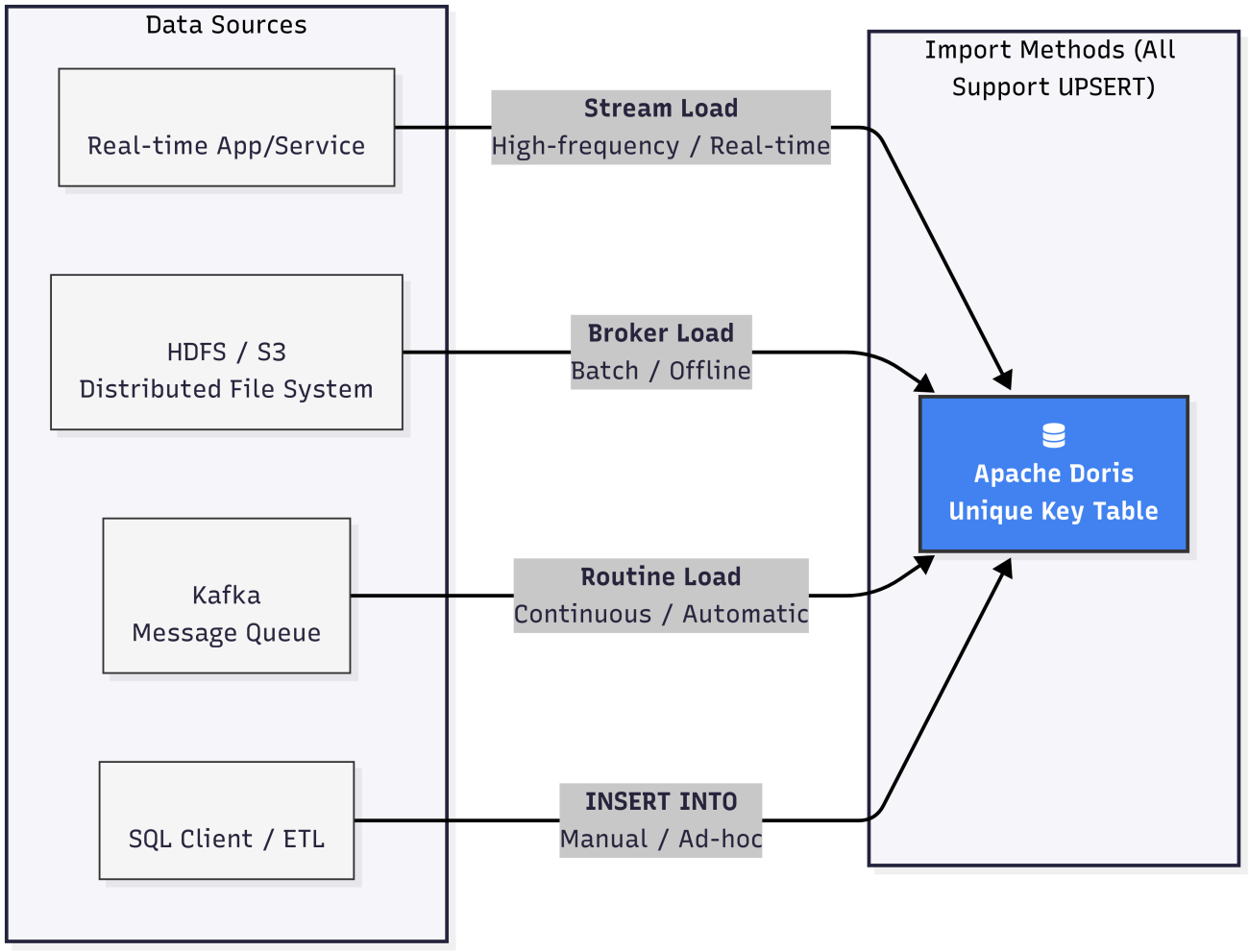
Table Model	Key Features	Update Capability	Use Cases
Unique Key Model	Built for real-time updates. Each data row is identified by a unique Primary Key, supporting row-level UPSERT (Update/Insert) and partial column updates.	Strongest, supports all update and deletion methods.	Order status updates, real-time user tag computation, CDC data synchronization, and other scenarios requiring frequent, real-time changes.
Aggregate Key Model	Pre-aggregates data based on specified Key columns. For rows with the same Key, Value columns are merged according to defined aggregation functions (such as SUM, MAX, MIN, REPLACE).	Limited, supports REPLACE-style updates and deletions based on Key columns.	Scenarios requiring real-time summary statistics, such as real-time reports, advertisement click statistics, etc.
Duplicate Key Model	Data only supports append-only writes, without any deduplication or aggregation operations. Even identical data rows are retained.	Limited, only supports conditional deletion through DELETE statements.	Log collection, user behavior tracking, and other scenarios that only need appending without updates.

1.2. Data Update Methods

Doris provides two major categories of data update methods: **updating through data load** and **updating through DML statements**.

1.2.1. Updating Through Load (UPSERT)

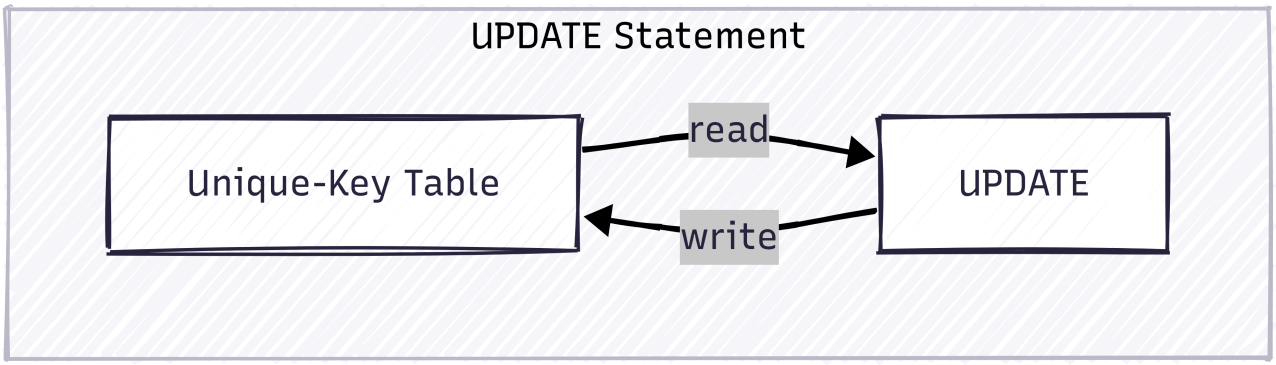
This is Doris's **recommended high-performance, high-concurrency** update method, primarily targeting the **Unique Key Model**. All load methods (Stream Load, Broker Load, Routine Load, `INSERT INTO`) naturally support `UPSERT` semantics. When new data is loaded, if its primary key already exists, Doris will overwrite the old row data with the new row data; if the primary key doesn't exist, it will insert a new row.



img

1.2.2. Updating Through `UPDATE` DML Statements

Doris supports standard SQL `UPDATE` statements, allowing users to update data based on conditions specified in the `WHERE` clause. This method is very flexible and supports complex update logic, such as cross-table join updates.



img

```
-- Simple update
UPDATE user_profiles SET age = age + 1 WHERE user_id = 1;

-- Cross-table join update
UPDATE sales_records t1
SET t1.user_name = t2.name
FROM user_profiles t2
WHERE t1.user_id = t2.user_id;
```

Note: The execution process of `UPDATE` statements involves first scanning data that meets the conditions, then rewriting the updated data back to the table. It's suitable for low-frequency, batch update tasks. **High-concurrency operations on `UPDATE` statements are not recommended** because concurrent `UPDATE` operations involving the same primary keys cannot guarantee data isolation.

1.2.3. Updating Through `INSERT INTO SELECT` DML Statements

Since Doris provides UPSERT semantics by default, using `INSERT INTO SELECT` can also achieve similar update effects as `UPDATE`.

1.3. Data Deletion Methods

Similar to updates, Doris also supports deleting data through both load and DML statements.

1.3.1. Mark Deletion Through Load

This is an efficient batch deletion method, primarily used for the **Unique Key Model**. Users can add a special hidden column `DORIS_DELETE_SIGN` when loading data. When the value of this column for a row is `1` or `true`, Doris will mark the corresponding data row with that primary key as deleted (the principle of delete sign will be explained in detail later).

```
// Stream Load load data, delete row with user_id = 2
// curl --location-trusted -u user:passwd -H "columns:user_id, __DORIS_DELETE_SIGN__" -T delete.json
http://fe_host:8030/api/db_name/table_name/_stream_load

// delete.json content
[
    {"user_id": 2, "__DORIS_DELETE_SIGN__": "1"}
]
```

1.3.2. Deletion Through `DELETE` DML Statements

Doris supports standard SQL `DELETE` statements that can delete data based on `WHERE` conditions.

- **Unique Key Model**: `DELETE` statements will rewrite the primary keys of rows meeting the conditions with deletion marks. Therefore, its performance is proportional to the amount of data to be deleted. The execution principle of `DELETE` statements on Unique Key Models is very similar to `UPDATE` statements, first reading the data to be deleted through queries, then writing it once more with deletion marks. Compared to `UPDATE` statements, DELETE statements only need to write Key columns and deletion mark columns, making them relatively lighter.
- **Duplicate/Aggregate Models**: `DELETE` statements are implemented by recording a delete predicate. During queries, this predicate serves as a runtime filter to filter out deleted data. Therefore, `DELETE` operations themselves are very fast, almost independent of the amount of deleted data. However, note that **high-frequency `DELETE` operations on Duplicate/Aggregate Models will accumulate many runtime filters, severely affecting subsequent query performance**.

```
DELETE FROM user_profiles WHERE last_login < '2022-01-01';
```

The following table provides a brief summary of using DML statements for deletion:

	Unique Key Model	Aggregate Model	Duplicate Model
Implementation	Delete Sign	Delete Predicate	Delete Predicate
Limitations	None	Delete conditions only for Key columns	None
Deletion Performance	Moderate	Fast	Fast

2. Deep Dive into Unique Key Model: Principles and Implementation

The Unique Key Model is the cornerstone of Doris's high-performance real-time updates. Understanding its internal working principles is crucial for fully leveraging its performance.

2.1. Merge-on-Write (MoW) vs. Merge-on-Read (MoR)

The Unique Key Model has two data merging strategies: Merge-on-Write (MoW) and Merge-on-Read (MoR). **Since Doris 2.1, MoW has become the default and recommended implementation**.

Feature	Merge-on-Write (MoW)	Merge-on-Read (MoR) - (Legacy)
Core Concept	Completes data deduplication and merging during data writing, ensuring only one latest record per primary key in storage.	Retains multiple versions during data writing, performs real-time merging during queries to return the latest version.
Query Performance	Extremely high. No additional merge operations needed during queries, performance approaches that of non-updated detail tables.	Poor. Requires data merging during queries, taking about 3-10 times longer than MoW and consuming more CPU and memory.
Write Performance	Has merge overhead during writing, with some performance loss compared to MoR (about 10-20% for small batches, 30-50% for large batches).	Fast writing speed, approaching detail tables.

Feature	Merge-on-Write (MoW)	Merge-on-Read (MoR) - (Legacy)
Resource Consumption	Consumes more CPU and memory during writing and background Compaction.	Consumes more CPU and memory during queries.
Use Cases	Most real-time update scenarios. Especially suitable for read-heavy, write-light businesses, providing ultimate query analysis performance.	Suitable for write-heavy, read-light scenarios, but no longer mainstream recommended.

The MoW mechanism trades a small cost during the writing phase for tremendous improvement in query performance, perfectly aligning with the OLAP system's "read-heavy, write-light" characteristics.

2.2. Conditional Updates (Sequence Column)

In distributed systems, out-of-order data arrival is a common problem. For example, an order status changes sequentially to "Paid" and "Shipped", but due to network delays, data representing "Shipped" might arrive at Doris before data representing "Paid".

To solve this problem, Doris introduces the **Sequence Column** mechanism. Users can specify a column (usually a timestamp or version number) as the Sequence column when creating tables. When processing data with the same primary key, Doris will compare their Sequence column values and **always retain the row with the largest Sequence value**, thus ensuring eventual consistency even when data arrives out of order.

```
CREATE TABLE order_status (
    order_id BIGINT,
    status_name STRING,
    update_time DATETIME
)
UNIQUE KEY(order_id)
DISTRIBUTED BY HASH(order_id)
PROPERTIES (
    "function_column.sequence_col" = "update_time" -- Specify update_time as Sequence column
);

-- 1. Write "Shipped" record (larger update_time)
-- {"order_id": 1001, "status_name": "Shipped", "update_time": "2023-10-26 12:00:00"}

-- 2. Write "Paid" record (smaller update_time, arrives later)
-- {"order_id": 1001, "status_name": "Paid", "update_time": "2023-10-26 11:00:00"}

-- Final query result, retains record with largest update_time
-- order_id: 1001, status_name: "Shipped", update_time: "2023-10-26 12:00:00"
```

2.3. Deletion Mechanism (`DORIS_DELETE_SIGN`) Workflow

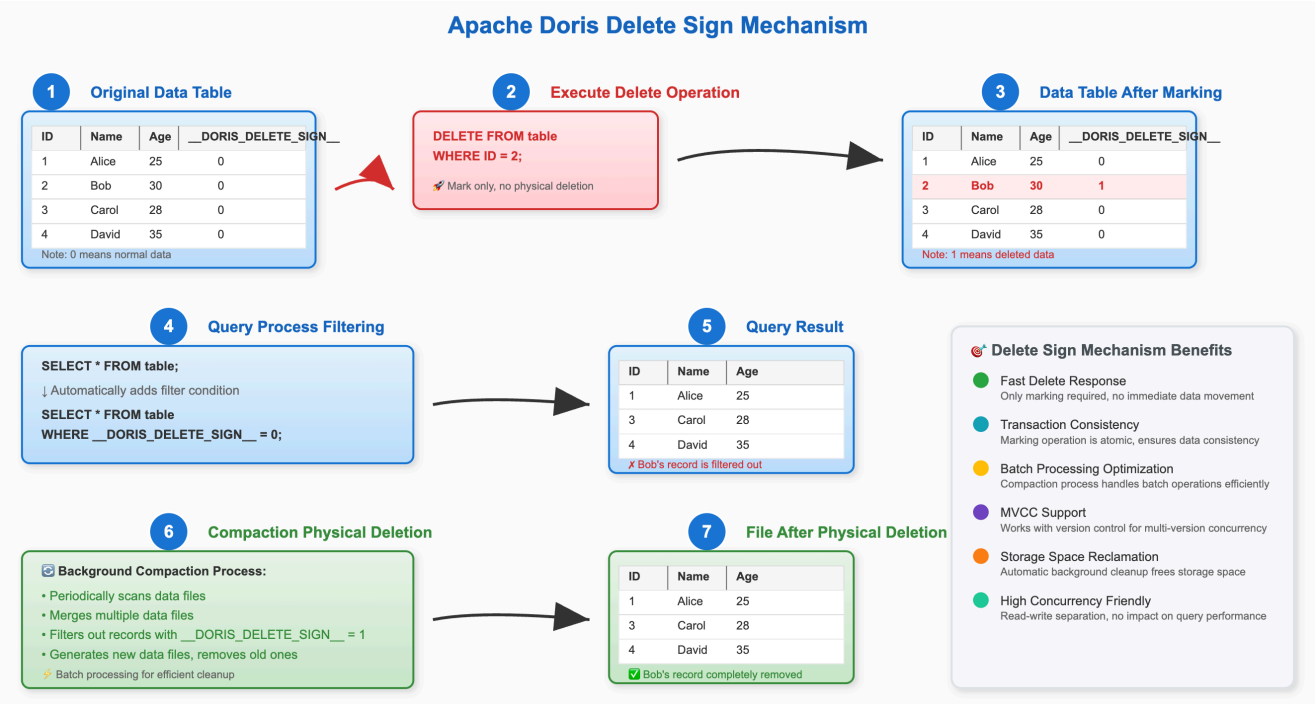
The working principle of `DORIS_DELETE_SIGN` can be summarized as "logical marking, background cleanup".

1. **Execute Deletion**: When users delete data through load or `DELETE` statements, Doris doesn't immediately remove data from physical files. Instead, it writes a new record for the primary key to be deleted, with the `DORIS_DELETE_SIGN` column marked as `1`.
2. **Query Filtering**: When users query data, Doris automatically adds a filter condition `WHERE DORIS_DELETE_SIGN = 0` to the query plan, thus hiding all data marked for deletion from query results.
3. **Background Compaction**: Doris's background Compaction process periodically scans data. When it finds a primary key with both normal records and deletion mark records, it will physically remove both

records during the merge process, eventually freeing storage space.

This mechanism ensures quick response to deletion operations while asynchronously completing physical cleanup through background tasks, avoiding performance impact on online business.

The following diagram shows how `DORIS_DELETE_SIGN` works:



img

2.4 Partial Column Update

Starting from version 2.0, Doris supports powerful partial column update capabilities on Unique Key Models (MoW). When loading data, users only need to provide the primary key and columns to be updated; unprovided columns will maintain their original values unchanged. This greatly simplifies ETL processes for scenarios like wide table joining and real-time tag updates.

To enable this feature, you need to enable Merge-on-Write (MoW) mode when creating the Unique Key Model table. For INSERT INTO, set the session variable `enable_unique_key_partial_update` to `true` to enable partial column updates; for Stream Load and other import methods, configure the `partial_columns` parameter to enable partial column updates.

```
CREATE TABLE user_profiles (
    user_id BIGINT,
    name STRING,
    age INT,
    last_login DATETIME
)
UNIQUE KEY(user_id)
DISTRIBUTED BY HASH(user_id)
PROPERTIES (
    "enable_unique_key_merge_on_write" = "true"
);

-- Initial data
-- user_id: 1, name: 'Alice', age: 30, last_login: '2023-10-01 10:00:00'
```

```
-- load partial update data through Stream Load, only updating age and last_login
-- {"user_id": 1, "age": 31, "last_login": "2023-10-26 18:00:00"}

-- Updated data
-- user_id: 1, name: 'Alice', age: 31, last_login: '2023-10-26 18:00:00'
```

Partial Column Update Principle Overview

Unlike traditional OLTP databases, Doris's partial column update is not in-place data update. To achieve better write throughput and query performance in Doris, partial column updates in Unique Key Models adopt an **"load-time missing field completion followed by full-row writing"** implementation approach.

Therefore, using Doris's partial column update has **"read amplification"** and **"write amplification"** effects. For example, updating 10 fields in a 100-column wide table requires Doris to complete the missing 90 fields during the write process. Assuming each field has similar size, a 1MB 10-field update will generate approximately 9MB of data reading (completing missing fields) and 10MB of data writing (writing the complete row to new files) in the Doris system, resulting in about 9x read amplification and 10x write amplification.

Partial Column Update Performance Recommendations

Due to read and write amplification in partial column updates, and since Doris is a columnar storage system, the data reading process may generate significant random I/O, requiring high random read IOPS from storage. Since traditional mechanical disks have significant bottlenecks in random I/O, **if you want to use partial column update functionality for high-frequency writes, SSD drives are recommended, preferably NVMe interface**, which can provide the best random I/O support.

Additionally, **if the table is very wide, enabling row storage is also recommended to reduce random I/O**. After enabling row storage, Doris will store an additional copy of row-based data alongside columnar storage. Since row-based data stores each row continuously, it can read entire rows with a single I/O operation (columnar storage requires N I/O operations to read all missing fields, such as the previous example of a 100-column wide table updating 10 columns, requiring 90 I/O operations per row to read all fields).

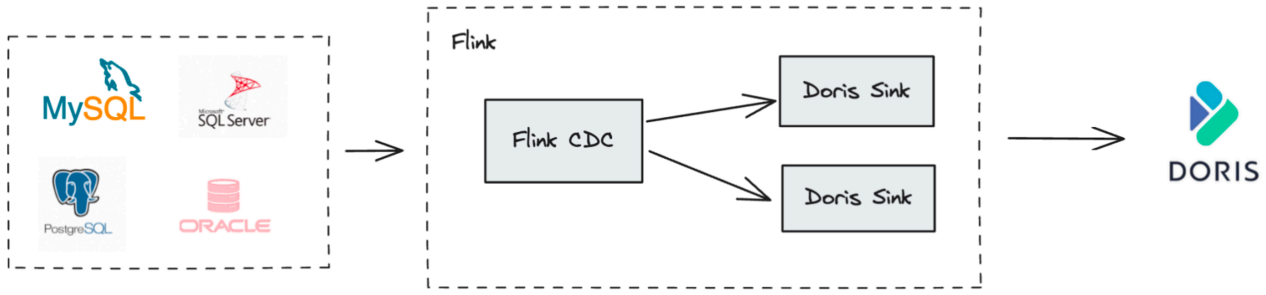
3. Typical Application Scenarios

Doris's powerful data update capabilities enable it to handle various demanding real-time analysis scenarios.

3.1. CDC Real-time Data Synchronization

Capturing change data (Binlog) from upstream business databases (such as MySQL, PostgreSQL, Oracle) through tools like Flink CDC and writing it in real-time to Doris Unique Key Model tables is the most classic scenario for building real-time data warehouses.

- **Whole Database Synchronization**: Flink Doris Connector internally integrates Flink CDC, enabling automated, end-to-end whole database synchronization from upstream databases to Doris without manual table creation and field mapping configuration.
- **Ensuring Consistency**: Utilizes the Unique Key Model's `UPSERT` capability to handle upstream `INSERT` and `UPDATE` operations, uses `DORIS_DELETE_SIGN` to handle `DELETE` operations, and combines with Sequence columns (such as timestamps in Binlog) to handle out-of-order data, perfectly replicating upstream database states and achieving millisecond-level data synchronization latency.



img

3.2. Real-time Wide Table Joining

In many analytical scenarios, data from different business systems needs to be joined into user-wide tables or product-wide tables. Traditional approaches use offline ETL tasks (such as Spark or Hive) for periodic (T+1) joining, which has poor real-time performance and high maintenance costs. Alternatively, using Flink for real-time wide table join calculations and writing joined data to databases typically requires significant computational resources.

Using Doris's **partial column update** capability can greatly simplify this process:

1. Create a Unique Key Model wide table in Doris.
2. Write data streams from different sources (such as user basic information, user behavior data, transaction data, etc.) to this wide table in real-time through Stream Load or Routine Load.
3. Each data stream only updates its relevant fields. For example, user behavior data streams only update `page_view_count`, `last_login_time`, and other fields; transaction data streams only update `total_orders`, `total_amount`, and other fields.

This approach not only transforms wide table construction from offline ETL to real-time stream processing, greatly improving data freshness, but also reduces I/O overhead by only writing changed columns, improving write performance.

4. Best Practices

Following these best practices can help you use Doris's data update functionality more stably and efficiently.

1. **Prioritize load Updates**: For high-frequency, large-volume update operations, prioritize load methods like Stream Load and Routine Load over `UPDATE` DML statements.
2. **Batch Writes**: Avoid using `INSERT INTO` statements for individual high-frequency writes (such as > 100 TPS), as each `INSERT` incurs transaction overhead. If necessary, consider enabling Group Commit functionality to merge multiple small batch commits into one large transaction.
3. **Use High-frequency `DELETE` Carefully**: On Duplicate and Aggregate models, avoid high-frequency `DELETE` operations to prevent query performance degradation.
4. **Use `TRUNCATE PARTITION` for Partition Data Deletion**: If you need to delete entire partition data, use `TRUNCATE PARTITION`, which is much more efficient than `DELETE`.
5. **Execute `UPDATE` Serially**: Avoid concurrent execution of `UPDATE` tasks that might affect the same data rows.

Conclusion

Apache Doris, with its powerful, flexible, and efficient data update capabilities centered on the Unique Key Model, truly breaks through the bottleneck of traditional OLAP systems in terms of data freshness. Whether through high-performance loads implementing `UPSERT` and partial column updates, or using Sequence columns to ensure consistency of out-of-order data, Doris provides complete solutions for building end-to-end real-time analytical applications.

By deeply understanding its core principles, mastering the applicable scenarios for different update methods, and following the best practices provided in this document, you will be able to fully unleash Doris's potential, making real-time data truly become a powerful engine driving business growth.

Updating Data with UPDATE Command

This document explains how to use the `UPDATE` command to modify data in Doris. The `UPDATE` command is only applicable to tables with a Unique data model.

Applicable Scenarios

- Small-scale data updates: Ideal for scenarios where a small amount of data needs to be corrected, such as fixing erroneous fields in certain records or updating the status of specific fields (e.g., order status updates).
- ETL batch processing of certain fields: Suitable for large-scale updates of a specific field, commonly seen in ETL processing scenarios. Note: Large-scale data updates should be infrequent.

How It Works

The query engine uses its own filtering logic to identify the rows that need to be updated. Then, using the Unique model's Value column logic to replace old data with new data. The rows to be updated are modified and reinserted into the table to achieve row-level updates.

Synchronization

The `UPDATE` syntax in Doris is synchronous, meaning that once the `UPDATE` statement is successfully executed, the update operation is completed and the data is immediately visible.

Performance

The performance of the `UPDATE` statement is closely related to the number of rows to be updated and the efficiency of the query conditions.

- Number of rows to be updated: The more rows that need to be updated, the slower the `UPDATE` statement will be. For small-scale updates, Doris supports a frequency similar to `INSERT INTO` statements. For large-scale updates, due to the long execution time, it is only suitable for infrequent calls.
- Efficiency of query conditions: The `UPDATE` implementation first reads the rows that meet the query conditions. Therefore, if the query conditions are efficient, the `UPDATE` speed will be fast. Ideally, the condition columns should hit the index or partition bucket pruning, so Doris does not need to scan the entire table and can quickly locate the rows that need to be updated, thereby improving update efficiency. It is strongly recommended not to include value columns in the condition columns.

Usage Example

Assume there is a transaction details table with the following structure in a financial risk control scenario:

```
CREATE TABLE transaction_details (
  transaction_id BIGINT NOT NULL,        -- Unique transaction ID
```

```
  user_id BIGINT NOT NULL,               -- User ID
  transaction_date DATE NOT NULL,        -- Transaction date
  transaction_time DATETIME NOT NULL,    -- Transaction time
  transaction_amount DECIMAL(18, 2),     -- Transaction amount
  transaction_device STRING,             -- Transaction device
  transaction_region STRING,             -- Transaction region
  average_daily_amount DECIMAL(18, 2),   -- Average daily transaction amount over the last 3 months
  recent_transaction_count INT,          -- Number of transactions in the last 7 days
  has_dispute_history BOOLEAN,           -- Whether there is a dispute history
  risk_level STRING                      -- Risk level
)
UNIQUE KEY(transaction_id)
DISTRIBUTED BY HASH(transaction_id) BUCKETS 16
PROPERTIES (
  "replication_num" = "3",               -- Number of replicas, default is 3
  "enable_unique_key_merge_on_write" = "true"  -- Enable MOW mode, support merge updates
);
```

The following transaction data exists:

```
+----------------+---------+------------------+---------------------+--------------------+--------------------+--------------------
+----------------------+--------------------------+--------------------+------------+
| transaction_id | user_id | transaction_date | transaction_time    | transaction_amount | transaction_device | transaction_region |
average_daily_amount | recent_transaction_count | has_dispute_history | risk_level |
+----------------+---------+------------------+---------------------+--------------------+--------------------+--------------------
+----------------------+--------------------------+--------------------+------------+
|           1001 |    5001 | 2024-11-24       | 2024-11-24 14:30:00 |             100.00 | iPhone 12          | New York           |
100.00 |                       10 |                    0 | NULL       |
|           1002 |    5002 | 2024-11-24       | 2024-11-24 03:30:00 |             120.00 | iPhone 12          | New York           |
100.00 |                       15 |                    0 | NULL       |
|           1003 |    5003 | 2024-11-24       | 2024-11-24 10:00:00 |             150.00 | Samsung S21        | Los Angeles        |
100.00 |                       30 |                    0 | NULL       |
|           1004 |    5004 | 2024-11-24       | 2024-11-24 16:00:00 |             300.00 | MacBook Pro        | high_risk_region1  |
200.00 |                        5 |                    0 | NULL       |
|           1005 |    5005 | 2024-11-24       | 2024-11-24 11:00:00 |            1100.00 | iPad Pro           | Chicago            |
200.00 |                       10 |                    0 | NULL       |
+----------------+---------+------------------+---------------------+--------------------+--------------------+--------------------
+----------------------+--------------------------+--------------------+------------+
```

Update the risk level of all daily transaction records according to the following risk control rules: 1. Transactions with a dispute history have a risk level of high. 2. Transactions in high-risk regions have a risk level of high. 3. Transactions with abnormal amounts (exceeding 5 times the daily average) have a risk level of high. 4. Frequent transactions in the last 7 days: a. Transactions > 50 times have a risk level of high. b. Transactions between 20 and 50 times have a risk level of medium. 5. Transactions during non-working hours (2 AM to 4 AM) have a risk level of medium. 6. The default risk level is low.

```
UPDATE transaction_details
SET risk_level = CASE
  -- Transactions with a dispute history or in high-risk regions
  WHEN has_dispute_history = TRUE THEN 'high'
  WHEN transaction_region IN ('high_risk_region1', 'high_risk_region2') THEN 'high'

  -- Abnormal transaction amount
  WHEN transaction_amount > 5 * average_daily_amount THEN 'high'

  -- High transaction frequency in the last 7 days
  WHEN recent_transaction_count > 50 THEN 'high'
  WHEN recent_transaction_count BETWEEN 20 AND 50 THEN 'medium'

  -- Transactions during non-working hours
```

```
  WHEN HOUR(transaction_time) BETWEEN 2 AND 4 THEN 'medium'

  -- Default risk level
  ELSE 'low'
END
WHERE transaction_date = '2024-11-24';
```

The updated data is as follows:

```
+----------------+---------+------------------+---------------------+--------------------+--------------------+--------------------
+----------------------+--------------------------+---------------------+------------+
| transaction_id | user_id | transaction_date | transaction_time    | transaction_amount | transaction_device | transaction_region |
average_daily_amount | recent_transaction_count | has_dispute_history | risk_level |
+----------------+---------+------------------+---------------------+--------------------+--------------------+--------------------
+----------------------+--------------------------+---------------------+------------+
|           1001 |    5001 | 2024-11-24       | 2024-11-24 14:30:00 |             100.00 | iPhone 12          | New York           |
100.00 |                       10 |                   0 | low        |
|           1002 |    5002 | 2024-11-24       | 2024-11-24 03:30:00 |             120.00 | iPhone 12          | New York           |
100.00 |                       15 |                   0 | medium     |
|           1003 |    5003 | 2024-11-24       | 2024-11-24 10:00:00 |             150.00 | Samsung S21        | Los Angeles        |
100.00 |                       30 |                   0 | medium     |
|           1004 |    5004 | 2024-11-24       | 2024-11-24 16:00:00 |             300.00 | MacBook Pro        | high_risk_region1  |
200.00 |                        5 |                   0 | high       |
|           1005 |    5005 | 2024-11-24       | 2024-11-24 11:00:00 |            1100.00 | iPad Pro           | Chicago            |
200.00 |                       10 |                   0 | high       |
+----------------+---------+------------------+---------------------+--------------------+--------------------+--------------------
+----------------------+--------------------------+---------------------+------------+
```

More Help

For more detailed syntax on data updates, please refer to the UPDATE command manual. You can also enter `HELP UPDATE` in the MySQL client command line for more help.

Updating Data on Unique Key Model

This document introduces how to update data in the Doris unique key model using various load methods.

Whole Row Update

When loading data into the unique key model using Doris-supported methods like Stream Load, Broker Load, Routine Load, Insert Into, etc., new data is inserted if there is no existing primary key data row. If there is an existing primary key data row, it is updated. This means the load operation in the Doris unique key model works in an “upsert” mode. The process of updating existing records is the same as loading new records by default, so you can refer to the data load documentation for more details.

Partial Column Update

Partial column update allows you to update specific fields in a table without modifying all fields. You can use the Update statement to perform this operation, which typically involves reading the entire row, updating the desired fields, and writing it back. This read-write transaction is time-consuming and not suitable for large-scale data writing. Doris provides a feature to directly insert or update partial column data in the unique key model load update, bypassing the need to read the entire row first, thus significantly improving update efficiency.

> **Note**
>
> 1. Version 2.0 only supports partial column updates in the Merge-on-Write implementation of the Unique Key.
> 2. Starting from version 2.0.2, partial column updates are supported using INSERT INTO.
> 3. Partial column updates are not supported on tables with synchronized materialized views.
> 4. Partial column updates are not allowed on tables doing schema change.

Applicable Scenarios

- Real-time dynamic column updates, requiring frequent updates of specific fields in the table. For example, updating fields related to the latest user behavior in a user tag table for real-time analysis and decision-making in advertising/recommendation systems.
- Merging multiple source tables into one large wide table.
- Data correction.

Usage Example

Assume there is an order table `order_tbl` in Doris, where the order id is the Key column, and the order status and order amount are the Value columns. The data status is as follows:

Order id	Order Amount	Order Status
1	100	Pending Payment

```
+----------+--------------+--------------+
| order_id | order_amount | order_status |
+----------+--------------+--------------+
| 1        |          100 | Pending Payment |
+----------+--------------+--------------+
1 row in set (0.01 sec)
```

After the user clicks to pay, the Doris system needs to change the order status of the order with order id '1' to 'Pending Shipment'.

Partial Column Update Using Load Methods

StreamLoad/BrokerLoad/RoutineLoad

Prepare the following CSV file:

```
1,Pending Shipment
```

Add the following header during load:

```
partial_columns:true
```

Specify the columns to be loaded in `columns` (must include all key columns). Below is an example of Stream Load:

```
curl --location-trusted -u root: -H "partial_columns:true" -H "column_separator:," -H "columns:order_id,order_status" -T
/tmp/update.csv http://127.0.0.1:8030/api/db1/order_tbl/_stream_load
```

INSERT INTO

In all data models, the default behavior of `INSERT INTO` when given partial columns is to write the entire row. To prevent misuse, in the Merge-on-Write implementation, `INSERT INTO` maintains the semantics of whole row UPSERT by default. To enable partial column updates, set the following session variable:

```
SET enable_unique_key_partial_update=true;
INSERT INTO order_tbl (order_id, order_status) VALUES (1, 'Pending Shipment');
```

Note that the session variable `enable_insert_strict` defaults to true, enabling strict mode by default. In strict mode, partial column updates do not allow updating non-existent keys. To insert non-existent keys using the insert statement for partial column updates, set `enable_unique_key_partial_update` to true and `enable_insert_strict` to false.

Flink Connector

If using Flink Connector, add the following configuration:

```
'sink.properties.partial_columns' = 'true',
```

Specify the columns to be loaded in `sink.properties.column` (must include all key columns).

Update Result

The result after the update is as follows:

```
+----------+--------------+--------------+
| order_id | order_amount | order_status |
+----------+--------------+--------------+
| 1        |          100 | Pending Shipment |
+----------+--------------+--------------+
1 row in set (0.01 sec)
```

Usage Notes

Since the Merge-on-Write implementation needs to complete the entire row of data during writing to ensure optimal query performance, using it for partial column updates may decrease partial load performance.

Performance optimization suggestions:

- Use SSDs equipped with NVMe or high-speed SSD cloud disks, as completing data will read a large amount of historical data, generating high read IOPS and throughput.
- Enabling row storage can reduce the IOPS generated when completing data, significantly improving load performance. Enable row storage by setting the following property when creating a table:

```
"store_row_column" = "true"
```

Currently, all rows in the same batch data writing task (whether a load task or `INSERT INTO`) can only update the same columns. To update data with different columns, write in different batches.

Future versions will support flexible column updates, allowing users to update different columns for each row in the same batch load.

Updating Data on Aggregate Key Model

This document primarily introduces how to update the Doris Aggregate model based on data load.

Whole Row Update

When loading data into the Aggregate model table using Doris-supported methods such as Stream Load, Broker Load, Routine Load, Insert Into, etc., the new values will be aggregated with the old values according to the column's aggregation function to produce new aggregated values. This value may be generated at the time of insertion or during asynchronous compaction, but users will get the same return value when querying.

Partial Column Update of Aggregate Model

The Aggregate table is mainly used in pre-aggregation scenarios rather than data update scenarios, but partial column updates can be achieved by setting the aggregation function to REPLACE_IF_NOT_NULL.

Create Table

Set the aggregation function of the fields that need to be updated to `REPLACE_IF_NOT_NULL`.

```
CREATE TABLE order_tbl (
  order_id int(11) NULL,
  order_amount int(11) REPLACE_IF_NOT_NULL NULL,
  order_status varchar(100) REPLACE_IF_NOT_NULL NULL
) ENGINE=OLAP
AGGREGATE KEY(order_id)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(order_id) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

Data Insertion

Whether it is Stream Load, Broker Load, Routine Load, or `INSERT INTO`, directly write the data of the fields to be updated.

Example

Similar to the previous examples, the corresponding Stream Load command is (no additional header required):

```
$ cat update.csv

1,To be shipped

curl  --location-trusted -u root: -H "column_separator:," -H "columns:order_id,order_status" -T ./update.csv
http://127.0.0.1:8030/api/db1/order_tbl/_stream_load
```

The corresponding `INSERT INTO` statement is (no additional session variable settings required):

```
INSERT INTO order_tbl (order_id, order_status) values (1,'Shipped');
```

Notes on Partial Column Updates

The Aggregate Key model does not perform any additional processing during the write process, so the write performance is not affected and is the same as normal data load. However, the cost of aggregation during query is relatively high, and the typical aggregation query performance is 5-10 times lower than the Merge-on-Write implementation of the Unique Key model.

Since the `REPLACE_IF_NOT_NULL` aggregation function only takes effect when the value is not NULL, users cannot change a field value to NULL.

Concurrency Control for Updates in the Primary Key Model

Overview

Doris employs Multi-Version Concurrency Control (MVCC) to handle concurrent updates. Each data write operation is assigned a write transaction, ensuring atomicity (i.e., the write operation either fully succeeds or fully fails). Upon committing the write transaction, the system assigns it a version number. In the Unique Key model, when loading data multiple times, if there are duplicate primary keys, Doris determines the overwrite order based on the version number: data with a higher version number will overwrite data with a lower version number.

In some scenarios, users may need to adjust the effective order of data by specifying a sequence column in the table creation statement. For example, when synchronizing data to Doris concurrently through multiple threads, data from different threads may arrive out of order. In this case, old data arriving later may incorrectly overwrite new data. To solve this problem, users can assign a lower sequence value to old data and a higher sequence value to new data, allowing Doris to correctly determine the update order based on the user-provided sequence values.

Additionally, the `UPDATE` statement differs significantly from updates implemented through data loads at the underlying mechanism level. The `UPDATE` operation involves two steps: reading the data to be updated from the database and writing the updated data. By default, the `UPDATE` statement provides transaction capabilities with Serializable isolation level through table-level locks, meaning multiple `UPDATE` operations can only be executed serially. Users can also bypass this restriction by adjusting the configuration, as detailed in the following sections.

UPDATE Concurrency Control

By default, concurrent `UPDATE` operations on the same table are not allowed.

The main reason is that Doris currently supports row updates, which means that even if the user declares `SET v2 = 1`, all other value columns will also be overwritten (even if the values have not changed).

This can lead to a problem where if two `UPDATE` operations update the same row simultaneously, the behavior may be indeterminate, potentially resulting in dirty data.

However, in practical applications, if users can ensure that concurrent updates do not operate on the same row simultaneously, they can manually enable concurrent updates. By modifying the FE configuration `enable_concurrent_update`, setting this configuration value to `true` will disable transaction guarantees for update commands.

Sequence Column

The Unique model is mainly for scenarios requiring unique primary keys, ensuring the uniqueness constraint of the primary key. The replacement order of data loaded in the same batch or different batches is not

guaranteed. Without a guaranteed replacement order, the specific data ultimately loaded into the table is uncertain.

To solve this problem, Doris supports sequence columns. By specifying a sequence column during loading, data with the same key column is replaced based on the sequence column value, with larger values replacing smaller ones, and vice versa. This method allows users to control the replacement order.

In implementation, Doris adds a hidden column **DORIS_SEQUENCE_COL**, whose type is specified by the user during table creation. The specific value of this column is determined during data loading, and the effective row for the same key column is decided based on this value.

> **Note**
>
> The sequence column currently only supports the Unique model.

Enabling Sequence Column Support

When creating a new table, if `function_column.sequence_col` or `function_column.sequence_type` is set, the new table will support sequence columns.

For a table that does not support sequence columns, you can enable this feature using the following statement: `ALTER TABLE example_db.my_table ENABLE FEATURE "SEQUENCE_LOAD" WITH PROPERTIES ("function_column.sequence_type" = "Date")`.

To check if a table supports sequence columns, you can set a session variable to display hidden columns `SET show_hidden_columns=true`, then use `desc tablename`. If the output includes the `__DORIS_SEQUENCE_COL__` column, it is supported; otherwise, it is not.

Usage Example

Below is an example of using Stream Load:

1. Create a table supporting sequence columns

Create a unique model `test_table` and map the sequence column to the `modify_date` column.

```
CREATE TABLE test.test_table
(
    user_id bigint,
    date date,
    group_id bigint,
    modify_date date,
    keyword VARCHAR(128)
)
UNIQUE KEY(user_id, date, group_id)
DISTRIBUTED BY HASH (user_id) BUCKETS 32
PROPERTIES(
    "function_column.sequence_col" = 'modify_date',
    "replication_num" = "1",
    "in_memory" = "false"
);
```

The `sequence_col` specifies the mapping of the sequence column to a column in the table. This column can be of integer or date/time type (DATE, DATETIME) and cannot be changed after creation.

The table structure is as follows:

```
MySQL>  desc test_table;
+-------------+--------------+------+-------+---------+---------+
| Field       | Type         | Null | Key   | Default | Extra   |
+-------------+--------------+------+-------+---------+---------+
| user_id     | BIGINT       | No   | true  | NULL    |         |
| date        | DATE         | No   | true  | NULL    |         |
| group_id    | BIGINT       | No   | true  | NULL    |         |
| modify_date | DATE         | No   | false | NULL    | REPLACE |
| keyword     | VARCHAR(128) | No   | false | NULL    | REPLACE |
+-------------+--------------+------+-------+---------+---------+
```

In addition to specifying the sequence column through column mapping, Doris also supports creating a sequence column based on a specified type. This method does not require a column in the schema for mapping. The syntax is as follows:

```
PROPERTIES (
    "function_column.sequence_type" = 'Date',
);
```

The `sequence_type` specifies the type of the sequence column, which can be integer or date/time type (DATE, DATETIME).

2. Load Data:

Using column mapping (`function_column.sequence_col`) to specify the sequence column does not require modifying any parameters. Below is an example of loading data using Stream Load:

```
1   2020-02-22  1   2020-02-21  a
1   2020-02-22  1   2020-02-22  b
1   2020-02-22  1   2020-03-05  c
1   2020-02-22  1   2020-02-26  d
1   2020-02-22  1   2020-02-23  e
1   2020-02-22  1   2020-02-24  b
```

Stream load command:

```
curl --location-trusted -u root: -T testData http://host:port/api/test/test_table/_stream_load
```

Result:

```
MySQL> select * from test_table;
+---------+------------+----------+-------------+---------+
| user_id | date       | group_id | modify_date | keyword |
+---------+------------+----------+-------------+---------+
|       1 | 2020-02-22 |        1 | 2020-03-05  | c       |
+---------+------------+----------+-------------+---------+
```

In this load job, the value '2020-03-05' in the sequence column (modify_date) is the largest, so the keyword column retains 'c'.

If the sequence column is specified using `function_column.sequence_col` during table creation, the sequence column mapping must be specified during load.

1. Stream Load

In Stream Load, specify the sequence column mapping in the header:

```
curl --location-trusted -u root -H "columns: k1,k2,source_sequence,v1,v2" -H "function_column.sequence_col: source_sequence" -T
testData http://host:port/api/testDb/testTbl/_stream_load
```

2. Broker Load

Set the hidden column mapping in the `ORDER BY` clause:

```
LOAD LABEL db1.label1
(
    DATA INFILE("hdfs://host:port/user/data/*/test.txt")
    INTO TABLE `tbl1`
    COLUMNS TERMINATED BY ","
    (k1,k2,source_sequence,v1,v2)
    ORDER BY source_sequence
)
WITH BROKER 'broker'
(
    "username"="user",
    "password"="pass"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

3. Routine Load

The mapping method is the same as above. Example:

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
    [WITH MERGE|APPEND|DELETE]
    COLUMNS(k1, k2, source_sequence, v1, v2),
    WHERE k1 > 100 and k2 like "%doris%"
    [ORDER BY source_sequence]
    PROPERTIES
    (
        "desired_concurrent_number"="3",
        "max_batch_interval" = "20",
        "max_batch_rows" = "300000",
        "max_batch_size" = "209715200",
        "strict_mode" = "false"
    )
    FROM KAFKA
    (
        "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
        "kafka_topic" = "my_topic",
        "kafka_partitions" = "0,1,2,3",
```

```
        "kafka_offsets" = "101,0,0,200"
    );
```

3. Ensuring Replacement Order

After completing the above steps, load the following data:

```
1   2020-02-22  1   2020-02-22  a
1   2020-02-22  1   2020-02-23  b
```

Query data:

```
MySQL [test]> select * from test_table;
+---------+------------+----------+-------------+---------+
| user_id | date       | group_id | modify_date | keyword |
+---------+------------+----------+-------------+---------+
|       1 | 2020-02-22 |        1 | 2020-03-05  | c       |
+---------+------------+----------+-------------+---------+
```

In this load, the sequence column value '2020-03-05' is the largest, so the keyword column retains 'c'.

4. Try Loading the Following Data

```
1   2020-02-22  1   2020-02-22  a
1   2020-02-22  1   2020-03-23  w
```

Query data:

```
MySQL [test]> select * from test_table;
+---------+------------+----------+-------------+---------+
| user_id | date       | group_id | modify_date | keyword |
+---------+------------+----------+-------------+---------+
|       1 | 2020-02-22 |        1 | 2020-03-23  | w       |
+---------+------------+----------+-------------+---------+
```

This time, the data in the table is replaced. In summary, during the load process, the sequence column values of all batches are compared, and the record with the largest value is loaded into the Doris table.

Note

1. To prevent misuse, in Stream Load/Broker Load load tasks and row update insert statements, users must explicitly specify the sequence column (unless the default value of the sequence column is CURRENT_TIMESTAMP), otherwise, the following error message will be received:

```
Table test_tbl has sequence column, need to specify the sequence column
```

2. When you use the Insert statement to insert data, you must display the specified sequence column; otherwise, the preceding exception is reported. In order to facilitate the use of Doris in some scenarios (table replication, internal data migration, etc.), Doris can be controlled by the session parameter to close the check constraint on the sequence column:

```
set require_sequence_in_insert = false;
```

3. Since version 2.0, Doris supports partial column update capability for Unique Key tables with Merge-on-Write implementation. In partial column update loads, users can update only a portion of the columns each time, so it is not necessary to include the sequence column. If the load task submitted by the user includes the sequence column, the behavior is unaffected; if the load task does not include the sequence column, Doris will use the sequence column from the matching historical data as the value of the sequence column for the updated row. If there is no matching key column in the historical data, null or the default value will be used.

4. During concurrent loads, Doris uses the MVCC mechanism to ensure data correctness. If two batches of data loads update different columns of the same key, the load task with the higher system version will use the data row written by the lower version load task to fill in the same key after the lower version load task succeeds.

Delete Overview

In Apache Doris, the delete operation is essential for managing and cleaning data to meet the flexible needs of users in large-scale data analysis scenarios.

Doris offers a variety of delete functionalities, including DELETE statements, delete sign, partition deletion, full table deletion, and atomic overwrite using temporary partitions. The following sections will detail each feature:

DELETE Statement

The DELETE statement is the most commonly used method for deleting data and supports all table models. Users can use it to delete data that meets specific conditions.

The syntax of the DELETE statement is as follows:

```
DELETE FROM table_name WHERE condition;
```

While the DELETE statement can handle most deletion needs, it may not be the most efficient in some scenarios. To address various deletion requirements flexibly and efficiently, Doris also provides the following methods.

Truncate Partition

In Doris, managing data through date partitions and other methods is common. Many users only need to retain data for a recent period (e.g., 7 days). For expired data partitions, the truncate partition feature can be used for efficient deletion.

Compared to the DELETE statement, truncate partition only requires modifying some partition metadata to complete the deletion, making it the best method in this scenario.

The syntax of partition deletion is as follows:

```
TRUNCATE TABLE tbl PARTITION(p1, p2);
```

Truncate Table

Truncate table is suitable for quickly clearing a table while retaining its structure, such as when redoing data in offline analysis scenarios.

The syntax of full truncate table is as follows:

```
TRUNCATE TABLE table_name;
```

Delete Sign

Data deletion can be considered a type of data update. Therefore, on the primary key model (Unique Key) with update capabilities, users can use the delete sign feature to perform delete operations as data updates.

For example, in CDC data synchronization scenarios, the CDC program can mark a DELETE operation binlog with a delete sign. When this data is written to Doris, the corresponding primary key will be deleted.

This method can perform batch deletion of a large number of primary keys, which is more efficient than the DELETE statement.

The delete sign is an advanced feature and is more complex to use compared to the previous methods. For detailed usage, please refer to the document Batch Deletion.

Atomic Overwrite Using Temporary Partitions

In some cases, users want to rewrite the data of a partition. However, if the data is deleted and then loaded, there will be a period when the data is unavailable. In this case, users can first create a corresponding temporary partition, load the new data into the temporary partition, and then replace the original partition atomically. For detailed usage, please refer to the document Atomic Table Replacement.

Precautions

1. The delete operation will generate new data versions, so frequent deletions may increase the number of versions, thereby affecting query performance.
2. Deleted data will still occupy storage until merge and compression are completed, so the delete operation itself will not immediately reduce storage usage.

Deleting Data with DELETE Command

The `DELETE` statement removes data from a specified table or partition based on conditions through the MySQL protocol. It supports specifying the data to be deleted using simple predicate combinations and also supports using the `USING` clause to join multiple tables for deletion on primary key tables.

Delete by Specifying Filter Predicates

```
DELETE FROM table_name [table_alias]
  [PARTITION partition_name | PARTITIONS (partition_name [, partition_name])]
  WHERE column_name op { value | value_list } [ AND column_name op { value | value_list } ...];
```

Required Parameters

- `table_name`: The table from which data needs to be deleted.
- `column_name`: A column belonging to `table_name`.
- `op`: Logical comparison operators, including: =, >, <, >=, <=, !=, in, not in.
- `value | value_list`: The value or list of values for logical comparison.

Optional Parameters

- `PARTITION partition_name | PARTITIONS (partition_name [, partition_name])`: Specifies the partition name where the data deletion is to be executed. If the table does not have this partition, an error will be reported.
- `table_alias`: Alias for the table.

Usage Restrictions

- When using the Aggregate table model, conditions can only be specified on Key columns. If the selected Key column does not exist in a Rollup, deletion cannot be performed.
- For partitioned tables, partitions need to be specified. If not specified, Doris will infer the partition from the conditions.
    - Doris cannot infer the partition from the conditions in two cases:
        1. The conditions do not include partition columns.
        2. The `op` of the partition column is `not in`.
    - When the partitioned table is not a Unique table, and it does not specify a partition or cannot infer the partition from the conditions, the session variable `delete_without_partition` needs to be set to `true`, and the delete operation will apply to all partitions.

Examples

1. Delete rows in partition `p1` of `my_table` where the value of column `k1` is 3

```
DELETE FROM my_table PARTITION p1
  WHERE k1 = 3;
```

2. Delete rows in partition `p1` of `my_table` where the value of column `k1` is greater than or equal to 3 and the value of column `status` is "outdated"

```
DELETE FROM my_table PARTITION p1
WHERE k1 >= 3 AND status = "outdated";
```

3. Delete rows in partitions `p1` and `p2` of `my_table` where the value of column `k1` is greater than or equal to 3 and the value of column `dt` is between "2024-10-01" and "2024-10-31"

```
DELETE FROM my_table PARTITIONS (p1, p2)
WHERE k1 >= 3 AND dt >= "2024-10-01" AND dt <= "2024-10-31";
```

Delete Using the `USING` Clause

In some scenarios, users need to join multiple tables to accurately determine the data to be deleted. In such cases, the `USING` clause is very useful. The syntax is as follows:

```
DELETE FROM table_name [table_alias]
  [PARTITION partition_name | PARTITIONS (partition_name [, partition_name])]
  [USING additional_tables]
  WHERE condition
```

Required Parameters

- `table_name`: The table from which data needs to be deleted.
- `WHERE condition`: Specifies the condition for selecting the rows to be deleted.

Optional Parameters

- `PARTITION partition_name | PARTITIONS (partition_name [, partition_name])`: Specifies the partition name where the data deletion is to be executed. If the table does not have this partition, an error will be reported.
- `table_alias`: Alias for the table.

Notes

- This form can only be used on UNIQUE KEY model tables.

Example

Using the join result of tables `t2` and `t3`, delete data from `t1`. The table to be deleted only supports the unique model.

```
-- Create tables t1, t2, t3
CREATE TABLE t1
  (id INT, c1 BIGINT, c2 STRING, c3 DOUBLE, c4 DATE)
UNIQUE KEY (id)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1', "function_column.sequence_col" = "c4");

CREATE TABLE t2
  (id INT, c1 BIGINT, c2 STRING, c3 DOUBLE, c4 DATE)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1');

CREATE TABLE t3
  (id INT)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1');

-- Insert data
INSERT INTO t1 VALUES
  (1, 1, '1', 1.0, '2000-01-01'),
  (2, 2, '2', 2.0, '2000-01-02'),
  (3, 3, '3', 3.0, '2000-01-03');

INSERT INTO t2 VALUES
  (1, 10, '10', 10.0, '2000-01-10'),
  (2, 20, '20', 20.0, '2000-01-20'),
  (3, 30, '30', 30.0, '2000-01-30'),
  (4, 4, '4', 4.0, '2000-01-04'),
  (5, 5, '5', 5.0, '2000-01-05');

INSERT INTO t3 VALUES
  (1),
  (4),
  (5);

-- Delete data from t1
DELETE FROM t1
  USING t2 INNER JOIN t3 ON t2.id = t3.id
  WHERE t1.id = t2.id;
```

The expected result is to delete the row in table `t1` where `id` is 1.

```
+----+----+----+--------+------------+
| id | c1 | c2 | c3     | c4         |
+----+----+----+--------+------------+
| 2  | 2  | 2  |    2.0 | 2000-01-02 |
| 3  | 3  | 3  |    3.0 | 2000-01-03 |
+----+----+----+--------+------------+
```

Related Configuration

Timeout Configuration

- `insert_timeout`: Since the delete operation is an SQL command and is considered a special load, the delete statement is affected by the `insert_timeout` value in the Session. You can increase the timeout by `SET insert_timeout = xxx`, where the unit is seconds.

IN Predicate Configuration

- `max_allowed_in_element_num_of_delete`: If the user needs to use a large number of elements in the `in` predicate, this item can be adjusted to increase the allowed element limit. The default value is 1024.

View History

Users can view the history of completed delete records using the `SHOW DELETE` statement.

The syntax is as follows:

```
SHOW DELETE [FROM db_name]
```

Example:

```
mysql> show delete from test_db;
+-----------+---------------+---------------------+-----------------+----------+
| TableName | PartitionName | CreateTime          | DeleteCondition | State    |
+-----------+---------------+---------------------+-----------------+----------+
| empty_tbl | p3            | 2020-04-15 23:09:35 | k1 EQ "1"       | FINISHED |
| test_tbl  | p4            | 2020-04-15 23:09:53 | k1 GT "80"      | FINISHED |
+-----------+---------------+---------------------+-----------------+----------+
2 rows in set (0.00 sec)
```

Performance Recommendations

1. On detail tables (Duplicate Key) and aggregate tables (Aggregate Key), the delete operation executes quickly, but a large number of delete operations in a short period will affect query performance.
2. On primary key tables (Unique Key), the delete operation is converted into an `INSERT INTO` statement. When deleting a large range, the execution speed is slow, but a large number of delete operations in a short period will not significantly affect query performance.

Syntax

For detailed delete syntax, refer to the DELETE syntax manual.

Batch Deletion Based on Load

Batch Deletion Based on Load

The delete operation is a special form of data update. In the primary key model (Unique Key) table, Doris supports deletion by adding a delete sign when loading data.

Compared to the `DELETE` statement, using delete signs offers better usability and performance in the following scenarios:

1. **CDC Scenario**: When synchronizing data from an OLTP database to Doris, Insert and Delete operations in the binlog usually appear alternately. The `DELETE` statement cannot efficiently handle these operations. Using delete signs allows Insert and Delete operations to be processed uniformly, simplifying the CDC code for writing to Doris and improving data load and query performance.
2. **Batch Deletion of Specified Primary Keys**: If a large number of primary keys need to be deleted, using the `DELETE` statement is inefficient. Each execution of `DELETE` generates an empty rowset to record the delete condition and produces a new data version. Frequent deletions or too many delete conditions can severely affect query performance.

Working Principle of Delete Signs

Principle Explanation

- **Table Structure**: The delete sign is stored as a hidden column `__DORIS_DELETE_SIGN__` in the primary key table. When the value of this column is 1, it indicates that the delete sign is effective.
- **Data Load**: Users can specify the mapping condition of the delete sign column in the load task. The usage varies for different load tasks, as detailed in the syntax explanation below.
- **Query**: During the query, Doris FE automatically adds the filter condition `__DORIS_DELETE_SIGN__ != true` in the query plan to filter out data with a delete sign value of 1.
- **Data Compaction**: Doris's background data compaction periodically cleans up data with a delete sign value of 1.

Data Example

Table Structure

Create an example table:

```
CREATE TABLE example_table (
    id BIGINT NOT NULL,
    value STRING
)
UNIQUE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 10
PROPERTIES (
    "replication_num" = "3"
);
```

Use the session variable `show_hidden_columns` to view hidden columns:

```
mysql> set show_hidden_columns=true;

mysql> desc example_table;
+-----------------------+---------+------+-------+---------+-------+
| Field                 | Type    | Null | Key   | Default | Extra |
+-----------------------+---------+------+-------+---------+-------+
| id                    | bigint  | No   | true  | NULL    |       |
| value                 | text    | Yes  | false | NULL    | NONE  |
| __DORIS_DELETE_SIGN__ | tinyint | No   | false | 0       | NONE  |
| __DORIS_VERSION_COL__ | bigint  | No   | false | 0       | NONE  |
+-----------------------+---------+------+-------+---------+-------+
```

Data Load

The table has the following existing data:

```
+------+-------+
| id   | value |
+------+-------+
|    1 | foo   |
|    2 | bar   |
+------+-------+
```

Insert a delete sign for id 1 (this is only for principle demonstration, not introducing various methods of using delete signs in load):

```
mysql> insert into example_table (id, __DORIS_DELETE_SIGN__) values (1, 1);
```

Query

Directly view the data, and you can find that the record with id 1 has been deleted:

```
mysql> select * from example_table;
+------+-------+
| id   | value |
+------+-------+
|    2 | bar   |
+------+-------+
```

Use the session variable `show_hidden_columns` to view hidden columns, and you can see that the row with id 1 has not been actually deleted. Its hidden column `__DORIS_DELETE_SIGN__` value is 1 and is filtered out during the query:

```
mysql> set show_hidden_columns=true;
mysql> select * from example_table;
+------+-------+-----------------------+-----------------------+
| id   | value | __DORIS_DELETE_SIGN__ | __DORIS_VERSION_COL__ |
+------+-------+-----------------------+-----------------------+
|    1 | NULL  |                     1 |                     3 |
```

```
|    2 | bar   |                    0 |                    2 |
+------+-------+----------------------+----------------------+
```

Syntax Explanation

Different load types have different syntax for setting delete signs. Below are the usage syntax for delete signs in various load types.

Load Merge Type Selection

There are several merge types when loading data:

1. **APPEND**: All data is appended to the existing data.
2. **DELETE**: Delete all rows with the same key column values as the loaded data.
3. **MERGE**: Decide whether to APPEND or DELETE based on the DELETE ON condition.

Stream Load

The `Stream Load` syntax is to add a field for setting the delete sign column in the header's columns field, for example: `-H "columns: k1, k2, label_c3" -H "merge_type: [MERGE|APPEND|DELETE]" -H "delete: label_c3=1"`.

For usage examples of Stream Load, please refer to the "Specify merge_type for Delete Operation" and "Specify merge_type for Merge Operation" sections in the Stream Load Manual.

Broker Load

The `Broker Load` syntax is to set the delete sign column field in `PROPERTIES`, as follows:

```
LOAD LABEL db1.label1
(
    [MERGE|APPEND|DELETE] DATA INFILE("hdfs://abc.com:8888/user/palo/test/ml/file1")
    INTO TABLE tbl1
    COLUMNS TERMINATED BY ","
    (tmp_c1,tmp_c2, label_c3)
    SET
    (
        id=tmp_c2,
        name=tmp_c1,
    )
    [DELETE ON label_c3=true]
)
WITH BROKER 'broker'
(
    "username"="user",
    "password"="pass"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Routine Load

The `Routine Load` syntax is to add a mapping in the `columns` field, with the same mapping method as above, as follows:

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
 [WITH MERGE|APPEND|DELETE]
 COLUMNS(k1, k2, k3, v1, v2, label),
 WHERE k1  100 and k2 like "%doris%"
 [DELETE ON label=true]
 PROPERTIES
 (
     "desired_concurrent_number"="3",
     "max_batch_interval" = "20",
     "max_batch_rows" = "300000",
     "max_batch_size" = "209715200",
     "strict_mode" = "false"
 )
 FROM KAFKA
 (
     "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
     "kafka_topic" = "my_topic",
     "kafka_partitions" = "0,1,2,3",
     "kafka_offsets" = "101,0,0,200"
 );
```

Deleting Data with TRUNCATE Command

Truncate

Use this statement to clear data from a specified table and its partitions.

Syntax

```
TRUNCATE TABLE [db.]tbl [PARTITION(p1, p2, ...)];
```

- This statement clears the data but retains the table or partition structure.
- Unlike DELETE, TRUNCATE only performs metadata operations, making it faster and not affecting query performance.
- Data removed by this operation cannot be recovered.
- The table status must be NORMAL, and there should be no ongoing SCHEMA CHANGE operations.
- This command may cause ongoing import tasks to fail.

Examples

1. Clear the table `tbl` in the `example_db` database

```
TRUNCATE TABLE example_db.tbl;
```

2. Clear the `p1` and `p2` partitions of the table `tbl`

```
TRUNCATE TABLE tbl PARTITION(p1, p2);
```

Replacing Atomic Table

Doris supports atomic replacement operations between two tables, applicable only to OLAP tables.

Applicable Scenarios

Sometimes, users need to rewrite table data, but deleting and then importing the data would result in a period of unavailability. In such cases, users can create a new table with the same structure using the `CREATE TABLE LIKE` statement, import the new data into the new table, and then perform an atomic replacement of the old table. For partition-level atomic overwrite operations, refer to the temporary partition documentation.

Syntax

```
ALTER TABLE [db.]tbl1 REPLACE WITH TABLE tbl2
[PROPERTIES('swap' = 'true')];
```

Replace table `tbl1` with table `tbl2`.

If the `swap` parameter is `true`, after the replacement, the data in `tbl1` will be the original data in `tbl2`, and the data in `tbl2` will be the original data in `tbl1`, meaning the data in the two tables will be swapped.

If the `swap` parameter is `false`, after the replacement, the data in `tbl1` will be the original data in `tbl2`, and `tbl2` will be deleted.

Principle

The table replacement function turns the following set of operations into an atomic operation.

Assuming table A is replaced with table B, and `swap` is `true`, the operations are as follows:

1. Rename table B to table A.
2. Rename table A to table B.

If `swap` is `false`, the operations are as follows:

1. Delete table A.
2. Rename table B to table A.

Notes

- If the `swap` parameter is `false`, the replaced table (table A) will be deleted and cannot be recovered.
- The replacement operation can only occur between two OLAP tables and does not check if the table structures are consistent.
- The replacement operation will not change the original permission settings, as permission checks are based on table names.

Temporary Partition

Apache Doris supports adding temporary partitions to partitioned tables. Temporary partitions differ from regular partitions in that they are not retrieved by standard queries and can only be accessed through specific query statements.

- Temporary partition columns must match those of regular partitions and cannot be modified.
- Temporary partition ranges cannot overlap with each other, but they can overlap with regular partition ranges.
- Temporary partition names must be unique and cannot duplicate regular partition names or other temporary partition names.

Use Cases

Temporary partitions are primarily used for:

- **Atomic overwrite operations**: You can rewrite partition data without losing data between deletion and import operations. Create a temporary partition, import new data into it, then atomically replace the original partition through a replace operation. For atomic overwrite operations on non-partitioned tables, see the replace table documentation.
- **Bucket count modification**: If an inappropriate bucket count was used when creating a partition, you can create a new temporary partition with the desired bucket count. Import the regular partition data into the temporary partition using the `INSERT INTO` command, then atomically replace the original partition through a replace operation.
- **Partition merging or splitting**: You can modify partition ranges, such as merging two partitions or splitting a large partition into multiple smaller ones. Create a new temporary partition, import the regular partition data using the `INSERT INTO` command, then atomically replace the original partition through a replace operation.

Adding Temporary Partitions

Use the `ALTER TABLE ADD TEMPORARY PARTITION` statement to add temporary partitions:

```
ALTER TABLE tbl1 ADD TEMPORARY PARTITION tp1 VALUES LESS THAN("2020-02-01");

ALTER TABLE tbl2 ADD TEMPORARY PARTITION tp1 VALUES [("2020-01-01"), ("2020-02-01"));

ALTER TABLE tbl1 ADD TEMPORARY PARTITION tp1 VALUES LESS THAN("2020-02-01")
("replication_num" = "1")
DISTRIBUTED BY HASH(k1) BUCKETS 5;

ALTER TABLE tbl3 ADD TEMPORARY PARTITION tp1 VALUES IN ("Beijing", "Shanghai");

ALTER TABLE tbl4 ADD TEMPORARY PARTITION tp1 VALUES IN ((1, "Beijing"), (1, "Shanghai"));

ALTER TABLE tbl3 ADD TEMPORARY PARTITION tp1 VALUES IN ("Beijing", "Shanghai")
```

```
("replication_num" = "1")
DISTRIBUTED BY HASH(k1) BUCKETS 5;
```

Dropping Temporary Partitions

Use the `ALTER TABLE DROP TEMPORARY PARTITION` statement to drop temporary partitions:

```
ALTER TABLE tbl1 DROP TEMPORARY PARTITION tp1;
```

Replacing Regular Partitions

Use the `ALTER TABLE REPLACE PARTITION` statement to replace regular partitions with temporary partitions:

```
ALTER TABLE tbl1 REPLACE PARTITION (p1) WITH TEMPORARY PARTITION (tp1);

ALTER TABLE tbl1 REPLACE PARTITION (p1, p2) WITH TEMPORARY PARTITION (tp1, tp2, tp3);

ALTER TABLE tbl1 REPLACE PARTITION (p1, p2) WITH TEMPORARY PARTITION (tp1, tp2)
PROPERTIES (
  "strict_range" = "false",
  "use_temp_partition_name" = "true"
);
```

The replace operation supports two optional parameters:

1. `strict_range`

Default value: `true`

For Range partitions: When `true`, the union of all replaced regular partition ranges must exactly match the union of the replacing temporary partition ranges. When `false`, only ensure that the new regular partition ranges don't overlap after replacement.

For List partitions: This parameter is always `true`. The enumeration values of all replaced regular partitions must exactly match the enumeration values of the replacing temporary partitions.

Example 1: Range partitions with matching unions

```
-- The range of partitions p1, p2, p3 to be replaced (=> union):
[10, 20), [20, 30), [40, 50) => [10, 30), [40, 50)

-- The range of replacing partitions tp1, tp2 (=> union):
[10, 30), [40, 45), [45, 50) => [10, 30), [40, 50)

-- Range unions match, so tp1 and tp2 can replace p1, p2, p3.
```

Example 2: Range partitions with non-matching unions

```
-- The range of partition p1 to be replaced (=> union):
[10, 50) => [10, 50)

-- The range of replacing partitions tp1, tp2 (=> union):
```

```
[10, 30), [40, 50) => [10, 30), [40, 50)

-- The union of ranges is not the same. If strict_range is true, tp1 and tp2 cannot replace p1. If it is false, and the ranges of
the two partitions after replacement [10, 30), [40, 50) do not overlap with other regular partitions, they can replace p1.
```

Example 3: List partitions with matching unions

```
-- The enumeration values of partitions p1, p2 to be replaced (=> union):
(1, 2, 3), (4, 5, 6) => (1, 2, 3, 4, 5, 6)

-- The enumeration values of replacing partitions tp1, tp2, tp3 (=> union):
(1, 2, 3), (4), (5, 6) => (1, 2, 3, 4, 5, 6)

-- Enumeration unions match, so tp1, tp2, tp3 can replace p1, p2.
```

Example 4: Multi-column list partitions with matching unions

```
-- The enumeration values of partitions p1, p2, p3 to be replaced (=> union):
(("1","beijing"), ("1", "shanghai")), (("2","beijing"), ("2", "shanghai")), (("3","beijing"), ("3", "shanghai")) =>
(("1","beijing"), ("1", "shanghai"), ("2","beijing"), ("2", "shanghai"), ("3","beijing"), ("3", "shanghai"))

-- The enumeration values of replacing partitions tp1, tp2 (=> union):
(("1","beijing"), ("1", "shanghai")), (("2","beijing"), ("2", "shanghai"), ("3","beijing"), ("3", "shanghai")) => (("1","beijing"),
("1", "shanghai"), ("2","beijing"), ("2", "shanghai"), ("3","beijing"), ("3", "shanghai"))

-- Enumeration unions match, so tp1, tp2 can replace p1, p2, p3.
```

2. `use_temp_partition_name`

Default value: `false`

When `false` and the number of partitions to replace equals the number of replacing partitions, partition names remain unchanged after replacement.

When `true`, partition names after replacement become the names of the replacing partitions.

Example 1: Same partition count

```
ALTER TABLE tbl1 REPLACE PARTITION (p1) WITH TEMPORARY PARTITION (tp1);
```

- With `use_temp_partition_name` defaulting to `false`, the partition name remains `p1` after replacement, but data and properties are replaced with those of `tp1`.
- With `use_temp_partition_name` set to `true`, the partition name becomes `tp1` after replacement, and `p1` no longer exists.

Example 2: Different partition counts

```
ALTER TABLE tbl1 REPLACE PARTITION (p1, p2) WITH TEMPORARY PARTITION (tp1);
```

- With `use_temp_partition_name` defaulting to `false`, but since partition counts differ, this parameter is ignored. After replacement, the partition name becomes `tp1`, and `p1` and `p2` no longer exist.

> ****Important: **After successful partition replacement, the replaced partitions are permanently deleted and cannot be recovered.**

Importing Data into Temporary Partitions

The syntax for importing data into temporary partitions varies slightly depending on the import method:

```
INSERT INTO tbl TEMPORARY PARTITION(tp1, tp2, ...) SELECT ....

curl --location-trusted -u root: -H "label:123" -H "temporary_partitions: tp1, tp2, ..." -T testData
http://host:port/api/testDb/testTbl/_stream_load

LOAD LABEL example_db.label1
(
DATA INFILE("hdfs://hdfs_host:hdfs_port/user/palo/data/input/file")
INTO TABLE my_table
TEMPORARY PARTITION (tp1, tp2, ...)
...
)
WITH BROKER hdfs ("username"="hdfs_user", "password"="hdfs_password");

CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100),
TEMPORARY PARTITIONS(tp1, tp2, ...),
WHERE k1 > 100
PROPERTIES
(...)
FROM KAFKA
(...);
```

Querying Temporary Partitions

```
SELECT ... FROM
tbl1 TEMPORARY PARTITION(tp1, tp2, ...)
JOIN
tbl2 TEMPORARY PARTITION(tp1, tp2, ...)
ON ...
WHERE ...;
```

Interaction with Other Operations

DROP Operations

- **Database/Table DROP**: After dropping a database or table, you can recover it using the Recover command (within time limits), but temporary partitions are not recovered.
- **Regular Partition DROP**: After dropping a regular partition, you can recover it using the Recover command (within time limits). This operation doesn't affect temporary partitions.
- **Temporary Partition DROP**: After dropping a temporary partition, it cannot be recovered using the Recover command.

TRUNCATE Operations

- **Table TRUNCATE**: Truncating a table removes all temporary partitions, and they cannot be recovered.
- **Regular Partition TRUNCATE**: Truncating a regular partition doesn't affect temporary partitions.
- **Temporary Partition TRUNCATE**: The TRUNCATE command cannot be used on temporary partitions.

ALTER Operations

- **Schema Changes**: When a table has temporary partitions, you cannot perform Schema Change, Rollup, or other modification operations.
- **Adding Temporary Partitions**: You cannot add temporary partitions to a table that is undergoing modification operations.

Transaction

A transaction is an operation that contains one or more SQL statements. The execution of these statements must either be completely successful or completely fail. It is an indivisible work unit.

Introduction

Queries and DDL single statements are implicit transactions and are not supported within multi-statement transactions. Each individual write is an implicit transaction by default, and multiple writes can form an explicit transaction. Currently, Doris does not support nested transactions.

Explicit and Implicit Transactions

Explicit Transactions

Explicit transactions require users to actively start, commit transactions. Only insert into values statement is supported in 2.1.

```
BEGIN;
[INSERT INTO VALUES]
COMMIT;
```

Rollback is not supported in 2.1.

Implicit Transactions

Implicit transactions refer to SQL statements that are executed without explicitly adding statements to start and commit transactions before and after the statements.

In Doris, except for Group Commit, each import statement opens a transaction when it starts executing. The transaction is automatically committed after the statement is executed, or automatically rolled back if the statement fails. Each query or DDL statement is also an implicit transaction.

Isolation Level

The only isolation level currently supported by Doris is READ COMMITTED. Under the READ COMMITTED isolation level, a statement sees only data that was committed before the statement began execution. It does not see uncommitted data.

When a single statement is executed, it captures a snapshot of the tables involved at the start of the statement, meaning that a single statement can only see commits from other transactions made before it began execution. Other transactions' commits are not visible during the execution of a single statement.

When a statement is executed inside a multi-statement transaction:

- It sees only data that was committed before the statement began execution. If another transaction commits between the execution of the first and the second statements, two successive statements in the same transaction may see different data.
- Currently, it cannot see changes made by previous statements within the same transaction.

No Duplicates, No Loss

Doris supports mechanisms to ensure no duplicates and no loss during data writes. The Label mechanism ensures no duplicates within a single transaction, while two-phase commit coordinates to prevent duplicates across multiple transactions.

Label Mechanism

Transactions or writes in Doris can be assigned a Label. This Label is typically a user-defined string with some business logic attributes. If not set, a UUID string will be generated internally. The main purpose of a Label is to uniquely identify a transaction or import task and ensure that a transaction or import with the same Label will only execute successfully once. The Label mechanism ensures that data imports are neither lost nor duplicated. If the upstream data source guarantees at-least-once semantics, combined with Doris's Label mechanism, exactly-once semantics can be achieved. Labels are unique within a database.

Doris will clean up Labels based on time and number. By default, if the number of Labels exceeds 2000, cleanup will be triggered. Labels older than three days will also be cleaned up by default. Once a Label is cleaned up, a Label with the same name can execute successfully again, meaning it no longer has deduplication semantics.

Labels are usually set in the format of `business_logic+timestamp`, such as `my_business1_20220330_125000`. This Label typically represents a batch of data generated by the business `my_business1` at `2022-03-30 12:50:00`. By setting Labels this way, the business can query the import task status using the Label to clearly determine whether the batch of data at that time has been successfully imported. If not, the import can be retried using the same Label.

StreamLoad 2PC

StreamLoad 2PC is mainly used to support exactly-once semantics (EOS) when writing to Doris with Flink.

Transaction Operations

Start a Transaction

```
BEGIN;

BEGIN WITH LABEL {user_label};
```

If this statement is executed while the current session is in the middle of a transaction, Doris will ignore the statement, which can also be understood as transactions cannot be nested.

Commit a Transaction

```
COMMIT;
```

Used to commit all modifications made in the current transaction.

Transaction with multiple sql statements

Currently, Doris supports only one way of transaction loading.

Multiple `INSERT INTO VALUES` for one table

Suppose the table schema is:

```
CREATE TABLE `dt` (
    `id` INT(11) NOT NULL,
    `name` VARCHAR(50) NULL,
    `score` INT(11) NULL
) ENGINE=OLAP
UNIQUE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
    "replication_num" = "1"
);
```

Do transaction load:

```
mysql> BEGIN;
Query OK, 0 rows affected (0.01 sec)
{'label':'txn_insert_b55db21aad7451b-b5b6c339704920c5', 'status':'PREPARE', 'txnId':''}

mysql> INSERT INTO dt (id, name, score) VALUES (1, "Emily", 25), (2, "Benjamin", 35), (3, "Olivia", 28), (4, "Alexander", 60), (5,
"Ava", 17);
Query OK, 5 rows affected (0.08 sec)
{'label':'txn_insert_b55db21aad7451b-b5b6c339704920c5', 'status':'PREPARE', 'txnId':'10013'}

mysql> INSERT INTO dt VALUES (6, "William", 69), (7, "Sophia", 32), (8, "James", 64), (9, "Emma", 37), (10, "Liam", 64);
Query OK, 5 rows affected (0.00 sec)
{'label':'txn_insert_b55db21aad7451b-b5b6c339704920c5', 'status':'PREPARE', 'txnId':'10013'}

mysql> COMMIT;
Query OK, 0 rows affected (1.02 sec)
{'label':'txn_insert_b55db21aad7451b-b5b6c339704920c5', 'status':'VISIBLE', 'txnId':'10013'}
```

This method not only achieves atomicity, but also in Doris, it enhances the writing performance of `INSERT INTO VALUES`.

If user enables `Group Commit` and transaction insert at the same time, the transaction insert will work.

QA

- Writing to multiple tables must belong to the same Database; otherwise, you will encounter the error `Transaction insert must be in the same database`

- If the time-consuming from `BEGIN` statement exceeds the timeout configured in Doris, the transaction will be rolled back. Currently, the timeout uses the maximum value of session variables `insert_timeout` and `query_timeout`.
- When using JDBC to connect to Doris for transaction operations, please add `useLocalSessionState=true` in the JDBC URL; otherwise, you may encounter the error `This is in a transaction, only insert, update, delete, commit, rollback is acceptable`.

Stream Load 2PC

1. Enable two-phase commit by setting `two_phase_commit:true` in the HTTP Header.

```
curl --location-trusted -u user:passwd -H "two_phase_commit:true" -T test.txt http://fe_host:http_port/api/{db}/{table}/_stream_load
{
    "TxnId": 18036,
    "Label": "55c8ffc9-1c40-4d51-b75e-f2265b3602ef",
    "TwoPhaseCommit": "true",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 100,
    "NumberLoadedRows": 100,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 1031,
    "LoadTimeMs": 77,
    "BeginTxnTimeMs": 1,
    "StreamLoadPutTimeMs": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 58,
    "CommitAndPublishTimeMs": 0
}
```

2. Trigger the commit operation for a transaction (can be sent to FE or BE).

- Specify the transaction using the Transaction ID:

```
curl -X PUT --location-trusted -u user:passwd -H "txn_id:18036" -H "txn_operation:commit"
http://fe_host:http_port/api/{db}/{table}/_stream_load_2pc
{
    "status": "Success",
    "msg": "transaction [18036] commit successfully."
}
```

- Specify the transaction using the label:

```
curl -X PUT --location-trusted -u user:passwd -H "label:55c8ffc9-1c40-4d51-b75e-f2265b3602ef" -H "txn_operation:commit"
http://fe_host:http_port/api/{db}/{table}/_stream_load_2pc
{
    "status": "Success",
    "msg": "label [55c8ffc9-1c40-4d51-b75e-f2265b3602ef] commit successfully."
}
```

3. Trigger the abort operation for a transaction (can be sent to FE or BE).

- Specify the transaction using the Transaction ID:

```
curl -X PUT --location-trusted -u user:passwd -H "txn_id:18037" -H "txn_operation:abort"
http://fe_host:http_port/api/{db}/{table}/_stream_load_2pc
{
    "status": "Success",
    "msg": "transaction [18037] abort successfully."
}
```

- Specify the transaction using the label:

```
curl -X PUT --location-trusted -u user:passwd -H "label:55c8ffc9-1c40-4d51-b75e-f2265b3602ef" -H "txn_operation:abort"
http://fe_host:http_port/api/{db}/{table}/_stream_load_2pc
{
    "status": "Success",
    "msg": "label [55c8ffc9-1c40-4d51-b75e-f2265b3602ef] abort successfully."
}
```

Broker Load into muti tables with a transaction

All Broker Load tasks are atomic and ensure atomicity even when loading multiple tables within the same task. The Label mechanism can be used to ensure data load without loss or duplication.

The following example demonstrates loading data from HDFS by using wildcard patterns to match two sets of files and load them into two different tables.

```
LOAD LABEL example_db.label2
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/file-10*")
    INTO TABLE `my_table1`
    PARTITION (p1)
    COLUMNS TERMINATED BY ","
    (k1, tmp_k2, tmp_k3)
    SET (
        k2 = tmp_k2 + 1,
        k3 = tmp_k3 + 1
    )
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/file-20*")
    INTO TABLE `my_table2`
    COLUMNS TERMINATED BY ","
    (k1, k2, k3)
)
WITH BROKER hdfs
(
    "username"="hdfs_user",
    "password"="hdfs_password"
);
```

The wildcard pattern is used to match and load two sets of files, `file-10*` and `file-20*`, into `my_table1` and `my_table2` respectively. In the case of `my_table1`, the load is specified to the `p1` partition, and the values of thesecond and third columns in the source file are incremented by 1 before being loaded.

Export Overview

The data export function is used to write the query result set or Doris table data into the specified storage system in the specified file format.

The differences between the export function and the data backup function are as follows:

	Data Export	Data Backup
Final Storage Location	HDFS, Object Storage, Local File System	HDFS, Object Storage
Data Format	Open file formats such as Parquet, ORC, CSV	Doris internal storage format
Execution Speed	Moderate (requires reading data and converting to the target data format)	Fast (no parsing and conversion required, directly upload Doris data files)
Flexibility	Can flexibly define the data to be exported through SQL statements	Only supports table-level full backup
Use Cases	Result set download, data exchange between different systems	Data backup, data migration between Doris clusters

Choosing Export Methods

Doris provides three different data export methods:

- **SELECT INTO OUTFILE**: Supports the export of any SQL result set.
- **EXPORT**: Supports the export of partial or full table data.
- **MySQL DUMP**: Compatible with the MySQL dump command for data export.

The similarities and differences between the three export methods are as follows:

	SELECT INTO OUTFILE	EXPORT	MySQL DUMP
Synchronous/Asynchronous	Synchronous	Asynchronous (submit EXPORT tasks and check task progress via SHOW EXPORT command)	Synchronous
Supports any SQL	Yes	No	No
Export specific partitions	Yes	Yes	No
Export specific tablets	Yes	No	No
Concurrent export	Supported with high concurrency (depends on whether the SQL statement has operators such as ORDER BY that need to be processed on a single node)	Supported with high concurrency (supports tablet-level concurrent export)	Not supported, single-threaded export only
Supported export data formats	Parquet, ORC, CSV	Parquet, ORC, CSV	MySQL Dump proprietary format

	SELECT INTO OUTFILE	EXPORT	MySQL DUMP
Supports exporting external tables	Yes	Partially supported	No
Supports exporting views	Yes	Yes	Yes
Supported export locations	S3, HDFS	S3, HDFS	LOCAL

SELECT INTO OUTFILE

Suitable for the following scenarios:

- Data needs to be exported after complex calculations, such as filtering, aggregation, joins, etc.
- Suitable for scenarios that require synchronous tasks.

EXPORT

Suitable for the following scenarios:

- Large-scale single table export, with simple filtering conditions.
- Scenarios that require asynchronous task submission.

MySQL Dump

Suitable for the following scenarios:

- Compatible with the MySQL ecosystem, requires exporting both table structure and data.
- Only for development testing or scenarios with very small data volumes.

Export File Column Type Mapping

Parquet and ORC file formats have their own data types. Doris's export function can automatically map Doris's data types to the corresponding data types in Parquet and ORC file formats. The CSV format does not have types, all data is output as text.

The following table shows the mapping between Doris data types and Parquet, ORC file format data types:

- ORC

Doris Type	Orc Type
boolean	boolean
tinyint	tinyint
smallint	smallint
int	int
bigint	bigint
largeInt	string
date	string

Doris Type	Orc Type
datev2	string
datetime	string
datetimev2	timestamp
float	float
double	double
char / varchar / string	string
decimal	decimal
struct	struct
map	map
array	array
json	string
variant	string
bitmap	binary
quantile_state	binary
hll	binary

- Parquet

  When Doris is exported to the Parquet file format, the Doris memory data is first converted to the Arrow memory data format, and then written out to the Parquet file format by Arrow.

Doris Type	Arrow Type	Parquet Physical Type	Parquet Logical Type
boolean	boolean	BOOLEAN	
tinyint	int8	INT32	INT_8
smallint	int16	INT32	INT_16
int	int32	INT32	INT_32
bigint	int64	INT64	INT_64
largeInt	utf8	BYTE_ARRAY	UTF8
date	utf8	BYTE_ARRAY	UTF8
datev2	date32	INT32	DATE
datetime	utf8	BYTE_ARRAY	UTF8
datetimev2	timestamp	INT96/INT64	TIMESTAMP(MICROS/MILLIS/SECONDS)
float	float32	FLOAT	
double	float64	DOUBLE	
char / varchar / string	utf8	BYTE_ARRAY	UTF8

Doris Type	Arrow Type	Parquet Physical Type	Parquet Logical Type
decimal	decimal128	FIXED_LEN_BYTE_ARRAY	DECIMAL(scale, precision)
struct	struct		Parquet Group
map	map		Parquet Map
array	list		Parquet List
json	utf8	BYTE_ARRAY	UTF8
variant	utf8	BYTE_ARRAY	UTF8
bitmap	binary	BYTE_ARRAY	
quantile_state	binary	BYTE_ARRAY	
hll	binary	BYTE_ARRAY	

> Note: In versions 2.1.11 and 3.0.7, you can specify the `parquet.enable_int96_timestamps` property to determine whether Doris's datetimev2 type uses Parquet's INT96 storage or INT64. INT96 is used by default. However, INT96 has been deprecated in the Parquet standard and is only used for compatibility with some older systems (such as versions before Hive 4.0).

Export Manual | Export

Export Manual

This document will introduce how to use the `EXPORT` command to export the data stored in Doris.

`Export` is a function provided by Doris for asynchronously exporting data. This function can export the data of the tables or partitions specified by users in the specified file format to the target storage systems, including object storage, HDFS or local file system.

`Export` is an asynchronously executed command. Once the command is executed successfully, it will return the result immediately. Users can view the detailed information of the Export task through the `Show Export` command.

For the detailed introduction of the `EXPORT` command, please refer to: EXPORT

Regarding how to choose between `SELECT INTO OUTFILE` and `EXPORT`, please refer to Export Overview.

Applicable Scenarios

`Export` is applicable to the following scenarios:

- Exporting a single table with a large amount of data, only requiring simple filtering conditions.
- Scenarios where tasks need to be submitted asynchronously.

The following limitations should be noted when using `Export`:

- Currently, exporting in compressed text file formats is not supported.
- Exporting the result set of `Select` is not supported. If you need to export the `Select` result set, please use OUTFILE Export.

Quick Start

Table Creation and Data Import

```
CREATE TABLE IF NOT EXISTS tbl (
    `c1` int(11) NULL,
    `c2` string NULL,
    `c3` bigint NULL
)
DISTRIBUTED BY HASH(c1) BUCKETS 20
PROPERTIES("replication_num" = "1");


insert into tbl values
    (1, 'doris', 18),
    (2, 'nereids', 20),
    (3, 'pipelibe', 99999),
    (4, 'Apache', 122123455),
    (5, null, null);
```

Create an Export Job

Export to HDFS

Export all data from the `tbl` table to HDFS. Set the file format of the export job to csv (the default format) and set the column delimiter to `,`.

```
EXPORT TABLE tbl
TO "hdfs://host/path/to/export/"
PROPERTIES
(
    "line_delimiter" = ","
)
with HDFS (
    "fs.defaultFS"="hdfs://hdfs_host:port",
    "hadoop.username" = "hadoop"
);
```

Export to Object Storage

Export all the data in the `tbl` table to the object storage, set the file format of the export job to csv (the default format), and set the column delimiter to `,`.

```
EXPORT TABLE tbl TO "s3://bucket/export/export_"
PROPERTIES (
    "line_delimiter" = ","
) WITH s3 (
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx"
);
```

View Export Jobs

After submitting a job, you can query the status of the export job via the SHOW EXPORT command. An example of the result is as follows:

```
mysql> show export\G
*************************** 1. row ***************************
      JobId: 143265
      Label: export_0aa6c944-5a09-4d0b-80e1-cb09ea223f65
      State: FINISHED
   Progress: 100%
   TaskInfo: {"partitions":
[],"parallelism":5,"data_consistency":"partition","format":"csv","broker":"S3","column_separator":"\t","line_delimiter":"\n","max_fi
le_size":"2048MB","delete_existing_files":"","with_bom":"false","db":"tpch1","tbl":"lineitem"}
       Path: s3://bucket/export/export_
 CreateTime: 2024-06-11 18:01:18
  StartTime: 2024-06-11 18:01:18
 FinishTime: 2024-06-11 18:01:31
    Timeout: 7200
   ErrorMsg: NULL
OutfileInfo: [
  [
```

```
    {
      "fileNumber": "1",
      "totalRows": "6001215",
      "fileSize": "747503989",
      "url": "s3://bucket/export/export_6555cd33e7447c1-baa9568b5c4eb0ac_*"
    }
  ]
]
1 row in set (0.00 sec)
```

For the detailed usage of the `show export` command and the meaning of each column in the returned results, please refer to SHOW EXPORT.

Cancel Export Jobs

After submitting an Export job, the export job can be cancelled via the CANCEL EXPORT command before the Export task succeeds or fails. An example of the cancellation command is as follows:

```
CANCEL EXPORT FROM dbName WHERE LABEL like "%export_%";
```

Export Instructions

Export Data Sources

`EXPORT` currently supports exporting the following types of tables or views:

- Internal tables in Doris
- Logical views in Doris
- Tables in External Catalog

Export Data Storage Locations

`Export` currently supports exporting to the following storage locations:

- Object storage: Amazon S3, COS, OSS, OBS, Google GCS
- HDFS

Export File Types

`EXPORT` currently supports exporting to the following file formats:

- Parquet
- ORC
- csv
- csv_with_names
- csv_with_names_and_types

Examples

Export to an HDFS Cluster with High Availability Enabled

If the HDFS has high availability enabled, HA information needs to be provided. For example:

```
EXPORT TABLE tbl
TO "hdfs://HDFS8000871/path/to/export_"
PROPERTIES
(
    "line_delimiter" = ","
)
with HDFS (
    "fs.defaultFS" = "hdfs://HDFS8000871",
    "hadoop.username" = "hadoop",
    "dfs.nameservices" = "your-nameservices",
    "dfs.ha.namenodes.your-nameservices" = "nn1,nn2",
    "dfs.namenode.rpc-address.HDFS8000871.nn1" = "ip:port",
    "dfs.namenode.rpc-address.HDFS8000871.nn2" = "ip:port",
    "dfs.client.failover.proxy.provider.HDFS8000871" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

Export to an HDFS Cluster with High Availability and Kerberos Authentication Enabled

If the HDFS cluster has both high availability enabled and Kerberos authentication enabled, you can refer to the following SQL statements:

```
EXPORT TABLE tbl
TO "hdfs://HDFS8000871/path/to/export_"
PROPERTIES
(
    "line_delimiter" = ","
)
with HDFS (
    "fs.defaultFS"="hdfs://hacluster/",
    "hadoop.username" = "hadoop",
    "dfs.nameservices"="hacluster",
    "dfs.ha.namenodes.hacluster"="n1,n2",
    "dfs.namenode.rpc-address.hacluster.n1"="192.168.0.1:8020",
    "dfs.namenode.rpc-address.hacluster.n2"="192.168.0.2:8020",
    "dfs.client.failover.proxy.provider.hacluster"="org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider",
    "dfs.namenode.kerberos.principal"="hadoop/_HOST@REALM.COM",
    "hadoop.security.authentication"="kerberos",
    "hadoop.kerberos.principal"="doris_test@REALM.COM",
    "hadoop.kerberos.keytab"="/path/to/doris_test.keytab"
);
```

Specify Partition for Export

The export job supports exporting only some partitions of the internal tables in Doris. For example, only export partitions p1 and p2 of the `test` table.

```
EXPORT TABLE test
PARTITION (p1,p2)
TO "s3://bucket/export/export_"
PROPERTIES (
```

```
    "columns" = "k1,k2"
) WITH s3 (
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx"
);
```

Filter Data During Export

The export job supports filtering data according to predicate conditions during the export process, exporting only the data that meets the conditions. For example, only export the data that satisfies the condition `k1 < 50`.

```
EXPORT TABLE test
WHERE k1 < 50
TO "s3://bucket/export/export_"
PROPERTIES (
    "columns" = "k1,k2",
    "column_separator"=","
) WITH s3 (
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx"
);
```

Export External Table Data

The export job supports the data of tables in External Catalog.

```
-- Create a hive catalog
CREATE CATALOG hive_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://172.0.0.1:9083'
);

-- Export hive table
EXPORT TABLE hive_catalog.sf1.lineitem TO "s3://bucket/export/export_"
PROPERTIES(
    "format" = "csv",
    "max_file_size" = "1024MB"
) WITH s3 (
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx"
);
```

Clearing the Export Directory Before Exporting

```
EXPORT TABLE test TO "s3://bucket/export/export_"
PROPERTIES (
    "format" = "parquet",
    "max_file_size" = "512MB",
    "delete_existing_files" = "true"
) WITH s3 (
```

```
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx"
);
```

If `"delete_existing_files" = "true"` is set, the export job will first delete all files and directories under the `s3://bucket/export/` directory, and then export the data to this directory.

If you want to use the `delete_existing_files` parameter, you also need to add the configuration `enable_delete_existing_files = true` in the `fe.conf` and restart the FE. Only then will the `delete_existing_files` take effect. This operation will delete the data of the external system and is a high-risk operation. Please ensure the permissions and data security of the external system on your own.

Setting the Size of Exported Files

The export job supports setting the size of the export file. If the size of a single file exceeds the set value, it will be divided into multiple files for export according to the specified size.

```
EXPORT TABLE test TO "s3://bucket/export/export_"
PROPERTIES (
    "format" = "parquet",
    "max_file_size" = "512MB"
) WITH s3 (
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx"
);
```

By setting `"max_file_size" = "512MB"`, the maximum size of a single exported file is 512MB.

`max_file_size` cannot be less than 5MB and cannot be greater than 2GB.

In versions 2.1.11 and 3.0.7, the maximum limit of 2GB was removed, leaving only the minimum limit of 5MB.

Notice

- Export Data Volume

  It is not recommended to export a large amount of data at one time. The recommended maximum export data volume for an Export job is several tens of gigabytes. Excessive exports will lead to more junk files and higher retry costs. If the table data volume is too large, it is recommended to export by partitions.

  In addition, the Export job will scan data and occupy IO resources, which may affect the query latency of the system.

- Export File Management

  If the Export job fails, the files that have already been generated will not be deleted, and users need to delete them manually.

- Export Timeout

If the amount of exported data is very large and exceeds the export timeout period, the Export task will fail. At this time, you can specify the `timeout` parameter in the Export command to increase the timeout period and retry the Export command.

- Export Failure

  During the operation of the Export job, if the FE restarts or switches the master, the Export job will fail, and the user needs to resubmit it. You can check the status of the Export task through the `show export` command.

- Number of Exported Partitions

  The maximum number of partitions allowed to be exported by an Export Job is 2000. You can add the parameter `maximum_number_of_export_partitions` in fe.conf and restart the FE to modify this setting.

- Data Integrity

  After the export operation is completed, it is recommended to verify whether the exported data is complete and correct to ensure the quality and integrity of the data.

Appendix

Basic Principle

The underlying layer of the Export task is to execute the `SELECT INTO OUTFILE` SQL statement. After a user initiates an Export task, Doris will construct one or more `SELECT INTO OUTFILE` execution plans according to the table to be exported by Export, and then submit these `SELECT INTO OUTFILE` execution plans to the Job Schedule task scheduler of Doris. The Job Schedule task scheduler will automatically schedule and execute these tasks.

Exporting to the Local File System

The function of exporting to the local file system is disabled by default. This function is only used for local debugging and development, and should not be used in the production environment.

If you want to enable this function, please add `enable_outfile_to_local=true` in the `fe.conf` and restart the FE.

Example: Export all the data in the `tbl` table to the local file system, set the file format of the export job to csv (the default format), and set the column delimiter to `,`.

```
EXPORT TABLE tbl TO "file:///path/to/result_"
PROPERTIES (
  "format" = "csv",
  "line_delimiter" = ","
);
```

This function will export and write data to the disk of the node where the BE is located. If there are multiple BE nodes, the data will be scattered on different BE nodes according to the concurrency of the export task, and each node will have a part of the data.

As in this example, a set of files similar to `result_7052bac522d840f5-972079771289e392_0.csv` will eventually be produced under `/path/to/` of the BE node.

The specific BE node IP can be viewed in the `OutfileInfo` column in the `SHOW EXPORT` result, such as:

```
[
    [
        {
            "fileNumber": "1",
            "totalRows": "0",
            "fileSize": "8388608",
            "url": "file:///172.20.32.136/path/to/result_7052bac522d840f5-972079771289e392_*"
        }
    ],
    [
        {
            "fileNumber": "1",
            "totalRows": "0",
            "fileSize": "8388608",
            "url": "file:///172.20.32.137/path/to/result_22aba7ec933b4922-ba81e5eca12bf0c2_*"
        }
    ]
]
```

> **This function is not applicable to the production environment, and please ensure the permissions of the export directory and data security on your own.**

SELECT INTO OUTFILE

This document will introduce how to use the `SELECT INTO OUTFILE` command to perform the export operation of query results.

The `SELECT INTO OUTFILE` command exports the result data of the `SELECT` part to the target storage system in the specified file format, including object storage or HDFS.

The `SELECT INTO OUTFILE` is a synchronous command, the return of the command means the export is finished. If the export is successful, information such as the number, size, and path of the exported files will be returned. If the export fails, error information will be returned.

Regarding how to choose between `SELECT INTO OUTFILE` and `EXPORT`, please refer to Export Overview.

For a detailed introduction to the `SELECT INTO OUTFILE` command, please refer to: SELECT INTO OUTFILE

Applicable Scenarios

The `SELECT INTO OUTFILE` is applicable to the following scenarios:

- When the exported data needs to go through complex calculation logic, such as filtering, aggregation, joining, etc.
- It is suitable for scenarios where synchronous tasks need to be executed.

The following limitations should be noted when using the `SELECT INTO OUTFILE`:

- Compression formats for text is not supported.
- The pipeline engine in version 2.1 does not support concurrent exports.

Quick Start

Creating Tables and Importing Data

```
CREATE TABLE IF NOT EXISTS tbl (
    `c1` int(11) NULL,
    `c2` string NULL,
    `c3` bigint NULL
)
DISTRIBUTED BY HASH(c1) BUCKETS 20
PROPERTIES("replication_num" = "1");


insert into tbl values
    (1, 'doris', 18),
    (2, 'nereids', 20),
    (3, 'pipelibe', 99999),
    (4, 'Apache', 122123455),
    (5, null, null);
```

Export to HDFS

Export the query results to the directory `hdfs://path/to/` and specify the export format as Parquet:

```
SELECT c1, c2, c3 FROM tbl
INTO OUTFILE "hdfs://ip:port/path/to/result_"
FORMAT AS PARQUET
PROPERTIES
(
    "fs.defaultFS" = "hdfs://ip:port",
    "hadoop.username" = "hadoop"
);
```

Export to Object Storage

Export the query results to the `s3://bucket/export/` directory in the s3 storage, specify the export format as ORC, and information such as `sk` (secret key) and `ak` (access key) needs to be provided.

```
SELECT * FROM tbl
INTO OUTFILE "s3://bucket/export/result_"
FORMAT AS ORC
PROPERTIES(
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx"
);
```

Export Instructions

Export Desctination

The `SELECT INTO OUTFILE` currently supports exporting to the following storage locations:

- Object Storage: Amazon S3, COS, OSS, OBS, Google GCS
- HDFS

Supported File Types

The `SELECT INTO OUTFILE` currently supports exporting the following file formats:

- Parquet
- ORC
- csv
- csv_with_names
- csv_with_names_and_types

Export concurrency

You can enable concurrent export through the session variable `enable_parallel_outfile`.

```
SET enable_parallel_outfile=true;
```

Concurrent export will use multi-node and multi-thread to export result data to improve the overall export throughout. However, concurrent export may generate more files.

Note that some queries cannot perform concurrent export even if this variable is turned on, such as queries containing global sorting. If the number of rows returned by the export command is greater than 1, it means that concurrent export is enabled.

Export Examples

Export to an HDFS Cluster with High Availability Enabled

If HDFS has high availability enabled, HA (High Availability) information needs to be provided, such as:

```
SELECT c1, c2, c3 FROM tbl
INTO OUTFILE "hdfs://HDFS8000871/path/to/result_"
FORMAT AS PARQUET
PROPERTIES
(
    "fs.defaultFS" = "hdfs://HDFS8000871",
    "hadoop.username" = "hadoop",
    "dfs.nameservices" = "your-nameservices",
    "dfs.ha.namenodes.your-nameservices" = "nn1,nn2",
    "dfs.namenode.rpc-address.HDFS8000871.nn1" = "ip:port",
    "dfs.namenode.rpc-address.HDFS8000871.nn2" = "ip:port",
    "dfs.client.failover.proxy.provider.HDFS8000871" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

Export to an HDFS Cluster with High Availability and Kerberos Authentication Enabled

If the HDFS cluster has high availability enabled and Kerberos authentication is enabled, you can refer to the following SQL statement:

```
SELECT * FROM tbl
INTO OUTFILE "hdfs://path/to/result_"
FORMAT AS PARQUET
PROPERTIES
(
    "fs.defaultFS"="hdfs://hacluster/",
    "hadoop.username" = "hadoop",
    "dfs.nameservices"="hacluster",
    "dfs.ha.namenodes.hacluster"="n1,n2",
    "dfs.namenode.rpc-address.hacluster.n1"="192.168.0.1:8020",
    "dfs.namenode.rpc-address.hacluster.n2"="192.168.0.2:8020",
    "dfs.client.failover.proxy.provider.hacluster"="org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider",
    "dfs.namenode.kerberos.principal"="hadoop/_HOST@REALM.COM",
    "hadoop.security.authentication"="kerberos",
    "hadoop.kerberos.principal"="doris_test@REALM.COM",
    "hadoop.kerberos.keytab"="/path/to/doris_test.keytab"
);
```

Generating an Export Success Indicator File

The `SELECT INTO OUTFILE` command is a synchronous command. Therefore, it is possible that the task connection is disconnected during the execution of the SQL, making it impossible to know whether the

exported data has ended normally or is complete. At this time, you can use the `success_file_name` parameter to require that a file indicator be generated in the directory after a successful export.

Similar to Hive, users can judge whether the export has ended normally and whether the files in the export directory are complete by checking whether there is a file specified by the `success_file_name` parameter in the export directory.

For example: Export the query results of the select statement to the object storage: `s3://bucket/export/`. Specify the export format as `csv`. Specify the name of the export success indicator file as `SUCCESS`. After the export is completed, a indicator file will be generated.

```
SELECT k1,k2,v1 FROM tbl1 LIMIT 100000
INTO OUTFILE "s3://bucket/export/result_"
FORMAT AS CSV
PROPERTIES
(
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx",
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "success_file_name" = "SUCCESS"
);
```

After the export is completed, one more file will be written, and the file name of this file is `SUCCESS`.

Clearing the Export Directory Before Exporting

```
SELECT * FROM tbl1
INTO OUTFILE "s3://bucket/export/result_"
FORMAT AS CSV
PROPERTIES
(
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx",
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "delete_existing_files" = "true"
);
```

If `"delete_existing_files" = "true"` is set, the export job will first delete all files and directories under the `s3://bucket/export/` directory, and then export the data to this directory.

If you want to use the `delete_existing_files` parameter, you also need to add the configuration `enable_delete_existing_files = true` in `fe.conf` and restart the FE. Only then will the `delete_existing_files` parameter take effect. This operation will delete the data of the external system, which is a high-risk operation. Please ensure the permissions and data security of the external system on your own.

Setting the Size of Exported Files

```
SELECT * FROM tbl
INTO OUTFILE "s3://path/to/result_"
FORMAT AS ORC
PROPERTIES(
    "s3.endpoint" = "xxxxx",
    "s3.region" = "xxxxx",
    "s3.secret_key"="xxxx",
    "s3.access_key" = "xxxxx",
    "max_file_size" = "2048MB"
);
```

Since `"max_file_size" = "2048MB"` is specified, if the final generated file is not larger than 2GB, there will be only one file. If it is larger than 2GB, there will be multiple files.

Notice

1. Export Data Volume and Export Efficiency

   The `SELECT INTO OUTFILE` function is essentially executing an SQL query command. If concurrent exports are not enabled, the query results are exported by a single BE node in a single thread. Therefore, the entire export time includes the time consumed by the query itself and the time consumed by writing out the final result set. Enabling concurrent exports can reduce the export time.

2. Export Timeout

   The timeout period of the export command is the same as that of the query. If the export data times out due to a large amount of data, you can set the session variable `query_timeout` to appropriately extend the query timeout period.

3. Management of Exported Files

   Doris does not manage the exported files. Whether the files are successfully exported or left over after a failed export, users need to handle them on their own.

   In addition, the `SELECT INTO OUTFILE` command does not check whether files or file paths exist. Whether the `SELECT INTO OUTFILE` command will automatically create paths or overwrite existing files is completely determined by the semantics of the remote storage system.

4. If the Query Result Set Is Empty

   For an export with an empty result set, an empty file will still be generated.

5. File Splitting

   File splitting ensures that a row of data is completely stored in a single file. Therefore, the size of the file is not strictly equal to `max_file_size`.

6. Functions with Non-visible Characters

   For some functions whose output is non-visible characters, such as BITMAP and HLL types, when exported to the CSV file format, the output is `\N`.

Appendix

Export to Local File System

The function of exporting to the local file system is turned off by default. This function is only used for local debugging and development, and should not be used in the production environment.

If you want to enable this function, please add `enable_outfile_to_local=true` in `fe.conf` and restart the FE.

Example: Export all the data in the tbl table to the local file system, set the file format of the export job to csv (the default format), and set the column separator to `,`.

```
SELECT c1, c2 FROM db.tbl
INTO OUTFILE "file:///path/to/result_"
FORMAT AS CSV
PROPERTIES(
    "column_separator" = ","
);
```

This function will export and write data to the disk of the node where the BE is located. If there are multiple BE nodes, the data will be scattered on different BE nodes according to the concurrency of the export task, and each node will have a part of the data.

As in this example, a set of files similar to `result_c6df5f01bd664dde-a2168b019b6c2b3f_0.csv` will eventually be produced under `/path/to/` of the BE node.

The specific BE node IP will be displayed in the returned results, such as:

```
+------------+-----------+----------+--------------------------------------------------------------------------------+
| FileNumber | TotalRows | FileSize | URL                                                                            |
+------------+-----------+----------+--------------------------------------------------------------------------------+
|          1 |   1195072 |  4780288 | file:///172.20.32.136/path/to/result_c6df5f01bd664dde-a2168b019b6c2b3f_* |
|          1 |   1202944 |  4811776 | file:///172.20.32.136/path/to/result_c6df5f01bd664dde-a2168b019b6c2b40_* |
|          1 |   1198880 |  4795520 | file:///172.20.32.137/path/to/result_c6df5f01bd664dde-a2168b019b6c2b43_* |
|          1 |   1198880 |  4795520 | file:///172.20.32.137/path/to/result_c6df5f01bd664dde-a2168b019b6c2b45_* |
+------------+-----------+----------+--------------------------------------------------------------------------------+
```

> **This function is not suitable for the production environment, and please ensure the permissions and data security of the export directory on your own.**

Using MySQL Dump

Doris has supported exporting data or table structures through the `mysqldump` tool after version 0.15

Examples

Export

1. Export the table1 table in the test database: `mysqldump -h127.0.0.1 -P9030 -uroot --no-tablespaces --databases test --tables table1`
2. Export the table1 table structure in the test database: `mysqldump -h127.0.0.1 -P9030 -uroot --no-tablespaces --databases test --tables table1 --no-data`
3. Export all tables in the test1, test2 database: `mysqldump -h127.0.0.1 -P9030 -uroot --no-tablespaces --databases test1 test2`
4. Export all databases and tables `mysqldump -h127.0.0.1 -P9030 -uroot --no-tablespaces --all-databases` For more usage parameters, please refer to the manual of `mysqldump`

Import

The results exported by `mysqldump` can be redirected to a file, which can then be imported into Doris through the source command `source filename.sql`

Notice

1. Since there is no concept of tablespace in mysql in Doris, add the `--no-tablespaces` parameter when using `mysqldump`
2. Using mysqldump to export data and table structure is only used for development and testing or when the amount of data is small. Do not use it in a production environment with a large amount of data.

Best Practices | Export

Best Practices

This document mainly introduces how to determine whether resource utilization is reasonable during data export operations, and how to adjust resource utilization to achieve better data export efficiency.

SELECT INTO OUTFILE

1. Enable parallel export

   The number of rows returned by `SELECT INTO OUTFILE` represents the number of parallel Writers. The more Writers, the higher the export concurrency, but the number of output files will also increase. If you find there is only one Writer, you can try enabling the parallel export feature.

   `SET enable_parallel_outfile=true`

   After enabling, the `SELECT INTO OUTFILE` operation generates the corresponding number of Writers based on the query's parallelism. The query parallelism is controlled by the session variable `parallel_pipeline_task_num`. By default, it's half of the number of CPU cores per BE.

   For example, in a cluster with 3 BE nodes, where each node has 8 CPU cores. When parallel export is enabled, it will generate (4*3=) 12 Writers.

   Note that even if `enable_parallel_outfile` is enabled, not all queries can be exported in parallel. For example, if the query contains global sorting and aggregation semantics, it cannot be exported in parallel. For example:

   ```
   SELECT * FROM table ORDER BY id;

   SELECT SUM(cost) FROM table;
   ```

2. Determine export speed

   Each row result returned by `SELECT INTO OUTFILE` contains the time (in seconds) and speed (in KB/s) of the corresponding Writer's output.

   Adding up the speeds of multiple Writers on the same node gives you the write speed of a single node. You can compare this speed with disk bandwidth (e.g., when exporting to local) or network bandwidth (e.g., when exporting to object storage) to see if it has reached the bandwidth bottleneck.

Export

1. Determine export execution status from the return results

   The Export command essentially breaks down the task into multiple `SELECT INTO OUTFILE` clauses for execution.

The results returned by the `SHOW EXPORT` command contain a JSON string, which is a two-dimensional array. The first dimension represents the number of concurrent threads in Export, with the number of concurrent threads representing how many Outfile statements were initiated concurrently. The second dimension represents the return results of a single Outfile statement. Example:


```
[
    [
        {
            "fileNumber": "1",
            "totalRows": "640321",
            "fileSize": "350758307",
            "url": "file:///127.0.0.1/mnt/disk2/ftw/tmp/export/exp_59fd917c43874adc-9b1c3e9cd6e655be_*",
            "writeTime": "17.989",
            "writeSpeed": "19041.66"
        },
        {...},
        {...},
        {...}
    ],
    [
        {
            "fileNumber": "1",
            "totalRows": "646609",
            "fileSize": "354228704",
            "url": "file:///127.0.0.1/mnt/disk2/ftw/tmp/export/exp_c75b9d4b59bf4943-92eb94a7b97e46cb_*",
            "writeTime": "17.249",
            "writeSpeed": "20054.64"
        },
        {...},
        {...},
        {...}
    ]
]
```


In the above example, 2 Outfile commands were initiated. Each command has 4 Writers writing concurrently.

By adjusting the `parallelism` parameter in the Export command properties, you can control the number of concurrent Outfile operations, thereby controlling the concurrency level.

2. Parameters affecting parallelism

   The parallelism of Export jobs depends on two parameters:

   - `parallelism`

     Used to set the maximum number of Outfile commands to split into.

   - `data_consistency`

     Whether to split Outfile commands within partitions. This parameter defaults to `partition`, which means no further splitting within partitions. That is, the number of Outfile commands will only be less than or equal to the number of partitions involved. If set to `none`, a partition will be further split, which can improve concurrency, but if the partition is being written to, it may sacrifice export consistency (i.e., different Outfile commands for the same partition may export different versions of data from that partition).

For details, please refer to Export Command Manual

- `async_task_consumer_thread_num`

  This is an FE configuration parameter that indicates the number of Export Tasks that the current cluster can run concurrently. The default value is 64. An Export Job is split into multiple Export Tasks based on concurrency. All Export Tasks share this threshold. To increase the overall number of concurrent export tasks that the cluster can execute, increase this parameter and restart the FE node.

MySQL Compatibility

Doris is highly compatible with MySQL syntax and supports standard SQL. However, there are several differences between Doris and MySQL, as outlined below.

Data Types

Numeric Types

Type	MySQL	Doris
Boolean		
Bit		Not supported
Tinyint		
Smallint		
Mediumint		- Not supported
Int		
Bigint		
Largeint	- Not supported	
Decimal		
Float/Double		

Date Types

Type	MySQL	Doris
Date		
DateTime		
Timestamp		- Not supported
Time		- Not supported
Year		- Not supported

String Types

Type	MySQL	Doris
Char		

Type	MySQL	Doris
Varchar		
String	- Not supported	
Binary		- Not supported
Varbinary		
Blob		- Not supported
Text		- Not supported
Enum		- Not supported
Set		- Not supported

JSON Type

Type	MySQL	Doris
JSON	Supported	Supported

Doris unique data type

Doris has several unique data types. Here are the details:

- **HyperLogLog**

  HLL (HyperLogLog) is a data type that cannot be used as a key column. In an aggregate model table, the corresponding aggregation type for HLL is HLL_UNION. The length and default value do not need to be specified. The length is controlled internally based on the data aggregation level. HLL columns can only be queried or used with `HLL_UNION_AGG`, `HLL_RAW_AGG`, `HLL_CARDINALITY`, `HLL_HASH`, and other related functions.

  HLL is used for approximate fuzzy deduplication and performs better than count distinct when dealing with large amounts of data. The typical error rate of HLL is around 1%, sometimes reaching up to 2%.

- **Bitmap**

  Bitmap is a data type that cannot be used as a key column. In aggregate model table, the corresponding aggregation type for BITMAP is BITMAP_UNION. Similar to HLL, the length and default values do not need to be specified, and the length is controlled internally based on the data aggregation level. Bitmap columns can only be queried or used with functions like `BITMAP_UNION_COUNT`, `BITMAP_UNION`, `BITMAP_HASH`, `BITMAP_HASH64` and others.

  Using BITMAP in traditional scenarios may impact loading speed, but it generally performs better than Count Distinct when dealing with large amounts of data. Please note that in real-time scenarios, using BITMAP without a global dictionary and with bitmap_hash() function may introduce an error of around 0.1%. If this error is not acceptable, you can use bitmap_hash64 instead.

- **QUANTILE_PERCENT**

  QUANTILE_STATE is a data type that cannot be used as a key column. In an aggregate model table, the corresponding aggregation type for QUANTILE_STATE is QUANTILE_UNION. The length and default value do not need to be specified, and the length is controlled internally based on the data aggregation level. QUANTILE_STATE columns can only be queried or used with functions like `QUANTILE_PERCENT`, `QUANTILE_UNION`, `TO_QUANTILE_STATE` and others.

  QUANTILE_STATE is used for calculating approximate quantile values. During import, it performs pre-aggregation on the same key with different values. When the number of values does not exceed 2048, it stores all the data in detail. When the number of values exceeds 2048, it uses the TDigest algorithm to aggregate (cluster) the data and save the centroids of the clusters.

- **Array<T>**

  Array is a data type in Doris that represents an array composed of elements of type T. It cannot be used as a key column.

- **MAP<K, V>**

  MAP is a data type in Doris that represents a map composed of elements of types K and V.

- **STRUCT<field_name:field_type,...>**

  A structure (STRUCT) is composed of multiple fields. It can also be identified as a collection of multiple columns.

  - field_name: The identifier of the field, which must be unique.
  - field_type: The type of field.

- **Agg_State**

  AGG_STATE is a data type in Doris that cannot be used as a key column. During table creation, the signature of the aggregation function needs to be declared.

  The length and default value do not need to be specified, and the actual storage size depends on the implementation of the function.

  AGG_STATE can only be used in combination with STATE / MERGE/ UNION functions from the SQL manual for aggregators.

Syntax

DDL

01 Create Table Syntax in Doris

```
CREATE TABLE [IF NOT EXISTS] [database.]table
(
    column_definition_list
    [, index_definition_list]
)
```

```
[engine_type]
[keys_type]
[table_comment]
[partition_info]
distribution_desc
[rollup_list]
[properties]
[extra_properties]
```

02 Differences with MySQL

Parameter	Differences from MySQL
Column_definition_list	- Field list definition: The basic syntax is similar to MySQL but includes an additional operation for aggregate types. - The aggregate type operation primarily supports Aggregate. - When creating a table, MySQL allows adding constraints like Index (e.g., Primary Key, Unique Key) after the field list definition, while Doris supports these constraints and computations by defining data models.
Index_definition_list	- Index list definition: The basic syntax is similar to MySQL, supporting bitmap indexes, inverted indexes, and N-Gram indexes, but Bloom filter indexes are set through properties. - MySQL supports B+Tree and Hash indexes.
Engine_type	- Table engine type: Optional. - The currently supported table engine is mainly the OLAP native engine. - MySQL supports storage engines such as Innodb, MyISAM, etc.
Keys_type	- Data model: Optional. - Supported types include: 1) DUPLICATE KEY (default): The specified columns are sort columns. 2) AGGREGATE KEY: The specified columns are dimension columns. 3) UNIQUE KEY: The specified columns are primary key columns. - MySQL does not have the concept of a data model.
Table_comment	Table comment
Partition_info	- Partitioning algorithm: Optional. Doris supported partitioning algorithms include: - LESS THAN: Only defines the upper bound of partitions. The lower bound is determined by the upper bound of the previous partition. - FIXED RANGE: Defines left-closed and right-open intervals for partitions. - MULTI RANGE: Creates multiple RANGE partitions in bulk, defining left-closed and right-open intervals, setting time units and steps. Time units support years, months, days, weeks, and hours. MySQL supports algorithms such as Hash, Range, List, Key. MySQL also supports subpartitions, with only Hash and Key supported for subpartitions.
Distribution_desc	- Bucketing algorithm: Required. Includes: 1) Hash bucketing syntax: DISTRIBUTED BY HASH (k1[,k2 …]) [BUCKETS num|auto]. Description: Uses specified key columns for hash bucketing. 2) Random bucketing syntax: DISTRIBUTED BY RANDOM [BUCKETS num|auto]. Description: Uses random numbers for bucketing. - MySQL does not have a bucketing algorithm.
Rollup_list	- Multiple sync materialized views can be created while creating the table. - Syntax: `rollup_name (col1[, col2, ...]) [DUPLICATE KEY(col1[, col2, ...])][PROPERTIES("key" = "value")]` - MySQL does not support this.
Properties	Table properties: They differ from MySQL's table properties, and the syntax for defining table properties also differs from MySQL.

03 CREATE INDEX

```
CREATE INDEX [IF NOT EXISTS] index_name ON table_name (column [, ...],) [USING BITMAP];
```

- Doris currently supports Bitmap index, Inverted index, and N-Gram index. BloomFilter index are supported as well, but they have a separate syntax for setting them.
- MySQL supports index algorithms such as B+Tree and Hash.

04 CREATE VIEW

```
CREATE VIEW [IF NOT EXISTS]
 [db_name.]view_name
 (column1[ COMMENT "col comment"][, column2, ...])
AS query_stmt

CREATE MATERIALIZED VIEW (IF NOT EXISTS)? mvName=multipartIdentifier
        (LEFT_PAREN cols=simpleColumnDefs RIGHT_PAREN)? buildMode?
        (REFRESH refreshMethod? refreshTrigger?)?
        (KEY keys=identifierList)?
        (COMMENT STRING_LITERAL)?
        (PARTITION BY LEFT_PAREN partitionKey = identifier RIGHT_PAREN)?
        (DISTRIBUTED BY (HASH hashKeys=identifierList | RANDOM) (BUCKETS (INTEGER_VALUE | AUTO))?)?
        propertyClause?
        AS query
```

- The basic syntax is consistent with MySQL.
- Doris supports logical view and supports two types of materialized views: synchronous materialized views and asynchronous materialized views
- MySQL do not supports asynchronous materialized views.

05 ALTER TABLE / ALTER INDEX

The syntax of Doris ALTER is basically the same as that of MySQL.

DROP TABLE / DROP INDEX

The syntax of Doris DROP is basically the same as MySQL.

DML

INSERT

```
INSERT INTO table_name
    [ PARTITION (p1, ...) ]
    [ WITH LABEL label]
    [ (column [, ...]) ]
    [ [ hint [, ...] ] ]
    { VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

The Doris INSERT syntax is basically the same as MySQL.

UPDATE

```
UPDATE target_table [table_alias]
    SET assignment_list
    WHERE condition

assignment_list:
    assignment [, assignment] ...

assignment:
    col_name = value

value:
    {expr | DEFAULT}
```

The Doris UPDATE syntax is basically the same as MySQL, but it should be noted that the `WHERE` **condition must be added.**

Delete

```
DELETE FROM table_name [table_alias]
    [PARTITION partition_name | PARTITIONS (partition_name [, partition_name])]
    WHERE column_name op { value | value_list } [ AND column_name op { value | value_list } ...];
```

The syntax can only specify filter predicates

```
DELETE FROM table_name [table_alias]
    [PARTITION partition_name | PARTITIONS (partition_name [, partition_name])]
    [USING additional_tables]
    WHERE condition
```

This syntax can only be used on the UNIQUE KEY model table.

The DELETE syntax in Doris is basically the same as in MySQL. However, since Doris is an analytical database, deletions cannot be too frequent.

SELECT

```
SELECT
    [hint_statement, ...]
    [ALL | DISTINCT]
    select_expr [, select_expr ...]
    [EXCEPT ( col_name1 [, col_name2, col_name3, ...] )]
    [FROM table_references
      [PARTITION partition_list]
      [TABLET tabletid_list]
      [TABLESAMPLE sample_value [ROWS | PERCENT]
        [REPEATABLE pos_seek]]
    [WHERE where_condition]
    [GROUP BY [GROUPING SETS | ROLLUP | CUBE] {col_name | expr | position}]
    [HAVING where_condition]
```

```
    [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
    [LIMIT {[offset_count,] row_count | row_count OFFSET offset_count}]
    [INTO OUTFILE 'file_name']
```

The Doris SELECT syntax is basically the same as MySQL.

SQL Function

Doris Function covers most MySQL functions.

SQL Mode

Name	Behavior when enabled	Behavior when disabled	Notes
PIPES_AS_CONCAT	Parses || as the `concat` function	Parses || as the logical AND operator	-
NO_BACKSLASH_ESCAPES	Treats backslashes in strings as literal characters	Treats backslashes in strings as escape characters	-

Join

What is JOIN

In relational databases, data is distributed across multiple tables, which are interconnected through specific relationships. SQL JOIN operations allow users to combine different tables into a more complete result set based on these relationships.

JOIN types supported by Doris

- **INNER JOIN**: Comparing each row of the left table with all rows of the right table based on the JOIN condition, returning matching rows from both tables. For more details, refer to the syntax definition for JOIN queries in SELECT.
- **LEFT JOIN**: Building on the result set of an INNER JOIN, if a row from the left table does not have a match in the right table, all rows from the left table are returned, with corresponding columns from the right table shown as NULL.
- **RIGHT JOIN**: The opposite of LEFT JOIN; if a row from the right table does not have a match in the left table, all rows from the right table are returned, with corresponding columns from the left table shown as NULL.
- **FULL JOIN**: Building on the result set of an INNER JOIN, returning all rows from both tables, filling in NULL where there are no matches.
- **CROSS JOIN**: Having no JOIN condition, returning the Cartesian product of the two tables, where each row from the left table is combined with each row from the right table.
- **LEFT SEMI JOIN**: Comparing each row of the left table with all rows of the right table based on the JOIN condition. If a match exists, the corresponding row from the left table is returned.
- **RIGHT SEMI JOIN**: The opposite of LEFT SEMI JOIN; comparing each row of the right table with all rows of the left table, returning the corresponding row from the right table if a match exists.
- **LEFT ANTI JOIN**: Comparing each row of the left table with all rows of the right table based on the JOIN condition. If there is no match, the corresponding row from the left table is returned.
- **RIGHT ANTI JOIN**: The opposite of LEFT ANTI JOIN; comparing each row of the right table with all rows of the left table, returning rows from the right table that do not have matches.
- **NULL AWARE LEFT ANTI JOIN**: Similar to LEFT ANTI JOIN but ignoring rows in the left table where the matching column is NULL.

Implementation of JOIN in Doris

Doris supports two implementation methods for JOIN: **Hash Join** and **Nested Loop Join**.

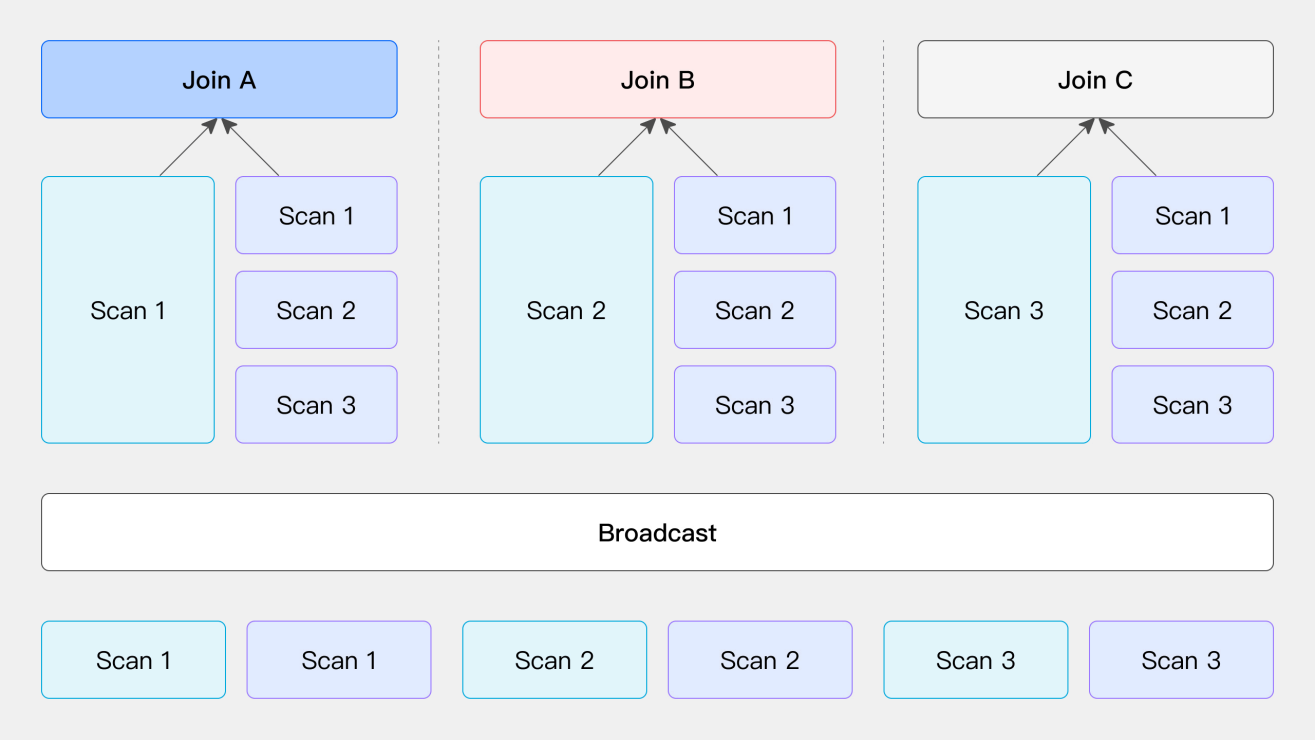
- **Hash Join**: A hash table is built on the right table based on the equality JOIN columns, and the data from the left table is streamed through this hash table for JOIN calculations. This method is limited to cases where equality JOIN conditions are applicable.
- **Nested Loop Join**: This method uses two nested loops, driven by the left table, to iterate through each row of the left table and compare it with every row of the right table based on the JOIN condition. It is suitable for all JOIN scenarios, including those that Hash Join cannot handle, such as queries involving GREATER THAN or LESS THAN comparisons, or cases requiring Cartesian products. However, compared to Hash Join, Nested Loop Join may have inferior performance.

Implementation of Hash Join in Doris

As a distributed MPP database, Apache Doris requires data shuffling during the Hash Join process to ensure the correctness of the JOIN results. Below are several data shuffling methods:

Broadcast Join As illustrated, the Broadcast Join process involves sending all data from the right table to all nodes participating in the JOIN computation, including the nodes scanning the left table's data, while the left table's data remains stationary. In this process, each node receives a complete copy of the right table's data (with a total volume of T(R)) to ensure that all nodes have the necessary data to perform the JOIN operation.

This method is suitable for various scenarios but is not applicable for RIGHT OUTER, RIGHT ANTI, and RIGHT SEMI types of Hash Join. Its network overhead is calculated as the number of JOIN nodes N multiplied by the volume of right table's data T(R).

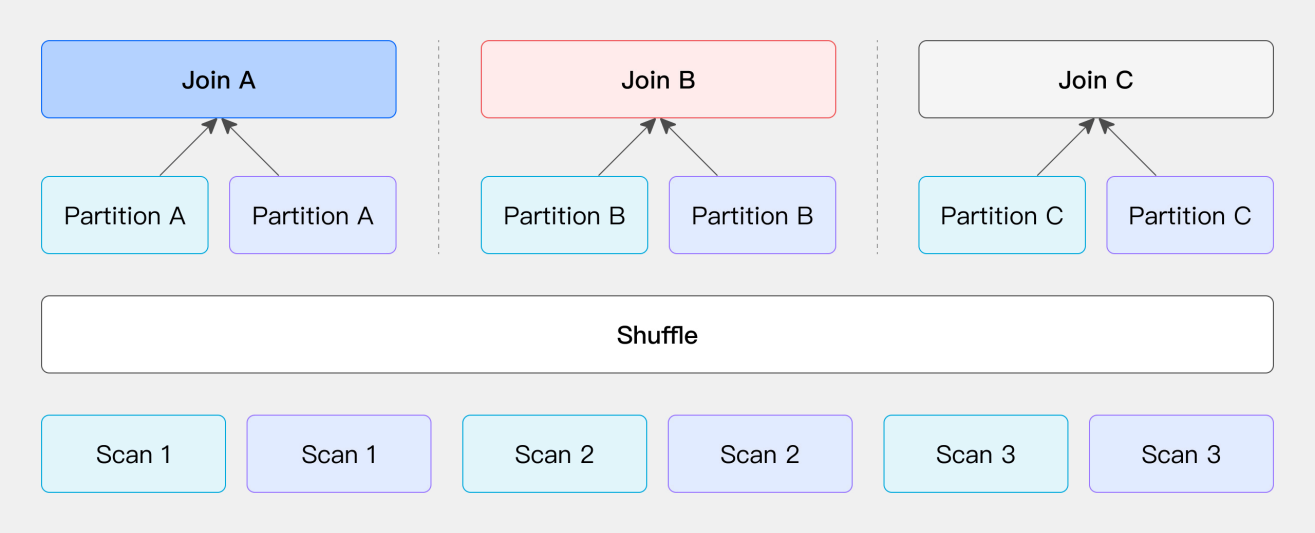


Implementation of Hash Join in Doris

Partition Shuffle Join

This method computes hash values based on the JOIN conditions and performs bucketing. Specifically, the data from both the left and right tables is partitioned according to the hash values calculated from the JOIN conditions, and these partitioned data sets are then sent to the corresponding partition nodes (as illustrated).

The network overhead of this method mainly includes two parts: the cost of transferring the left table's data T(S) and the cost of transferring the right table's data T(R). This method only supports Hash Join operations because it relies on the JOIN conditions to perform data bucketing.



Partition Shuffle Join

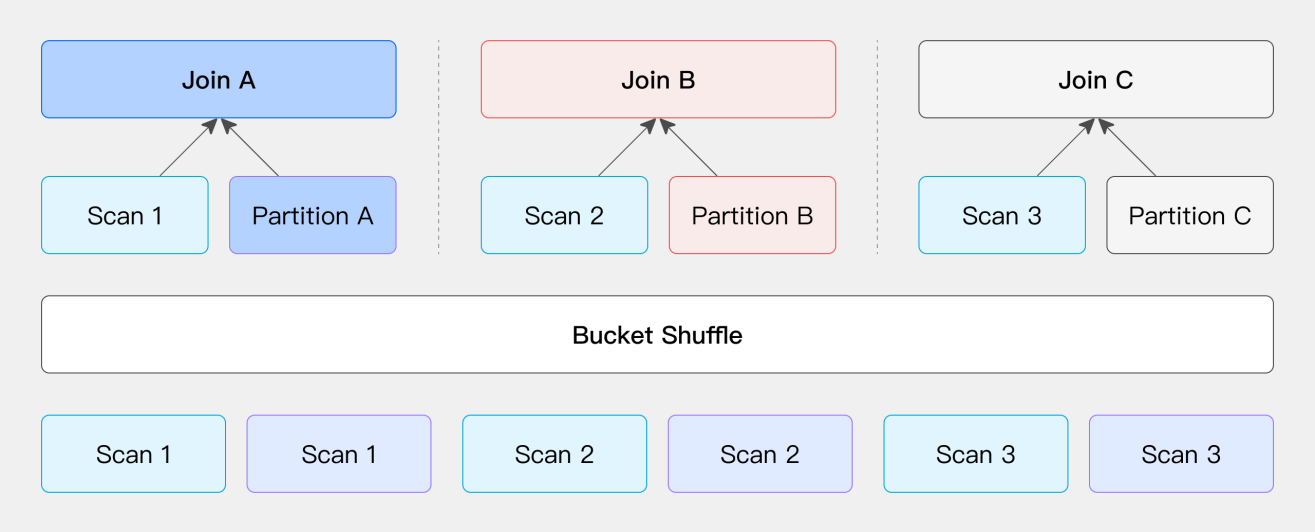
Bucket Shuffle Join

When the JOIN condition includes the bucketed column from the left table, the left table's data location remains unchanged while the right table's data is distributed to the left table's nodes for the JOIN, reducing network overhead.

When one side of the table involved in the JOIN operation has its data already hash-distributed according to the JOIN condition column, users can choose to keep this side's data location unchanged while distributing the other side's data based on the same JOIN condition column and hash distribution. (The term "table" here refers not only to physically stored tables but also to the output results of any operators in SQL queries. Users can flexibly choose to keep either the left or right table's data location unchanged while only moving and distributing the other side's table.)

For example, in the case of physical tables of Doris, since the table data is stored in a bucketed manner through hash computation, users can directly leverage this feature to optimize the data shuffle process for the JOIN operation. Suppose you have two tables that need to be joined, and the JOIN column is the bucketed column from the left table. In this case, you do not need to move the left table's data; you only need to distribute the right table's data to the appropriate locations based on the left table's bucket information to complete the JOIN computation.

The primary network overhead for this process comes from the movement of the right table's data, denoted as T(R).



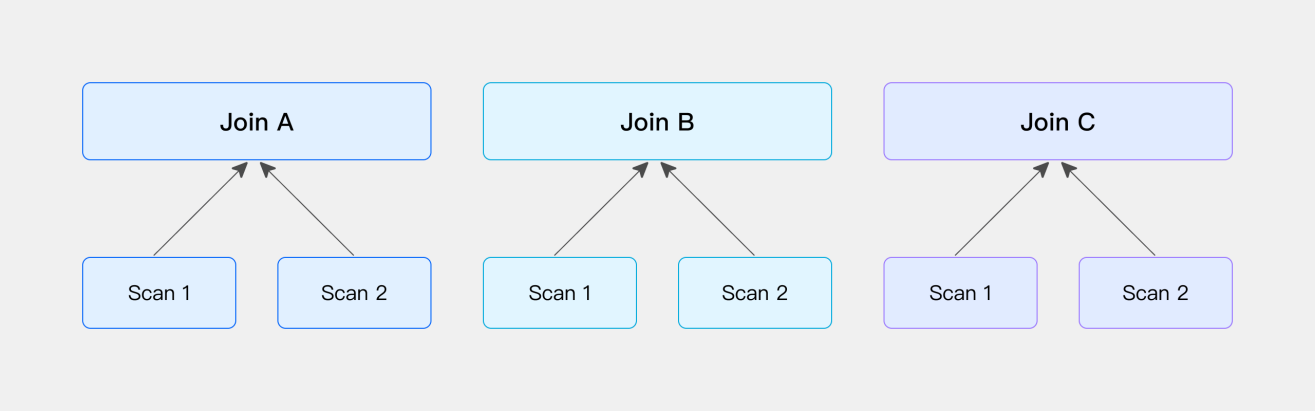
Bucket Shuffle Join

Colocate Join

Similar to Bucket Shuffle Join, if both tables involved in the Join are already distributed by Hash according to the Join condition columns, the Shuffle process can be skipped, and the Join calculation can be performed directly on the local data. This can be illustrated with physical tables:

When creating a table in Doris with the specification of DISTRIBUTED BY HASH, the system distributes data based on the Hash distribution key during data import. If the Hash distribution keys of both tables happen to match the Join condition columns, it can be said that the data in these two tables is already pre-distributed according to the Join requirements, eliminating the need for additional Shuffle operations. Therefore, during actual queries, the Join calculation can be executed directly on these two tables.

> **For scenarios where Join is executed after directly scanning data, certain conditions must be met during table creation; please refer to the subsequent restrictions regarding Colocate Join between the two physical tables.**



Colocate Join

Bucket Shuffle Join VS Colocate Join

As mentioned earlier, for both Bucket Shuffle Join and Colocate Join, the join operations can be executed as long as the distribution of the participating tables meets specific conditions (the term “tables” here refers to any output from SQL query operators).

Next, we will provide a more detailed explanation of the generalized Bucket Shuffle Join and Colocate Join using two tables, t1 and t2, along with relevant SQL examples. First, here are the table creation statements for both tables:

```
create table t1
(
    c1 bigint,
    c2 bigint
)
DISTRIBUTED BY HASH(c1) BUCKETS 3
PROPERTIES ("replication_num" = "1");

create table t2
(
    c1 bigint,
    c2 bigint
)
DISTRIBUTED BY HASH(c1) BUCKETS 3
PROPERTIES ("replication_num" = "1");
```

Example of Bucket Shuffle Join

In the following example, both tables t1 and t2 have been processed by the GROUP BY operator, resulting in new tables (at this point, the tx table is hash-distributed by c1, while the ty table is hash-distributed by c2). The subsequent JOIN condition is tx.c1 = ty.c2, which perfectly meets the conditions for a Bucket Shuffle Join.

```
explain select *
from
    (
        -- The t1 table is hash-distributed by c1, and after the GROUP BY operator, it still maintains the hash distribution by c1.
        select c1 as c1, sum(c2) as c2
        from t1
        group by c1
    ) tx
join
    (
        -- The t2 table is hash-distributed by c1, but after the GROUP BY operator, the data is redistributed to be hash-distributed
by c2.
        select c2 as c2, sum(c1) as c1
        from t2
        group by c2
    ) ty
on tx.c1 = ty.c2;
```

From the following Explain execution plan, it can be observed that the left child node of the Hash Join node 7 is the aggregation node 6, while the right child node is the Exchange node 4. This indicates that the data from the left child node, after aggregation, remains in the same location, while the data from the right child node is distributed to the node where the left child node resides using the Bucket Shuffle method, in order to perform the subsequent Hash Join operation.

```
+------------------------------------------------------------+
| Explain String(Nereids Planner)                            |
+------------------------------------------------------------+
| PLAN FRAGMENT 0                                            |
|   OUTPUT EXPRS:                                            |
|     c1[#18]                                                |
|     c2[#19]                                                |
|     c2[#20]                                                |
|     c1[#21]                                                |
|   PARTITION: HASH_PARTITIONED: c1[#8]                      |
|                                                            |
|   HAS_COLO_PLAN_NODE: true                                 |
|                                                            |
|   VRESULT SINK                                             |
|      MYSQL_PROTOCAL                                        |
|                                                            |
|   7:VHASH JOIN(364)                                        |
|   |  join op: INNER JOIN(BUCKET_SHUFFLE)[]                 |
|   |  equal join conjunct: (c1[#12] = c2[#6])               |
|   |  cardinality=10                                        |
|   |  vec output tuple id: 8                                |
|   |  output tuple id: 8                                    |
|   |  vIntermediate tuple ids: 7                            |
|   |  hash output slot ids: 6 7 12 13                       |
|   |  final projections: c1[#14], c2[#15], c2[#16], c1[#17] |
|   |  final project output tuple id: 8                      |
|   |  distribute expr lists: c1[#12]                        |
|   |  distribute expr lists: c2[#6]                         |
|   |                                                        |
|   |----4:VEXCHANGE                                         |
|   |       offset: 0                                        |
|   |       distribute expr lists: c2[#6]                    |
|   |                                                        |
|   6:VAGGREGATE (update finalize)(342)                      |
|   |  output: sum(c2[#9])[#11]                              |
|   |  group by: c1[#8]                                      |
|   |  sortByGroupKey:false                                  |
|   |  cardinality=10                                        |
|   |  final projections: c1[#10], c2[#11]                   |
|   |  final project output tuple id: 6                      |
|   |  distribute expr lists: c1[#8]                         |
|   |                                                        |
|   5:VOlapScanNode(339)                                     |
|      TABLE: tt.t1(t1), PREAGGREGATION: ON                  |
|      partitions=1/1 (t1)                                   |
|      tablets=1/1, tabletList=491188                        |
|      cardinality=21, avgRowSize=0.0, numNodes=1            |
|      pushAggOp=NONE                                        |
|                                                            |
| PLAN FRAGMENT 1                                            |
|                                                            |
|   PARTITION: HASH_PARTITIONED: c2[#2]                      |
|                                                            |
|   HAS_COLO_PLAN_NODE: true                                 |
|                                                            |
|   STREAM DATA SINK                                         |
|     EXCHANGE ID: 04                                        |
|     BUCKET_SHFFULE_HASH_PARTITIONED: c2[#6]                |
|                                                            |
|   3:VAGGREGATE (merge finalize)(355)                       |
|   |  output: sum(partial_sum(c1)[#3])[#5]                  |
|   |  group by: c2[#2]                                      |
|   |  sortByGroupKey:false                                  |
|   |  cardinality=5                                         |
```

```
|   |  final projections: c2[#4], c1[#5]                    |
|   |  final project output tuple id: 3                     |
|   |  distribute expr lists: c2[#2]                        |
|   |                                                       |
|   2:VEXCHANGE                                             |
|      offset: 0                                            |
|      distribute expr lists:                               |
|                                                           |
| PLAN FRAGMENT 2                                           |
|                                                           |
|   PARTITION: HASH_PARTITIONED: c1[#0]                     |
|                                                           |
|   HAS_COLO_PLAN_NODE: false                               |
|                                                           |
|   STREAM DATA SINK                                        |
|     EXCHANGE ID: 02                                       |
|     HASH_PARTITIONED: c2[#2]                              |
|                                                           |
|   1:VAGGREGATE (update serialize)(349)                    |
|   |  STREAMING                                            |
|   |  output: partial_sum(c1[#0])[#3]                      |
|   |  group by: c2[#1]                                     |
|   |  sortByGroupKey:false                                 |
|   |  cardinality=5                                        |
|   |  distribute expr lists: c1[#0]                        |
|   |                                                       |
|   0:VOlapScanNode(346)                                    |
|      TABLE: tt.t2(t2), PREAGGREGATION: ON                 |
|      partitions=1/1 (t2)                                  |
|      tablets=1/1, tabletList=491198                       |
|      cardinality=10, avgRowSize=0.0, numNodes=1           |
|      pushAggOp=NONE                                       |
|                                                           |
|                                                           |
| Statistics                                                |
|  planed with unknown column statistics                    |
+-----------------------------------------------------------+
97 rows in set (0.01 sec)
```

Example of Colocate Join

In the following example, both tables t1 and t2 have been processed by the GROUP BY operator, resulting in new tables (at this point, both tx and ty are hash-distributed by c2). The subsequent JOIN condition is tx.c2 = ty.c2, which perfectly meets the conditions for a Colocate Join.

```
explain select *
from
    (
        -- The t1 table is initially hash-distributed by c1, but after the GROUP BY operator, the data distribution changes to be
hash-distributed by c2.
        select c2 as c2, sum(c1) as c1
        from t1
        group by c2
    ) tx
join
    (
        -- The t2 table is initially hash-distributed by c1, but after the GROUP BY operator, the data distribution changes to be
hash-distributed by c2.
        select c2 as c2, sum(c1) as c1
        from t2
        group by c2
```

```
    ) ty
on tx.c2 = ty.c2;
```

From the results of the following Explain execution plan, it can be seen that the left child node of Hash Join node 8 is aggregation node 7, and the right child node is aggregation node 3, with no Exchange node present. This indicates that the aggregated data from both the left and right child nodes remains in its original location, eliminating the need for data movement and allowing the subsequent Hash Join operation to be performed directly locally.

```
+------------------------------------------------------------+
| Explain String(Nereids Planner)                            |
+------------------------------------------------------------+
| PLAN FRAGMENT 0                                            |
|   OUTPUT EXPRS:                                            |
|     c2[#20]                                                |
|     c1[#21]                                                |
|     c2[#22]                                                |
|     c1[#23]                                                |
|   PARTITION: HASH_PARTITIONED: c2[#10]                     |
|                                                            |
|   HAS_COLO_PLAN_NODE: true                                 |
|                                                            |
|   VRESULT SINK                                             |
|      MYSQL_PROTOCAL                                        |
|                                                            |
|   8:VHASH JOIN(373)                                        |
|   |  join op: INNER JOIN(PARTITIONED)[]                    |
|   |  equal join conjunct: (c2[#14] = c2[#6])               |
|   |  cardinality=10                                        |
|   |  vec output tuple id: 9                                |
|   |  output tuple id: 9                                    |
|   |  vIntermediate tuple ids: 8                            |
|   |  hash output slot ids: 6 7 14 15                       |
|   |  final projections: c2[#16], c1[#17], c2[#18], c1[#19] |
|   |  final project output tuple id: 9                      |
|   |  distribute expr lists: c2[#14]                        |
|   |  distribute expr lists: c2[#6]                         |
|   |                                                        |
|   |----3:VAGGREGATE (merge finalize)(367)                  |
|   |    |  output: sum(partial_sum(c1)[#3])[#5]             |
|   |    |  group by: c2[#2]                                 |
|   |    |  sortByGroupKey:false                             |
|   |    |  cardinality=5                                    |
|   |    |  final projections: c2[#4], c1[#5]                |
|   |    |  final project output tuple id: 3                 |
|   |    |  distribute expr lists: c2[#2]                    |
|   |    |                                                   |
|   |    2:VEXCHANGE                                         |
|   |       offset: 0                                        |
|   |       distribute expr lists:                           |
|   |                                                        |
|   7:VAGGREGATE (merge finalize)(354)                       |
|   |  output: sum(partial_sum(c1)[#11])[#13]                |
|   |  group by: c2[#10]                                     |
|   |  sortByGroupKey:false                                  |
|   |  cardinality=10                                        |
|   |  final projections: c2[#12], c1[#13]                   |
|   |  final project output tuple id: 7                      |
|   |  distribute expr lists: c2[#10]                        |
|   |                                                        |
|   6:VEXCHANGE                                              |
|      offset: 0                                             |
```

```
|      distribute expr lists:                                |
|                                                            |
| PLAN FRAGMENT 1                                            |
|                                                            |
|   PARTITION: HASH_PARTITIONED: c1[#8]                      |
|                                                            |
|   HAS_COLO_PLAN_NODE: false                                |
|                                                            |
|   STREAM DATA SINK                                         |
|     EXCHANGE ID: 06                                        |
|     HASH_PARTITIONED: c2[#10]                              |
|                                                            |
|   5:VAGGREGATE (update serialize)(348)                     |
|   |  STREAMING                                             |
|   |  output: partial_sum(c1[#8])[#11]                      |
|   |  group by: c2[#9]                                      |
|   |  sortByGroupKey:false                                  |
|   |  cardinality=10                                        |
|   |  distribute expr lists: c1[#8]                         |
|   |                                                        |
|   4:VOlapScanNode(345)                                     |
|      TABLE: tt.t1(t1), PREAGGREGATION: ON                  |
|      partitions=1/1 (t1)                                   |
|      tablets=1/1, tabletList=491188                        |
|      cardinality=21, avgRowSize=0.0, numNodes=1            |
|      pushAggOp=NONE                                        |
|                                                            |
| PLAN FRAGMENT 2                                            |
|                                                            |
|   PARTITION: HASH_PARTITIONED: c1[#0]                      |
|                                                            |
|   HAS_COLO_PLAN_NODE: false                                |
|                                                            |
|   STREAM DATA SINK                                         |
|     EXCHANGE ID: 02                                        |
|     HASH_PARTITIONED: c2[#2]                               |
|                                                            |
|   1:VAGGREGATE (update serialize)(361)                     |
|   |  STREAMING                                             |
|   |  output: partial_sum(c1[#0])[#3]                       |
|   |  group by: c2[#1]                                      |
|   |  sortByGroupKey:false                                  |
|   |  cardinality=5                                         |
|   |  distribute expr lists: c1[#0]                         |
|   |                                                        |
|   0:VOlapScanNode(358)                                     |
|      TABLE: tt.t2(t2), PREAGGREGATION: ON                  |
|      partitions=1/1 (t2)                                   |
|      tablets=1/1, tabletList=491198                        |
|      cardinality=10, avgRowSize=0.0, numNodes=1            |
|      pushAggOp=NONE                                        |
|                                                            |
|                                                            |
| Statistics                                                 |
|  planed with unknown column statistics                     |
+------------------------------------------------------------+
105 rows in set (0.06 sec)
```

Comparison of four shuffle methods

Shuffle Methods	Network Overhead	Physical Operator	Applicable Scenarios
Broadcast	N * T(R)	Hash Join /Nest Loop Join	General
Shuffle	T(S) + T(R)	Hash Join	General
Bucket Shuffle	T(R)	Hash Join	JOIN condition includes the left table's bucketed column, with the left table being single-partitioned.
Colocate	0	Hash Join	JOIN condition includes the left table's bucketed column, and both tables belong to the same Colocate Group.

> **NOTE**
>
> N: Number of instances participating in the Join calculation
>
> T(Relation): Number of tuples in the relation

The flexibility of the four Shuffle methods decreases in order, and their requirements for data distribution become increasingly strict. In most cases, as the requirements for data distribution increase, the performance of Join calculations tends to improve gradually. It is important to note that if the number of buckets in a table is small, Bucket Shuffle or Colocate Join may experience a decrease in performance due to lower parallelism, potentially resulting in slower performance than Shuffle Join. This is because the Shuffle operation can more effectively balance data distribution, thereby providing higher parallelism in subsequent processing.

FAQ

Bucket Shuffle Join and Colocate Join have specific limitations regarding data distribution and JOIN conditions when applied. Below, we will elaborate on the specific restrictions for each of these JOIN methods.

Limitations of Bucket Shuffle Join

When directly scanning two physical tables for a Bucket Shuffle Join, the following conditions must be met:

1. **Equality Join condition**: Bucket Shuffle Join is only applicable for scenarios where the JOIN condition is based on equality, as it relies on hash calculations to determine data distribution.
2. **Inclusion of bucketed columns in equality conditions**: The equality JOIN condition must include the bucketed columns from both tables. When the left table's bucketed column is used as the equality JOIN condition, it is more likely to be planned as a Bucket Shuffle Join.
3. **Table type restrictions**: Bucket Shuffle Join is only applicable to native OLAP tables in Doris. For external tables such as ODBC, MySQL, and ES, Bucket Shuffle Join cannot be effective when they are used as the left table.
4. **Single Partition Requirement**: For partitioned tables, since the data distribution may differ across partitions, Bucket Shuffle Join is only guaranteed to be effective when the left table is a single partition.

Therefore, when executing SQL, it is advisable to use `WHERE` conditions to enable partition pruning strategies whenever possible.

Limitations of Colocate Join

When directly scanning two physical tables, Colocate Join has stricter limitations compared to Bucket Shuffle Join. In addition to meeting all the conditions for Bucket Shuffle Join, the following requirements must also be satisfied:

1. **bucket column types and counts are Same**: Not only must the types of the bucketed columns match, but the number of buckets must also be the same to ensure data distribution consistency.
2. **Explicit specification of Colocation Group**: A Colocation Group must be explicitly specified; only tables within the same Colocation Group can participate in a Colocate Join.
3. **Unstable state during replica repair or balancing**: During operations such as replica repair or balancing, the Colocation Group may be in an unstable state. In this case, the Colocate Join will degrade to a regular Join operation.

Subquery

A Subquery is an SQL query nested within another query (usually a SELECT statement). It can be used in the SELECT, FROM, WHERE, or HAVING clauses to provide data or conditions for the outer query. The use of subqueries makes SQL queries more flexible and powerful, as they allow us to solve more complex problems within a single query.

Some important features of subqueries are as follows:

1. Position of Subqueries: Subqueries can be placed in multiple SQL clauses, such as the WHERE clause, HAVING clause, and FROM clause. They can be used with SELECT, UPDATE, INSERT, DELETE statements, and expression operators (such as comparison operators =, >, <, <=, as well as IN, EXISTS, etc.).

2. Relationship between Main Query and Subquery: A subquery is a query nested inside another query. The outer query is referred to as the main query, while the inner query is referred to as the subquery.

3. Execution Order: When there is no correlation between the subquery and the main query, the subquery is usually executed first. When there is a correlation, the parser decides which query to execute first in real-time as needed and uses the output of the subquery accordingly.

4. Use of Parentheses: Subqueries must be enclosed in parentheses to distinguish them as nested within another query.

Below, we will use tables t1 and t2 and related SQL to introduce the basic features and usage of subqueries. The table creation statements are as follows:

```
create table t1
(
    c1 bigint,
    c2 bigint
)
DISTRIBUTED BY HASH(c1) BUCKETS 3
PROPERTIES ("replication_num" = "1");

create table t2
(
    c1 bigint,
    c2 bigint
)
DISTRIBUTED BY HASH(c1) BUCKETS 3
PROPERTIES ("replication_num" = "1");
```

Classification of Subqueries

Classification Based on the Characteristics of Data Returned by Subqueries

Subqueries can be classified into scalar and non-scalar subqueries based on the characteristics of the data they return:

1. Scalar Subquery

A subquery that always returns a single value (essentially equivalent to a one-row, one-column Relation). If the subquery does not return any data, it returns a NULL value. Scalar subqueries can theoretically appear anywhere a single-value expression is allowed.

2. Non-scalar Subquery

A subquery that returns a Relation (different from the return value of a scalar subquery, this Relation can contain multiple rows and columns). If the subquery does not return any data, it returns an empty set (0 rows). Non-scalar subqueries can theoretically appear anywhere a relation (set) is allowed.

The following examples illustrate scalar and non-scalar subqueries (for the two subqueries in parentheses, when t2 is an empty table, the results returned by the two subqueries are different):

```
-- Scalar subquery, when t2 is an empty table, the subquery returns the scalar value null
select * from t1 where t1.c1 > (select sum(t2.c1) from t2);

-- Non-scalar subquery, when t2 is an empty table, the subquery returns an empty set (0 rows)
select * from t1 where t1.c1 in (select t2.c1 from t2);
```

Classification Based on Whether the Subquery References Columns from the Outer Query

Subqueries can be classified into correlated subqueries and non-correlated subqueries based on whether they reference columns from the outer query:

1. Non-correlated Subquery

A subquery that does not reference any columns from the outer query. Non-correlated subqueries can often be computed independently and return the corresponding results once for the outer query to use.

2. Correlated Subquery

A subquery that references one or more columns from the main query (also known as the outer query) (the referenced outer columns are often in the WHERE condition of the subquery). Correlated subqueries can often be seen as a filtering operation on the externally associated table, as for each row of data in the outer table, the subquery is computed and returns the corresponding result.

The following examples illustrate correlated and non-correlated subqueries:

```
-- Correlated subquery, the subquery internally uses the column t1.c2 from the outer table
select * from t1 where t1.c1 in (select t2.c1 from t2 where t2.c2 = t1.c2);

-- Non-correlated subquery, the subquery internally does not use any columns from the outer table t1
select * from t1 where t1.c1 in (select t2.c1 from tt2);
```

Subqueries Supported by Doris

Doris supports all non-correlated subqueries and provides partial support for correlated subqueries as follows:

- Supports correlated scalar subqueries in the `WHERE` and `HAVING` clauses.

- Supports correlated `IN`, `NOT IN`, `EXISTS`,`NOT EXISTS` non-scalar subqueries in the `WHERE` and `HAVING` clauses.
- Supports correlated scalar subqueries in the `SELECT` list.
- For nested subqueries, Doris only supports subqueries correlated to their immediate parent query and does not support cross-level correlation to outer queries beyond the parent.

Limitations of Correlated Subqueries

Limitations of Correlated Scalar Subqueries

- The correlation condition must be an equality condition.
- The output of the subquery must be the result of a single aggregate function without a GROUP BY clause.

```
-- Single aggregate function without GROUP BY, supported
select * from t1 where t1.c1 < (select max(t2.c1) from t2 where t1.c2 = t2.c2);

-- Equivalent rewritten SQL as follows:
select t1.* from t1 inner join (select t2.c2 as c2, max(t2.c1) as c1 from t2 group by t2.c2) tx on t1.c1 < tx.c1 and t1.c2 = tx.c2;

-- Non-equality condition, not supported
select * from t1 where t1.c1 = (select max(t2.c1) from t2 where t1.c2 > t2.c2);

-- No aggregate function, not supported
select * from t1 where t1.c1 = (select t2.c1 from t2 where t1.c2 = t2.c2);

-- With aggregate function but includes GROUP BY, not supported
select * from t1 where t1.c1 = (select max(t2.c1) from t2 where t1.c2 = t2.c2 group by t2.c2);
```

Limitations of Correlated (NOT) EXISTS Subqueries

- The subquery cannot have both OFFSET and LIMIT.

```
-- With LIMIT but no OFFSET, supported
select * from t1 where exists (select t2.c1 from t2 where t1.c2 = t2.c2 limit 2);

-- Equivalent rewritten SQL as follows:
select * from t1 left semi join t2 on t1.c2 = t2.c2;

-- With OFFSET and LIMIT, not supported
select * from t1 where exists (select t2.c1 from t2 where t1.c2 = t2.c2 limit 2, 3);
```

Limitations of Correlated (NOT) IN Subqueries

- The output of the subquery must be a single column.
- The subquery cannot have LIMIT.
- The subquery cannot have aggregate functions or GROUP BY clauses.

```
-- Supported subquery
select * from t1 where t1.c1 in (select t2.c1 from t2 where t1.c2 = t2.c2);
```

```
-- Equivalent rewritten SQL as follows:
select * from t1 left semi join t2 on t1.c1 = t2.c1 and t1.c2 = t2.c2;

-- Subquery output is multiple columns, not supported
select * from t1 where (t1.a, t1.c) in (select t2.c1, t2.c from t2 where t1.c2 = t2.c2);

-- Subquery with LIMIT, not supported
select * from t1 where t1.c1 in (select t2.c1 from t2 where t1.c2 = t2.c2 limit 3);

-- With GROUP BY clause, not supported
select * from t1 where t1.c1 in (select t2.c1 from t2 where t1.c2 = t2.c2 group by t2.c1);

-- With aggregate function, not supported
select * from t1 where t1.c1 in (select sum(t2.c1) from t2 where t1.c2 = t2.c2);
```

Limitations of Nested Subqueries

Currently, only subqueries that correlate directly with their immediate parent queries are supported. Correlation with outer layers of the parent query is not supported.

Assume there is another table `t3` with the following creation statement:

```
create table t3
(
    c1 bigint,
    c2 bigint
)
DISTRIBUTED BY HASH(c1) BUCKETS 3
PROPERTIES ("replication_num" = "1");
```

- Supported when the subquery only uses columns from its immediate parent query:

  ```
  select
      t1.c1
  from
      t1
  where not exists (
      select
          t2.c1
      from
          t2
      where not exists (
          select
              t3.c1
          from
              t3
          where
              t3.c2 = t2.c2
      ) and t2.c2 = t1.c2
  );
  ```

- Not supported when the innermost subquery uses columns from its immediate parent query `t2.c2` and also columns from the outermost query `t1.c1`:

  ```
  select
      t1.c1
  from
  ```

```
    t1
where not exists (
    select
        t2.c1
    from
        t2
    where not exists (
        select
            t3.c1
        from
            t3
        where
            t3.c2 = t2.c2 and t3.c1 = t1.c1
    )
);
```

Mark Join

In `where` conditions, clauses with `or` relationships composed of subqueries using `(not) in` or `(not) exists` and other filtering conditions require special handling to produce correct results. An example is given below:

```
select
    t1.c1,
    t1.c2
from t1
where exists (
    select
        t2.c1
    from t2
    where
        t1.c2 = t2.c2
    ) or t1.c1 > 0;
```

If the `exists` clause in this SQL is directly implemented using `left semi join`, according to the semantics of `left semi join`, only rows from `t1` that satisfy `t1.c2 = t2.c2` will be output. However, rows that actually satisfy the condition `t1.c1 > 0` should also be output. To achieve this, the mechanism of `Mark Join` is introduced.

> **Note**
>
> `right semi join` is similar but differs in the left and right tables. Here, we use `left semi join` as an example.

Example SQL is as follows:

```
-- This SQL cannot be executed and is only for demonstration purposes
select
    tx.c1,
    tx.c2
from
    (
        select
            t1.c1,
            t1.c2,
            mark_join_flag
        from
```

```
            t1 left (mark) semi join t2 on t1.c2 = t2.c2
    ) tx
where
    tx.mark_join_flag or tx.c1 > 0;
```

The difference between `Mark Join` and a regular `left semi join` is that a regular `left semi join` directly outputs rows from the left table that meet the condition, while `Mark Join` outputs the original left table with an additional flag column (the `mark_join_flag` in the example) that can be `true`, `false`, or `null`. The value of the flag is determined by the `join` condition expression `t1.c2 = t2.c2`, with each row corresponding to a flag value. The calculation of flag values is shown in the table below:

t1.c1	t2.c1	mark_join_flag
1	1	TRUE
1	2	FALSE
1	NULL	NULL
NULL	1	NULL
NULL	NULL	NULL

With this flag, the `where` filtering condition can be rewritten as `where mark_join_flag or t1.c1 > 0` to obtain the correct results.

Usage notes

Since the output of a scalar subquery must be a single value, a runtime error will be reported when the subquery returns more than one row of data.

For Correlated Scalar Subqueries

When using a correlated quantifier subquery, if the subquery that satisfies the correlation condition returns more than one row of data, a runtime error will be reported.

Please refer to the following SQL example:

```
-- If there are more than 1 row in the t2 table that satisfies t1.c2 = t2.c2 in the associated scalar subquery, a runtime error will be reported
select t1.*, (select t2.c1 from t2 where t1.c2 = t2.c2) from t1;

-- Example error message
ERROR 1105 (HY000): errCode = 2, detailMessage = (127.0.0.1)[INVALID_ARGUMENT][E33] correlate scalar subquery must return only 1 row
```

For Non-Correlated Scalar Subqueries

Doris will add an `assert num rows` operator at runtime. If the subquery returns more than one row of data, a runtime error will be reported.

Please refer to the following SQL example:

```
-- Non-correlated scalar subquery, will report an error if table t2 has more than 1 row of data
select t1.*, (select t2.c1 from t2) from t1;

-- Example error message
ERROR 1105 (HY000): errCode = 2, detailMessage = (127.0.0.1)[CANCELLED]Expected EQ 1 to be returned by expression
```

Multi-Dimensional Analytics

In a database, ROLLUP, CUBE, and GROUPING SETS are advanced SQL statements used for multidimensional data aggregation. These features significantly enhance the capabilities of the GROUP BY clause, enabling users to obtain multiple levels of summary results in a single query, which is semantically equivalent to using UNION ALL to connect multiple aggregation statements.

- **ROLLUP**: ROLLUP is an operation used to generate hierarchical summaries. It aggregates data according to the specified column order, gradually summarizing from the finest granularity to the highest level. For example, in sales data, ROLLUP can be used to summarize by region and time, providing sales for each region per month, total sales for each region, and overall total sales. ROLLUP is suitable for scenarios requiring step-by-step summaries.
- **CUBE**: CUBE is a more powerful aggregation operation that generates all possible summary combinations. Unlike ROLLUP, CUBE calculates subsets for all dimensions. For instance, for sales data aggregated by product and region, CUBE will compute sales for each product in each region, total sales for each product, total sales for each region, and overall total sales. CUBE is applicable to scenarios requiring comprehensive multidimensional analysis, such as business analysis and market research.
- **GROUPING SETS**: GROUPING SETS offer flexibility in aggregating specific grouping sets. It allows users to specify a set of column combinations for independent aggregation, rather than generating all possible combinations as in ROLLUP and CUBE. For example, one can define summaries for specific combinations of region and time without needing all combinations of each dimension. GROUPING SETS are suitable for scenarios requiring customized summaries, providing flexible aggregation control.

ROLLUP, CUBE, and GROUPING SETS provide powerful multidimensional data summary functions, catering to various data analysis and reporting needs, and making complex aggregation calculations simpler and more efficient. The following sections will detail the usage scenarios, syntax, and examples of these features.

ROLLUP

Use Case

ROLLUP is particularly useful for summarizing data along hierarchical dimensions such as time, geography, and category. For instance, queries can specify `ROLLUP(year, month, day)` or `(country, Province, city)`.

Syntax and Example

The syntax for ROLLUP is as follows:

```
SELECT ... GROUP BY ROLLUP(grouping_column_reference_list)
```

Here's an example query that analyzes sales sums by year and month:

```
SELECT
        YEAR(d_date),
```

```
        MONTH(d_date),
        SUM(ss_net_paid) AS total_sum
FROM
        store_sales,
        date_dim d1
WHERE
        d1.d_date_sk = ss_sold_date_sk
        AND YEAR(d_date) IN (2001, 2002)
        AND MONTH(d_date) IN (1, 2, 3)
GROUP BY
        ROLLUP(YEAR(d_date), MONTH(d_date))
ORDER BY
        YEAR(d_date), MONTH(d_date);
```

This query summarizes data by time, calculating subtotals for sales by year, sales by month within each year, and the grand total of sales. The query result is as follows:

```
+--------------+---------------+-------------+
| YEAR(d_date) | MONTH(d_date) | total_sum   |
+--------------+---------------+-------------+
|         NULL |          NULL | 54262669.17 |
|         2001 |          NULL | 26640320.46 |
|         2001 |             1 |  9982165.83 |
|         2001 |             2 |  8454915.34 |
|         2001 |             3 |  8203239.29 |
|         2002 |          NULL | 27622348.71 |
|         2002 |             1 | 11260654.35 |
|         2002 |             2 |  7722750.61 |
|         2002 |             3 |  8638943.75 |
+--------------+---------------+-------------+
9 rows in set (0.08 sec)
```

CUBE

Use Case

CUBE is best suited for queries involving columns from multiple independent dimensions, rather than columns representing different levels of a single dimension. For example, a common usage scenario is summarizing all combinations of month, region, and product. These are three independent dimensions, and it is common to analyze all possible subtotal combinations. By contrast, cross-tabulating all possible combinations of year, month, and day would include several unnecessary values due to the natural hierarchy in the time dimension. In most analyses, subtotals like profits calculated by month and day are unnecessary. Relatively few users need to ask, “What is the total sales for the 16th of each month throughout the year?”

Syntax and Example

The syntax for CUBE is as follows:

```
SELECT ... GROUP BY CUBE(grouping_column_reference_list)
```

Example usage:

```
SELECT
        YEAR(d_date),
```

```
        i_category,
        ca_state,
        SUM(ss_net_paid) AS total_sum
FROM
        store_sales,
        date_dim d1,
        item,
        customer_address ca
WHERE
        d1.d_date_sk = ss_sold_date_sk
        AND i_item_sk = ss_item_sk
        AND ss_addr_sk = ca_address_sk
        AND i_category IN ("Books", "Electronics")
        AND YEAR(d_date) IN (1998, 1999)
        AND ca_state IN ("LA", "AK")
GROUP BY CUBE(YEAR(d_date), i_category, ca_state)
ORDER BY YEAR(d_date), i_category, ca_state;
```

The query result is as follows, calculating:

- The total sales sum;
- Subtotals for sales by year, by product category, and by state;
- Subtotals for sales by product category within each year, by state for each product, by state within each year, and by product category within each state and year.

```
+--------------+-------------+----------+------------+
| YEAR(d_date) | i_category  | ca_state | total_sum  |
+--------------+-------------+----------+------------+
|         NULL | NULL        | NULL     | 8690374.60 |
|         NULL | NULL        | AK       | 2675198.33 |
|         NULL | NULL        | LA       | 6015176.27 |
|         NULL | Books       | NULL     | 4238177.69 |
|         NULL | Books       | AK       | 1310791.36 |
|         NULL | Books       | LA       | 2927386.33 |
|         NULL | Electronics | NULL     | 4452196.91 |
|         NULL | Electronics | AK       | 1364406.97 |
|         NULL | Electronics | LA       | 3087789.94 |
|         1998 | NULL        | NULL     | 4369656.14 |
|         1998 | NULL        | AK       | 1402539.19 |
|         1998 | NULL        | LA       | 2967116.95 |
|         1998 | Books       | NULL     | 2213703.82 |
|         1998 | Books       | AK       |  719911.29 |
|         1998 | Books       | LA       | 1493792.53 |
|         1998 | Electronics | NULL     | 2155952.32 |
|         1998 | Electronics | AK       |  682627.90 |
|         1998 | Electronics | LA       | 1473324.42 |
|         1999 | NULL        | NULL     | 4320718.46 |
|         1999 | NULL        | AK       | 1272659.14 |
|         1999 | NULL        | LA       | 3048059.32 |
|         1999 | Books       | NULL     | 2024473.87 |
|         1999 | Books       | AK       |  590880.07 |
|         1999 | Books       | LA       | 1433593.80 |
|         1999 | Electronics | NULL     | 2296244.59 |
|         1999 | Electronics | AK       |  681779.07 |
|         1999 | Electronics | LA       | 1614465.52 |
+--------------+-------------+----------+------------+
27 rows in set (0.21 sec)
```

GROUPING FUNCTION

This section introduces how to address two challenges when using ROLLUP and CUBE:

1. How to programmatically identify which rows in the result set represent subtotals and accurately determine the aggregation level corresponding to a given subtotal. Since subtotals are often needed for calculations such as percentage of totals, we require a convenient method to identify these subtotal rows.

2. When the query results contain both actual stored NULL values and “NULL” values generated by ROLLUP or CUBE operations, another problem arises: how to distinguish between these two types of NULL values?

GROUPING, GROUPING_ID, and GROUPING SETS can effectively solve the aforementioned challenges.

GROUPING

1. Principles

GROUPING uses a single column as a parameter and returns 1 when encountering a NULL value created by ROLLUP or CUBE operations, indicating that the row is a subtotal. Any other type of value (including NULLs inherently present in the table data) returns 0.

Example:

```
select
        year(d_date),
        month(d_date),
        sum(ss_net_paid) as total_sum,
        grouping(year(d_date)),
        grouping(month(d_date))
from
        store_sales,
        date_dim d1
where
        d1.d_date_sk = ss_sold_date_sk
        and year(d_date) in (2001, 2002)
        and month(d_date) in (1, 2, 3)
group by
        rollup(year(d_date), month(d_date))
order by
        year(d_date), month(d_date);
```

- The GROUPING function result for the (YEAR(d_date), MONTH(d_date)) group is (0,0) for aggregation by year and month.
- The GROUPING function result for the (YEAR(d_date)) group is (0,1) for aggregation by year.
- The GROUPING function result for the () group is (1,1) for the total aggregation.

Query result:

```
+--------------+---------------+-------------+-----------------------+-------------------------+
| year(d_date) | month(d_date) | total_sum   | Grouping(year(d_date)) | Grouping(month(d_date)) |
+--------------+---------------+-------------+-----------------------+-------------------------+
```

```
|         NULL |          NULL | 54262669.17 |                     1 |                        1 |
|         2001 |          NULL | 26640320.46 |                     0 |                        1 |
|         2001 |             1 |  9982165.83 |                     0 |                        0 |
|         2001 |             2 |  8454915.34 |                     0 |                        0 |
|         2001 |             3 |  8203239.29 |                     0 |                        0 |
|         2002 |          NULL | 27622348.71 |                     0 |                        1 |
|         2002 |             1 | 11260654.35 |                     0 |                        0 |
|         2002 |             2 |  7722750.61 |                     0 |                        0 |
|         2002 |             3 |  8638943.75 |                     0 |                        0 |
+--------------+---------------+-------------+-----------------------+------------------------+
9 rows in set (0.06 sec)
```

2. Usage Scenarios, Syntax, and Examples

The GROUPING function can be used to filter results. Example:

```
select
        year(d_date),
        i_category,
        ca_state,
        sum(ss_net_paid) as total_sum
from
        store_sales,
        date_dim d1,
        item,
        customer_address ca
where
        d1.d_date_sk = ss_sold_date_sk
        and i_item_sk = ss_item_sk
        and ss_addr_sk=ca_address_sk
        and i_category in ("Books", "Electronics")
        and year(d_date) in(1998, 1999)
        and ca_state in ("LA", "AK")
group by cube(year(d_date), i_category, ca_state)
having grouping(year(d_date))=1 and grouping(i_category)=1 and grouping(ca_state)=1
or grouping(year(d_date))=0 and grouping(i_category)=1 and grouping(ca_state)=1
or grouping(year(d_date))=1 and grouping(i_category)=1 and grouping(ca_state)=0
order by year(d_date), i_category, ca_state;
```

Using the GROUPING function in the HAVING clause retains only the total sales, sales summarized by year, and sales summarized by region. Query result:

```
+---------------------+------------+----------+------------+
| year(`d1`.`d_date`) | i_category | ca_state | total_sum  |
+---------------------+------------+----------+------------+
|                NULL | NULL       | NULL     | 8690374.60 |
|                NULL | NULL       | AK       | 2675198.33 |
|                NULL | NULL       | LA       | 6015176.27 |
|                1998 | NULL       | NULL     | 4369656.14 |
|                1999 | NULL       | NULL     | 4320718.46 |
+---------------------+------------+----------+------------+
5 rows in set (0.13 sec)
```

You can also use the GROUPING function with the IF function to enhance query readability. Example:

```
select
        if(grouping(year(d_date)) = 1, "Multi-year sum", year(d_date)) as year,
        if(grouping(i_category) = 1, "Multi-category sum", i_category) as category,
        sum(ss_net_paid) as total_sum
```

```
from
        store_sales,
        date_dim d1,
        item,
        customer_address ca
where
        d1.d_date_sk = ss_sold_date_sk
        and i_item_sk = ss_item_sk
        and ss_addr_sk = ca_address_sk
        and i_category in ("Books", "Electronics")
        and year(d_date) in (1998, 1999)
        and ca_state in ("LA", "AK")
group by cube(year(d_date), i_category)
```

Query result:

```
+----------------+--------------------+------------+
| year           | category           | total_sum  |
+----------------+--------------------+------------+
| 1998           | Books              | 2213703.82 |
| 1998           | Electronics        | 2155952.32 |
| 1999           | Electronics        | 2296244.59 |
| 1999           | Books              | 2024473.87 |
| 1998           | Multi-category sum | 4369656.14 |
| 1999           | Multi-category sum | 4320718.46 |
| Multi-year sum | Books              | 4238177.69 |
| Multi-year sum | Electronics        | 4452196.91 |
| Multi-year sum | Multi-category sum | 8690374.60 |
+----------------+--------------------+------------+
9 rows in set (0.09 sec)
```

GROUPING_ID

1. Usage Scenarios

In a database, both GROUPING_ID and GROUPING functions serve as auxiliary functions for handling multidimensional data aggregation queries, such as ROLLUP and CUBE, aiding users in distinguishing between different levels of aggregation results. If you wish to determine the aggregation level of a particular row, you need to use the GROUPING function to compute all GROUP BY columns, as the computation result of a single column alone is insufficient.

The GROUPING_ID function is more powerful than GROUPING because it can detect multiple columns simultaneously. The GROUPING_ID function accepts multiple columns as parameters and returns an integer that represents the aggregation status of these columns through binary bits. When using tables or materialized views to store computation results, using GROUPING to represent different levels of aggregation can consume considerable storage space. In such scenarios, GROUPING_ID is more appropriate.

Taking CUBE(a, b) as an example, its GROUPING_ID can be represented as follows:

Aggregation Level	Bit Vector	GROUPING_ID	GROUPING(a)	GROUPING(b)
a,b	0 0	0	0	0
a	0 1	1	0	1
b	1 0	2	1	0

Aggregation Level	Bit Vector	GROUPING_ID	GROUPING(a)	GROUPING(b)
Grand Total	1 1	3	1	1

2. Syntax and Example

Here is an example SQL query:

```
SELECT
    year(d_date),
    i_category,
    SUM(ss_net_paid) AS total_sum,
    GROUPING(year(d_date)),
    GROUPING(i_category),
    GROUPING_ID(year(d_date), i_category)
FROM
    store_sales,
    date_dim d1,
    item,
    customer_address ca
WHERE
    d1.d_date_sk = ss_sold_date_sk
    AND i_item_sk = ss_item_sk
    AND ss_addr_sk = ca_address_sk
    AND i_category IN ('Books', 'Electronics')
    AND year(d_date) IN (1998, 1999)
    AND ca_state IN ('LA', 'AK')
GROUP BY CUBE(year(d_date), i_category);
```

The query results are as follows:

```
+--------------+-------------+------------+-----------------------+---------------------+--------------------------------------+
| year(d_date) | i_category  | total_sum  | GROUPING(year(d_date)) | GROUPING(i_category) | GROUPING_ID(year(d_date), i_category) |
+--------------+-------------+------------+-----------------------+---------------------+--------------------------------------+
| 1998         | Electronics | 2155952.32 | 0                     | 0                   | 0                                    |
| 1998         | Books       | 2213703.82 | 0                     | 0                   | 0                                    |
| 1999         | Electronics | 2296244.59 | 0                     | 0                   | 0                                    |
| 1999         | Books       | 2024473.87 | 0                     | 0                   | 0                                    |
| 1998         | NULL        | 4369656.14 | 0                     | 1                   | 1                                    |
| 1999         | NULL        | 4320718.46 | 0                     | 1                   | 1                                    |
| NULL         | Electronics | 4452196.91 | 1                     | 0                   | 2                                    |
| NULL         | Books       | 4238177.69 | 1                     | 0                   | 2                                    |
| NULL         | NULL        | 8690374.60 | 1                     | 1                   | 3                                    |
+--------------+-------------+------------+-----------------------+---------------------+--------------------------------------+
9 rows in set (0.12 sec)
```

GROUPING SETS

1. Usage Scenarios

When there is a need to selectively specify the group sets to create, the `GROUPING SETS` expression can be used in the `GROUP BY` clause. This method allows users to precisely specify across multiple dimensions without computing the entire CUBE.

Since CUBE queries typically consume significant resources, using `GROUPING SETS` can enhance query execution efficiency when only a few dimensions are of interest.

2. Syntax and Examples

The syntax for `GROUPING SETS` is as follows:

```
SELECT ... GROUP BY GROUPING SETS(grouping_column_reference_list)
```

If you need:

- Subtotals of sales for each product category per year
- Subtotals of sales for each state per year
- Subtotals of sales for each product in each state per year

You can use `GROUPING SETS` to specify these dimensions and perform the aggregation. Here is an example:

```
SELECT
    YEAR(d_date),
    i_category,
    ca_state,
    SUM(ss_net_paid) AS total_sum
FROM
    store_sales,
    date_dim d1,
    item,
    customer_address ca
WHERE
    d1.d_date_sk = ss_sold_date_sk
    AND i_item_sk = ss_item_sk
    AND ss_addr_sk = ca_address_sk
    AND i_category IN ('Books', 'Electronics')
    AND YEAR(d_date) IN (1998, 1999)
    AND ca_state IN ('LA', 'AK')
GROUP BY GROUPING SETS(
    (YEAR(d_date), i_category),
    (YEAR(d_date), ca_state),
    (YEAR(d_date), ca_state, i_category)
)
ORDER BY YEAR(d_date), i_category, ca_state;
```

Query Result:

```
+--------------+-------------+----------+------------+
| YEAR(d_date) | i_category  | ca_state | total_sum  |
+--------------+-------------+----------+------------+
| 1998         | NULL        | AK       | 1402539.19 |
| 1998         | NULL        | LA       | 2967116.95 |
| 1998         | Books       | NULL     | 2213703.82 |
| 1998         | Books       | AK       |  719911.29 |
| 1998         | Books       | LA       | 1493792.53 |
| 1998         | Electronics | NULL     | 2155952.32 |
| 1998         | Electronics | AK       |  682627.90 |
| 1998         | Electronics | LA       | 1473324.42 |
| 1999         | NULL        | AK       | 1272659.14 |
| 1999         | NULL        | LA       | 3048059.32 |
| 1999         | Books       | NULL     | 2024473.87 |
| 1999         | Books       | AK       |  590880.07 |
| 1999         | Books       | LA       | 1433593.80 |
| 1999         | Electronics | NULL     | 2296244.59 |
```

```
| 1999         | Electronics | AK       |  681779.07 |
| 1999         | Electronics | LA       | 1614465.52 |
+--------------+-------------+----------+------------+
16 rows in set (0.11 sec)
```

The above approach is equivalent to using CUBE but specifies concrete `grouping_id`, thereby reducing unnecessary calculations:

```
SELECT
    SUM(ss_net_paid) AS total_sum,
    YEAR(d_date),
    i_category,
    ca_state
FROM
    store_sales,
    date_dim d1,
    item,
    customer_address ca
WHERE
    d1.d_date_sk = ss_sold_date_sk
    AND i_item_sk = ss_item_sk
    AND ss_addr_sk = ca_address_sk
    AND i_category IN ('Books', 'Electronics')
    AND YEAR(d_date) IN (1998, 1999)
    AND ca_state IN ('LA', 'AK')
GROUP BY CUBE(YEAR(d_date), ca_state, i_category)
HAVING grouping_id(YEAR(d_date), ca_state, i_category) = 0
    OR grouping_id(YEAR(d_date), ca_state, i_category) = 2
    OR grouping_id(YEAR(d_date), ca_state, i_category) = 1;
```

> **Note**
>
> Using `CUBE` computes all possible aggregation levels (eight in this case), but in practice, you may only be interested in a few of them.

3. Semantic Equivalents

- GROUPING SETS vs. GROUP BY UNION ALL

  The `GROUPING SETS` statement:

  ```
  SELECT k1, k2, SUM(k3) FROM t GROUP BY GROUPING SETS ((k1, k2), (k1), (k2), ());
  ```

  Is equivalent in query results to multiple `GROUP BY` queries connected with `UNION ALL`:

  ```
  SELECT k1, k2, SUM(k3) FROM t GROUP BY k1, k2
  UNION ALL
  SELECT k1, NULL, SUM(k3) FROM t GROUP BY k1
  UNION ALL
  SELECT NULL, k2, SUM(k3) FROM t GROUP BY k2
  UNION ALL
  SELECT NULL, NULL, SUM(k3) FROM t;
  ```

Using `UNION ALL` results in a longer query and requires multiple scans of the base table, making it less efficient in both writing and execution.

- GROUPING SETS vs. ROLLUP

  `ROLLUP` is an extension of `GROUPING SETS`. For example:

  ```
  SELECT a, b, c, SUM(d) FROM tab1 GROUP BY ROLLUP(a, b, c);
  ```

  This `ROLLUP` is equivalent to the following `GROUPING SETS`:

  ```
  GROUPING SETS (
      (a, b, c),
      (a, b),
      (a),
      ()
  );
  ```

- GROUPING SETS vs. CUBE

  `CUBE(a, b, c)` is equivalent to the following `GROUPING SETS`:

  ```
  GROUPING SETS (
      (a, b, c),
      (a, b),
      (a, c),
      (a),
      (b, c),
      (b),
      (c),
      ()
  );
  ```

APPENDIX

For table creation statements and data files, see the Window Function appendix.

Window Function

Analytic functions, also known as window functions, are functions in SQL queries that perform complex calculations on rows in a data set. The characteristic of window functions is that they do not reduce the number of rows in the query result, but instead add a new computed result for each row. Window functions are applicable to various analysis scenarios, such as calculating running totals, rankings, and moving averages.

Below is an example of using a window function to calculate the three-day moving average of sales for each store before and after a given date:

```
CREATE TABLE daily_sales
(store_id INT, sales_date DATE, sales_amount DECIMAL(10, 2))
PROPERTIES (
  "replication_num" = "1"
);
INSERT INTO daily_sales (store_id, sales_date, sales_amount) VALUES (1, '2023-01-01', 100.00), (1, '2023-01-02', 150.00), (1, '2023-01-03', 200.00), (1, '2023-01-04', 250.00), (1, '2023-01-05', 300.00), (1, '2023-01-06', 350.00), (1, '2023-01-07', 400.00), (1, '2023-01-08', 450.00), (1, '2023-01-09', 500.00), (2, '2023-01-01', 110.00), (2, '2023-01-02', 160.00), (2, '2023-01-03', 210.00), (2, '2023-01-04', 260.00), (2, '2023-01-05', 310.00), (2, '2023-01-06', 360.00), (2, '2023-01-07', 410.00), (2, '2023-01-08', 460.00), (2, '2023-01-09', 510.00);

SELECT
        store_id,
        sales_date,
        sales_amount,
        AVG(sales_amount) OVER ( PARTITION BY store_id ORDER BY sales_date
        ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING ) AS moving_avg_sales
FROM
        daily_sales;
```

The query result is as follows:

```
+----------+------------+--------------+------------------+
| store_id | sales_date | sales_amount | moving_avg_sales |
+----------+------------+--------------+------------------+
|        1 | 2023-01-01 |       100.00 |         175.0000 |
|        1 | 2023-01-02 |       150.00 |         200.0000 |
|        1 | 2023-01-03 |       200.00 |         225.0000 |
|        1 | 2023-01-04 |       250.00 |         250.0000 |
|        1 | 2023-01-05 |       300.00 |         300.0000 |
|        1 | 2023-01-06 |       350.00 |         350.0000 |
|        1 | 2023-01-07 |       400.00 |         375.0000 |
|        1 | 2023-01-08 |       450.00 |         400.0000 |
|        1 | 2023-01-09 |       500.00 |         425.0000 |
|        2 | 2023-01-01 |       110.00 |         185.0000 |
|        2 | 2023-01-02 |       160.00 |         210.0000 |
|        2 | 2023-01-03 |       210.00 |         235.0000 |
|        2 | 2023-01-04 |       260.00 |         260.0000 |
|        2 | 2023-01-05 |       310.00 |         310.0000 |
|        2 | 2023-01-06 |       360.00 |         360.0000 |
|        2 | 2023-01-07 |       410.00 |         385.0000 |
|        2 | 2023-01-08 |       460.00 |         410.0000 |
|        2 | 2023-01-09 |       510.00 |         435.0000 |
+----------+------------+--------------+------------------+
18 rows in set (0.09 sec)
```

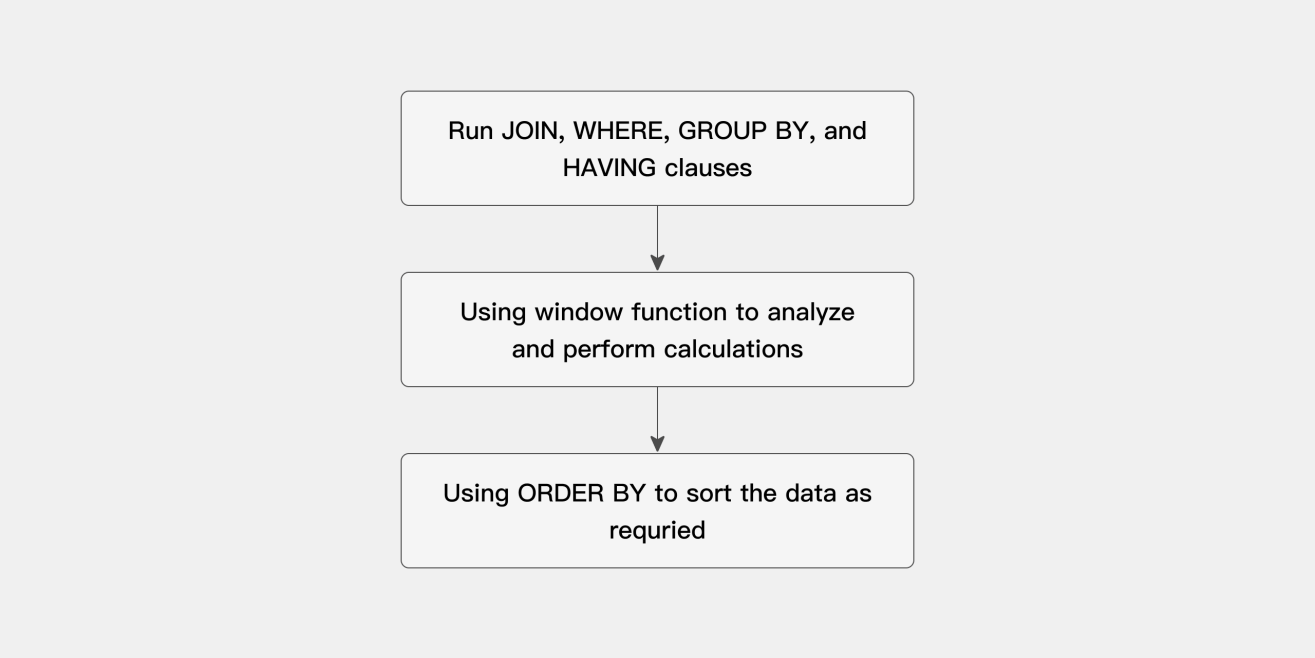
Introduction to Basic Concepts

Processing Order

The processing of queries using analytic functions can be divided into three stages.

1. Execute all joins, WHERE, GROUP BY, and HAVING clauses.
2. Provide the result set to the analytic functions and perform all necessary calculations.
3. If the query ends with an ORDER BY clause, process this clause to achieve precise output sorting.

The processing order of the query is illustrated as follows:



processing order

Result Set Partitioning

Partitions are created after defining groups using the PARTITION BY clause. Analytic functions allow users to divide the query result set into groups of rows called partitions.

> **Note**
>
> The term “partition” used in analytic functions is unrelated to the table partitioning feature. In this chapter, the term “partition” refers only to its meaning related to analytic functions.

Window

For each row in a partition, you can define a sliding data window. This window determines the range of rows involved in performing calculations for the current row. A window has a starting row and an ending row, and depending on its definition, the window can slide at one or both ends. For example, for a cumulative sum

function, the starting row is fixed at the first row of its partition, while the ending row slides from the start to the last row of the partition. Conversely, for a moving average, both the start and end points slide.

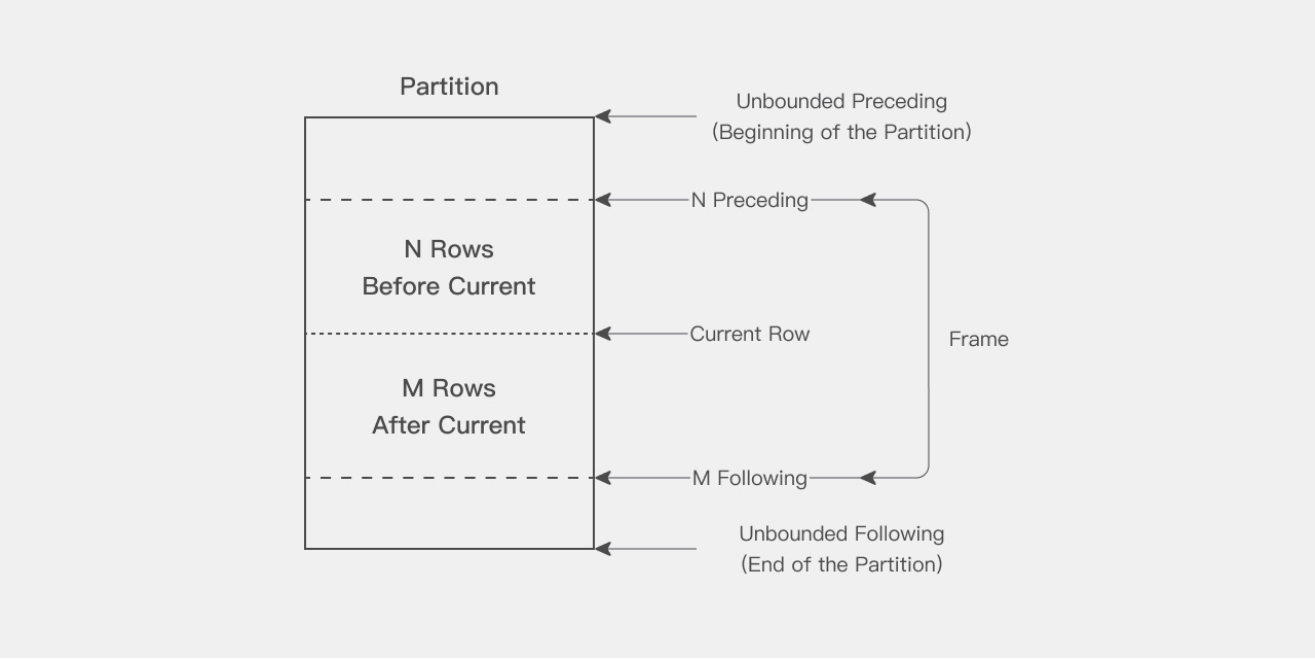
The size of the window can be set to be as large as all rows in the partition or as small as a sliding window that only includes one row within the partition. It should be noted that when the window is near the boundaries of the partition, due to boundary restrictions, the range of calculations may be reduced, and the function only returns the computed results of the available rows.

When using window functions, the current row is included in the calculation. Therefore, when processing n items, it should be specified as (n-1). For example, if you need to calculate a five-day average, the window should be specified as “rows between 4 preceding and current row,” which can also be abbreviated as “rows 4 preceding.”

Current Row

Each calculation performed using analytic functions is based on the current row within the partition. The current row serves as the reference point for determining the start and end of the window, as illustrated below.

For instance, a window can be used to define a centered moving average calculation that includes the current row, the 6 rows before the current row, and the 6 rows after the current row. This creates a sliding window containing 13 rows.



Current Row

Sorting Function

In a sorting function, query results are deterministic only when the specified sorting column is unique; if the sorting column contains duplicate values, the query results may vary each time.

NTILE Function

NTILE is a window function in SQL used to divide a query result set into a specified number of buckets (groups) and assign a bucket number to each row. This is particularly useful in data analysis and reporting, especially when data needs to be grouped and sorted.

1. Function Syntax

```
NTILE(num_buckets) OVER ([PARTITION BY partition_expression] ORDER BY order_expression)
```

- `num_buckets`: The number of buckets into which to divide the rows.
- `PARTITION BY partition_expression` (optional): Defines how to partition the data.
- `ORDER BY order_expression`: Defines how to sort the data.

2. Using the NTILE Function

Suppose there is a table `class_student_scores` containing students' exam scores, and you want to divide the students into 4 groups based on their scores, with the number of students in each group being as uniform as possible.

First, create the `class_student_scores` table and insert data:

```
CREATE TABLE class_student_scores (
    class_id INT,
    student_id INT,
    student_name VARCHAR(50),
    score INT
)distributed by hash(student_id) properties('replication_num'=1);

INSERT INTO class_student_scores VALUES
(1, 1, 'Alice', 85),
(1, 2, 'Bob', 92),
(1, 3, 'Charlie', 87),
(2, 4, 'David', 78),
(2, 5, 'Eve', 95),
(2, 6, 'Frank', 80),
(2, 7, 'Grace', 90),
(2, 8, 'Hannah', 84);
```

Then, use the NTILE function to divide the students into 4 groups based on their scores:

```
SELECT
    student_id,
    student_name,
    score,
    NTILE(4) OVER (ORDER BY score DESC) AS bucket
FROM
    class_student_scores;
```

The results are as follows:

```
+------------+--------------+-------+--------+
| student_id | student_name | score | bucket |
```

```
+------------+--------------+-------+--------+
|          5 | Eve          |    95 |      1 |
|          2 | Bob          |    92 |      1 |
|          7 | Grace        |    90 |      2 |
|          3 | Charlie      |    87 |      2 |
|          1 | Alice        |    85 |      3 |
|          8 | Hannah       |    84 |      3 |
|          6 | Frank        |    80 |      4 |
|          4 | David        |    78 |      4 |
+------------+--------------+-------+--------+
8 rows in set (0.12 sec)
```

In this example, the `NTILE(4)` function divides the students into 4 groups (buckets) based on their scores, with the number of students in each group being as uniform as possible.

> **Notes**
>
> - If rows cannot be evenly distributed into buckets, some buckets may have one extra row.
> - The `NTILE` function works within each partition. If the `PARTITION BY` clause is used, data within each partition will be separately assigned to buckets.

3. Using NTILE with PARTITION BY

Suppose you want to group students by class, and then divide them into 3 groups within each class based on their scores. You can use the `PARTITION BY` and `NTILE` functions:

```
SELECT
    class_id,
    student_id,
    student_name,
    score,
    NTILE(3) OVER (PARTITION BY class_id ORDER BY score DESC) AS bucket
FROM
    class_student_scores;
```

The results are as follows:

```
+----------+------------+--------------+-------+--------+
| class_id | student_id | student_name | score | bucket |
+----------+------------+--------------+-------+--------+
|        1 |          2 | Bob          |    92 |      1 |
|        1 |          3 | Charlie      |    87 |      2 |
|        1 |          1 | Alice        |    85 |      3 |
|        2 |          5 | Eve          |    95 |      1 |
|        2 |          7 | Grace        |    90 |      1 |
|        2 |          8 | Hannah       |    84 |      2 |
|        2 |          6 | Frank        |    80 |      2 |
|        2 |          4 | David        |    78 |      3 |
+----------+------------+--------------+-------+--------+
8 rows in set (0.05 sec)
```

In this example, students are partitioned by class, and then within each class, they are divided into 3 groups based on their scores. The number of students in each group is as uniform as possible.

Analytic Functions

Using the Analytic Function SUM to Calculate Cumulative Values

Here is an example:

```
SELECT
        i_category,
        year(d_date),
        month(d_date),
        sum(ss_net_paid) as total_sales,
        sum(sum(ss_net_paid)) over(partition by i_category order by year(d_date),month(d_date) ROWS UNBOUNDED PRECEDING) cum_sales
FROM
        store_sales,
        date_dim d1,
        item
WHERE
        d1.d_date_sk = ss_sold_date_sk
        and i_item_sk = ss_item_sk
        and year(d_date) =2000
        and i_category in ('Books','Electronics')
GROUP BY
        i_category,
        year(d_date),
        month(d_date)
```

The query result is as follows:

```
+-------------+--------------+---------------+-------------+-------------+
| i_category  | year(d_date) | month(d_date) | total_sales | cum_sales   |
+-------------+--------------+---------------+-------------+-------------+
| Books       |         2000 |             1 |  5348482.88 |  5348482.88 |
| Books       |         2000 |             2 |  4353162.03 |  9701644.91 |
| Books       |         2000 |             3 |  4466958.01 | 14168602.92 |
| Books       |         2000 |             4 |  4495802.19 | 18664405.11 |
| Books       |         2000 |             5 |  4589913.47 | 23254318.58 |
| Books       |         2000 |             6 |  4384384.00 | 27638702.58 |
| Books       |         2000 |             7 |  4488018.76 | 32126721.34 |
| Books       |         2000 |             8 |  9909227.94 | 42035949.28 |
| Books       |         2000 |             9 | 10366110.30 | 52402059.58 |
| Books       |         2000 |            10 | 10445320.76 | 62847380.34 |
| Books       |         2000 |            11 | 15246901.52 | 78094281.86 |
| Books       |         2000 |            12 | 15526630.11 | 93620911.97 |
| Electronics |         2000 |             1 |  5534568.17 |  5534568.17 |
| Electronics |         2000 |             2 |  4472655.10 | 10007223.27 |
| Electronics |         2000 |             3 |  4316942.60 | 14324165.87 |
| Electronics |         2000 |             4 |  4211523.06 | 18535688.93 |
| Electronics |         2000 |             5 |  4723661.00 | 23259349.93 |
| Electronics |         2000 |             6 |  4127773.06 | 27387122.99 |
| Electronics |         2000 |             7 |  4286523.05 | 31673646.04 |
| Electronics |         2000 |             8 | 10004890.96 | 41678537.00 |
| Electronics |         2000 |             9 | 10143665.77 | 51822202.77 |
| Electronics |         2000 |            10 | 10312020.35 | 62134223.12 |
| Electronics |         2000 |            11 | 14696000.54 | 76830223.66 |
| Electronics |         2000 |            12 | 15344441.52 | 92174665.18 |
+-------------+--------------+---------------+-------------+-------------+
24 rows in set (0.13 sec)
```

In this example, the analytic function SUM defines a window for each row, starting from the beginning of the partition (UNBOUNDED PRECEDING) and ending at the current row by default. In this case, nested use of SUM is required because we need to perform SUM on the result that is itself a SUM. Nested aggregation is frequently used in analytic aggregation functions.

Using the Analytic Function AVG to Calculate Moving Averages

Here is an example:

```
SELECT
        i_category,
        year(d_date),
        month(d_date),
        sum(ss_net_paid) as total_sales,
        avg(sum(ss_net_paid)) over(order by year(d_date),month(d_date) ROWS 2 PRECEDING) avg
FROM
        store_sales,
        date_dim d1,
        item
WHERE
        d1.d_date_sk = ss_sold_date_sk
        and i_item_sk = ss_item_sk
        and year(d_date) =2000
        and i_category='Books'
GROUP BY
        i_category,
        year(d_date),
        month(d_date)
```

The query result is as follows:

```
+------------+--------------+---------------+-------------+---------------+
| i_category | year(d_date) | month(d_date) | total_sales | avg           |
+------------+--------------+---------------+-------------+---------------+
| Books      |         2000 |             1 |  5348482.88 |  5348482.8800 |
| Books      |         2000 |             2 |  4353162.03 |  4850822.4550 |
| Books      |         2000 |             3 |  4466958.01 |  4722867.6400 |
| Books      |         2000 |             4 |  4495802.19 |  4438640.7433 |
| Books      |         2000 |             5 |  4589913.47 |  4517557.8900 |
| Books      |         2000 |             6 |  4384384.00 |  4490033.2200 |
| Books      |         2000 |             7 |  4488018.76 |  4487438.7433 |
| Books      |         2000 |             8 |  9909227.94 |  6260543.5666 |
| Books      |         2000 |             9 | 10366110.30 |  8254452.3333 |
| Books      |         2000 |            10 | 10445320.76 | 10240219.6666 |
| Books      |         2000 |            11 | 15246901.52 | 12019444.1933 |
| Books      |         2000 |            12 | 15526630.11 | 13739617.4633 |
+------------+--------------+---------------+-------------+---------------+
12 rows in set (0.13 sec)
```

> **Note**
>
> In the output data, the AVG column for the first two rows does not calculate a three-day moving average because there are not enough preceding rows for the boundary data (the number of rows specified in SQL is 3).

Additionally, it is possible to calculate window aggregate functions centered on the current row. For instance, this example calculates the centered moving average of monthly sales for products in the “Books” category in the year 2000, specifically averaging the total sales of the month before the current row, the current row, and the month after the current row.

```
SELECT
        i_category,
        YEAR(d_date) AS year,
        MONTH(d_date) AS month,
        SUM(ss_net_paid) AS total_sales,
        AVG(SUM(ss_net_paid)) OVER (ORDER BY YEAR(d_date), MONTH(d_date) ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS avg_sales
FROM
        store_sales,
        date_dim d1,
        item
WHERE
        d1.d_date_sk = ss_sold_date_sk
        AND i_item_sk = ss_item_sk
        AND YEAR(d_date) = 2000
        AND i_category = 'Books'
GROUP BY
        i_category,
        YEAR(d_date),
        MONTH(d_date)
```

> **Note**
>
> The centered moving averages for the starting and ending rows in the output data are calculated based on only two days because there are not enough rows before and after the boundary data.

Reporting Function

A reporting function refers to a scenario where the window range for each row covers the entire Partition. The primary advantage of reporting functions is their ability to pass data multiple times within a single query block, thereby enhancing query performance. For example, queries such as “For each year, find the product category with the highest sales” do not require JOIN operations when using reporting functions. An example is provided below:

```
SELECT year, category, total_sum FROM (
    SELECT
        YEAR(d_date) AS year,
        i_category AS category,
        SUM(ss_net_paid) AS total_sum,
        MAX(SUM(ss_net_paid)) OVER (PARTITION BY YEAR(d_date)) AS max_sales
    FROM
        store_sales,
        date_dim d1,
        item
    WHERE
        d1.d_date_sk = ss_sold_date_sk
        AND i_item_sk = ss_item_sk
        AND YEAR(d_date) IN (1998, 1999)
    GROUP BY
        YEAR(d_date), i_category
```

```
) t
WHERE total_sum = max_sales;
```

The inner query result for reporting `MAX(SUM(ss_net_paid))` is as follows:

```
SELECT year, category, total_sum FROM (
    SELECT
        YEAR(d_date) AS year,
        i_category AS category,
        SUM(ss_net_paid) AS total_sum,
        MAX(SUM(ss_net_paid)) OVER (PARTITION BY YEAR(d_date)) AS max_sales
    FROM
        store_sales,
        date_dim d1,
        item
    WHERE
        d1.d_date_sk = ss_sold_date_sk
        AND i_item_sk = ss_item_sk
        AND YEAR(d_date) IN (1998, 1999)
    GROUP BY
        YEAR(d_date), i_category
) t
WHERE total_sum = max_sales;
```

The complete query result is as follows:

```
+------+-------------+-------------+
| year | category    | total_sum   |
+------+-------------+-------------+
| 1998 | Electronics | 91723676.27 |
| 1999 | Electronics | 90310850.54 |
+------+-------------+-------------+
2 rows in set (0.12 sec)
```

You can combine reporting aggregation with nested queries to solve some complex problems, such as finding the best-selling products within important product subcategories. For example, to “Find subcategories where product sales account for more than 20% of total sales in their product category, and select the top five products from these subcategories,” the query statement is as follows:

```
SELECT i_category AS categ, i_class AS sub_categ, i_item_id
FROM
    (
    SELECT
        i_item_id, i_class, i_category, SUM(ss_net_paid) AS sales,
        SUM(SUM(ss_net_paid)) OVER (PARTITION BY i_category) AS cat_sales,
        SUM(SUM(ss_net_paid)) OVER (PARTITION BY i_class) AS sub_cat_sales,
        RANK() OVER (PARTITION BY i_class ORDER BY SUM(ss_net_paid)) AS rank_in_line
    FROM
        store_sales,
        item
    WHERE
        i_item_sk = ss_item_sk
    GROUP BY i_class, i_category, i_item_id
    ) t
WHERE sub_cat_sales > 0.2 * cat_sales AND rank_in_line <= 5;
```

LAG / LEAD

The LAG and LEAD functions are suitable for comparisons between values. Both functions can access multiple rows in a table simultaneously without requiring self-joins, thereby enhancing the speed of query processing. Specifically, the LAG function provides access to a row at a given offset before the current row, while the LEAD function provides access to a row at a given offset after the current row.

Below is an example of an SQL query using the LAG function. This query aims to select the total sales for each product category in specific years (1999, 2000, 2001, 2002), the total sales of the previous year, and the difference between them:

```
select year, category, total_sales, before_year_sales, total_sales - before_year_sales from
(
select
        sum(ss_net_paid) as total_sales,
        year(d_date) year,
        i_category category,
        lag(sum(ss_net_paid), 1,0) over(PARTITION BY i_category ORDER BY YEAR(d_date)) AS before_year_sales
from
        store_sales,
        date_dim d1,
        item
where
        d1.d_date_sk = ss_sold_date_sk
        and i_item_sk = ss_item_sk
GROUP BY
        YEAR(d_date), i_category
) t
where year in (1999, 2000, 2001, 2002)
```

The query results are as follows:

```
+------+-------------+-------------+-------------------+----------------------------------+
| year | category    | total_sales | before_year_sales | (total_sales - before_year_sales) |
+------+-------------+-------------+-------------------+----------------------------------+
| 1999 | Books       | 88993351.11 |       91307909.84 |                      -2314558.73 |
| 2000 | Books       | 93620911.97 |       88993351.11 |                       4627560.86 |
| 2001 | Books       | 90640097.99 |       93620911.97 |                      -2980813.98 |
| 2002 | Books       | 89585515.90 |       90640097.99 |                      -1054582.09 |
| 1999 | Electronics | 90310850.54 |       91723676.27 |                      -1412825.73 |
| 2000 | Electronics | 92174665.18 |       90310850.54 |                       1863814.64 |
| 2001 | Electronics | 92598527.85 |       92174665.18 |                        423862.67 |
| 2002 | Electronics | 94303831.84 |       92598527.85 |                       1705303.99 |
+------+-------------+-------------+-------------------+----------------------------------+
8 rows in set (0.16 sec)
```

Unique Ordering for Analytic Function Data

1. Issue of Inconsistent Return Results

When the `ORDER BY` clause of a window function fails to produce a unique ordering of the data, such as when the `ORDER BY` expression results in duplicate values, the order of the rows becomes indeterminate. This means that the return order of these rows may vary across multiple query executions, leading to inconsistent results from the window function.

The following example illustrates how the query returns different results on successive runs. The inconsistency arises primarily because `ORDER BY dateid` does not provide a unique ordering for the `SUM` window function.

```
CREATE TABLE test_window_order
    (item_id int,
    date_time date,
    sales double)
distributed BY hash(item_id)
properties("replication_num" = 1);

INSERT INTO test_window_order VALUES
(1, '2024-07-01', 100),
(2, '2024-07-01', 100),
(3, '2024-07-01', 140);

SELECT
    item_id, date_time, sales,
    sum(sales) OVER (ORDER BY date_time ROWS BETWEEN
        UNBOUNDED PRECEDING AND CURRENT ROW) sum
FROM
    test_window_order;
```

Due to duplicate values in the sorting column `date_time`, the following two query results may be observed:

```
+---------+------------+-------+------+
| item_id | date_time  | sales | sum  |
+---------+------------+-------+------+
|       1 | 2024-07-01 |   100 |  100 |
|       3 | 2024-07-01 |   140 |  240 |
|       2 | 2024-07-01 |   100 |  340 |
+---------+------------+-------+------+
3 rows in set (0.03 sec)
```

2. Solution

To address this issue, you can add a unique value column, such as `item_id`, to the `ORDER BY` clause to ensure the uniqueness of the ordering.

```
SELECT
        item_id,
        date_time,
        sales,
        sum(sales) OVER (
        ORDER BY item_id,
        date_time ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) sum
FROM
        test_window_order;
```

This results in a consistent query output:

```
+---------+------------+-------+------+
| item_id | date_time  | sales | sum  |
+---------+------------+-------+------+
|       1 | 2024-07-01 |   100 |  100 |
|       2 | 2024-07-01 |   100 |  200 |
|       3 | 2024-07-01 |   140 |  340 |
```

```
+---------+------------+-------+------+
3 rows in set (0.03 sec)
```

For more information on analytic functions, refer to the Oracle official documentation SQL for Analysis and Reporting

Reference

The table creation statement used in the example is as follows:

```
CREATE DATABASE IF NOT EXISTS doc_tpcds;
USE doc_tpcds;

CREATE TABLE IF NOT EXISTS item (
    i_item_sk bigint not null,
    i_item_id char(16) not null,
    i_rec_start_date date,
    i_rec_end_date date,
    i_item_desc varchar(200),
    i_current_price decimal(7,2),
    i_wholesale_cost decimal(7,2),
    i_brand_id integer,
    i_brand char(50),
    i_class_id integer,
    i_class char(50),
    i_category_id integer,
    i_category char(50),
    i_manufact_id integer,
    i_manufact char(50),
    i_size char(20),
    i_formulation char(20),
    i_color char(20),
    i_units char(10),
    i_container char(10),
    i_manager_id integer,
    i_product_name char(50)
)
DUPLICATE KEY(i_item_sk)
DISTRIBUTED BY HASH(i_item_sk) BUCKETS 12
PROPERTIES (
  "replication_num" = "1"
);

CREATE TABLE IF NOT EXISTS store_sales (
    ss_item_sk bigint not null,
    ss_ticket_number bigint not null,
    ss_sold_date_sk bigint,
    ss_sold_time_sk bigint,
    ss_customer_sk bigint,
    ss_cdemo_sk bigint,
    ss_hdemo_sk bigint,
    ss_addr_sk bigint,
    ss_store_sk bigint,
    ss_promo_sk bigint,
    ss_quantity integer,
    ss_wholesale_cost decimal(7,2),
    ss_list_price decimal(7,2),
    ss_sales_price decimal(7,2),
    ss_ext_discount_amt decimal(7,2),
    ss_ext_sales_price decimal(7,2),
    ss_ext_wholesale_cost decimal(7,2),
```

```
    ss_ext_list_price decimal(7,2),
    ss_ext_tax decimal(7,2),
    ss_coupon_amt decimal(7,2),
    ss_net_paid decimal(7,2),
    ss_net_paid_inc_tax decimal(7,2),
    ss_net_profit decimal(7,2)
)
DUPLICATE KEY(ss_item_sk, ss_ticket_number)
DISTRIBUTED BY HASH(ss_item_sk, ss_ticket_number) BUCKETS 32
PROPERTIES (
  "replication_num" = "1"
);

CREATE TABLE IF NOT EXISTS date_dim (
    d_date_sk bigint not null,
    d_date_id char(16) not null,
    d_date date,
    d_month_seq integer,
    d_week_seq integer,
    d_quarter_seq integer,
    d_year integer,
    d_dow integer,
    d_moy integer,
    d_dom integer,
    d_qoy integer,
    d_fy_year integer,
    d_fy_quarter_seq integer,
    d_fy_week_seq integer,
    d_day_name char(9),
    d_quarter_name char(6),
    d_holiday char(1),
    d_weekend char(1),
    d_following_holiday char(1),
    d_first_dom integer,
    d_last_dom integer,
    d_same_day_ly integer,
    d_same_day_lq integer,
    d_current_day char(1),
    d_current_week char(1),
    d_current_month char(1),
    d_current_quarter char(1),
    d_current_year char(1)
)
DUPLICATE KEY(d_date_sk)
DISTRIBUTED BY HASH(d_date_sk) BUCKETS 12
PROPERTIES (
  "replication_num" = "1"
);

CREATE TABLE IF NOT EXISTS customer_address (
    ca_address_sk bigint not null,
    ca_address_id char(16) not null,
    ca_street_number char(10),
    ca_street_name varchar(60),
    ca_street_type char(15),
    ca_suite_number char(10),
    ca_city varchar(60),
    ca_county varchar(30),
    ca_state char(2),
    ca_zip char(10),
    ca_country varchar(20),
    ca_gmt_offset decimal(5,2),
    ca_location_type char(20)
)
DUPLICATE KEY(ca_address_sk)
```

```
DISTRIBUTED BY HASH(ca_address_sk) BUCKETS 12
PROPERTIES (
  "replication_num" = "1"
);
```

Execute the following command on the terminal to download the data to the local computer and load the data into the table using the Stream Load method:

```
curl -L https://assets-cdn.selectdb.com/media/static/doc_ddl_dir_d27a752a7b.tar -o - | tar -xf -

curl --location-trusted \
-u "root:" \
-H "column_separator:|" \
-H "columns: i_item_sk, i_item_id, i_rec_start_date, i_rec_end_date, i_item_desc, i_current_price, i_wholesale_cost, i_brand_id,
i_brand, i_class_id, i_class, i_category_id, i_category, i_manufact_id, i_manufact, i_size, i_formulation, i_color, i_units,
i_container, i_manager_id, i_product_name" \
-T "doc_ddl_dir/item_1_10.dat" \
http://127.0.0.1:8030/api/doc_tpcds/item/_stream_load

curl --location-trusted \
-u "root:" \
-H "column_separator:|" \
-H "columns: d_date_sk, d_date_id, d_date, d_month_seq, d_week_seq, d_quarter_seq, d_year, d_dow, d_moy, d_dom, d_qoy, d_fy_year,
d_fy_quarter_seq, d_fy_week_seq, d_day_name, d_quarter_name, d_holiday, d_weekend, d_following_holiday, d_first_dom, d_last_dom,
d_same_day_ly, d_same_day_lq, d_current_day, d_current_week, d_current_month, d_current_quarter, d_current_year" \
-T "doc_ddl_dir/date_dim_1_10.dat" \
http://127.0.0.1:8030/api/doc_tpcds/date_dim/_stream_load

curl --location-trusted \
-u "root:" \
-H "column_separator:|" \
-H "columns: ss_sold_date_sk, ss_sold_time_sk, ss_item_sk, ss_customer_sk, ss_cdemo_sk, ss_hdemo_sk, ss_addr_sk, ss_store_sk,
ss_promo_sk, ss_ticket_number, ss_quantity, ss_wholesale_cost, ss_list_price, ss_sales_price, ss_ext_discount_amt,
ss_ext_sales_price, ss_ext_wholesale_cost, ss_ext_list_price, ss_ext_tax, ss_coupon_amt, ss_net_paid, ss_net_paid_inc_tax,
ss_net_profit" \
-T "doc_ddl_dir/store_sales.csv" \
http://127.0.0.1:8030/api/doc_tpcds/store_sales/_stream_load

curl --location-trusted \
-u "root:" \
-H "column_separator:|" \
-H "columns: ca_address_sk, ca_address_id, ca_street_number, ca_street_name, ca_street_type, ca_suite_number, ca_city, ca_county,
ca_state, ca_zip, ca_country, ca_gmt_offset, ca_location_type" \
-T "doc_ddl_dir/customer_address_1_10.dat" \
http://127.0.0.1:8030/api/doc_tpcds/customer_address/_stream_load
```

The data files `item_1_10.dat`, `date-dim_1_10.dat`, `store_stales.csv`, and `customer-address_1_10.dat` can be downloaded by clicking on the link.

Common Table Expression

Description

Common Table Expression (CTE) define a temporary result set that can be referenced multiple times within the scope of an SQL statement. CTEs are primarily used in SELECT statements.

To specify a CTE, use the `WITH` clause with one or more comma-separated clauses. Each clause provides a subquery that generates a result set and associates a name with the subquery.

Doris supports nested CTE. Within the statement that contains the `WITH` clause, you can reference each CTE name to access the corresponding CTE result set. CTE names can be referenced in other CTE, allowing you to define CTE based on other CTE.

Doris **DOES NOT** support recursive CTE. For more information, please read MySQL manual about recursive CTE

Example

Simple CTE

The following example defines CTE named cte1 and cte2 within the WITH clause and refers to them in the top-level SELECT below the WITH clause:

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

Nested CTE

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM cte1)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

Recursive CTE (NOT Support)

```sql WITH r_cte AS ( SELECT 1 AS user_id, 2 as manager_id UNION ALL SELECT user_id, manager_id FROM r_cte INNER JOIN (SELECT 1 AS user_id, 2 as manager_id) t ON r_cte.manager_id = t.user_id ) SELECT * FROM r_cte
```

Alias Function

Introduction

An alias function refers to assigning an alias name to a function. By registering a new signature for a function or expression fragment in the system, it aims to enhance compatibility or increase convenience.

Alias functions, like other custom functions, support two scopes: `LOCAL` and `GLOBAL`.

- `LOCAL`: Alias functions are registered under the current database scope. If the alias function needs to be used under other databases, its fully qualified name should be used, i.e., `<Database Name>.<Function Name>`.
- `GLOBAL`: Alias functions are registered under the global scope. They can be directly accessed by their function names under any database.

Use Case

Assigning Aliases to Functions

This scenario is common in system migration. When users have existing queries targeting other database systems, there may be functions in these queries that have the same functionality as a function in Doris but different names. In this case, by defining a new alias function for this function, migration can be completed without users noticing any changes.

Simplifying Query Statements

This scenario is often seen in complex analyses. When writing complex query statements, there may be a large number of repetitive expression fragments within a statement or across different statements. By creating an alias function for this complex expression fragment, the query statement can be simplified, enhancing writing convenience and maintainability.

Supported Scope

Expression Requirements

Currently, alias functions require that the root node of the actual expression they point to must be a function expression.

Legal Examples:

```
-- Create an alias function named func with parameters INT, INT, actually pointing to the expression abs(foo + bar);
CREATE ALIAS FUNCTION func(INT, INT) WITH PARAMETER(foo, bar) AS abs(foo + bar);
-- Create an alias function named func with parameters DATETIMEV2(3), INT, actually pointing to the expression
date_trunc(days_sub(foo, bar), 'day')
CREATE ALIAS FUNCTION func(DATETIMEV2(3), INT) WITH PARAMETER (foo, bar) AS date_trunc(days_sub(foo, bar), 'day')
```

Illegal Example:

```
-- The root expression is not a function
CREATE ALIAS FUNCTION func(INT, INT) WITH PARAMETER(foo, bar) AS foo + bar;
```

Parameter Requirements

Currently, alias functions do not support variable-length parameters and must have at least one parameter.

Java UDF, UDAF, UDWF, UDTF

Overview

Java UDF provides a Java interface for users to implement user-defined functions (UDFs) conveniently using the Java programming language. Doris supports the use of Java to develop UDFs, UDAFs, and UDTFs. Unless otherwise specified, “UDF” in the following text refers to all types of user-defined functions.

1. Java UDF: A Java UDF is a commonly used scalar function, where each input row produces a corresponding output row. Common examples include ABS and LENGTH. Notably, Hive UDFs can be directly migrated to Doris, which is convenient for users.

2. Java UDAF: A Java UDAF is a user-defined aggregate function that aggregates multiple input rows into a single output row. Common examples include MIN, MAX, and COUNT.

3. Java UDWF: stands for User-Defined Window Function, which returns a computed value for each row based on a window (one or multiple rows). Common examples include ROW_NUMBER, RANK, and DENSE_RANK.

4. Java UDTF: A Java UDTF is a user-defined table function, where a single input row can generate one or multiple output rows. In Doris, UDTFs must be used with Lateral View to achieve row-to-column transformations. Common examples include EXPLODE and EXPLODE_SPLIT. **Java UDTF is available from version 3.0.0 and onwards.**

Type Correspondence

Type	UDF Argument Type
Bool	Boolean
TinyInt	Byte
SmallInt	Short
Int	Integer
BigInt	Long
LargeInt	BigInteger
Float	Float
Double	Double
Date	LocalDate
Datetime	LocalDateTime
String	String
Decimal	BigDecimal
`array<Type>`	`ArrayList<Type>`

Type	UDF Argument Type
map<Type1,Type2>	HashMap<Type1,Type2>
struct<Type...>	ArrayList<Object> (from version 3.0.0)

> **array/map/struct types can be nested with other types. For instance, Doris: array<array<int>> corresponds to JAVA UDF Argument Type: ArrayList<ArrayList<Integer>>. Other types follow the same pattern.**

> **Warning**
>
> When creating functions, avoid using varchar in place of string, as this may cause the function to fail.

Usage Notes

1. Complex data types (HLL, Bitmap) are not supported.
2. Users are currently allowed to specify the maximum JVM heap size. The configuration item is the -Xmx part of JAVA_OPTS in be.conf. The default is 1024m. If you need to aggregate data, it is recommended to increase this value to enhance performance and reduce the risk of memory overflow.
3. Due to issues with JVM loading classes with the same name, do not use multiple classes with the same name as UDF implementations simultaneously. If you want to update a UDF with a class of the same name, you need to restart BE to reload the classpath.

Getting Started

This section mainly introduces how to develop a Java UDF. Examples are provided in samples/doris-demo/java-udf-demo/ for reference. Click here to view details.

The usage of UDFs is identical to standard functions, with the primary distinction being that built-in functions have a global scope, while UDFs are scoped within the DB.

When the session is linked within the database, directly using the UDF name will search for the corresponding UDF within the current DB. Otherwise, users must explicitly specify the UDF's database name, for example, dbName.funcName.

In the following sections, examples will use the table test_table. The corresponding table creation script is as follows:

```
CREATE TABLE `test_table` (
    id int NULL,
    d1 double NULL,
    str string NULL
) ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
```

```
"replication_num" = "1");

insert into test_table values (1, 111.11, "a,b,c");
insert into test_table values (6, 666.66, "d,e");
```

Introduction to Java-UDF Example

When writing a UDF in Java, the main entry point must be the `evaluate` function. This is consistent with other engines like Hive. In this example, we write an `AddOne` UDF to perform an increment operation on integer inputs.

1. Write the corresponding Java code and package it into a JAR file.

```
public class AddOne extends UDF {
    public Integer evaluate(Integer value) {
        return value == null ? null : value + 1;
    }
}
```

2. Register and create the Java-UDF function in Doris. For more details on the syntax, refer to CREATE FUNCTION.

```
CREATE FUNCTION java_udf_add_one(int) RETURNS int PROPERTIES (
    "file"="file:///path/to/java-udf-demo-jar-with-dependencies.jar",
    "symbol"="org.apache.doris.udf.AddOne",
    "always_nullable"="true",
    "type"="JAVA_UDF"
);
```

> **Note**
>
> When using `file://`, Doris reads the JAR file on the FE node during `CREATE FUNCTION` to compute the checksum, and reads the JAR file on BE nodes during execution. Therefore, the same JAR file must exist at the same absolute path on all FE and BE nodes, and the file content must be identical. If the file exists only on BE nodes, `CREATE FUNCTION` fails. If the file content differs between FE and BE nodes, execution fails with a checksum mismatch.

3. To utilize UDFs, users must possess the `SELECT` privilege for the corresponding database. And to verify the successful registration of the UDF, you can use the SHOW FUNCTIONS command.

```
select id,java_udf_add_one(id) from test_table;
+------+----------------------+
| id   | java_udf_add_one(id) |
+------+----------------------+
|    1 |                    2 |
|    6 |                    7 |
+------+----------------------+
```

4. If a UDF is no longer needed, it can be dropped using the following command, as detailed in DROP FUNCTION.

Additionally, if your UDF requires loading large resource files or defining global static variables, you can refer to the method for loading static variables described later in this document.

Introduction to Java-UDAF Example

When writing a `UDAF` using Java, there are some functions that must be implemented (marked as required) along with an internal class State. The following example will illustrate how to implement them.

1. Write the corresponding Java UDAF code and package it into a JAR file.

** Example 1: SimpleDemo will implement a simple function similar to sum, where the input parameter is INT and the output parameter is INT.**

```
package org.apache.doris.udf;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.logging.Logger;

public class SimpleDemo  {

    Logger log = Logger.getLogger("SimpleDemo");

    //Need an inner class to store data
    /*required*/
    public static class State {
        /*some variables if you need */
        public int sum = 0;
    }

    /*required*/
    public State create() {
        /* here could do some init work if needed */
        return new State();
    }

    /*required*/
    public void destroy(State state) {
        /* here could do some destroy work if needed */
    }

    /*Not Required*/
    public void reset(State state) {
        /*if you want this udaf function can work with window function.*/
        /*Must impl this, it will be reset to init state after calculate every window frame*/
        state.sum = 0;
    }

    /*required*/
    //first argument is State, then other types your input
    public void add(State state, Integer val) throws Exception {
        /* here doing update work when input data*/
        if (val != null) {
            state.sum += val;
        }
    }

    /*required*/
    public void serialize(State state, DataOutputStream out) throws Exception {
        /* serialize some data into buffer */
```

```
        out.writeInt(state.sum);
    }

    /*required*/
    public void deserialize(State state, DataInputStream in) throws Exception {
        /* deserialize get data from buffer before you put */
        int val = in.readInt();
        state.sum = val;
    }

    /*required*/
    public void merge(State state, State rhs) throws Exception {
        /* merge data from state */
        state.sum += rhs.sum;
    }

    /*required*/
    //return Type you defined
    public Integer getValue(State state) throws Exception {
        /* return finally result */
        return state.sum;
    }
}
```

** Example 2: MedianUDAF is a function that calculates the median. The input types are (DOUBLE, INT), and the output type is DOUBLE. **

```
package org.apache.doris.udf.demo;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.math.BigDecimal;
import java.util.Arrays;
import java.util.logging.Logger;

/* UDAF to calculate the median */
public class MedianUDAF {
    Logger log = Logger.getLogger("MedianUDAF");

    // State storage
    public static class State {
        // Precision of the return result
        int scale = 0;
        // Whether it is the first time to execute the add method for a certain aggregation condition under a certain tablet
        boolean isFirst = true;
        // Data storage
        public StringBuilder stringBuilder;
    }

    // Initialize the state
    public State create() {
        State state = new State();
        // Pre-initialize based on the amount of data that needs to be aggregated under each aggregation condition of each tablet to
increase performance
        state.stringBuilder = new StringBuilder(1000);
        return state;
    }

    // Process each data under respective aggregation conditions for each tablet
    public void add(State state, Double val, int scale) {
        if (val != null && state.isFirst) {
            state.stringBuilder.append(scale).append(",").append(val).append(",");
```

```
            state.isFirst = false;
        } else if (val != null) {
            state.stringBuilder.append(val).append(",");
        }
    }

    // Data needs to be output for aggregation after processing
    public void serialize(State state, DataOutputStream out) throws IOException {
        // Currently, only DataOutputStream is provided. If serialization of objects is required, methods such as concatenating
strings, converting to JSON, or serializing into byte arrays can be considered
        // If the State object needs to be serialized, it may be necessary to implement a serialization interface for the State
inner class
        // Ultimately, everything needs to be transmitted via DataOutputStream
        out.writeUTF(state.stringBuilder.toString());
    }

    // Obtain the output data from the data processing execution unit
    public void deserialize(State state, DataInputStream in) throws IOException {
        String string = in.readUTF();
        state.scale = Integer.parseInt(String.valueOf(string.charAt(0)));
        StringBuilder stringBuilder = new StringBuilder(string.substring(2));
        state.stringBuilder = stringBuilder;
    }

    // The aggregation execution unit merges the processing results of data under certain aggregation conditions for a given key.
The state1 parameter is the initialized instance during the first merge of each key
    public void merge(State state1, State state2) {
        state1.scale = state2.scale;
        state1.stringBuilder.append(state2.stringBuilder.toString());
    }

    // Output the final result after merging the data for each key
    public Double getValue(State state) {
        String[] strings = state.stringBuilder.toString().split(",");
        double[] doubles = new double[strings.length];
        for (int i = 0; i < strings.length - 1; i++) {
            doubles[i] = Double.parseDouble(strings[i + 1]);
        }

        Arrays.sort(doubles);
        double n = doubles.length;
        if (n == 0) {
            return 0.0;
        }
        double index = (n - 1) / 2.0;

        int low = (int) Math.floor(index);
        int high = (int) Math.ceil(index);

        double value = low == high ? (doubles[low] + doubles[high]) / 2 : doubles[high];

        BigDecimal decimal = new BigDecimal(value);
        return decimal.setScale(state.scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }

    // Executed after each execution unit completes
    public void destroy(State state) {
    }
}
```

2. Register and create the Java-UDAF function in Doris. For more syntax details, please refer to CREATE FUNCTION.

```
CREATE AGGREGATE FUNCTION simple_demo(INT) RETURNS INT PROPERTIES (
    "file"="file:///pathTo/java-udaf.jar",
    "symbol"="org.apache.doris.udf.SimpleDemo",
    "always_nullable"="true",
    "type"="JAVA_UDF"
);
```

3. When using Java-UDAF, you can perform aggregation either by grouping or by aggregating all results:

```
select simple_demo(id) from test_table group by id;
+-----------------+
| simple_demo(id) |
+-----------------+
|               1 |
|               6 |
+-----------------+
```

```
select simple_demo(id) from test_table;
+-----------------+
| simple_demo(id) |
+-----------------+
|               7 |
+-----------------+
```

Introduction to Java-UDWF Example

1. The implementation is similar to Java UDAF, but requires an additional reset() method to clear the state.

```
void reset(State state)
```

2. Register and create the Java-UDWF function same as UDAF in Doris. For more syntax details, please refer to CREATE FUNCTION.

```
CREATE AGGREGATE FUNCTION simple_demo_window(INT) RETURNS INT PROPERTIES (
    "file"="file:///pathTo/java-udaf.jar",
    "symbol"="org.apache.doris.udf.SimpleDemo",
    "always_nullable"="true",
    "type"="JAVA_UDF"
);
```

3. Java UDWF allows querying computed results within specific window frames. For detailed syntax, refer to Window Function

```
select id, simple_demo_window(id) over(partition by id order by d1 rows between 1 preceding and 1 following) as res from
test_table;
    +------+------+
    | id   | res  |
    +------+------+
    |    1 |    1 |
    |    6 |    6 |
    +------+------+
```

Introduction to Java-UDTF Example

> **UDTF is supported starting from Doris version 3.0.**

Best Practices

Loading static variables

Currently, in Doris, executing a UDF function, e.g., `select udf(col) from table`, will load the udf.jar package for each concurrent instance, and unload the udf.jar package when the instance finish.

If the udf.jar file needs to load a file of several hundred MBs, the memory usage will increase sharply due to concurrency, potentially leading to OOM (Out of Memory).

Alternatively, if you want to use a connection pool, this approach will not allow you to initialize it only once in the static area.

Here are two solutions, with the second solution requiring Doris version branch-3.0 or above.

Solution 1:

The solution is to split the resource loading code, generate a separate jar package, and have other packages directly reference this resource jar package.

Assume the files have been split into `DictLibrary` and `FunctionUdfAR`.

```
public class DictLibrary {
    private static HashMap<String, String> res = new HashMap<>();

    static {
        // suppose we built this dictionary from a certain local file.
        res.put("key1", "value1");
        res.put("key2", "value2");
        res.put("key3", "value3");
        res.put("0", "value4");
        res.put("1", "value5");
        res.put("2", "value6");
    }

    public static String evaluate(String key) {
        if (key == null) {
            return null;
        }
        return res.get(key);
    }
}
```

```
public class FunctionUdf {
    public String evaluate(String key) {
        String value = DictLibrary.evaluate(key);
        return value;
    }
}
```

1. Compile the DictLibrary file separately to generate an independent jar package, resulting in a resource file DictLibrary.jar:

```
javac ./DictLibrary.java
jar -cf ./DictLibrary.jar ./DictLibrary.class
```

2. Then compile the FunctionUdf file, directly referencing the resource package from the previous step, resulting in the FunctionUdf.jar package:

```
javac -cp ./DictLibrary.jar ./FunctionUdf.java
jar -cvf ./FunctionUdf.jar ./FunctionUdf.class
```

3. After the above two steps, you will get two jar packages. To allow the resource jar package to be referenced by all concurrent instances, place it in the deployment path `be/custom_lib`. After the restarting, it will be loaded with the JVM startup. As a result, the resources will be loaded when the service starts and released when the service stops.

4. Finally, use the `create function` statement to create a UDF function

```
CREATE FUNCTION java_udf_dict(string) RETURNS string PROPERTIES (
 "file"="file:///pathTo/FunctionUdf.jar",
 "symbol"="org.apache.doris.udf.FunctionUdf",
 "always_nullable"="true",
 "type"="JAVA_UDF"
);
```

Solution 2:

The BE (Backend) globally caches the JAR file and customizes the expiration and eviction time. When creating a function, two additional properties are added:

static_load: This defines whether to use the static cache loading method. expiration_time: This defines the expiration time of the JAR file, in minutes. If the static cache loading method is used, the UDF instance will be cached after the first call and initialization. On subsequent calls to the UDF, the system will first search in the cache. If not found, the initialization process will be triggered.

Additionally, a background thread regularly checks the cache. If the function has not been called within the configured expiration time, it will be evicted from the cache. If the function is called, the cache timestamp will be automatically updated.

```
public class Print extends UDF {
    static Integer val = 0;
    public Integer evaluate() {
        val = val + 1;
        return val;
    }
}
```

```
CREATE FUNCTION print_12() RETURNS int
PROPERTIES (
    "file" = "file:///path/to/java-udf-demo-jar-with-dependencies.jar",
    "symbol" = "org.apache.doris.udf.Print",
```

```
    "always_nullable"="true",
    "type" = "JAVA_UDF",
    "static_load" = "true", // default value is false
    "expiration_time" = "60" // default value is 360 minutes
);
```

As we can see, the result keeps incrementing, which proves that the loaded JAR file is not being unloaded and reloaded. Instead, the variables are being re-initialized to 0.

```
mysql [test_query_qa]>select print_12();
+------------+
| print_12() |
+------------+
|          1 |
+------------+
1 row in set (0.40 sec)

mysql [test_query_qa]>select print_12();
+------------+
| print_12() |
+------------+
|          2 |
+------------+
1 row in set (0.03 sec)

mysql [test_query_qa]>select print_12();
+------------+
| print_12() |
+------------+
|          3 |
+------------+
1 row in set (0.04 sec)
```

Complex Type

Doris supports complex types such as Array, Map, Struct, and JSON.

Doris provides various functions specifically designed for these complex types.

For detailed information on the supported functions, please refer to the SQL manual - Array functions, SQL manual - Map functions data type, SQL manual - Struct functions, and SQL manual - Json functions.

Column to Row (Lateral View)

Column to Row (Lateral View)

Used in conjunction with generator functions such as `EXPLODE`, will generate a virtual table containing one or more rows. `LATERAL VIEW` applies rows to each raw input row.

Grammar

```
LATERAL VIEW  generator_function ( expression [, ...] ) table_identifier AS column_identifier [, ...]
```

Parameters

- generator_function

  Generator functions (EXPLODE, EXPLODE_SPLIT, etc.).

- table_identifier

  Alias for `generator_function`.

- column_identifier

  List column alias `generator_function`, which can be used to reference the output columns. The number of column identifiers must match the number of columns returned by the generator function.

Example

```
CREATE TABLE `person` (
  `id` int(11) NULL,
  `name` text NULL,
  `age` int(11) NULL,
  `class` int(11) NULL,
  `address` text NULL
) ENGINE=OLAP
UNIQUE KEY(`id`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1",
"in_memory" = "false",
"storage_format" = "V2",
"disable_auto_compaction" = "false"
);

INSERT INTO person VALUES
    (100, 'John', 30, 1, 'Street 1'),
    (200, 'Mary', NULL, 1, 'Street 2'),
    (300, 'Mike', 80, 3, 'Street 3'),
    (400, 'Dan', 50, 4, 'Street 4');
```

```
mysql> SELECT * FROM person
    ->     LATERAL VIEW EXPLODE(ARRAY(30, 60)) tableName AS c_age;
+------+------+------+-------+----------+-------+
| id   | name | age  | class | address  | c_age |
+------+------+------+-------+----------+-------+
|  100 | John |   30 |     1 | Street 1 |    30 |
|  100 | John |   30 |     1 | Street 1 |    60 |
|  200 | Mary | NULL |     1 | Street 2 |    30 |
|  200 | Mary | NULL |     1 | Street 2 |    60 |
|  300 | Mike |   80 |     3 | Street 3 |    30 |
|  300 | Mike |   80 |     3 | Street 3 |    60 |
|  400 | Dan  |   50 |     4 | Street 4 |    30 |
|  400 | Dan  |   50 |     4 | Street 4 |    60 |
+------+------+------+-------+----------+-------+
8 rows in set (0.12 sec)
```

Tuning Overview

Query performance tuning is a systematic process that requires multi-level and multi-dimensional adjustments to the database system. Below is an overview of the tuning process and methodology:

1. Firstly, business personnel and database administrators (DBAs) need to have a comprehensive understanding of the database system being used, including the hardware utilized by the business system, the scale of the cluster, the version of the database software being used, as well as the features provided by the specific software version.
2. Secondly, an effective performance diagnostic tool is a necessary prerequisite for identifying performance issues. Only by efficiently and quickly locating problematic SQL queries or slow SQL queries can subsequent specific performance tuning processes be carried out.
3. After entering the performance tuning phase, a range of commonly used performance analysis tools are indispensable. These include specialized tools provided by the currently running database system, as well as general tools at the operating system level.
4. With these tools in place, specialized tools can be used to obtain detailed information about SQL queries running on the current database system, aiding in the identification of performance bottlenecks. Meanwhile, general tools can serve as auxiliary analysis methods to assist in locating issues.

In summary, performance tuning requires evaluating the current system's performance status from a holistic perspective. Firstly, it is necessary to identify business SQL queries with performance issues, then utilize analysis tools to discover performance bottlenecks, and finally implement specific tuning operations.

Based on the aforementioned tuning process and methodology, Apache Doris provides corresponding tools at each of these levels. The following sections will introduce the performance diagnostic tools, analysis tools, and tuning process respectively.

Diagnostic Tools

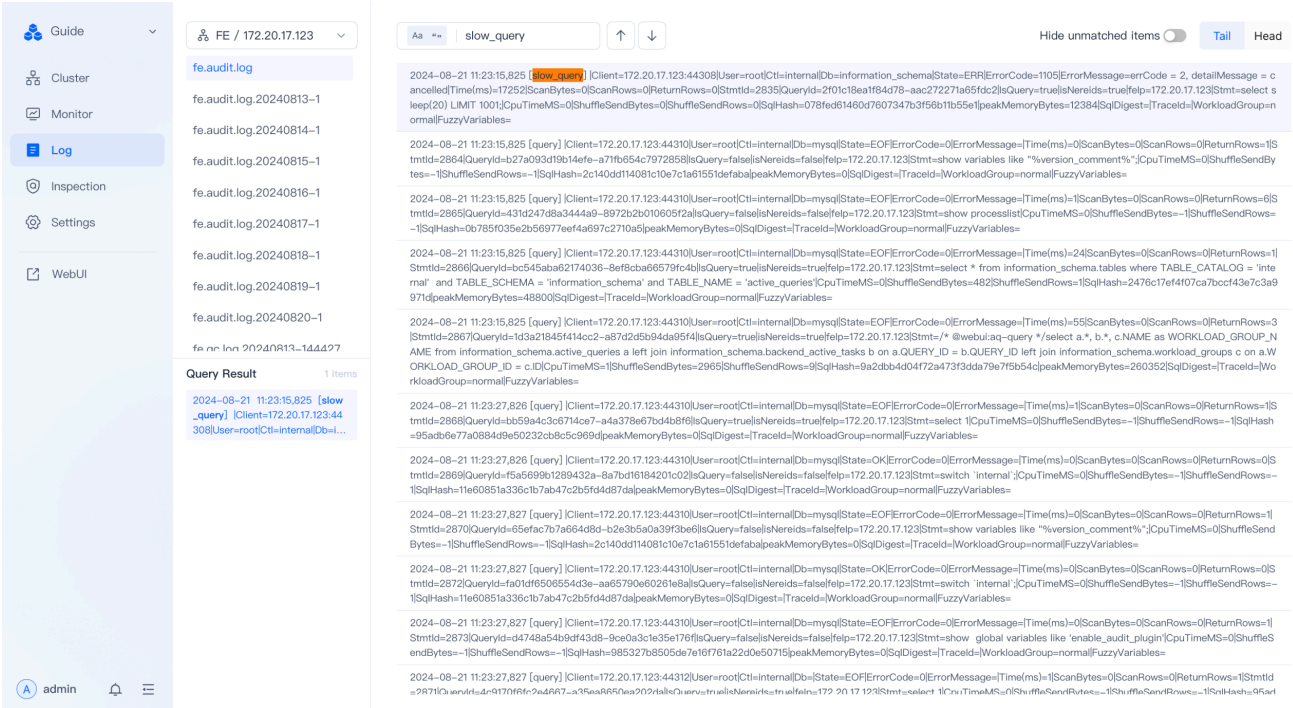
Overview

Efficient and effective performance diagnostic tools are crucial for database system tuning, as they determine whether problematic business SQL queries can be quickly identified, and subsequently, performance bottlenecks can be rapidly pinpointed and resolved, ensuring that the database system meets its Service Level Agreements (SLAs).

Currently, Doris considers SQL queries with execution times exceeding 5 seconds as slow SQL by default. This threshold can be configured via `config.qe_slow_log_ms`. Doris currently offers the following three diagnostic channels to help quickly identify slow SQL queries with performance issues:

Doris Manager Logs

The log module in Doris Manager provides a slow SQL filtering function. Users can view slow SQL by selecting the `fe.audit.log` on a specific FE node. By simply entering `slow_query` in the search box, the historical slow SQL information of the current system will be displayed on the page, as shown in the figure below:



Doris Manager Monitoring and Logging

Audit Log

Currently, Doris FE provides four types of Audit Logs, including `slow_query`, `query`, `load`, and `stream_load`. Besides accessing the logs through the log page on the cluster where the Manager service is installed and deployed, Audit Logs can also be directly obtained by accessing the `fe/log/fe.audit.log` file on the node where FE is located.

By directly searching for the `slow_query` tag in `fe.audit.log`, you can quickly filter out slow-executing SQL queries, as shown below:

```
2024-07-18 11:23:13,042 [slow_query]
|Client=127.0.0.1:63510|User=root|Ctl=internal|Db=tpch_sf1000|State=EOF|ErrorCode=0|ErrorMessage=|Time(ms)=11603|ScanBytes=236667379
712|ScanRows=13649979418|ReturnRows=100|StmtId=1689|QueryId=91ff336304f14182-
9ca537eee75b3856|IsQuery=true|isNereids=true|feIp=172.21.0.10|Stmt=select     c_name,     c_custkey,     o_orderkey,
o_orderdate,     o_totalprice,     sum(l_quantity) from     customer,     orders,     lineitem where     o_orderkey  in  (
select             l_orderkey         from             lineitem         group  by             l_orderkey  having
sum(l_quantity)  >  300     )     and  c_custkey  =  o_custkey     and  o_orderkey  =  l_orderkey group  by     c_name,
c_custkey,     o_orderkey,     o_orderdate,     o_totalprice order  by     o_totalprice  desc,     o_orderdate limit
100|CpuTimeMS=918556|ShuffleSendBytes=3267419|ShuffleSendRows=89668|SqlHash=b4e1de9f251214a30188180f37907f7d|peakMemoryBytes=3872093
5552|SqlDigest=f23c7a7ecff61da33f537b2699e9b053|cloudClusterName=UNKNOWN|TraceId=|WorkloadGroup=normal|FuzzyVariables=|scanBytesFrom
LocalStorage=0|scanBytesFromRemoteStorage=0
2024-07-18 11:23:33,043 [slow_query]
|Client=127.0.0.1:26672|User=root|Ctl=internal|Db=tpch_sf1000|State=EOF|ErrorCode=0|ErrorMessage=|Time(ms)=8978|ScanBytes=3349855559
68|ScanRows=10717654374|ReturnRows=100|StmtId=1815|QueryId=6e1fae453cb04d9a-
b1e5f94d9cea1885|IsQuery=true|isNereids=true|feIp=172.21.0.10|Stmt=select     s_name,     count(*) as numwait from     supplier,
lineitem l1,     orders,     nation where     s_suppkey = l1.l_suppkey     and o_orderkey = l1.l_orderkey     and o_orderstatus =
'F'     and l1.l_receiptdate > l1.l_commitdate     and exists (         select             *         from             lineitem l2
where                 l2.l_orderkey = l1.l_orderkey           and l2.l_suppkey <> l1.l_suppkey     )     and not exists (
select             *         from             lineitem l3         where                 l3.l_orderkey = l1.l_orderkey           and
l3.l_suppkey <> l1.l_suppkey           and l3.l_receiptdate > l3.l_commitdate     )     and s_nationkey = n_nationkey     and n_name
= 'SAUDI ARABIA' group by     s_name order by     numwait desc,     s_name limit
100|CpuTimeMS=990127|ShuffleSendBytes=59208164|ShuffleSendRows=3651504|SqlHash=f8a30e4182d72cce3eff6cb385005b1f|peakMemoryBytes=1049
5660672|SqlDigest=fec5a7136f9375aa968a4de971b994da|cloudClusterName=UNKNOWN|TraceId=|WorkloadGroup=normal|FuzzyVariables=|scanBytesF
romLocalStorage=0|scanBytesFromRemoteStorage=0
2024-07-18 11:23:41,044 [slow_query]
|Client=127.0.0.1:26684|User=root|Ctl=internal|Db=tpch_sf1000|State=EOF|ErrorCode=0|ErrorMessage=|Time(ms)=8514|ScanBytes=3349865512
96|ScanRows=10717654374|ReturnRows=100|StmtId=1833|QueryId=4f91483464ce4aa8-
beeed7dcb8675bc8|IsQuery=true|isNereids=true|feIp=172.21.0.10|Stmt=select     s_name,     count(*) as numwait from     supplier,
lineitem l1,     orders,     nation where     s_suppkey = l1.l_suppkey     and o_orderkey = l1.l_orderkey     and o_orderstatus =
'F'     and l1.l_receiptdate > l1.l_commitdate     and exists (         select             *         from             lineitem l2
where                 l2.l_orderkey = l1.l_orderkey           and l2.l_suppkey <> l1.l_suppkey     )     and not exists (
select             *         from             lineitem l3         where                 l3.l_orderkey = l1.l_orderkey           and
l3.l_suppkey <> l1.l_suppkey           and l3.l_receiptdate > l3.l_commitdate     )     and s_nationkey = n_nationkey     and n_name
= 'SAUDI ARABIA' group by     s_name order by     numwait desc,     s_name limit
100|CpuTimeMS=925841|ShuffleSendBytes=59223190|ShuffleSendRows=3651602|SqlHash=f8a30e4182d72cce3eff6cb385005b1f|peakMemoryBytes=1050
5123104|SqlDigest=fec5a7136f9375aa968a4de971b994da|cloudClusterName=UNKNOWN|TraceId=|WorkloadGroup=normal|FuzzyVariables=|scanBytesF
romLocalStorage=0|scanBytesFromRemoteStorage=0
2024-07-18 11:23:49,044 [slow_query]
|Client=127.0.0.1:10748|User=root|Ctl=internal|Db=tpch_sf1000|State=EOF|ErrorCode=0|ErrorMessage=|Time(ms)=8660|ScanBytes=3349876736
00|ScanRows=10717654374|ReturnRows=100|StmtId=1851|QueryId=4599cb1bab204f80-
ac430dd78b45e3da|IsQuery=true|isNereids=true|feIp=172.21.0.10|Stmt=select     s_name,     count(*) as numwait from     supplier,
lineitem l1,     orders,     nation where     s_suppkey = l1.l_suppkey     and o_orderkey = l1.l_orderkey     and o_orderstatus =
'F'     and l1.l_receiptdate > l1.l_commitdate     and exists (         select             *         from             lineitem l2
where                 l2.l_orderkey = l1.l_orderkey           and l2.l_suppkey <> l1.l_suppkey     )     and not exists (
select             *         from             lineitem l3         where                 l3.l_orderkey = l1.l_orderkey           and
l3.l_suppkey <> l1.l_suppkey           and l3.l_receiptdate > l3.l_commitdate     )     and s_nationkey = n_nationkey     and n_name
= 'SAUDI ARABIA' group by     s_name order by     numwait desc,     s_name limit
100|CpuTimeMS=932664|ShuffleSendBytes=59223178|ShuffleSendRows=3651991|SqlHash=f8a30e4182d72cce3eff6cb385005b1f|peakMemoryBytes=1053
2849344|SqlDigest=fec5a7136f9375aa968a4de971b994da|cloudClusterName=UNKNOWN|TraceId=|WorkloadGroup=normal|FuzzyVariables=|scanBytesF
romLocalStorage=0|scanBytesFromRemoteStorage=0
```

The slow SQL obtained through `fe.audit.log` allows users to easily access detailed information such as execution time, number of rows scanned, number of rows returned, and the SQL statement itself, laying the foundation for further reproducing and locating performance issues.

Additionally, the Audit Log includes a `SqlDigest` field (e.g., `SqlDigest=...` in the example above). This field is a hash value generated from the structure of the SQL statement (with specific parameter values removed). By aggregating and analyzing `SqlDigest` in `slow_query`, you can identify “patterns” of slow queries. This means that

even if specific SQL statements differ slightly due to parameters, their `SqlDigest` will be identical as long as the structure is the same.

Using `SqlDigest`, users can determine which SQL patterns appear most frequently or consume the most time, allowing them to prioritize optimization for these “high-frequency” or “high-latency” patterns. This approach significantly improves the efficiency of slow query optimization by avoiding the inefficiency of analyzing individual SQL statements one by one.

It is important to note that `SqlDigest` itself is just a hash value and is not directly readable. Once the slow query pattern to be optimized is identified, you need to refer to the `Stmt` field in the Audit Log to get the specific SQL statement content corresponding to that pattern. Furthermore, the `QueryId` field can be used to retrieve detailed Profile information for the query (Profile retrieval and analysis will be detailed in subsequent sections) for in-depth performance analysis and optimization.

Slow Query Analysis Example

Taking the 4 slow query logs in the `fe.audit.log` above as an example, we can observe:

1. The first log (`Time(ms)=11603`) has a `SqlDigest` of `f23c7a7ecff61da33f537b2699e9b053`.
2. The subsequent three logs (`Time(ms)=8978`, `8514`, `8660`) all have the `SqlDigest` of `fec5a7136f9375aa968a4de971b994da`.

This indicates that the latter three slow queries belong to the same SQL pattern (structurally identical), although their specific execution times (`Time(ms)`) and details differ slightly.

In real-world optimization scenarios, if we find that a certain `SqlDigest` (such as `fec5a7136f9375aa968a4de971b994da`) appears repeatedly in the slow query logs, or accounts for a high proportion of the total execution time, we should prioritize optimizing this pattern.

Recommended Optimization Steps:

1. **Identify Business Logic**: By examining the `Stmt` field in any log entry of this pattern:

   ```
   select    s_name,    count(*) as numwait from    supplier,    lineitem l1,    orders,    nation where ...
   ```

   We can identify the specific SQL logic.

2. **Deep Analysis**: Use the `QueryId` (e.g., `6e1fae453cb04d9a-b1e5f94d9cea1885`) from the log to retrieve the corresponding Query Profile in Doris. The Profile helps analyze whether the issue is due to excessive data scanning, prolonged Join operations, or other reasons, enabling targeted optimization strategies (such as adding indexes, optimizing SQL writing, or adjusting table structures).

Once the issue with this pattern is resolved, all slow queries belonging to this pattern will be improved.

audit_log System Table

Starting from Doris version 2.1, the `audit_log` system table is provided under the `__internal_schema` database for users to view the execution status of SQL queries. Before using it, the global configuration `set global enable_audit_plugin=true;` needs to be enabled (this switch is disabled by default).

```
mysql> use __internal_schema;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------------------+
| Tables_in___internal_schema |
+-----------------------------+
| audit_log                   |
| column_statistics           |
| histogram_statistics        |
| partition_statistics        |
+-----------------------------+
4 rows in set (0.00 sec)

mysql> desc audit_log;
+-------------------+--------------+------+-------+---------+-------+
| Field             | Type         | Null | Key   | Default | Extra |
+-------------------+--------------+------+-------+---------+-------+
| query_id          | varchar(48)  | Yes  | true  | NULL    |       |
| time              | datetime     | Yes  | true  | NULL    |       |
| client_ip         | varchar(128) | Yes  | true  | NULL    |       |
| user              | varchar(128) | Yes  | false | NULL    | NONE  |
| catalog           | varchar(128) | Yes  | false | NULL    | NONE  |
| db                | varchar(128) | Yes  | false | NULL    | NONE  |
| state             | varchar(128) | Yes  | false | NULL    | NONE  |
| error_code        | int          | Yes  | false | NULL    | NONE  |
| error_message     | text         | Yes  | false | NULL    | NONE  |
| query_time        | bigint       | Yes  | false | NULL    | NONE  |
| scan_bytes        | bigint       | Yes  | false | NULL    | NONE  |
| scan_rows         | bigint       | Yes  | false | NULL    | NONE  |
| return_rows       | bigint       | Yes  | false | NULL    | NONE  |
| stmt_id           | bigint       | Yes  | false | NULL    | NONE  |
| is_query          | tinyint      | Yes  | false | NULL    | NONE  |
| frontend_ip       | varchar(128) | Yes  | false | NULL    | NONE  |
| cpu_time_ms       | bigint       | Yes  | false | NULL    | NONE  |
| sql_hash          | varchar(128) | Yes  | false | NULL    | NONE  |
| sql_digest        | varchar(128) | Yes  | false | NULL    | NONE  |
| peak_memory_bytes | bigint       | Yes  | false | NULL    | NONE  |
| stmt              | text         | Yes  | false | NULL    | NONE  |
+-------------------+--------------+------+-------+---------+-------+
```

Through the `audit_log` internal table, users can query detailed SQL execution information and perform detailed statistical analysis such as slow query filtering.

Summary

Doris Manager logs, audit logs, and the `audit_log` system table provide capabilities such as automatic or manual filtering of slow SQL queries, as well as fine-grained statistical analysis of SQL execution information. These tools offer powerful support for systematic performance diagnosis and tuning.

Analysis Tools

Overview

The previous section on diagnostic tools helped business and operations personnel pinpoint specific slow SQL queries. This section will introduce how to analyze the performance bottlenecks of slow SQL to determine which part of the SQL execution process is causing the slowdown.

The execution process of an SQL query can be roughly divided into two stages: plan generation and plan execution. The former is responsible for generating the execution plan, while the latter executes the specific plan. Issues in either part can lead to performance bottlenecks. For example, if a poor plan is generated, no matter how excellent the executor is, good performance cannot be achieved. Similarly, even with a correct plan, inappropriate execution methods can also lead to performance bottlenecks. Furthermore, the performance of the executor is closely related to the current hardware and system architecture. Deficiencies in infrastructure or incorrect configurations can also cause performance issues.

All three types of problems require the support of good analysis tools. Based on this, the Doris system provides two performance analysis tools to analyze bottlenecks in planning and execution respectively. Additionally, the system level also offers corresponding performance monitoring tools to assist in locating performance bottlenecks. The following sections will introduce these three aspects:

Doris Explain

An execution plan describes the specific execution method and process of an SQL query. For example, for an SQL query that joins two tables, the execution plan will show information such as how the tables are accessed, the join method, and the join order.

Doris provides the Explain tool, which conveniently displays detailed information about an SQL query's execution plan. By analyzing the plan output by Explain, users can quickly locate bottlenecks at the planning level and perform plan-level tuning based on different situations.

Doris offers multiple Explain tools with different levels of granularity, such as Explain Verbose, Explain All Plan, Explain Memo Plan, and Explain Shape Plan, which are used to display the final physical plan, logical plans at various stages, plans based on cost optimization processes, and plan shapes, respectively. For detailed information, please refer to the Execution Plan Explain section to learn about the usage of various Explain tools and the interpretation of their output information.

By analyzing the output of Explain, business personnel and DBAs can quickly locate performance bottlenecks in the current plan. For example, by analyzing the execution plan, it may be discovered that filters are not pushed down to the base tables, resulting in data not being filtered early and an excessive amount of data being involved in calculations, leading to performance issues. Another example is that in an Inner equi-join of two tables, the filter conditions on one side of the join condition are not derived to the other side, resulting in the data of the other table not being filtered early, which may also lead to suboptimal performance. Such performance bottlenecks can be located and resolved by analyzing the output of Explain.

For cases of using Doris Explain output to perform plan-level tuning, please refer to the Plan Tuning section.

Doris Profile

The Explain tool described above outlines the execution plan for an SQL query, such as planning a join operation between tables t1 and t2 as a Hash Join, with t1 designated as the build side and t2 as the probe side. When the SQL query is actually executed, understanding how much time each specific execution step takes—for instance, how long the build phase lasts and how long the probe phase lasts—is crucial for performance analysis and tuning. The Profile tool provides detailed execution information for this purpose. The following section first gives an overview of the Profile file structure and then introduces the meanings of execution times in Merged Profile, Execution Profile, and PipelineTask.

Profile File Structure

A Profile file contains several main sections:

1. Basic query information: including ID, time, database, etc.
2. The SQL statement and its execution plan.
3. Time spent by the Frontend (FE) on tasks like Plan Time, Schedule Time, etc.
4. Execution time spent by each operator during the Backend (BE) processing (including Merged Profile and Execution Profile).
5. The detailed information about the execution side is mainly contained in the last part. Next, we will mainly introduce what information the Profile can provide for performance analysis.

Merged Profile

To help users more accurately analyze performance bottlenecks, Doris provides aggregated profile results for each operator. Taking the EXCHANGE_OPERATOR as an example:

```
EXCHANGE_OPERATOR  (id=4):
    -  BlocksProduced:  sum  0,  avg  0,  max  0,  min  0
    -  CloseTime:  avg  34.133us,  max  38.287us,  min  29.979us
    -  ExecTime:  avg  700.357us,  max  706.351us,  min  694.364us
    -  InitTime:  avg  648.104us,  max  648.604us,  min  647.605us
    -  MemoryUsage:  sum  ,  avg  ,  max  ,  min
    -  PeakMemoryUsage:  sum  0.00  ,  avg  0.00  ,  max  0.00  ,  min  0.00
    -  OpenTime:  avg  4.541us,  max  5.943us,  min  3.139us
    -  ProjectionTime:  avg  0ns,  max  0ns,  min  0ns
    -  RowsProduced:  sum  0,  avg  0,  max  0,  min  0
    -  WaitForDependencyTime:  avg  0ns,  max  0ns,  min  0ns
    -  WaitForData0:  avg  9.434ms,  max  9.476ms,  min  9.391ms
```

The Merged Profile consolidates key metrics for each operator, with the core metrics and their meanings outlined below:

Metric Name	Metric Definition
BlocksProduced	Number of Data Blocks produced
CloseTime	Time spent by the Operator during the close phase
ExecTime	Total execution time of the Operator across all phases

Metric Name	Metric Definition
InitTime	Time spent by the Operator during the initialization phase
MemoryUsage	Memory usage of the Operator during execution
OpenTime	Time spent by the Operator during the open phase
ProjectionTime	Time spent by the Operator on projections
RowsProduced	Number of rows returned by the Operator
WaitForDependencyTime	Time the Operator waits for its execution dependencies

In Doris, each operator executes concurrently based on the concurrency level set by the user. Therefore, the Merged Profile calculates the Max, Avg, and Min values for each metric across all concurrent executions.

WaitForDependencyTime varies for each Operator, as the execution dependencies differ. For instance, in the case of an EXCHANGE_OPERATOR, the dependency is on data being sent by upstream operators via RPC. Thus, WaitForDependencyTime in this context specifically refers to the time spent waiting for upstream operators to send data.

Execution Profile

Unlike the Merged Profile, the Execution Profile displays detailed metrics for a specific concurrent execution. Taking the exchange operator with id=4 as an example:

```
EXCHANGE_OPERATOR  (id=4):(ExecTime:  706.351us)
      -  BlocksProduced:  0
      -  CloseTime:  38.287us
      -  DataArrivalWaitTime:  0ns
      -  DecompressBytes:  0.00
      -  DecompressTime:  0ns
      -  DeserializeRowBatchTimer:  0ns
      -  ExecTime:  706.351us
      -  FirstBatchArrivalWaitTime:  0ns
      -  InitTime:  647.605us
      -  LocalBytesReceived:  0.00
      -  MemoryUsage:
      -  PeakMemoryUsage:  0.00
      -  OpenTime:  5.943us
      -  ProjectionTime:  0ns
      -  RemoteBytesReceived:  0.00
      -  RowsProduced:  0
      -  SendersBlockedTotalTimer(*):  0ns
      -  WaitForDependencyTime:  0ns
      -  WaitForData0:  9.476ms
```

In this profile, for instance, LocalBytesReceived is a metric specific to the exchange operator and not found in other operators, hence it is not included in the Merged Profile.

PipelineTask Execution Time

In Doris, a PipelineTask consists of multiple operators. When analyzing the execution time of a PipelineTask, several key aspects need to be focused on: 1. ExecuteTime: The actual execution time of the entire PipelineTask, which is approximately equal to the sum of the ExecTime of all operators in this task 2.

WaitWorkerTime: The time that a task waits for a worker to execute. When a task is in the runnable state, it has to wait for an idle worker to execute it. The time it takes depends mainly on the cluster load. 3. Waiting time for executing dependencies: A task can be executed only when all the dependencies of each operator meet the execution conditions, and the time a task waits for executing dependencies is the sum of the waiting times of these dependencies. For example, simplifying one of the tasks in this example:

```sql
PipelineTask  (index=1):(ExecTime:  4.773ms)
  -  ExecuteTime:  1.656ms
      -  CloseTime:  90.402us
      -  GetBlockTime:  11.235us
      -  OpenTime:  1.448ms
      -  PrepareTime:  1.555ms
      -  SinkTime:  14.228us
  -  WaitWorkerTime:  63.868us
    DATA_STREAM_SINK_OPERATOR  (id=8,dst_id=8):(ExecTime:  1.688ms)
      -  WaitForDependencyTime:  0ns
          -  WaitForBroadcastBuffer:  0ns
          -  WaitForRpcBufferQueue:  0ns
    AGGREGATION_OPERATOR  (id=7  ,  nereids_id=648):(ExecTime:  398.12us)
      -  WaitForDependency[AGGREGATION_OPERATOR_DEPENDENCY]Time:  10.495ms
```

This task includes two operators (DATA_STREAM_SINK_OPERATOR - AGGREGATION_OPERATOR), of which DATA_STREAM_SINK_OPERATOR has two dependencies (WaitForBroadcastBuffer and WaitForRpcBufferQueue), and AGGREGATION_OPERATOR has one dependency (AGGREGATION_OPERATOR_DEPENDENCY), so the time consumption of the current task is distributed as follows:

```
1. ExecuteTime: 1.656ms (The actual execution time of the entire PipelineTask, which is approximately the sum of the ExecTime of all operators within the task).
2. WaitWorkerTime: 63.868us (The time the task waits for an execution worker. When the task is in a runnable state, it waits for an available worker to execute it, and this duration primarily depends on the cluster load).
3. Time Waiting for Execution Dependencies: 10.495ms (WaitForBroadcastBuffer + WaitForRpcBufferQueue + WaitForDependency[AGGREGATION_OPERATOR_DEPENDENCY]Time). The time a task waits for execution dependencies is the sum of the waiting times for these dependencies.
```

For cases of using Profile for execution-level tuning, please refer to the Tuning Execution section.

System-Level Performance Tools

Commonly used system tools can assist in identifying performance bottlenecks during execution. For instance, widely used Linux tools such as top, free, perf, sar, and iostat can be utilized to observe the CPU, memory, I/O, and network status of the system while SQL is running, thereby aiding in the identification of performance bottlenecks.

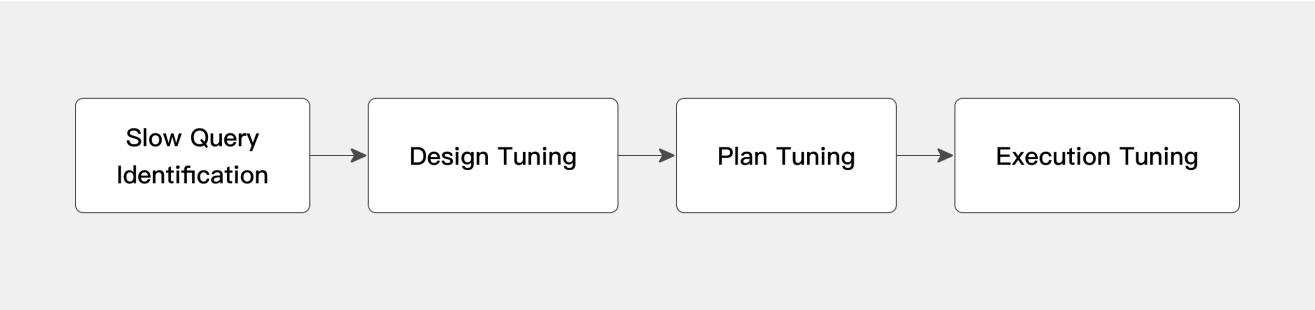
Summary

Effective performance analysis tools are crucial for quickly identifying performance bottlenecks. Doris provides Explain and Profile, offering powerful support for analyzing issues with execution plans and identifying which operations consume the most time during execution. Additionally, proficient use of system-level analysis tools can greatly assist in locating performance bottlenecks.

Tuning Process

Overview

Performance tuning is a systematic process that requires a comprehensive methodology and implementation framework for systematic diagnosis and optimization. With the strong support of diagnostic tools and analysis tools, the Doris system can efficiently diagnose, analyze, locate, and resolve performance issues. The complete four-step process for tuning is as follows:



Tuning process

Step 1: Use Performance Diagnostic Tools to Identify Slow Queries

For business systems running on Doris, use the aforementioned performance diagnostic tools to identify slow SQL queries.

- If Doris Manager is installed, it is recommended to use the Manager's log page for convenient visual identification of slow queries.
- If Manager is not installed, you can directly check the `fe.audit.log` file on the FE node or the audit_log system table to obtain a list of slow SQL queries and prioritize them for tuning.

Step 2: Schema Design and Tuning

After identifying specific slow SQL queries, the first priority is to inspect and tune the business schema design to eliminate performance issues caused by unreasonable schema design.

Schema design tuning can be divided into three aspects:

- Table-level Schema Design Tuning, such as adjusting the number of partitions and buckets, and field optimization;
- Index Design and Tuning
- The use of specific optimization techniques, such as Optimizing Join with Colocate Group. The main goal is to eliminate performance issues caused by unreasonable schema design or failure to fully leverage Doris's existing optimization capabilities.

For detailed tuning examples, please refer to the documentation on Plan Tuning.

Step 3: Plan Tuning

After inspecting and tuning the business schema, the main task of tuning begins: plan tuning and execution tuning. As mentioned above, at this stage, the primary task is to make full use of the various levels of Explain tools provided by Doris to systematically analyze the execution plans of slow SQL queries and identify key optimization points for targeted optimization.

- For single-table query and analysis scenarios, you can analyze the execution plan to check if partition pruning is working properly and use single-table materialized views for query acceleration.
- For complex multi-table analysis scenarios, you can analyze the Join Order to determine if it is reasonable and identify specific performance bottlenecks. You can also use multi-table materialized views for transparent rewriting to accelerate queries. If unexpected situations occur, such as unreasonable Join Order, you can manually specify the Join Hint to bind the execution plan, such as using the Leading hint to control Join Order, using the Shuffle Hint to adjust the Join shuffle method, and using Hints to control cost-based optimization rules, to achieve the goal of tuning the execution plan.
- For specific scenarios, you can also leverage advanced features provided by Doris, such as using SQL Cache to accelerate queries.

For detailed tuning examples, please refer to the documentation on Plan Tuning.

Step 4: Execution Tuning

In the execution tuning stage, you need to validate the effectiveness of plan tuning based on the actual execution of SQL queries. Additionally, within the framework of the existing plan, continue to analyze bottlenecks on the execution side, identify which execution stages are slow, or other common issues such as suboptimal parallelism.

Taking multi-table analysis queries as an example, you can analyze the Profile to check if the planned Join order is reasonable, if Runtime Filters are effective, and if the parallelism meets expectations. Furthermore, the Profile can provide feedback on machine load, such as slow I/O or unexpected network transmission performance. When confirming and diagnosing such issues, system-level tools are needed to assist in diagnosis and tuning.

For detailed tuning examples, please refer to the documentation on Execution Tuning.

> **When analyzing specific performance issues, it is recommended to first check the plan and then tune the execution. Start by using the Explain tool to confirm the execution plan, and then use the Profile tool to locate and tune execution performance. Reversing the order may lead to inefficiencies and hinder the rapid identification of performance issues.**

Summary

Query tuning is a systematic process, and Doris provides users with tools across various dimensions to facilitate the diagnosis, identification, analysis, and resolution of performance issues at different levels. By familiarizing themselves with these diagnostic and analysis tools and adopting reasonable tuning methods, business personnel and DBAs can quickly and effectively address performance bottlenecks, better unleash Doris's powerful performance advantages, and better adapt to business scenarios for business enablement.

Materialized View Overview

Materialized views are entities that contain both computation logic and data. Unlike views, which only contain computation logic and do not store data themselves, materialized views do.

Use Cases for Materialized Views

Materialized views calculate and store data based on SQL definitions and update periodically or in real-time according to policies. They can be queried directly or used for transparent query rewriting. They are applicable in the following scenarios:

Query Acceleration

In decision support systems, such as BI reports and ad-hoc queries, these analytical queries often involve aggregation operations and may include multi-table joins. Since calculating the results of such queries can be resource-intensive and response times may reach minutes, while business scenarios often require second-level responses, materialized views can be constructed to accelerate common queries.

Lightweight ETL (Data Modeling)

In data layering scenarios, nested materialized views can be used to construct DWD and DWM layers, leveraging the scheduling and refresh capabilities of materialized views.

Lakehouse Integration

For multiple external data sources, materialized views can be constructed for the tables used by these data sources to save costs from importing external tables to internal tables and accelerate the query process.

Classification of Materialized Views

Classified by Data Timeliness: Synchronous vs Asynchronous

- Synchronous materialized views need to maintain strong consistency with the base table data.
- Asynchronous materialized views maintain eventual consistency with the base table data and may have some delay. They are typically used in scenarios where data timeliness is not critical, often using T+1 or hourly data to construct materialized views. If high timeliness is required, consider using synchronous materialized views.

Currently, synchronous materialized views do not support direct queries, while asynchronous materialized views do.

Classified by SQL Mode Supporting Transparent Rewriting: Single Table vs Multi-Table

The SQL definition of a materialized view can include single-table queries or multi-table queries. From the perspective of the number of tables used, materialized views can be classified as single-table or multi-table

materialized views.

- For asynchronous materialized views, both single-table and multi-table can be used.
- For synchronous materialized views, only single-table can be used.

Classified by Materialized View Refresh: Full Refresh vs Partition Incremental Refresh vs Real-Time Refresh

For Asynchronous Materialized Views

- Full Refresh: Computes all data of the materialized view's SQL definition.
- Partition Incremental Refresh: When the partition data of the base table of the materialized view changes, it identifies the partitions of the materialized view that correspond to the changes and only refreshes those partitions, achieving partition incremental refresh without refreshing the entire materialized view.

For Synchronous Materialized Views

- Can be understood as real-time refresh, maintaining consistency with the base table data.

Sync-Materialized View

What is a Synchronous Materialized View

A synchronous materialized view is a special type of table in Doris that stores pre-computed data sets based on defined SELECT statements. Doris automatically maintains the data in synchronous materialized views, ensuring that any new imports or deletions in the base table are reflected in the materialized view in real-time, maintaining data consistency without requiring any additional manual maintenance. When querying, Doris automatically selects the optimal materialized view and retrieves data directly from it.

Applicable Scenarios

- Accelerating time-consuming aggregation operations
- Queries requiring prefix index matching
- Reducing the amount of data scanned by pre-filtering
- Speeding up queries by pre-computing complex expressions

Limitations

- Synchronous materialized views only support SELECT statements for a single table, including WHERE, GROUP BY, and ORDER BY clauses, but not JOIN, HAVING, LIMIT clauses, LATERAL VIEW.
- Unlike asynchronous materialized views, synchronous materialized views cannot be queried directly.
- The SELECT list cannot include auto-increment columns, constants, duplicate expressions, or window functions.
- If the SELECT list contains aggregation functions, these must be root expressions (e.g., `sum(a + 1)` is supported, but `sum(a) + 1` is not), and no non-aggregation function expressions can follow the aggregation function (e.g., `SELECT x, sum(a)` is allowed, but `SELECT sum(a), x` is not).
- If the condition column for a DELETE statement exists in the materialized view, the DELETE operation cannot proceed. If data deletion is necessary, the materialized view must be dropped first.
- Excessive materialized views on a single table can impact import efficiency. When importing data, both the materialized views and the base table are updated synchronously. Excessive materialized views on a table can slow down imports, similar to importing data into multiple tables simultaneously.
- Materialized views on Unique Key data models can only reorder columns and do not support aggregation. Therefore, coarse-grained aggregation operations cannot be performed through materialized views on Unique Key models.

Using Materialized Views

Doris provides a comprehensive set of DDL syntax for materialized views, including creation, viewing, and deletion. Below is an example demonstrating how to use materialized views to accelerate aggregation calculations. Suppose a user has a sales record detail table that stores transaction IDs, salespersons, stores, sale dates, and amounts. The table creation and data insertion statements are as follows:

```
-- Create a test_db
create database test_db;
use test_db;

-- Create table
create table sales_records
(
    record_id int,
    seller_id int,
    store_id int,
    sale_date date,
    sale_amt bigint
)
distributed by hash(record_id)
properties("replication_num" = "1");

-- Insert data
insert into sales_records values(1,1,1,"2020-02-02",1), (1,1,1,"2020-02-02",2);
```

Creating a Materialized View

If users frequently need to analyze sales volumes by different stores, they can create a materialized view for the `sales_records` table, grouped by store ID and summing sales amounts for each store. The creation statement is as follows:

```
create materialized view store_amt as
select store_id, sum(sale_amt) from sales_records group by store_id;
```

Checking if the Materialized View is Created

Since creating a materialized view is an asynchronous operation, users need to check the status of the materialized view creation task asynchronously after submitting it. The command is as follows:

```
show alter table materialized view from test_db;
```

The output will show all materialized view creation tasks for that database. A sample output is:

```
+--------+---------------+---------------------+---------------------+---------------+-----------------+----------+---------------+-
---------+------+----------+---------+
| JobId  | TableName     | CreateTime          | FinishTime          | BaseIndexName | RollupIndexName | RollupId | TransactionId |
State    | Msg  | Progress | Timeout |
+--------+---------------+---------------------+---------------------+---------------+-----------------+----------+---------------+-
---------+------+----------+---------+
| 494349 | sales_records | 2020-07-30 20:04:56 | 2020-07-30 20:04:57 | sales_records | store_amt       | 494350   | 133107        |
FINISHED |      | NULL     | 2592000 |
```

```
+--------+--------------+--------------------+--------------------+--------------+----------------+----------+--------------+-
---------+------+----------+---------+
```

The `State` column indicates the status. When the state changes to `FINISHED`, the materialized view is successfully created.

Canceling Materialized View Creation

If the background asynchronous task for creating the materialized view has not yet completed, it can be canceled with the following command:

```
cancel alter table materialized view from test_db.sales_records;
```

If the materialized view has already been created, it cannot be canceled, but it can be deleted using the DROP command.

Viewing the Materialized View Structure

The structure of all materialized views created on a target table can be viewed using the following command:

```
desc sales_records all;
```

Viewing the Creation Statement of a Materialized View

The creation statement for a materialized view can be viewed with the following command:

```
show create materialized view store_amt on sales_records;
```

Querying Materialized Views

Once a materialized view is created, when users query sales volumes for different stores, Doris will directly read the aggregated data from the newly created materialized view `store_amt`, thereby enhancing query efficiency. Users still specify the `sales_records` table in their queries, for example:

```
SELECT store_id, SUM(sale_amt) FROM sales_records GROUP BY store_id;
```

The above query will automatically match the `store_amt` materialized view. Users can use the following command to verify whether the current query has matched an appropriate materialized view.

```
EXPLAIN SELECT store_id, SUM(sale_amt) FROM sales_records GROUP BY store_id;
```

The result is as follows:

```
+------------------------------------------------------------------------+
| Explain String (Nereids Planner)                                       |
+------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                        |
```

```
|   OUTPUT EXPRS:                                                                |
|     store_id[#11]                                                              |
|     sum(sale_amt)[#12]                                                         |
|   PARTITION: HASH_PARTITIONED: mv_store_id[#7]                                 |
|                                                                                |
|   HAS_COLO_PLAN_NODE: true                                                     |
|                                                                                |
|   VRESULT SINK                                                                 |
|      MYSQL_PROTOCAL                                                            |
|                                                                                |
|   3:VAGGREGATE (merge finalize)(384)                                           |
|   |  output: sum(partial_sum(mva_SUM__`sale_amt`)[#8])[#10]                    |
|   |  group by: mv_store_id[#7]                                                 |
|   |  sortByGroupKey: false                                                     |
|   |  cardinality = 1                                                           |
|   |  final projections: mv_store_id[#9], sum(mva_SUM__`sale_amt`)[#10]         |
|   |  final project output tuple id: 4                                          |
|   |  distribute expr lists: mv_store_id[#7]                                    |
|   |                                                                            |
|   2:VEXCHANGE                                                                  |
|      offset: 0                                                                 |
|      distribute expr lists:                                                    |
|                                                                                |
| PLAN FRAGMENT 1                                                                |
|                                                                                |
|   PARTITION: HASH_PARTITIONED: record_id[#2]                                   |
|                                                                                |
|   HAS_COLO_PLAN_NODE: false                                                    |
|                                                                                |
|   STREAM DATA SINK                                                             |
|     EXCHANGE ID: 02                                                            |
|     HASH_PARTITIONED: mv_store_id[#7]                                          |
|                                                                                |
|   1:VAGGREGATE (update serialize)(374)                                         |
|   |  STREAMING                                                                 |
|   |  output: partial_sum(mva_SUM__`sale_amt`[#1])[#8]                          |
|   |  group by: mv_store_id[#0]                                                 |
|   |  sortByGroupKey: false                                                     |
|   |  cardinality = 1                                                           |
|   |  distribute expr lists:                                                    |
|   |                                                                            |
|   0:VOlapScanNode(369)                                                         |
|      TABLE: test_db.sales_records(store_amt), PREAGGREGATION: ON               |
|      partitions = 1/1 (sales_records)                                          |
|      tablets = 10/10, tabletList = 266568, 266570, 266572 ...                  |
|      cardinality = 1, avgRowSize = 1805.0, numNodes = 1                        |
|      pushAggOp = NONE                                                          |
|                                                                                |
|                                                                                |
| ========== MATERIALIZATIONS ==========                                         |
|                                                                                |
| MaterializedView                                                               |
| MaterializedViewRewriteSuccessAndChose:                                        |
|   internal.test_db.sales_records.store_amt chose,                              |
|                                                                                |
| MaterializedViewRewriteSuccessButNotChose:                                     |
|   not chose: none,                                                             |
|                                                                                |
| MaterializedViewRewriteFail:                                                   |
|                                                                                |
|                                                                                |
| ========== STATISTICS ==========                                               |
| planned with unknown column statistics                                         |
+--------------------------------------------------------------------------------+
```

`MaterializedViewRewriteSuccessAndChose` displays the materialized view that was successfully matched, as shown in the following example:

```
+------------------------------------------------------------------------+
| MaterializedViewRewriteSuccessAndChose:                                |
|   internal.test_db.sales_records.store_amt chose,                      |
+------------------------------------------------------------------------+
```

The above content indicates that the query successfully matched the materialized view named `store_amt`. It's worth noting that if there is no data in the target table, the materialized view may not be hit.

Detailed explanations on MATERIALIZATIONS:

- **MaterializedViewRewriteSuccessAndChose**: Displays the materialized view that was successfully selected and used for query optimization.
- **MaterializedViewRewriteSuccessButNotChose**: Displays materialized views that matched the query but were not selected (the optimizer chooses the optimal materialized view based on its cost, and these matched but unselected views indicate they were not the optimal choice).
- **MaterializedViewRewriteFail**: Displays materialized views that failed to match the query, meaning the original SQL query could not match any existing materialized views and therefore could not be optimized using them.

Dropping a Materialized View

```
drop materialized view store_amt on sales_records;
```

Usage Examples

Below are additional examples demonstrating the use of materialized views.

Example 1: Accelerating Aggregation Queries

Business Scenario: Calculating ad UV (Unique Visitors) and PV (Page Views).

1. Assuming the raw ad click data is stored in Doris, creating a materialized view with `bitmap_union` can speed up queries for ad PV and UV. First, create a table to store ad click details:

   ```
   create table advertiser_view_record
   (
       click_time datetime,
       advertiser varchar(10),
       channel varchar(10),
       user_id int
   ) distributed by hash(user_id) properties("replication_num" = "1");
   insert into advertiser_view_record values("2020-02-02 02:02:02",'a','a',1), ("2020-02-02 02:02:02",'a','a',2);
   ```

2. Since users want to query the UV value of advertisements, which requires an exact deduplication of users for the same advertisement, the typical query would be:

```
select
    advertiser,
    channel,
    count(distinct user_id)
from
    advertiser_view_record
group by
    advertiser, channel;
```

3. For this UV calculation scenario, we can create a materialized view with `bitmap_union` to achieve pre-exact deduplication. In Doris, the result of the `count(distinct)` aggregation is identical to the result of the `bitmap_union_count` aggregation. And `bitmap_union_count` is equivalent to counting the results of `bitmap_union`. Therefore, if the query involves `count(distinct)`, creating a materialized view with `bitmap_union` aggregation can speed up the query. Based on current usage scenarios, a materialized view can be created to group by advertisement and channel, with exact deduplication for `user_id`.

```
create materialized view advertiser_uv as
select
    advertiser,
    channel,
    bitmap_union(to_bitmap(user_id))
from
    advertiser_view_record
group by
    advertiser, channel;
```

4. Once the materialized view table is created, when querying the UV for advertisements, Doris will automatically retrieve data from the newly created materialized view `advertiser_uv`. If the previous SQL is executed:

```
select
    advertiser,
    channel,
    count(distinct user_id)
from
    advertiser_view_record
group by
    advertiser, channel;
```

5. After selecting the materialized view, the actual query will be transformed into:

```
select
    advertiser,
    channel,
    bitmap_union_count(to_bitmap(user_id))
from
    advertiser_uv
group by
    advertiser, channel;
```

6. Use the `explain` command to check if the query matches the materialized view:

```
explain select
    advertiser,
    channel,
```

```
    count(distinct user_id)
from
    advertiser_view_record
group by
    advertiser, channel;
```

7. The output will be:

```
+------------------------------------------------------------------------------------------------------------------------------
------------------------------+
| Explain String(Nereids Planner)
|
+------------------------------------------------------------------------------------------------------------------------------
------------------------------+
| PLAN FRAGMENT 0
|
|   OUTPUT EXPRS:
|
|     advertiser[#13]
|
|     channel[#14]
|
|     count(DISTINCT user_id)[#15]
|
|   PARTITION: HASH_PARTITIONED: mv_advertiser[#7], mv_channel[#8]
|
|
|
|   HAS_COLO_PLAN_NODE: true
|
|
|
|   VRESULT SINK
|
|      MYSQL_PROTOCAL
|
|
|
|   3:VAGGREGATE (merge finalize)(440)
|
|   |  output: bitmap_union_count(partial_bitmap_union_count(mva_BITMAP_UNION__to_bitmap_with_check(CAST(`user_id` AS
bigint)))[#9])[#12]                 |
|   |  group by: mv_advertiser[#7], mv_channel[#8]
|
|   |  sortByGroupKey:false
|
|   |  cardinality=1
|
|   |  final projections: mv_advertiser[#10], mv_channel[#11],
bitmap_union_count(mva_BITMAP_UNION__to_bitmap_with_check(CAST(`user_id` AS bigint)))[#12] |
|   |  final project output tuple id: 4
|
|   |  distribute expr lists: mv_advertiser[#7], mv_channel[#8]
|
|   |
|
|   2:VEXCHANGE
|
|      offset: 0
|
|      distribute expr lists:
|
|
```

```
|
| PLAN FRAGMENT 1
|
|
|
|   PARTITION: HASH_PARTITIONED: user_id[#6]
|
|
|
|   HAS_COLO_PLAN_NODE: false
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 02
|
|     HASH_PARTITIONED: mv_advertiser[#7], mv_channel[#8]
|
|
|
|   1:VAGGREGATE (update serialize)(430)
|
|   |  STREAMING
|
|   |  output: partial_bitmap_union_count(mva_BITMAP_UNION__to_bitmap_with_check(CAST(`user_id` AS bigint))[#2])[#9]
|
|   |  group by: mv_advertiser[#0], mv_channel[#1]
|
|   |  sortByGroupKey:false
|
|   |  cardinality=1
|
|   |  distribute expr lists:
|
|   |
|
|   0:VOlapScanNode(425)
|
|      TABLE: test_db.advertiser_view_record(advertiser_uv), PREAGGREGATION: ON
|
|      partitions=1/1 (advertiser_view_record)
|
|      tablets=10/10, tabletList=266637,266639,266641 ...
|
|      cardinality=1, avgRowSize=0.0, numNodes=1
|
|      pushAggOp=NONE
|
|
|
|
|
| ========== MATERIALIZATIONS ==========
|
|
|
| MaterializedView
|
| MaterializedViewRewriteSuccessAndChose:
|
|   internal.test_db.advertiser_view_record.advertiser_uv chose,
|
|
|
```

```
| MaterializedViewRewriteSuccessButNotChose:
|
|   not chose: none,
|
|
|
| MaterializedViewRewriteFail:
|
|
|
|
|
| ========== STATISTICS ==========
|
| planed with unknown column statistics
|
+-----------------------------------------------------------------------------------------------------------------------------
------------------------------+
```

8. In the result of the explain command, you can see that `internal.test_db.advertiser_view_record.advertiser_uv` was chosen. This indicates that the query will directly scan the data from the materialized view. This confirms that the match was successful. Secondly, the count(distinct) operation on the `user_id` field is rewritten as `bitmap_union_count(to_bitmap)`. This means that the exact deduplication effect is achieved through the use of Bitmap.

Example 2: Matching Different Prefix Indexes

Business Scenario: Matching prefix indexes.

1. If a table has prefix indexes on k1 and k2, but queries sometimes involve k3, a materialized view can be created with k3 as the first column to leverage indexing:

```
create table test_table
(
    k1 int,
    k2 int,
    k3 int,
    kx date
)
distributed by hash(k1)
properties("replication_num" = "1");

insert into test_table values(1,1,1,1),(3,3,3,3);
```

2. Create a materialized view with k3 as the prefix index:

```
create materialized view mv_1 as SELECT k3, k2, k1 FROM test_table;
```

3. Queries with `WHERE k3 = 3` will match the materialized view, as verified by `explain`.

```
explain select k1, k2, k3 from test_table where k3=3;
```

4. The output will be:

```
+------------------------------------------------------------+
| Explain String(Nereids Planner)                            |
+------------------------------------------------------------+
| PLAN FRAGMENT 0                                            |
|   OUTPUT EXPRS:                                            |
|     k1[#7]                                                 |
|     k2[#8]                                                 |
|     k3[#9]                                                 |
|   PARTITION: HASH_PARTITIONED: mv_k1[#2]                   |
|                                                            |
|   HAS_COLO_PLAN_NODE: false                                |
|                                                            |
|   VRESULT SINK                                             |
|      MYSQL_PROTOCAL                                        |
|                                                            |
|   0:VOlapScanNode(256)                                     |
|      TABLE: test_db.test_table(mv_1), PREAGGREGATION: ON   |
|      PREDICATES: (mv_k3[#0] = 3)                           |
|      partitions=1/1 (test_table)                           |
|      tablets=10/10, tabletList=271177,271179,271181 ...    |
|      cardinality=1, avgRowSize=0.0, numNodes=1             |
|      pushAggOp=NONE                                        |
|      final projections: mv_k1[#2], mv_k2[#1], mv_k3[#0]    |
|      final project output tuple id: 2                      |
|                                                            |
|                                                            |
| ========== MATERIALIZATIONS ==========                     |
|                                                            |
| MaterializedView                                           |
| MaterializedViewRewriteSuccessAndChose:                    |
|   internal.test_db.test_table.mv_1 chose,                  |
|                                                            |
| MaterializedViewRewriteSuccessButNotChose:                 |
|   not chose: none,                                         |
|                                                            |
| MaterializedViewRewriteFail:                               |
|                                                            |
|                                                            |
| ========== STATISTICS ==========                           |
| planed with unknown column statistics                      |
+------------------------------------------------------------+
```

5. In the result of the explain command, you can see that `internal.test_db.test_table.mv_1` was chosen, indicating that the query hit the materialized view.

Example 3: Pre-filtering and Expression Computation to Accelerate Queries

Business Scenario: Pre-filtering data or accelerating expression computation.

1. Create a table and materialized views for pre-filtering and expression computation:

```
create table d_table (
   k1 int null,
   k2 int not null,
   k3 bigint null,
   k4 date null
)
duplicate key (k1,k2,k3)
distributed BY hash(k1) buckets 3
properties("replication_num" = "1");
```

```
insert into d_table select 1,1,1,'2020-02-20';
insert into d_table select 2,2,2,'2021-02-20';
insert into d_table select 3,-3,null,'2022-02-20';
```

2. Creating Some Materialized Views:

```
-- mv1 Perform expression calculations ahead of time
create materialized view mv1 as
select
    abs(k1)+k2+1,
    sum(abs(k2+2)+k3+3)
from
    d_table
group by
    abs(k1)+k2+1;

-- mv2 Use where expressions to filter in advance to reduce the amount of data in materialized views
create materialized view mv2 as
select
    year(k4),
    month(k4)
from
    d_table
where
    year(k4) = 2020;
```

3. Testing Whether the Materialized Views Are Successfully Hit with Some Queries:

```
-- Hit mv1
select
    abs(k1)+k2+1,
    sum(abs(k2+2)+k3+3)
from
    d_table
group by
    abs(k1)+k2+1;

-- Hit mv1
select
    bin(abs(k1)+k2+1),
    sum(abs(k2+2)+k3+3)
from
    d_table
group by
    bin(abs(k1)+k2+1);

-- Hit mv2
select
    year(k4) + month(k4)
from
    d_table
where
    year(k4) = 2020;

-- Hit table d_table but not hit mv2, because where condition does not match
select
    year(k4),
    month(k4)
from
    d_table;
```

FAQ

1. Why isn’t the rewrite successful after creating a materialized view?

   If no matching data is found, it might be because the materialized view is still in the building process. In this case, you can use the following command to check the build status of the materialized view:

   ```
   show alter table materialized view from test_db;
   ```

   If the query result shows that the `status` field is not `FINISHED`, you need to wait until the status becomes `FINISHED` before the materialized view becomes available.

2. When upgrading from 2.x to 3.0.0, why aren’t the previous synchronous materialized views being hit?

   Starting from version 3.0.0, transparent rewriting of synchronous materialized views uses plan structure information by default. If you find that materialized views that previously worked in 2.x are not being hit in 3.0.0, you can disable the following switch (which is enabled by default):

   ``sqlSET enable_sync_mv_cost_based_rewrite = true;`

Overview of Asynchronous Materialized Views

Materialized views, as an efficient solution, combine the flexibility of views with the high performance advantages of physical tables. They can pre-compute and store the result sets of queries, allowing for quick retrieval of results directly from the stored materialized view when query requests arrive, thus avoiding the overhead of re-executing complex query statements.

Use Cases

- **Query Acceleration and Concurrency Improvement**: Materialized views can significantly enhance query speed while boosting the system's concurrent processing capabilities, effectively reducing resource consumption.
- **Simplifying ETL Processes**: During the Extract, Transform, Load (ETL) process, materialized views can streamline workflows, improve development efficiency, and make data processing smoother.
- **Accelerating External Table Queries in Lakehouse Architecture**: In a lakehouse architecture, materialized views can significantly enhance the query speed for external data sources, improving data access efficiency.
- **Improving Write Efficiency**: By reducing resource contention, materialized views can optimize the data writing process, enhance write efficiency, and ensure data consistency and integrity.

Limitations

- **Consistency of Asynchronous Materialized Views with Base Table Data**: Asynchronous materialized views will eventually be consistent with the base table data, but they cannot be synchronized in real-time, meaning real-time consistency cannot be maintained.
- **Support for Window Function Queries**: Currently, if a query contains window functions, it is not supported to transparently rewrite that query to utilize materialized views.
- **Materialized Views Joining More Tables than Query Tables**: If the number of tables joined in the materialized view exceeds the number of tables involved in the query (for example, if the query only involves t1 and t2, while the materialized view includes t1, t2, and an additional t3), the system currently does not support transparently rewriting that query to utilize the materialized view.
- If the materialized view contains set operations such as UNION ALL, LIMIT, ORDER BY, or CROSS JOIN, the materialized view can be built normally, but it cannot be used for transparent rewriting.

Principle Introduction

Materialized views, as an advanced feature in databases, essentially function as MTMV-type internal tables. When creating a materialized view, the system simultaneously registers a refresh task. This task will run when needed, executing an INSERT OVERWRITE statement to write the latest data into the materialized view.

Refresh Mechanism

Unlike the real-time incremental refresh used by synchronous materialized views, asynchronous materialized views offer more flexible refresh options.

- **Full Refresh**:
  In this mode, the system recalculates all data involved in the SQL definition of the materialized view and writes the complete results into the materialized view. This process ensures that the data in the materialized view remains consistent with the base table data, but it may consume more computational resources and time.

- **Partition Incremental Refresh**:
  When the partition data of the base table for the materialized view changes, the system can intelligently identify these changes and refresh only the affected partitions. This mechanism significantly reduces the computational resources and time required to refresh the materialized view while ensuring eventual data consistency.

Transparent Rewriting:
Transparent rewriting is an important means for databases to optimize query performance. When processing user queries, the system can automatically optimize and rewrite the SQL to improve execution efficiency and reduce computational costs. This rewriting process is transparent to the user, requiring no intervention.

Doris asynchronous materialized views utilize a transparent rewriting algorithm based on the SPJG (SELECT-PROJECT-JOIN-GROUP-BY) model. This algorithm can deeply analyze the structural information of SQL, automatically searching for and selecting suitable materialized views for transparent rewriting. When multiple materialized views are available, the algorithm will also choose the optimal materialized view to respond to the query SQL based on certain strategies (such as cost models), further enhancing query performance.

Support for Materialized Refresh Data Lake

The support for materialized refresh data lakes varies by table type and catalog.

Table Type	Catalog Type	Refresh Method		Triggered Refresh
		Full Refresh	Partition Refresh	Auto Trigger
Internal Table	Internal	Supported in 2.1	Supported in 2.1	Supported in 2.1.4
Hive	Hive	Supported in 2.1	Supported in 2.1	Not supported
Iceberg	Iceberg	Supported in 2.1	Not supported	Not supported
Paimon	Paimon	Supported in 2.1	Not supported	Not supported
Hudi	Hudi	Supported in 2.1	Not supported	Not supported
JDBC	JDBC	Supported in 2.1	Not supported	Not supported
ES	ES	Supported in 2.1	Not supported	Not supported

Transparent Rewriting Support for Data Lake

Currently, the transparent rewriting feature of asynchronous materialized views supports the following types of tables and catalogs.

Real-time Base Table Data Awareness: Refers to the materialized view's ability to detect changes in the underlying table data it uses and utilize the latest data during queries.

Table Type	Catalog Type	Transparent Rewriting Support	Real-time Base Table Data Awareness
Internal Table	Internal	Supported	Supported
Hive	Hive	Supported	3.1 Supported
Iceberg	Iceberg	Supported	3.1 Supported
Paimon	Paimon	Supported	3.1 Supported
Hudi	Hudi	Supported	3.1 Supported
JDBC	JDBC	Supported	Not Supported
ES	ES	Supported	Not Supported

Materialized views using external tables do not participate in transparent rewriting by default, because they cannot detect changes in external table data and cannot guarantee the data in the materialized view is up-to-date. If you want to enable transparent rewriting for materialized views containing external tables, you can set `SET materialized_view_rewrite_enable_contain_external_table = true`.

Since version 2.1.11, Doris has optimized the transparent rewriting performance for external tables, mainly improving the performance of obtaining available materialized views containing external tables.

For partitioned materialized views containing external tables, if transparent rewriting is slow, you need to configure in fe.conf: `max_hive_partition_cache_num = 20000`, the maximum number of Hive Metastore table-level partition caches, with a default value of 10000. If the external Hive table has many partitions, you can set this value higher.

`external_cache_expire_time_minutes_after_access`, the duration after last access when cache expires. Default is 10 minutes, can be appropriately increased. (Applies to external table schema cache and Hive metadata cache)

`external_cache_refresh_time_minutes = 60`, the automatic refresh interval for external table metadata cache. Default is 10 minutes, can be appropriately increased. This configuration is supported starting from version 3.1. For details about external table metadata cache configuration, see Metadata Cache

Relationship Between Materialized Views and OLAP Internal Tables

Asynchronous materialized views define SQL using the base table's table model without restrictions, which can be detail models, primary key models (merge-on-write and merge-on-read), aggregate models, etc.

The underlying implementation of materialized views relies on OLAP tables of the Duplicate model, which theoretically allows them to support all core functionalities of the Duplicate model. However, to ensure that materialized views can execute data refresh tasks stably and efficiently, we have imposed a series of necessary restrictions on their functionality. The specific restrictions are as follows:

- The partitions of materialized views are automatically created and maintained based on their base tables, so users cannot perform partition operations on materialized views.
- Since there are related jobs (JOB) that need to be processed behind materialized views, commands like DELETE TABLE or RENAME TABLE cannot be used to operate on materialized views. Instead, the commands specific to the materialized view must be used for these operations.

- The column data types of materialized views are automatically inferred based on the query statement specified at creation, so these data types cannot be modified. Otherwise, it may lead to failures in the refresh tasks of the materialized view.
- Materialized views have some properties that Duplicate tables do not possess, and these properties need to be modified through the commands of the materialized view. Other common properties should be modified using the ALTER TABLE command.

More References

For creating, querying, and maintaining asynchronous materialized views, you can refer to Creating, Querying, and Maintaining Asynchronous Materialized Views.

For best practices, you can refer to Best Practices.

For frequently asked questions, you can refer to Frequently Asked Questions.

Creating, Querying, and Maintaining Asynchronous Materialized Views

This document provides detailed information about materialized view creation, direct querying of materialized views, query rewriting, and common maintenance operations.

Creating Materialized Views

Permission Requirements

- Creating Materialized Views: Requires both materialized view creation permission (same as table creation permission) and query permission for the materialized view creation statement (same as SELECT permission).

Creation Syntax

```
CREATE MATERIALIZED VIEW
[ IF NOT EXISTS ] <materialized_view_name>
[ (<columns_definition>) ]
[ BUILD <build_mode> ]
[ REFRESH <refresh_method> [refresh_trigger]]
[ [DUPLICATE] KEY (<key_cols>) ]
[ COMMENT '<table_comment>' ]
[ PARTITION BY (
{ <partition_col>
| DATE_TRUNC(<partition_col>, <partition_unit>) }
)]
[ DISTRIBUTED BY { HASH (<distribute_cols>) | RANDOM }
[ BUCKETS { <bucket_count> | AUTO } ]
]
[ PROPERTIES (
-- Table property
<table_property>
-- Additional table properties
[ , ... ])
]
AS <query>
```

Refresh Configuration

build_mode Refresh Timing

Determines whether to refresh immediately after materialized view creation. - IMMEDIATE: Refresh immediately (default mode) - DEFERRED: Delayed refresh

refresh_method Refresh Method

- COMPLETE: Refresh all partitions

- AUTO: Attempt incremental refresh, only refreshing partitions with data changes since the last materialization. Falls back to full refresh of all partitions if data changes cannot be detected.

refresh_trigger Trigger Methods

- `ON MANUAL` **Manual Trigger**

  Users can trigger materialized view refreshes using SQL statements with the following strategies:

  Check for base table partition data changes since last refresh and refresh only changed partitions:

  ```
  REFRESH MATERIALIZED VIEW mvName AUTO;
  ```

  If the base table used in the SQL definition of the materialized view is a JDBC table, Doris cannot perceive changes in the table data. When refreshing the materialized view, it is necessary to specify COMPLETE. If AUTO is specified, it may result in the base table having data, but the materialized view being empty after the refresh. Currently, when refreshing the materialized view, Doris can only perceive data changes in internal tables and Hive data source tables; support for other data sources is being gradually implemented.

  Refresh all materialized view partitions without checking for base table changes:

  ```
  REFRESH MATERIALIZED VIEW mvName COMPLETE;
  ```

  Refresh only specified partitions:

  ```
  REFRESH MATERIALIZED VIEW mvName partitions(partitionName1,partitionName2);
  ```

  :::tip `partitionName` can be obtained using `SHOW PARTITIONS FROM mvName`. Starting from version 2.1.3, Hive supports detecting base table partition changes since last refresh. Other external tables don't support this yet. Internal tables have always supported this feature. :::

- `ON SCHEDULE` **Scheduled Trigger**

  Specify refresh intervals in the materialized view creation statement. You can specify the data refresh interval in the materialized view creation statement using refreshUnit, where the refresh time interval unit can be minute, hour, day, week, etc.

  Example of full refresh (`REFRESH COMPLETE`) every 10 hours, refreshing all partitions:

```
CREATE MATERIALIZED VIEW mv_6
REFRESH COMPLETE ON SCHEDULE EVERY 10 hour
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT FROM lineitem;
```

Example of incremental refresh (`REFRESH AUTO`) every 10 hours, only refreshing changed partitions or falling back to full refresh if needed (automatic Hive partition calculation supported from version 2.1.3):

```
CREATE MATERIALIZED VIEW mv_7
REFRESH AUTO ON SCHEDULE EVERY 10 hour
PARTITION by(l_shipdate)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT FROM lineitem;
```

- `ON COMMIT` **Automatic Trigger**

  :::tip This feature is available from Apache Doris version 2.1.4 onwards. :::

  Automatically triggers materialized view refresh when base table data changes, with refresh partition scope matching “scheduled trigger”.

  Example: When partition `t1` data changes in base table `lineitem`, it automatically triggers corresponding materialized view partition refresh:

```
CREATE MATERIALIZED VIEW mv_8
REFRESH AUTO ON COMMIT
PARTITION by(l_shipdate)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT FROM lineitem;
```

> **Not recommended for frequently changing base tables as it creates frequent materialized refresh tasks, consuming excessive resources.**

```
For more details, see [REFRESH MATERIALIZED VIEW](../../../sql-manual/sql-statements/table-and-view/async-materialized-view/REFRESH-MATERIALIZED-VIEW)
```

Examples

Table Creation Statements

```
CREATE TABLE IF NOT EXISTS lineitem (
    l_orderkey    integer not null,
    l_partkey     integer not null,
    l_suppkey     integer not null,
    l_linenumber  integer not null,
    l_quantity    decimalv3(15,2) not null,
    l_extendedprice  decimalv3(15,2) not null,
    l_discount    decimalv3(15,2) not null,
    l_tax         decimalv3(15,2) not null,
    l_returnflag  char(1) not null,
    l_linestatus  char(1) not null,
    l_shipdate    date not null,
    l_commitdate  date not null,
    l_receiptdate date not null,
    l_shipinstruct char(25) not null,
    l_shipmode     char(10) not null,
    l_comment      varchar(44) not null
    )
    DUPLICATE KEY(l_orderkey, l_partkey, l_suppkey, l_linenumber)
```

```
    PARTITION BY RANGE(l_shipdate)
    (FROM ('2023-10-17') TO ('2023-11-01') INTERVAL 1 DAY)
    DISTRIBUTED BY HASH(l_orderkey) BUCKETS 3;

INSERT INTO lineitem VALUES
(1, 2, 3, 4, 5.5, 6.5, 7.5, 8.5, 'o', 'k', '2023-10-17', '2023-10-17', '2023-10-17', 'a', 'b', 'yyyyyyyyy'),
(2, 4, 3, 4, 5.5, 6.5, 7.5, 8.5, 'o', 'k', '2023-10-18', '2023-10-18', '2023-10-18', 'a', 'b', 'yyyyyyyyy'),
(3, 2, 4, 4, 5.5, 6.5, 7.5, 8.5, 'o', 'k', '2023-10-19', '2023-10-19', '2023-10-19', 'a', 'b', 'yyyyyyyyy');

CREATE TABLE IF NOT EXISTS orders  (
    o_orderkey       integer not null,
    o_custkey        integer not null,
    o_orderstatus    char(1) not null,
    o_totalprice     decimalv3(15,2) not null,
    o_orderdate      date not null,
    o_orderpriority  char(15) not null,
    o_clerk          char(15) not null,
    o_shippriority   integer not null,
    o_comment        varchar(79) not null
    )
DUPLICATE KEY(o_orderkey, o_custkey)
PARTITION BY RANGE(o_orderdate)(
FROM ('2023-10-17') TO ('2023-11-01') INTERVAL 1 DAY)
DISTRIBUTED BY HASH(o_orderkey) BUCKETS 3;

INSERT INTO orders VALUES
(1, 1, 'o', 9.5, '2023-10-17', 'a', 'b', 1, 'yy'),
(1, 1, 'o', 10.5, '2023-10-18', 'a', 'b', 1, 'yy'),
(2, 1, 'o', 11.5, '2023-10-19', 'a', 'b', 1, 'yy'),
(3, 1, 'o', 12.5, '2023-10-19', 'a', 'b', 1, 'yy');

CREATE TABLE IF NOT EXISTS partsupp (
      ps_partkey     INTEGER NOT NULL,
      ps_suppkey     INTEGER NOT NULL,
      ps_availqty    INTEGER NOT NULL,
      ps_supplycost  DECIMALV3(15,2)  NOT NULL,
      ps_comment     VARCHAR(199) NOT NULL
    )
DUPLICATE KEY(ps_partkey, ps_suppkey)
DISTRIBUTED BY HASH(ps_partkey) BUCKETS 3;

INSERT INTO partsupp VALUES
(2, 3, 9, 10.01, 'supply1'),
(4, 3, 10, 11.01, 'supply2'),
(2, 3, 10, 11.01, 'supply3');
```

Refresh mechanism example 1

In the following example, the refresh timing is set to `BUILD IMMEDIATE` (refresh immediately after creation), the refresh method is set to `REFRESH AUTO` (attempt incremental refresh), which only refreshes partitions that have changed since the last materialization. If incremental refresh is not possible, it will perform a full refresh of all partitions. The trigger method is set to `ON MANUAL`. For non-partitioned full materialized views that have only one partition, if the base table data changes, a full refresh will be required.

```
CREATE MATERIALIZED VIEW mv_1_0
BUILD IMMEDIATE
REFRESH AUTO
ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
AS
```

```
SELECT
  l_linestatus,
  to_date(o_orderdate) as date_alias,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

Refresh mechanism example 2

In the following example, the refresh timing is set to delayed refresh (`BUILD DEFERRED`), the refresh method is set to full refresh (`REFRESH COMPLETE`), and the trigger timing is set to scheduled refresh (`ON SCHEDULE`). The first refresh time is `2024-12-01 20:30:00`, and it will refresh every day thereafter. If `BUILD DEFERRED` is specified as `BUILD IMMEDIATE`, the materialized view will refresh immediately upon creation. After that, it will refresh every day starting from `2024-12-01 20:30:00`.

> **The time specified in STARTS must be later than the current time.**

```
CREATE MATERIALIZED VIEW mv_1_1
BUILD DEFERRED
REFRESH COMPLETE
ON SCHEDULE EVERY 1 DAY STARTS '2024-12-01 20:30:00'
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
l_linestatus,
to_date(o_orderdate) as date_alias,
o_shippriority
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

Refresh mechanism example 3

In this example, the refresh timing is set to immediate refresh upon creation (`BUILD IMMEDIATE`), the refresh method is set to full refresh (`REFRESH COMPLETE`), and the trigger method is set to trigger refresh (`ON COMMIT`). When data in the `orders` or `lineitem` tables changes, it will automatically trigger the refresh of the materialized view.

```
CREATE MATERIALIZED VIEW mv_1_1
BUILD IMMEDIATE
REFRESH COMPLETE
ON COMMIT
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
l_linestatus,
to_date(o_orderdate) as date_alias,
o_shippriority
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

Partition Configuration

In the following example, when creating a partitioned materialized view, it is necessary to specify `PARTITION BY`. For expressions referencing partition fields, only the `date_trunc` function and identifiers are allowed. The following statement meets the requirements: the partition field references only the `date_trunc` function. The refresh method for partitioned materialized views is generally set to `AUTO`, which attempts incremental refresh, refreshing only the partitions that have changed since the last materialized refresh. If incremental refresh is not possible, it will refresh all partitions.

```
CREATE MATERIALIZED VIEW mv_2_0
BUILD IMMEDIATE
REFRESH AUTO
ON MANUAL
PARTITION BY (order_date_month)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
  l_linestatus,
  date_trunc(o_orderdate, 'month') as order_date_month,
  o_shippriority
FROM
  orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

The following statement will fail to create a partitioned materialized view because the partition field `order_date_month` uses the `date_add()` function, resulting in the error `because column to check use invalid implicit expression, invalid expression is date_add(o_orderdate#4, 2)`.

```
CREATE MATERIALIZED VIEW mv_2_1 BUILD IMMEDIATE REFRESH AUTO ON MANUAL
PARTITION BY (order_date_month)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
  l_linestatus,
  date_trunc(date_add(o_orderdate, INTERVAL 2 DAY), 'month') as order_date_month,
  o_shippriority
FROM
  orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

Base Table with Multiple Partition Columns

Currently, only Hive external tables support multiple partition columns. Hive external tables often have many multi-level partitions, such as a first-level partition by date and a second-level partition by region. Materialized views can choose one of Hive's partition columns as the partition column for the materialized view.

For example, the Hive table creation statement is as follows:

```
CREATE TABLE hive1 (
`k1` int)
PARTITIONED BY (
`year` int,
`region` string)
STORED AS ORC;
```

```
alter table hive1 add if not exists
partition(year=2020,region="bj")
partition(year=2020,region="sh")
partition(year=2021,region="bj")
partition(year=2021,region="sh")
partition(year=2022,region="bj")
partition(year=2022,region="sh")
```

When the materialized view creation statement is as follows, the materialized view `mv_hive` will have three partitions: `('2020')`, `('2021')`, and `('2022')`.

```
CREATE MATERIALIZED VIEW mv_hive
BUILD DEFERRED
REFRESH AUTO
ON MANUAL
PARTITION BY (year)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT k1, year, region FROM hive1;
```

When the materialized view creation statement is as follows, the materialized view `mv_hive2` will have the following two partitions: `('bj')` and `('sh')`:

```
CREATE MATERIALIZED VIEW mv_hive2
BUILD DEFERRED
REFRESH AUTO
ON MANUAL
PARTITION BY (region)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT k1, year, region FROM hive1;
```

Using Partial Partitions from the Base Table

Some base tables have many partitions, but the materialized view only focuses on the “hot” data from a recent period. This feature allows for that.

The base table creation statement is as follows:

```
CREATE TABLE t1 (
k1 INT,
k2 DATE NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(k1)
COMMENT 'OLAP'
PARTITION BY range(k2)
(
PARTITION p26 VALUES [("2024-03-26"),("2024-03-27")),
PARTITION p27 VALUES [("2024-03-27"),("2024-03-28")),
PARTITION p28 VALUES [("2024-03-28"),("2024-03-29"))
)
DISTRIBUTED BY HASH(k1) BUCKETS 2;
```

The materialized view creation statement is as follows, indicating that the materialized view only focuses on the data from the most recent day. If the current time is `2024-03-28 xx:xx:xx`, the materialized view will only have

one partition `[("2024-03-28"),("2024-03-29")]`:

```
CREATE MATERIALIZED VIEW mv1
BUILD DEFERRED
REFRESH AUTO
ON MANUAL
PARTITION BY (k2)
DISTRIBUTED BY RANDOM BUCKETS 2
PROPERTIES (
'partition_sync_limit'='1',
'partition_sync_time_unit'='DAY'
)
AS
SELECT FROM t1;
```

If the time passes another day, and the current time is `2024-03-29 xx:xx:xx`, `t1` will add a new partition `[("2024-03-29"),("2024-03-30")]`. If the materialized view is refreshed at this time, after the refresh is complete, the materialized view will only have one partition `[("2024-03-29"),("2024-03-30")]`.

Additionally, when the partition field is of string type, the materialized view property `partition_date_format` can be set, for example, `%Y-%m-%d`.

Partition Aggregation

> **Range partitioning is supported since Doris 2.1.5**

When the data in the base table is aggregated, the amount of data in each partition may significantly decrease. In this case, a partition aggregation strategy can be adopted to reduce the number of partitions in the materialized view.

Assuming the base table creation statement is as follows:

```
CREATE TABLE t1 (
k1 LARGEINT NOT NULL,
k2 DATE NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(k1)
COMMENT 'OLAP'
PARTITION BY range(k2)
(
PARTITION p_20200101 VALUES [("2020-01-01"),("2020-01-02")),
PARTITION p_20200102 VALUES [("2020-01-02"),("2020-01-03")),
PARTITION p_20200201 VALUES [("2020-02-01"),("2020-02-02"))
)
DISTRIBUTED BY HASH(k1) BUCKETS 2;
```

If the materialized view creation statement is as follows, the materialized view will contain two partitions: `[("2020-01-01","2020-02-01")]` and `[("2020-02-01","2020-03-01")]`.

```
CREATE MATERIALIZED VIEW mv_3
BUILD DEFERRED
REFRESH AUTO
ON MANUAL
```

```
PARTITION BY (date_trunc(k2,'month'))
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT FROM t1;
```

If the materialized view creation statement is as follows, the materialized view will only contain one partition: `[("2020-01-01","2021-01-01")]`.

```
CREATE MATERIALIZED VIEW mv_4
BUILD DEFERRED
REFRESH AUTO
ON MANUAL
PARTITION BY (date_trunc(k2,'year'))
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT FROM t1;
```

Additionally, if the partition field is of string type, the date format can be specified by setting the materialized view's `partition_date_format` property, for example, `'%Y-%m-%d'`.

For more details, refer to CREATE ASYNC MATERIALIZED VIEW.

SQL Definition

Asynchronous materialized views do not support creation using View.

Direct Querying of Materialized Views

Materialized views can be treated like tables, allowing for the addition of filtering conditions and aggregations for direct querying.

Definition of Materialized View:

```
CREATE MATERIALIZED VIEW mv_5
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 hour
DISTRIBUTED BY RANDOM BUCKETS 3
AS
SELECT t1.l_linenumber,
o_custkey,
o_orderdate
FROM (SELECT FROM lineitem WHERE l_linenumber > 1) t1
LEFT OUTER JOIN orders
ON l_orderkey = o_orderkey;
```

Original Query:

```
SELECT t1.l_linenumber,
o_custkey,
o_orderdate
FROM (SELECT FROM lineitem WHERE l_linenumber > 1) t1
LEFT OUTER JOIN orders
ON l_orderkey = o_orderkey
WHERE o_orderdate = '2023-10-18';
```

Equivalent Direct Query on Materialized View: Users need to manually modify the query.

```
SELECT l_linenumber,
o_custkey,
o_orderdate
FROM mv_5
WHERE o_orderdate = '2023-10-18';
```

Transparent Query Rewriting

Transparent rewriting means that when processing queries, users do not need to manually modify queries, as the system will automatically optimize and rewrite them. Doris asynchronous materialized views use a transparent rewriting algorithm based on the SPJG (SELECT-PROJECT-JOIN-GROUP-BY) pattern. This algorithm can analyze SQL structure information, automatically find suitable materialized views for transparent rewriting, and select the optimal materialized view to respond to query SQL. Doris provides rich and comprehensive transparent rewriting capabilities. For example, the following capabilities:

Condition Compensation

Query and materialized view conditions do not need to be exactly the same. By compensating conditions on materialized views to express queries, materialized views can be reused to the maximum extent, avoiding the need to repeatedly build materialized views.

When the `where` conditions in the materialized view and query are expressions connected by `and`:

1. **When the query's expressions contain the materialized view's expressions:**

   Condition compensation can be performed.

   For example, if the query condition is `a > 5 and b > 10 and c = 7`, and the materialized view condition is `a > 5 and b > 10`, the materialized view condition is a subset of the query condition, so only the `c = 7` condition needs to be compensated.

2. **When the query's expressions do not completely contain the materialized view's expressions:**

   When the query conditions can be derived from the materialized view conditions (common for comparison and range expressions like `>`, `<`, `=`, `in`, etc.), condition compensation can also be performed. The compensation result is the query condition itself.

   For example, if the query condition is `a > 5 and b = 10`, and the materialized view condition is `a > 1 and b > 8`, it can be seen that the materialized view condition contains the query condition, and the query condition can be derived from the materialized view condition, so compensation can be performed, with the compensation result being `a > 5 and b = 10`.

   Condition compensation usage restrictions:

3. For expressions connected by `or`, condition compensation cannot be performed; they must be exactly the same for successful rewriting.

4. For non-comparison and non-range expressions like `like`, condition compensation cannot be performed; they must be exactly the same for successful rewriting.

For example:

Materialized View Definition:

```sql
CREATE MATERIALIZED VIEW mv1
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 hour
DISTRIBUTED BY RANDOM BUCKETS 3
AS
SELECT t1.l_linenumber,
       o_custkey,
       o_orderdate
FROM (SELECT * FROM lineitem WHERE l_linenumber > 1) t1
LEFT OUTER JOIN orders
ON l_orderkey = o_orderkey;
```

The following queries can all hit the materialized view. Multiple queries can reuse one materialized view through transparent rewriting, reducing query rewriting time and saving materialized view construction costs.

```sql
SELECT l_linenumber,
       o_custkey,
       o_orderdate
FROM lineitem
LEFT OUTER JOIN orders
ON l_orderkey = o_orderkey
WHERE l_linenumber > 2;
```

```sql
SELECT l_linenumber,
       o_custkey,
       o_orderdate
FROM lineitem
LEFT OUTER JOIN orders
ON l_orderkey = o_orderkey
WHERE l_linenumber > 2 and o_orderdate = '2023-10-19';

```

JOIN Rewriting

JOIN rewriting refers to when the query and materialized view use the same tables, and conditions can be written in the materialized view, JOIN inputs, or outside the JOIN. The optimizer will attempt transparent rewriting for queries in this pattern.

Multiple table JOINs are supported, with the following supported JOIN types:

- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- LEFT SEMI JOIN
- RIGHT SEMI JOIN
- LEFT ANTI JOIN

- RIGHT ANTI JOIN

For example:

Materialized View Definition:

```
CREATE MATERIALIZED VIEW mv2
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 hour
DISTRIBUTED BY RANDOM BUCKETS 3
AS
SELECT t1.l_linenumber,
       o_custkey,
       o_orderkey,
       o_orderstatus,
       l_partkey,
       l_suppkey,
       l_orderkey
FROM (SELECT * FROM lineitem WHERE l_linenumber > 1) t1
INNER JOIN orders ON t1.l_orderkey = orders.o_orderkey;
```

The following query can be transparently rewritten. The condition `l_linenumber > 1` can be lifted up, enabling transparent rewriting to use the materialized view's pre-computed results to express the query. After hitting the materialized view, JOIN computation can be saved.

Query Statement:

```
SELECT l_linenumber,
       o_custkey
FROM lineitem
INNER JOIN orders ON l_orderkey = o_orderkey
WHERE l_linenumber > 1 and o_orderdate = '2023-10-18';
```

JOIN Derivation

When the JOIN types in the query and materialized view are inconsistent, if the materialized view can provide all the data needed by the query, transparent rewriting can still be performed by compensating predicates outside the JOIN.

For example:

Materialized View Definition:

```
CREATE MATERIALIZED VIEW mv3
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 hour
DISTRIBUTED BY RANDOM BUCKETS 3
AS
SELECT
    l_shipdate, l_suppkey, o_orderdate,
    sum(o_totalprice) AS sum_total,
    max(o_totalprice) AS max_total,
    min(o_totalprice) AS min_total,
    count(*) AS count_all,
    count(distinct CASE WHEN o_shippriority > 1 AND o_orderkey IN (1, 3) THEN o_custkey ELSE null END) AS bitmap_union_basic
FROM lineitem
LEFT OUTER JOIN orders ON lineitem.l_orderkey = orders.o_orderkey AND l_shipdate = o_orderdate
GROUP BY
l_shipdate,
```

```
l_suppkey,
o_orderdate;
```

Query Statement:

```
SELECT
    l_shipdate, l_suppkey, o_orderdate,
    sum(o_totalprice) AS sum_total,
    max(o_totalprice) AS max_total,
    min(o_totalprice) AS min_total,
    count(*) AS count_all,
    count(distinct CASE WHEN o_shippriority > 1 AND o_orderkey IN (1, 3) THEN o_custkey ELSE null END) AS bitmap_union_basic
FROM lineitem
INNER JOIN orders ON lineitem.l_orderkey = orders.o_orderkey AND l_shipdate = o_orderdate
WHERE o_orderdate = '2023-10-18' AND l_suppkey = 3
GROUP BY
l_shipdate,
l_suppkey,
o_orderdate;
```

Aggregate Rewriting

When the group dimensions in the query and materialized view definition are consistent, if the materialized view uses the same group by dimensions as the query, and the aggregate functions used in the query can be expressed using the materialized view's aggregate functions, transparent rewriting can be performed.

For example:

Materialized View Definition:

```
CREATE MATERIALIZED VIEW mv4
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 hour
DISTRIBUTED BY RANDOM BUCKETS 3
AS
SELECT
    o_shippriority, o_comment,
    count(distinct CASE WHEN o_shippriority > 1 AND o_orderkey IN (1, 3) THEN o_custkey ELSE null END) AS cnt_1,
    count(distinct CASE WHEN O_SHIPPRIORITY > 2 AND o_orderkey IN (2) THEN o_custkey ELSE null END) AS cnt_2,
    sum(o_totalprice),
    max(o_totalprice),
    min(o_totalprice),
    count(*)
FROM orders
GROUP BY
o_shippriority,
o_comment;
```

The following query can hit the materialized view, as it uses the same aggregation dimensions as the materialized view. The query can filter results using the materialized view's `o_shippriority` field. The query's group by dimensions and aggregate functions can be rewritten using the materialized view's group by dimensions and aggregate functions. After hitting the aggregate materialized view, aggregation computation can be reduced.

Query Statement:

```
SELECT
    o_shippriority, o_comment,
    count(distinct CASE WHEN o_shippriority > 1 AND o_orderkey IN (1, 3) THEN o_custkey ELSE null END) AS cnt_1,
    count(distinct CASE WHEN O_SHIPPRIORITY > 2 AND o_orderkey IN (2) THEN o_custkey ELSE null END) AS cnt_2,
    sum(o_totalprice),
    max(o_totalprice),
    min(o_totalprice),
    count(*)
FROM orders
WHERE o_shippriority in (1, 2)
GROUP BY
o_shippriority,
o_comment;
```

Aggregate Rewriting (Roll-up)

Even when the aggregation dimensions in the query and materialized view definition are inconsistent, rewriting can still be performed. The materialized view's `group by` dimensions need to include the query's `group by` dimensions, and the query may not have any `group by`. Additionally, the aggregate functions used in the query must be expressible using the materialized view's aggregate functions.

For example:

Materialized View Definition:

```
CREATE MATERIALIZED VIEW mv5
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 hour
DISTRIBUTED BY RANDOM BUCKETS 3
AS
SELECT
    l_shipdate, o_orderdate, l_partkey, l_suppkey,
    sum(o_totalprice) AS sum_total,
    max(o_totalprice) AS max_total,
    min(o_totalprice) AS min_total,
    count(*) AS count_all,
    bitmap_union(to_bitmap(CASE WHEN o_shippriority > 1 AND o_orderkey IN (1, 3) THEN o_custkey ELSE null END)) AS
bitmap_union_basic
FROM lineitem
LEFT OUTER JOIN orders ON lineitem.l_orderkey = orders.o_orderkey AND l_shipdate = o_orderdate
GROUP BY
l_shipdate,
o_orderdate,
l_partkey,
l_suppkey;
```

The following query can be transparently rewritten. The query and materialized view use different aggregation dimensions, but the materialized view's dimensions include the query's dimensions. The query can use fields from the dimensions to filter results. The query will attempt to roll up using the functions after the materialized view's `SELECT`, for example, the materialized view's `bitmap_union` will eventually roll up to `bitmap_union_count`, which maintains the same semantics as the query's `count(distinct)`.

Through aggregate roll-up, the same materialized view can be reused by multiple queries, saving materialized view construction costs.

Query Statement:

```
SELECT
    l_shipdate, l_suppkey,
    sum(o_totalprice) AS sum_total,
    max(o_totalprice) AS max_total,
    min(o_totalprice) AS min_total,
    count(*) AS count_all,
    count(distinct CASE WHEN o_shippriority > 1 AND o_orderkey IN (1, 3) THEN o_custkey ELSE null END) AS bitmap_union_basic
FROM lineitem
LEFT OUTER JOIN orders ON lineitem.l_orderkey = orders.o_orderkey AND l_shipdate = o_orderdate
WHERE o_orderdate = '2023-10-18' AND l_partkey = 3
GROUP BY
l_shipdate,
l_suppkey;
```

Currently supported aggregate roll-up functions are listed below:

Query Function	Materialized View Function	Function After Roll-up
max	max	max
min	min	min
sum	sum	sum
count	count	sum
count(distinct)	bitmap_union	bitmap_union_count
bitmap_union	bitmap_union	bitmap_union
bitmap_union_count	bitmap_union	bitmap_union_count
hll_union_agg, approx_count_distinct, hll_cardinality	hll_union or hll_raw_agg	hll_union_agg
any_value	any_value or column used after any_value in select	any_value

Multi-dimensional Aggregate Rewriting

Multi-dimensional aggregate transparent rewriting is supported, meaning that if the materialized view does not use `GROUPING SETS`, `CUBE`, or `ROLLUP`, but the query has multi-dimensional aggregation, and the materialized view's `group by` fields include all fields in the query's multi-dimensional aggregation, transparent rewriting can still be performed.

For example:

Materialized View Definition:

```
CREATE MATERIALIZED VIEW mv5_1
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 hour
DISTRIBUTED BY RANDOM BUCKETS 3
AS
select o_orderstatus, o_orderdate, o_orderpriority,
       sum(o_totalprice) as sum_total,
       max(o_totalprice) as max_total,
       min(o_totalprice) as min_total,
       count(*) as count_all
from orders
```

```
group by
o_orderstatus, o_orderdate, o_orderpriority;
```

The following query can hit the materialized view, reusing the materialized view's aggregate results and saving computation:

Query Statement:

```
select o_orderstatus, o_orderdate, o_orderpriority,
       sum(o_totalprice),
       max(o_totalprice),
       min(o_totalprice),
       count(*)
from orders
group by
GROUPING SETS ((o_orderstatus, o_orderdate), (o_orderpriority), (o_orderstatus), ());
```

Partition Compensation Rewriting

When a partitioned materialized view cannot provide all the data needed by the query, a `union all` approach can be used, combining data from the original table and the materialized view as the final result.

For example:

Materialized View Definition:

```
CREATE MATERIALIZED VIEW mv7
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
partition by(l_shipdate)
DISTRIBUTED BY RANDOM BUCKETS 2
as
select l_shipdate, o_orderdate, l_partkey,
       l_suppkey, sum(o_totalprice) as sum_total
from lineitem
left join orders on lineitem.l_orderkey = orders.o_orderkey and l_shipdate = o_orderdate
group by
    l_shipdate,
    o_orderdate,
    l_partkey,
    l_suppkey;
```

When the base table adds a partition `2023-10-21` and the materialized view hasn't been refreshed yet, results can be returned by using `union all` between the materialized view and the original table.

```
insert into lineitem values
(1, 2, 3, 4, 5.5, 6.5, 7.5, 8.5, 'o', 'k', '2023-10-21', '2023-10-21', '2023-10-21', 'a', 'b', 'yyyyyyyyy');
```

Query Statement:

```
select l_shipdate, o_orderdate, l_partkey, l_suppkey, sum(o_totalprice) as sum_total
from lineitem
left join orders on lineitem.l_orderkey = orders.o_orderkey and l_shipdate = o_orderdate
group by
    l_shipdate,
    o_orderdate,
```

```
    l_partkey,
    l_suppkey;
```

The query can partially use the materialized view's pre-computed results, saving this portion of computation.

Rewrite Result Illustration:

```
SELECT *
FROM mv7
union all
select t1.l_shipdate, o_orderdate, t1.l_partkey, t1.l_suppkey, sum(o_totalprice) as sum_total
from (select * from lineitem where l_shipdate = '2023-10-21') t1
left join orders on t1.l_orderkey = orders.o_orderkey and t1.l_shipdate = o_orderdate
group by
    t1.l_shipdate,
    o_orderdate,
    t1.l_partkey,
    t1.l_suppkey;
```

Nested Materialized View Rewriting

The SQL definition of a materialized view can use another materialized view; this is called a nested materialized view. There is theoretically no limit to the nesting depth, and this materialized view can be both directly queried and transparently rewritten. Nested materialized views can also participate in transparent rewriting.

For example:

Create inner materialized view `mv8_0_inner_mv`:

```
CREATE MATERIALIZED VIEW mv8_0_inner_mv
BUILD IMMEDIATE REFRESH COMPLETE ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
AS
select
l_linenumber,
o_custkey,
o_orderkey,
o_orderstatus,
l_partkey,
l_suppkey,
l_orderkey
from lineitem
inner join orders on lineitem.l_orderkey = orders.o_orderkey;
```

Create outer materialized view `mv8_0`:

```
CREATE MATERIALIZED VIEW mv8_0
BUILD IMMEDIATE REFRESH COMPLETE ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
AS
select
l_linenumber,
o_custkey,
o_orderkey,
o_orderstatus,
l_partkey,
```

```
l_suppkey,
l_orderkey,
ps_availqty
from mv8_0_inner_mv
inner join partsupp on l_partkey = ps_partkey AND l_suppkey = ps_suppkey;
```

For the following query, both `mv8_0_inner_mv` and `mv8_0` will be successfully rewritten, and the cost model will ultimately choose `mv8_0`.

Nested materialized views are commonly used in data modeling and particularly complex queries. If a single materialized view cannot be transparently rewritten, you can split the complex query and build nested materialized views. The transparent rewriting process will attempt to use nested materialized views for rewriting. If the rewrite is successful, it will save computation and improve query performance.

```
select lineitem.l_linenumber
from lineitem
inner join orders on l_orderkey = o_orderkey
inner join partsupp on  l_partkey = ps_partkey AND l_suppkey = ps_suppkey
where o_orderstatus = 'o'
```

Note:

1. The more layers of nested materialized views, the longer transparent rewriting will take. It is recommended that nested materialized views do not exceed 3 layers.
2. Nested materialized view transparent rewriting is disabled by default. See the related settings below for how to enable it.

Aggregate Query Using Non-Aggregate Materialized View Rewrite

If the query is an aggregate query and the materialized view does not contain aggregates, but the materialized view can provide all the columns used in the query, then it can also be rewritten. For example, if the query first performs a join and then a group by aggregation, hitting a materialized view that includes the join will also yield benefits.

```
CREATE MATERIALIZED VIEW mv10_0
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
as
select l_shipdate, o_orderdate, l_partkey,
       l_suppkey, o_totalprice
from lineitem
left join orders on lineitem.l_orderkey = orders.o_orderkey and l_shipdate = o_orderdate;
```

The following query can hit the mv10_0 materialized view, saving the computation of the lineitem join orders join:

```
select l_shipdate, o_orderdate, l_partkey,
       l_suppkey, sum(o_totalprice) as sum_total
from lineitem
left join orders on lineitem.l_orderkey = orders.o_orderkey and l_shipdate = o_orderdate
group by
    l_shipdate,
    o_orderdate,
```

```
    l_partkey,
    l_suppkey;
```

Explain Query Transparent Rewriting Status

To view materialized view transparent rewriting hits, used for viewing and debugging.

1. **To view materialized view transparent rewriting hit status, this statement will show brief process information about query transparent rewriting.**

   ```
   explain <query_sql>
   ```

   The returned information is as follows, with materialized view-related information excerpted here:

   ```
   | MaterializedView
   |
   | MaterializedViewRewriteSuccessAndChose:
   |
   |   Names: mv5
   |
   | MaterializedViewRewriteSuccessButNotChose:
   |
   |
   |
   | MaterializedViewRewriteFail:
   |
   |   Name: mv4
   |
   |   FailSummary: Match mode is invalid, View struct info is invalid
   |
   |   Name: mv3
   |
   |   FailSummary: Match mode is invalid, Rewrite compensate predicate by view fail, View struct info is invalid
   |
   |   Name: mv1
   |
   |   FailSummary: The columns used by query are not in view, View struct info is invalid
   |
   |   Name: mv2
   |
   |   FailSummary: The columns used by query are not in view, View struct info is invalid
   ```

- MaterializedViewRewriteSuccessAndChose: Indicates the list of materialized view names that were successfully transparently rewritten and chosen by CBO (Cost-Based Optimizer).
- MaterializedViewRewriteSuccessButNotChose: Indicates the list of materialized view names that were successfully transparently rewritten but ultimately not chosen by CBO.
- MaterializedViewRewriteFail: Lists the failed cases and summary reasons.

2. **To understand the detailed process information about materialized view candidacy, rewriting, and final selection, execute the following statement:**

   ```
   explain memo plan <query_sql>
   ```

Maintaining Materialized Views

Permission Requirements

- Dropping materialized views: Requires materialized view deletion permission (same as table deletion permission)
- Modifying materialized views: Requires materialized view modification permission (same as table modification permission)
- Pausing/resuming/canceling/refreshing materialized views: Requires materialized view creation permission

Modifying Materialized Views

Modifying Materialized View Properties

```
ALTER MATERIALIZED VIEW mv_1
SET(
  "grace_period" = "10"
);
```

For more details, see ALTER ASYNC MATERIALIZED VIEW

Materialized View Renaming, i.e., Atomic Replacement of Materialized Views

```
CREATE MATERIALIZED VIEW mv9_0
BUILD IMMEDIATE REFRESH COMPLETE ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
PROPERTIES ('replication_num' = '1')
AS
select
  l_linenumber,
  o_custkey,
  o_orderkey,
  o_orderstatus,
  l_partkey,
  l_suppkey,
  l_orderkey
from lineitem
inner join orders on lineitem.l_orderkey = orders.o_orderkey;
```

Replace the materialized view mv7 with mv9_0 and delete mv7:

```
ALTER MATERIALIZED VIEW mv7
REPLACE WITH MATERIALIZED VIEW mv9_0
PROPERTIES('swap' = 'false');
```

Dropping Materialized Views

```
DROP MATERIALIZED VIEW mv_1;
```

For more details, see DROP ASYNC MATERIALIZED VIEW

Viewing Materialized View Creation Statement

```
SHOW CREATE MATERIALIZED VIEW mv_1;
```

For more details, see SHOW CREATE MATERIALIZED VIEW

Pausing Materialized Views

For more details, see PAUSE MATERIALIZED VIEW

Resuming Materialized Views

For more details, see RESUME MATERIALIZED VIEW

Canceling Materialized View Refresh Tasks

For more details, see CANCEL MATERIALIZED VIEW TASK

Querying Materialized View Information

```
SELECT *
FROM mv_infos('database'='db_name')
WHERE Name = 'mv_name' \G
```

Example output:

```
*************************** 1. row ***************************
               Id: 139570
             Name: mv11
          JobName: inner_mtmv_139570
            State: NORMAL
SchemaChangeDetail:
      RefreshState: SUCCESS
       RefreshInfo: BUILD IMMEDIATE REFRESH AUTO ON MANUAL
          QuerySql: SELECT l_shipdate, l_orderkey, O_ORDERDATE, count(*)
FROM lineitem
LEFT OUTER JOIN orders on l_orderkey = o_orderkey
GROUP BY l_shipdate, l_orderkey, O_ORDERDATE
           EnvInfo: EnvInfo{ctlId='0', dbId='16813'}
      MvProperties: {}
   MvPartitionInfo: MTMVPartitionInfo{partitionType=FOLLOW_BASE_TABLE, relatedTable=lineitem, relatedCol='l_shipdate',
partitionCol='l_shipdate'}
SyncWithBaseTables: 1
```

- **SyncWithBaseTables:** Indicates whether the materialized view is synchronized with base tables.
    - For fully built materialized views, a value of 1 indicates the view is available for transparent rewriting.
    - For incrementally partitioned materialized views, availability is determined at the partition level. Even if some partitions are unavailable, the view can still be used for transparent rewriting if the

queried partitions are valid. The ability to use transparent rewriting depends on the `SyncWithBaseTables` value of the queried partitions - 1 means available, 0 means unavailable.

- **JobName:** Name of the materialized view's build job. Each materialized view has one Job, and each refresh creates a new Task, with a 1:n relationship between Jobs and Tasks.
- **State:** If changed to SCHEMA_CHANGE, indicates the base table's schema has changed. The materialized view cannot be used for transparent rewriting (but can still be queried directly). Will return to NORMAL after the next successful refresh task.
- **SchemaChangeDetail:** Explains the reason for SCHEMA_CHANGE.
- **RefreshState:** Status of the last refresh task. If FAIL, indicates execution failed - use the `tasks()` command to identify the cause. See Viewing Materialized View Task Status section.
- **SyncWithBaseTables:** Whether synchronized with base tables. 1 means synchronized, 0 means not synchronized. If not synchronized, use `show partitions` to check which partitions are out of sync. See the section below on checking SyncWithBaseTables status for partitioned materialized views.

For transparent rewriting, materialized views typically have two states:

- **Normal:** The materialized view is available for transparent rewriting.
- **Unavailable/Abnormal:** The materialized view cannot be used for transparent rewriting. However, it can still be queried directly.

For more details, see MV_INFOS

Querying Refresh Task Information

Each materialized view has one Job, and each refresh creates a new Task, with a 1:n relationship between Jobs and Tasks. To view a materialized view's Task status by name, run the following query to check refresh task status and progress:

```
SELECT *
FROM tasks("type"="mv")
WHERE
  MvDatabaseName = 'mv_db_name' and
  mvName = 'mv_name'
ORDER BY  CreateTime DESC \G
```

Example output:

```
*************************** 1. row ***************************
               TaskId: 167019363907545
                JobId: 139872
              JobName: inner_mtmv_139570
                MvId: 139570
              MvName: mv11
        MvDatabaseId: 16813
      MvDatabaseName: regression_test_nereids_rules_p0_mv
              Status: SUCCESS
            ErrorMsg:
          CreateTime: 2024-06-21 10:31:43
           StartTime: 2024-06-21 10:31:43
```

```
           FinishTime: 2024-06-21 10:31:45
           DurationMs: 2466
          TaskContext: {"triggerMode":"SYSTEM","isComplete":false}
          RefreshMode: COMPLETE
NeedRefreshPartitions:
["p_20231023_20231024","p_20231019_20231020","p_20231020_20231021","p_20231027_20231028","p_20231030_20231031","p_20231018_20231019"
,"p_20231024_20231025","p_20231021_20231022","p_20231029_20231030","p_20231028_20231029","p_20231025_20231026","p_20231022_20231023"
,"p_20231031_20231101","p_20231016_20231017","p_20231026_20231027"]
  CompletedPartitions:
["p_20231023_20231024","p_20231019_20231020","p_20231020_20231021","p_20231027_20231028","p_20231030_20231031","p_20231018_20231019"
,"p_20231024_20231025","p_20231021_20231022","p_20231029_20231030","p_20231028_20231029","p_20231025_20231026","p_20231022_20231023"
,"p_20231031_20231101","p_20231016_20231017","p_20231026_20231027"]
             Progress: 100.00% (15/15)
          LastQueryId: fe700ca3d6504521-bb522fc9ccf615e3
```

- NeedRefreshPartitions and CompletedPartitions record the partitions refreshed in this Task.
- Status: If FAILED, indicates execution failed. Check ErrorMsg for failure reason or use LastQueryId to search Doris logs for detailed error information. Currently, task failure makes existing materialized views unavailable. This will be changed so existing materialized views remain available for transparent rewriting even if tasks fail.
- ErrorMsg: Failure reason.
- RefreshMode: COMPLETE means all partitions were refreshed, PARTIAL means some partitions were refreshed, NOT_REFRESH means no partitions needed refreshing.

Note

- Currently, the default storage and display count for tasks is 100. This can be modified by configuring max_persistence_task_count in the fe.conf file. When exceeding this limit, older task records will be discarded. If the value is set to < 1, task persistence will be disabled. After modifying the configuration, a restart of the FE service is required for the changes to take effect.
- If the `grace_period` property was set when creating the materialized view, it may still be available for transparent rewriting in some cases even if `SyncWithBaseTables` is false or 0.
- `grace_period` is measured in seconds and specifies the allowed time for data inconsistency between the materialized view and base tables.
- If set to 0, requires exact consistency between materialized view and base table data for transparent rewriting.
- If set to 10, allows up to 10 seconds of delay between materialized view and base table data. The materialized view can be used for transparent rewriting during this 10-second window. :::

For more details, see TASKS

Querying Materialized View Jobs

```
SELECT *
FROM jobs("type"="mv")
WHERE Name="inner_mtmv_75043";
```

For more details, see JOBS

Querying Materialized View Partition Information

Checking SyncWithBaseTables Status for Partitioned Materialized Views

Run `show partitions from mv_name` to check if queried partitions are valid. Example output:

```
show partitions from mv11;
+-------------+--------------------+----------------+---------------------+--------+--------------+-------------------------
-------------------------------------------------------+----------------+---------+---------------+--------------+--------
-------------+--------------------+-------------------------+-----------+------------+-------------------------+-----------
+--------------------+--------------+
| PartitionId | PartitionName       | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy |
LastConsistencyCheckTime | DataSize  | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+--------------------+----------------+---------------------+--------+--------------+-------------------------
-------------------------------------------------------+----------------+---------+---------------+--------------+--------
-------------+--------------------+-------------------------+-----------+------------+-------------------------+-----------
+--------------------+--------------+
| 140189      | p_20231016_20231017 | 1              | 2024-06-21 10:31:45 | NORMAL | l_shipdate   | [types: [DATEV2]; keys:
[2023-10-16]; ..types: [DATEV2]; keys: [2023-10-17]; ) | l_orderkey      | 10      | 1              | HDD           | 9999-
12-31 23:59:59 |                     | NULL                    | 0.000     | false      | tag.location.default: 1 | true
| true               | []           |
| 139995      | p_20231018_20231019 | 2              | 2024-06-21 10:31:44 | NORMAL | l_shipdate   | [types: [DATEV2]; keys:
[2023-10-18]; ..types: [DATEV2]; keys: [2023-10-19]; ) | l_orderkey      | 10      | 1              | HDD           | 9999-
12-31 23:59:59 |                     | NULL                    | 880.000 B | false      | tag.location.default: 1 | true
| true               | []           |
| 139898      | p_20231019_20231020 | 2              | 2024-06-21 10:31:43 | NORMAL | l_shipdate   | [types: [DATEV2]; keys:
[2023-10-19]; ..types: [DATEV2]; keys: [2023-10-20]; ) | l_orderkey      | 10      | 1              | HDD           | 9999-
12-31 23:59:59 |                     | NULL                    | 878.000 B | false      | tag.location.default: 1 | true
| true               | []           |
+-------------+--------------------+----------------+---------------------+--------+--------------+-------------------------
-------------------------------------------------------+----------------+---------+---------------+--------------+--------
-------------+--------------------+-------------------------+-----------+------------+-------------------------+-----------
+--------------------+--------------+
```

Check the `SyncWithBaseTables` field - false indicates the partition is not available for transparent rewriting.

For more details, see SHOW PARTITIONS

Viewing Materialized View Table Structure

For more details, see DESCRIBE

Related Configuration

Session Variables

Variable	Description
SET enable_nereids_planner = true;	Async materialized views only work with the new optimizer. Enable this if materialized view transparent rewriting isn't working
SET enable_materialized_view_rewrite = true;	Enable/disable query transparent rewriting (enabled by default from version 2.1.5)

Variable	Description
SET materialized_view_rewrite_enable_contain_external_table = true;	Allow materialized views containing external tables to participate in transparent rewriting (disabled by default)
SET materialized_view_rewrite_success_candidate_num = 3;	Maximum number of successful rewrite results allowed in CBO candidates (default 3). Reduce if transparent rewriting is slow
SET enable_materialized_view_union_rewrite = true;	Allow UNION ALL between base table and materialized view when partitioned view doesn't provide all needed data (enabled by default)
SET enable_materialized_view_nest_rewrite = true;	Allow nested rewrites (disabled by default). Enable if complex queries require nested materialized views
SET materialized_view_relation_mapping_max_count = 8;	Maximum allowed relation mappings during transparent rewriting (default 8). Reduce if rewriting is slow
SET enable_dml_materialized_view_rewrite = true;	Enable structure-based materialized view transparent rewriting during DML (enabled by default)
SET enable_dml_materialized_view_rewrite_when_base_table_unawareness = true;	Enable structure-based materialized view transparent rewriting during DML when view contains external tables that can't be tracked in real-time (disabled by default)

fe.conf Configuration

- **job_mtmv_task_consumer_thread_num:** Controls the number of concurrent materialized view refresh tasks (default 10). Tasks exceeding this limit will be pending. Requires FE restart to take effect.

Best Practices | Async Materialized View

Best Practices

Principles for Using Asynchronous Materialized Views

- **Timeliness Consideration:** Asynchronous materialized views are typically used in scenarios where data timeliness is not critical, usually T+1 data. If high timeliness is required, consider using synchronous materialized views.
- **Acceleration Effect and Consistency Consideration:** In query acceleration scenarios, when creating materialized views, DBAs should group common query SQL patterns, aiming to minimize overlap between groups. The clearer the SQL pattern grouping, the higher the quality of the materialized view construction. A query may use multiple materialized views, and a materialized view may be used by multiple queries. Constructing materialized views requires comprehensive consideration of response time (acceleration effect), construction cost, and data consistency requirements.
- **Materialized View Definition and Construction Cost Consideration:**
    - The closer the materialized view definition is to the original query, the better the query acceleration effect, but the lower the generality and reusability of the materialization, meaning higher construction costs.
    - The more general the materialized view definition (e.g., without WHERE conditions and more aggregation dimensions), the lower the query acceleration effect, but the better the generality and reusability of the materialization, meaning lower construction costs.

> **Note**
>
> - **Control of Materialized View Quantity:** More materialized views are not necessarily better. Constructing and refreshing materialized views requires resources. Materialized views participate in transparent rewriting, and the CBO cost model needs time to select the optimal materialized view. In theory, the more materialized views, the longer the transparent rewriting time.
> - **Regularly Check the Usage Status of Materialized Views:** If not used, they should be deleted in time.
> - **Base Table Data Update Frequency:** If the base table data of the materialized view is frequently updated, it may not be suitable to use materialized views, as this will cause the materialized view to frequently become invalid and not usable for transparent rewriting (direct query). If you need to use such materialized views for transparent rewriting, you need to allow a certain timeliness delay in the queried data and can set a `grace_period`. See the applicable introduction of `grace_period` for details.
> :::

Principles for Choosing Materialized View Refresh Methods

When the following conditions are met, it is recommended to create partitioned materialized views:

- The base table data volume of the materialized view is large, and the base table is a partitioned table.
- The tables used by the materialized view, except for the partitioned table, do not change frequently.
- The definition SQL of the materialized view and the partition field meet the requirements of partition derivation, that is, meet the requirements of partition incremental update. Detailed requirements can be found in CREATE-ASYNC-MATERIALIZED-VIEW
- The number of partitions in the materialized view is not large, as too many partitions will lead to excessively long partition materialized view construction time.

When some partitions of the materialized view become invalid, transparent rewriting can use the valid partitions of the materialized view UNION ALL base table to return data.

If partitioned materialized views cannot be constructed, you can consider choosing fully refreshed materialized views.

Common Usage of Partitioned Materialized Views

When the materialized view's base table data volume is large and the base table is a partitioned table, if the materialized view's definition SQL and partition fields meet the requirements of partition derivation, this scenario is suitable for building partitioned materialized views. For detailed requirements of partition derivation, refer to CREATE-ASYNC-MATERIALIZED-VIEW and Async Materialized View FAQ Building Question 12.

The materialized view's partitions are created following the base table's partition mapping, generally having a 1:1 or 1:n relationship with the base table's partitions.

- If the base table's partitions undergo data changes, such as adding partitions or deleting partitions, the corresponding partitions in the materialized view will also become invalid. Invalid partitions cannot be used for transparent rewriting but can be directly queried. When transparent rewriting discovers that the materialized view's partition data is invalid, the invalid partitions will be handled by joining with the base table to respond to queries.

  For commands to check materialized view partition status, see viewing materialized view status, mainly using the `show partitions from mv_name` command.

- If non-partitioned tables referenced by the materialized view undergo data changes, it will trigger all partitions of the materialized view to become invalid, preventing the materialized view from being used for transparent rewriting. You need to refresh all partition data of the materialized view using the command `REFRESH MATERIALIZED VIEW mv1 AUTO;`. This command will attempt to refresh all partitions of the materialized view where data has changed.

  Therefore, it's generally recommended to place frequently changing data in partitioned tables referenced by the partitioned materialized view, and place infrequently changing dimension tables in non-referenced partition table positions.

- If non-partitioned tables referenced by the materialized view undergo data changes, and the non-partitioned table data is only being added without modifications, you can specify the attribute `excluded_trigger_tables = 'non_partition_table_name1,non_partition_table_name2'` when creating the materialized view. This way, data changes in non-partitioned tables won't invalidate all partitions of the materialized view, and the next refresh will only refresh the invalid partitions of the materialized view corresponding to the partition table.

Transparent rewriting of partitioned materialized views is at the partition granularity. Even if some partitions of the materialized view become invalid, the materialized view can still be used for transparent rewriting. However, if only one partition is queried and that partition's data in the materialized view is invalid, then the materialized view cannot be used for transparent rewriting.

For example:

```
CREATE TABLE IF NOT EXISTS lineitem (
    l_orderkey INTEGER NOT NULL,
    l_partkey INTEGER NOT NULL,
    l_suppkey INTEGER NOT NULL,
    l_linenumber INTEGER NOT NULL,
    l_ordertime DATETIME NOT NULL,
    l_quantity DECIMALV3(15, 2) NOT NULL,
    l_extendedprice DECIMALV3(15, 2) NOT NULL,
    l_discount DECIMALV3(15, 2) NOT NULL,
    l_tax DECIMALV3(15, 2) NOT NULL,
    l_returnflag CHAR(1) NOT NULL,
    l_linestatus CHAR(1) NOT NULL,
    l_shipdate DATE NOT NULL,
    l_commitdate DATE NOT NULL,
    l_receiptdate DATE NOT NULL,
    l_shipinstruct CHAR(25) NOT NULL,
    l_shipmode CHAR(10) NOT NULL,
    l_comment VARCHAR(44) NOT NULL
  ) DUPLICATE KEY(
    l_orderkey, l_partkey, l_suppkey,
    l_linenumber
  ) PARTITION BY RANGE(l_ordertime) (
    FROM
      ('2024-05-01') TO ('2024-06-30') INTERVAL 1 DAY
  )
DISTRIBUTED BY HASH(l_orderkey) BUCKETS 3;

INSERT INTO lineitem VALUES
(1, 2, 3, 4, '2024-05-01 01:45:05', 5.5, 6.5, 0.1, 8.5, 'o', 'k', '2024-05-01', '2024-05-01', '2024-05-01', 'a', 'b',
'yyyyyyyyy'),
(1, 2, 3, 4, '2024-05-15 02:35:05', 5.5, 6.5, 0.15, 8.5, 'o', 'k', '2024-05-15', '2024-05-15', '2024-05-15', 'a', 'b',
'yyyyyyyyy'),
(2, 2, 3, 5, '2024-05-25 08:30:06', 5.5, 6.5, 0.2, 8.5, 'o', 'k', '2024-05-25', '2024-05-25', '2024-05-25', 'a', 'b',
'yyyyyyyyy'),
(3, 4, 3, 6, '2024-06-02 09:25:07', 5.5, 6.5, 0.3, 8.5, 'o', 'k', '2024-06-02', '2024-06-02', '2024-06-02', 'a', 'b',
'yyyyyyyyy'),
(4, 4, 3, 7, '2024-06-15 13:20:09', 5.5, 6.5, 0, 8.5, 'o', 'k', '2024-06-15', '2024-06-15', '2024-06-15', 'a', 'b',
'yyyyyyyyy'),
(5, 5, 6, 8, '2024-06-25 15:15:36', 5.5, 6.5, 0.12, 8.5, 'o', 'k', '2024-06-25', '2024-06-25', '2024-06-25', 'a', 'b',
'yyyyyyyyy'),
(5, 5, 6, 9, '2024-06-29 21:10:52', 5.5, 6.5, 0.1, 8.5, 'o', 'k', '2024-06-30', '2024-06-30', '2024-06-30', 'a', 'b',
'yyyyyyyyy'),
(5, 6, 5, 10, '2024-06-03 22:05:50', 7.5, 8.5, 0.1, 10.5, 'k', 'o', '2024-06-03', '2024-06-03', '2024-06-03', 'c', 'd',
'xxxxxxxxx');

CREATE TABLE IF NOT EXISTS partsupp (
```

```
    ps_partkey INTEGER NOT NULL,
    ps_suppkey INTEGER NOT NULL,
    ps_availqty INTEGER NOT NULL,
    ps_supplycost DECIMALV3(15, 2) NOT NULL,
    ps_comment VARCHAR(199) NOT NULL
  )
DUPLICATE KEY(ps_partkey, ps_suppkey)
DISTRIBUTED BY HASH(ps_partkey) BUCKETS 3;


INSERT INTO partsupp VALUES
(2, 3, 9, 10.01, 'supply1'),
(4, 3, 9, 10.01, 'supply2'),
(5, 6, 9, 10.01, 'supply3'),
(6, 5, 10, 11.01, 'supply4');
```

In this example, the o_ordertime field in the orders table is the partition field, with type DATETIME, partitioned by day. The main query is based on a “day” granularity:

```
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  ps_partkey
FROM
  lineitem
  LEFT JOIN partsupp ON l_partkey = ps_partkey
  and l_suppkey = ps_suppkey
WHERE
  date_trunc(l_ordertime, 'day') <= DATE '2024-05-25'
  AND date_trunc(l_ordertime, 'day') >= DATE '2024-05-05'
GROUP BY
  l_linestatus,
  ps_partkey;
```

To avoid refreshing too many partitions each time in the materialized view, the partition granularity can be consistent with the base table orders, also partitioning by “day”.

The materialized view’s definition SQL can use “day” granularity and aggregate data by “day”:

```
CREATE MATERIALIZED VIEW rollup_partition_mv
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
partition by(order_date)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  ps_partkey,
  date_trunc(l_ordertime, 'day') as order_date
FROM
  lineitem
  LEFT JOIN partsupp ON l_partkey = ps_partkey
  and l_suppkey = ps_suppkey
GROUP BY
  l_linestatus,
```

```
  ps_partkey,
  date_trunc(l_ordertime, 'day');
```

Creating Partitioned Materialized Views with UNION ALL

Currently, Doris has a limitation where partitioned materialized view definitions cannot contain UNION ALL clauses. To create a materialized view that includes UNION ALL, you can use the following approach: For each input part of the UNION ALL, attempt to create a partitioned materialized view, and then create a regular view for the entire UNION ALL result set.

For example: The materialized view definition below contains a UNION ALL clause, which cannot be directly used to create a partitioned materialized view.

```
SELECT
l_linestatus,
sum(
l_extendedprice * (1 - l_discount)
) AS revenue,
ps_partkey,
date_trunc(l_ordertime, 'day') as order_date
FROM
lineitem
LEFT JOIN partsupp ON l_partkey = ps_partkey
and l_suppkey = ps_suppkey
GROUP BY
l_linestatus,
ps_partkey,
date_trunc(l_ordertime, 'day')
UNION ALL
SELECT
l_linestatus,
l_extendedprice,
ps_partkey,
date_trunc(l_ordertime, 'day') as order_date
FROM
lineitem
LEFT JOIN partsupp ON l_partkey = ps_partkey
and l_suppkey = ps_suppkey;
```

You can split the above SQL statement into two parts and create two partitioned materialized views separately.

```
CREATE MATERIALIZED VIEW union_sub_mv1
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
partition by(order_date)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
l_linestatus,
sum(
l_extendedprice * (1 - l_discount)
) AS revenue,
ps_partkey,
date_trunc(l_ordertime, 'day') as order_date
FROM
lineitem
```

```
LEFT JOIN partsupp ON l_partkey = ps_partkey
and l_suppkey = ps_suppkey
GROUP BY
l_linestatus,
ps_partkey,
date_trunc(l_ordertime, 'day');
```

```
CREATE MATERIALIZED VIEW union_sub_mv2
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
partition by(order_date)
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
l_linestatus,
l_extendedprice,
ps_partkey,
date_trunc(l_ordertime, 'day') as order_date
FROM
lineitem
LEFT JOIN partsupp ON l_partkey = ps_partkey
and l_suppkey = ps_suppkey;
```

Then create a regular view that performs UNION ALL on the result sets of the two partitioned materialized views. This view (union_all_view) can be exposed externally.

```
CREATE VIEW union_all_view
AS
SELECT *
FROM
union_sub_mv1
UNION ALL
SELECT *
FROM
union_sub_mv2;
```

Partitioned Materialized Views Retaining Only Recent Partition Data

> **Note**
>
> This feature has been supported since Apache Doris version 2.1.1.

Materialized views can be configured to retain data only from the most recent partitions, automatically deleting expired partition data during each refresh. This can be achieved by setting the following properties for the materialized view: partition_sync_limit, partition_sync_time_unit, and partition_sync_date_format.

`partition_sync_limit`: When the base table's partition field is time-based, this property configures the synchronization range for base table partitions, working in conjunction with partition_sync_time_unit. For example, setting it to 3 with partition_sync_time_unit as DAY means only partitions and data from the last 3 days of the base table will be synchronized.

`partition_sync_time_unit`: The time unit for partition refresh, supporting DAY/MONTH/YEAR (default is DAY).

`partition_date_format`: When the base table's partition field is a string type, this property sets the date format if you want to use the partition_sync_limit capability.

Example: The materialized view defined below will only retain data from the last 3 days. If there's no data in the recent 3 days, querying this materialized view directly will return no results.

```
CREATE MATERIALIZED VIEW latest_partition_mv
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
PARTITION BY(order_date)
DISTRIBUTED BY RANDOM BUCKETS 2
PROPERTIES (
"partition_sync_limit" = "3",
"partition_sync_time_unit" = "DAY",
"partition_date_format" = "yyyy-MM-dd"
)
AS
SELECT
l_linestatus,
sum(
l_extendedprice * (1 - l_discount)
) AS revenue,
ps_partkey,
date_trunc(l_ordertime, 'day') as order_date
FROM
lineitem
LEFT JOIN partsupp ON l_partkey = ps_partkey
AND l_suppkey = ps_suppkey
GROUP BY
l_linestatus,
ps_partkey,
date_trunc(l_ordertime, 'day');
```

How to Use Materialized Views to Accelerate Queries

To use materialized views for query acceleration, first check the profile file to find the operation that consumes the most time in a query, which usually appears in Join, Aggregate, Filter, or Calculated Expressions.

For Join, Aggregate, Filters, and Calculated Expressions, building materialized views can help accelerate queries. If a Join operation in a query consumes a large amount of computing resources while Aggregate consumes relatively fewer resources, you can build materialized views targeting the Join operation.

Next, we'll explain in detail how to build materialized views for these four operations:

1. **For Join**

   You can extract common table join patterns used in queries to build materialized views. If transparent rewriting uses this materialized view, it can save Join computation. Remove the Filters from the query to create a more general Join materialized view.

2. **For Aggregate**

   It is recommended to use low-cardinality fields as dimensions when building materialized views. If the dimensions are related, the number after aggregation can be reduced as much as possible.

For example, with table t1, if the original table has 1,000,000 records, and the SQL query has `group by a, b, c`. If the cardinality of a, b, c is 100, 50, and 15 respectively, then the aggregated data would be around 75,000, indicating that this materialized view is effective. If a, b, c are correlated, the amount of aggregated data will be further reduced.

If a, b, c have high cardinality, it will cause the aggregated data to expand rapidly. If the aggregated data is more than the original table data, this scenario might not be suitable for building materialized views. For example, if c's cardinality is 3,500, then the aggregated data would be around 17,000,000, much larger than the original table data, making the performance acceleration benefit of building such a materialized view low.

The aggregation granularity of the materialized view should be finer than the query, meaning the aggregation dimensions of the materialized view should include the query's aggregation dimensions to provide the data needed by the query. The query may not write Group By, and similarly, the aggregation functions of the materialized view should include the query's aggregation functions.

Taking aggregate query acceleration as an example:

Query 1:

```
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey
WHERE
  o_orderdate <= DATE '2024-06-30'
  AND o_orderdate >= DATE '2024-05-01'
GROUP BY
  l_linestatus,
  o_shippriority,
  l_partkey;
```

Query 2:

```
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey
WHERE
  o_orderdate <= DATE '2024-06-30'
  AND o_orderdate >= DATE '2024-05-01'
GROUP BY
  l_linestatus,
  o_shippriority,
  l_suppkey;
```

Based on the above two SQL queries, we can build a more general materialized view that includes Aggregate. In this materialized view, we include both l_partkey and l_suppkey as group by dimensions for aggregation, and use o_orderdate as a filter condition. Note that o_orderdate is not only used in the materialized view's condition compensation but also needs to be included in the materialized view's aggregation group by dimensions.

After building the materialized view this way, both Query 1 and Query 2 can hit this materialized view. The materialized view definition is as follows:

```
CREATE MATERIALIZED VIEW common_agg_mv
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  o_shippriority,
  l_suppkey,
  l_partkey,
  o_orderdate
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey
GROUP BY
  l_linestatus,
  o_shippriority,
  l_suppkey,
  l_partkey,
  o_orderdate;
```

3. **For Filter**

If filters on the same fields frequently appear in queries, adding corresponding Filters in the materialized view can reduce the amount of data in the materialized view, thereby improving the performance when queries hit the materialized view.

Note that the materialized view should have fewer Filters than those appearing in queries, and the query's Filters should include the materialized view's Filters. For example, if the query is `a > 10 and b > 5`, the materialized view can have no Filter, or if it has Filters, it should filter on a and b with a larger data range than the query, such as `a > 5 and b > 5`, `b > 0`, or just `a > 5`.

4. For Calculated Expressions

Taking examples like case when and string processing functions, these expression calculations are very performance-intensive. If these can be pre-calculated in the materialized view, using the pre-calculated materialized view through transparent rewriting can improve query performance.

It is recommended that the number of columns in the materialized view should not be too many. If a query uses multiple fields, you should build corresponding materialized views for different columns based on the initial SQL pattern grouping, avoiding too many columns in a single materialized view.

Usage Scenarios

Scenario One: Query Acceleration

In BI reporting scenarios or other acceleration scenarios, users are sensitive to query response times and typically require results to be returned in seconds. Queries usually involve multiple table joins followed by aggregate calculations, which consume significant computing resources and sometimes make it difficult to guarantee timeliness. Asynchronous materialized views can handle this well, supporting both direct queries and transparent rewriting, where the optimizer automatically selects the optimal materialized view to respond to requests based on rewriting algorithms and cost models.

Use Case 1: Multi-table Join Aggregate Query Acceleration

Building more general materialized views can accelerate multi-table join aggregate queries.

Taking the following three query SQLs as examples:

Query 1:

```
SELECT
  l_linestatus,
  l_extendedprice * (1 - l_discount),
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey
WHERE
  o_orderdate <= DATE '2024-06-30'
  AND o_orderdate >= DATE '2024-05-01';
```

Query 2:

```
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  o_orderdate,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey
WHERE
  o_orderdate <= DATE '2024-06-30'
  AND o_orderdate >= DATE '2024-05-01'
GROUP BY
  l_linestatus,
  o_orderdate,
  o_shippriority;
```

Query 3:

```
SELECT
  l_linestatus,
  l_extendedprice * (1 - l_discount),
```

```
  o_orderdate,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

For the above queries, we can build the following materialized view to satisfy all the above queries.

The materialized view definition removes the filter conditions from Query 1 and Query 2 to get a more general Join, and pre-calculates the expression `l_extendedprice * (1 - l_discount)`, so when queries hit the materialized view, it can save expression calculation:

```
CREATE MATERIALIZED VIEW common_join_mv
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
  l_linestatus,
  l_extendedprice * (1 - l_discount),
  o_orderdate,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

If the above materialized view cannot meet the acceleration performance requirements of Query 2, we can build an aggregate materialized view. To maintain generality, we can remove the filter condition on the `o_orderdate` field:

```
CREATE MATERIALIZED VIEW target_agg_mv
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 2
AS
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  o_orderdate,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey
GROUP BY
  l_linestatus,
  o_orderdate,
  o_shippriority;
```

Use Case 2: Log Query Acceleration

In log query acceleration scenarios, it is recommended not to limit yourself to using only asynchronous materialized views; they can be combined with synchronous materialized views.

Generally, the base table is a partitioned table, mostly partitioned by hour, with single-table aggregate queries, and filter conditions are usually based on time and some flag bits. Sometimes when query

response speed cannot meet requirements, synchronous materialized views can usually be built for acceleration.

For example, the base table definition might be as follows:

```
CREATE TABLE IF NOT EXISTS test (
`app_name` VARCHAR(64) NULL COMMENT 'identifier',
`event_id` VARCHAR(128) NULL COMMENT 'identifier',
`decision` VARCHAR(32) NULL COMMENT 'enum value',
`time` DATETIME NULL COMMENT 'query time',
`id` VARCHAR(35) NOT NULL COMMENT 'od',
`code` VARCHAR(64) NULL COMMENT 'identifier',
`event_type` VARCHAR(32) NULL COMMENT 'event type'
)
DUPLICATE KEY(app_name, event_id)
PARTITION BY RANGE(time)
(
    FROM ("2024-07-01 00:00:00") TO ("2024-07-15 00:00:00") INTERVAL 1 HOUR
)
DISTRIBUTED BY HASH(event_id)
BUCKETS 3;
```

The materialized view can aggregate data by minute, which can also achieve a certain aggregation effect. For example:

```
CREATE MATERIALIZED VIEW sync_mv
    AS
    SELECT
      decision,
      code,
      app_name,
      event_id,
      event_type,
      date_trunc(time, 'minute'),
      DATE_FORMAT(
        `time`, '%Y-%m-%d'
      ),
      cast(FLOOR(MINUTE(time) / 15) as decimal(9, 0)),
      count(id) as cnt
    from
      test
    group by
      code,
      app_name,
      event_id,
      event_type,
      date_trunc(time, 'minute'),
      decision,
      DATE_FORMAT(time, '%Y-%m-%d'),
      cast(FLOOR(MINUTE(`time`) / 15) as decimal(9, 0));
```

The query statement might be as follows:

```
SELECT
    decision,
    CONCAT(
        CONCAT(
          DATE_FORMAT(
            `time`, '%Y-%m-%d'
```

```
        ),
        '',
        LPAD(
          cast(FLOOR(MINUTE(`time`) / 15) as decimal(9, 0)) * 15,
          5,
          '00'
        ),
        ':00'
      )
    ) as time,
    count(id) as cnt
  from
    test
  where
  date_trunc(time, 'minute') BETWEEN '2024-07-02 18:00:00'
    AND '2024-07-03 20:00:00'
  group by
    decision,
    DATE_FORMAT(
      `time`, "%Y-%m-%d"
    ),
    cast(FLOOR(MINUTE(`time`) / 15) as decimal(9, 0));
```

Scenario Two: Data Modeling (ETL)

Data analysis work often requires joining and aggregating multiple tables, a process that typically involves complex and frequently repeated queries. These types of queries may lead to high query latency or high resource consumption issues. However, if using asynchronous materialized views to build layered data models, these problems can be well avoided. You can create higher-level materialized views based on existing materialized views (supported since version 2.1.3), flexibly meeting different requirements.

Different levels of materialized views can be set with their own trigger methods, for example:

- The first layer of materialized views can be set to refresh periodically, and the second layer set to trigger refresh. This way, when the first layer of materialized views completes refreshing, it will automatically trigger the refresh of the second layer materialized views.
- If each layer of materialized views is set to refresh periodically, then when the second layer materialized view refreshes, it won't consider whether the first layer's materialized view data is synchronized with the base table, it will just process the first layer's materialized view data and synchronize it to the second layer.

Next, we'll use the TPC-H dataset to illustrate the application of asynchronous materialized views in data modeling, taking the analysis of monthly order quantities and profits by region and country as an example:

Original query (without using materialized views):

```
SELECT
n_name,
date_trunc(o.o_orderdate, 'month') as month,
count(distinct o.o_orderkey) as order_count,
sum(l.l_extendedprice * (1 - l.l_discount)) as revenue
FROM orders o
JOIN lineitem l ON o.o_orderkey = l.l_orderkey
JOIN customer c ON o.o_custkey = c.c_custkey
JOIN nation n ON c.c_nationkey = n.n_nationkey
```

```
JOIN region r ON n.n_regionkey = r.r_regionkey
GROUP BY n_name, month;
```

Using asynchronous materialized views for layered modeling:

Build DWD layer (detailed data), process order detail wide table

```
CREATE MATERIALIZED VIEW dwd_order_detail
BUILD IMMEDIATE REFRESH AUTO ON COMMIT
DISTRIBUTED BY RANDOM BUCKETS 16
AS
select
o.o_orderkey,
o.o_custkey,
o.o_orderstatus,
o.o_totalprice,
o.o_orderdate,
c.c_name,
c.c_nationkey,
n.n_name as nation_name,
r.r_name as region_name,
l.l_partkey,
l.l_quantity,
l.l_extendedprice,
l.l_discount,
l.l_tax
from orders o
join customer c on o.o_custkey = c.c_custkey
join nation n on c.c_nationkey = n.n_nationkey
join region r on n.n_regionkey = r.r_regionkey
join lineitem l on o.o_orderkey = l.l_orderkey;
```

Build DWS layer (summary data), perform daily order summary

```
CREATE MATERIALIZED VIEW dws_daily_sales
BUILD IMMEDIATE REFRESH AUTO ON COMMIT
DISTRIBUTED BY RANDOM BUCKETS 16
AS
select
date_trunc(o_orderdate, 'month') as month,
nation_name,
region_name,
bitmap_union(to_bitmap(o_orderkey)) as order_count,
sum(l_extendedprice * (1 - l_discount)) as net_revenue
from dwd_order_detail
group by
date_trunc(o_orderdate, 'month'),
nation_name,
region_name;
```

The optimized query using materialized views is as follows:

```
SELECT
nation_name,
month,
bitmap_union_count(order_count),
sum(net_revenue) as revenue
FROM dws_daily_sales
GROUP BY nation_name, month;
```

Scenario Three: Lake-Warehouse Integration Federated Data Query

In modern data architectures, enterprises often adopt a lake-warehouse integration design to balance data storage costs and query performance. Under this architecture, two key challenges are frequently encountered: - Limited Query Performance: When frequently querying data from data lakes, performance may be affected by network latency and third-party services, leading to query delays and impacting user experience. - Complexity of Data Layer Modeling: In the data flow and transformation process from data lake to real-time data warehouse, complex ETL processes are usually required, which increases maintenance costs and development difficulty.

Using Doris asynchronous materialized views can effectively address these challenges: - Transparent Rewriting Accelerates Queries: Materialize commonly used data lake query results into Doris internal storage, using transparent rewriting to effectively improve query performance. - Simplify Layer Modeling: Support creating materialized views based on tables in the data lake, enabling convenient transformation from data lake to real-time data warehouse, greatly simplifying the data modeling process.

For example, using Hive:

Create Catalog based on Hive, using TPC-H dataset

```
CREATE CATALOG hive_catalog PROPERTIES (
'type'='hms', -- hive meta store address
'hive.metastore.uris' = 'thrift://172.21.0.1:7004'
);
```

Create materialized view based on Hive Catalog

```
-- Materialized views can only be created on internal catalog, switch to internal catalog
switch internal;
create database hive_mv_db;
use hive_mv_db;

CREATE MATERIALIZED VIEW external_hive_mv
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
DISTRIBUTED BY RANDOM BUCKETS 12
AS
SELECT
n_name,
o_orderdate,
sum(l_extendedprice * (1 - l_discount)) AS revenue
FROM
customer,
orders,
lineitem,
supplier,
nation,
region
WHERE
c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND l_suppkey = s_suppkey
AND c_nationkey = s_nationkey
AND s_nationkey = n_nationkey
AND n_regionkey = r_regionkey
AND r_name = 'ASIA'
GROUP BY
```

```
n_name,
o_orderdate;
```

Run the following query, which will automatically use the materialized view for acceleration through transparent rewriting.

```
SELECT
n_name,
sum(l_extendedprice * (1 - l_discount)) AS revenue
FROM
customer,
orders,
lineitem,
supplier,
nation,
region
WHERE
c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND l_suppkey = s_suppkey
AND c_nationkey = s_nationkey
AND s_nationkey = n_nationkey
AND n_regionkey = r_regionkey
AND r_name = 'ASIA'
AND o_orderdate >= DATE '1994-01-01'
AND o_orderdate < DATE '1994-01-01' + INTERVAL '1' YEAR
GROUP BY
n_name
ORDER BY
revenue DESC;
```

Note

Doris currently cannot detect data changes in external tables other than Hive. When external table data is inconsistent, using materialized views may result in data inconsistency. The following switch indicates: whether materialized views participating in transparent rewriting are allowed to include external tables, default false. If you accept data inconsistency or ensure external table data consistency through periodic refresh, you can set this switch to true. Set whether materialized views containing external tables can be used for transparent rewriting, default not allowed, if you can accept data inconsistency or can ensure data consistency yourself, you can enable

```
SET materialized_view_rewrite_enable_contain_external_table = true;
```

If the materialized view is in MaterializedViewRewriteSuccessButNotChose status, it means the rewrite was successful but the plan was not chosen by CBO, possibly due to incomplete statistics of external tables. Enable getting row count from file list for statistics

```
SET enable_get_row_count_from_file_list = true;
```

View external table statistics to confirm if they are complete

```
SHOW TABLE STATS external_table_name;
```

Scenario Four: Improving Write Efficiency, Reducing Resource Contention

In high-throughput data write scenarios, system stability and efficient data processing are equally important. Through the flexible refresh strategies of asynchronous materialized views, users can choose appropriate refresh methods based on specific scenarios, thereby reducing write pressure and avoiding resource contention.

Compared to synchronous materialized views, asynchronous materialized views provide three flexible refresh strategies: manual trigger, trigger-based, and periodic trigger. Users can choose suitable refresh strategies based on scenario requirements. When base table data changes, it won't immediately trigger materialized view refresh, and delayed refresh helps reduce resource pressure, effectively avoiding write resource contention.

As shown below, the chosen refresh method is periodic refresh, refreshing every 2 hours. When orders and lineitem import data, it won't immediately trigger materialized view refresh.

```
CREATE MATERIALIZED VIEW common_schedule_join_mv
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 2 HOUR
DISTRIBUTED BY RANDOM BUCKETS 16
AS
SELECT
l_linestatus,
l_extendedprice * (1 - l_discount),
o_orderdate,
o_shippriority
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

Transparent rewriting can rewrite query SQL and achieve query acceleration, while also being able to rewrite import SQL to improve import efficiency. Starting from version 2.1.6, when materialized view and base table data are strongly consistent, DML operations like Insert Into or Insert Overwrite can be transparently rewritten, which significantly improves performance for data import scenarios.

1. Create target table for Insert Into data

```
CREATE TABLE IF NOT EXISTS target_table  (
orderdate      DATE NOT NULL,
shippriority   INTEGER NOT NULL,
linestatus     CHAR(1) NOT NULL,
sale           DECIMALV3(15,2) NOT NULL
)
DUPLICATE KEY(orderdate, shippriority)
DISTRIBUTED BY HASH(shippriority) BUCKETS 3;
```

2. common_schedule_join_mv

```
CREATE MATERIALIZED VIEW common_schedule_join_mv
BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 2 HOUR
DISTRIBUTED BY RANDOM BUCKETS 16
AS
SELECT
l_linestatus,
l_extendedprice * (1 - l_discount),
o_orderdate,
```

```
o_shippriority
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

Import statement before rewriting:

```
INSERT INTO target_table
SELECT
o_orderdate,
o_shippriority,
l_linestatus,
l_extendedprice * (1 - l_discount)
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey;
```

After transparent rewriting, the statement becomes:

```
INSERT INTO target_table
SELECT *
FROM common_schedule_join_mv;
```

Note that: If the DML operation involves external tables whose data changes cannot be detected, transparent rewriting may cause the latest base table data to not be imported into the target table in real-time. If users can accept data inconsistency or can ensure data consistency themselves, they can enable the following switch:

For DML, when the materialized view contains external tables whose data cannot be detected in real-time, whether to enable materialized view transparent rewriting based on structure information, default disabled

```
SET enable_dml_materialized_view_rewrite_when_base_table_unawareness = true;
```

Use Advice

Overview

Asynchronous materialized views improve query performance by precomputing and storing query results, but each refresh may incur significant overhead. This document provides usage recommendations for asynchronous materialized views.
For the refresh principles of materialized views, refer to: Refresh Principles

Recommended Usage Scenarios

Recommended Scenarios

Complex Aggregation Queries

- **Scenario**: Queries involving multi-table joins, complex aggregate functions (e.g., SUM, AVG, COUNT), or window functions
- **Advantage**: Avoids recomputing complex logic during each execution

Reporting

- **Scenario**: Reports requiring consistent snapshots at fixed time points (e.g., daily midnight)
- **Advantage**: Ensures all users see data from the same point in time

Computation-Intensive Analysis

- **Scenario**: Analytical queries involving complex mathematical calculations or data transformations, such as customer lifetime value calculations or predictive modeling
- **Advantage**: Precomputes results to reduce runtime resource consumption

Star/Snowflake Schemas in Data Warehouses

- **Scenario**: Fact tables joined with multiple dimension tables, e.g., sales fact tables joined with product, time, and region dimension tables
- **Advantage**: Pre-materializes join results to accelerate analytical queries

Data Lake Acceleration

- **Scenario**: Queries on data lakes may slow down due to network latency and object storage throughput limitations
- **Advantage**: Leverages Doris's local storage advantages to accelerate data lake analysis

Data Warehouse Layering

- **Scenario**: Base tables contain large amounts of raw data, and queries require complex ETL operations
- **Advantage**: Implements data warehouse layering by building multi-level asynchronous materialized views

Not Recommended Scenarios

Frequently Updated Base Tables

- **Scenario**: Source table data changes very frequently (e.g., multiple updates per minute)
- **Issue**: Asynchronous materialized views struggle to stay synchronized, and refresh costs are too high. Consider periodic refreshes instead.

Simple Queries

- **Scenario**: Queries involving only single-table scans or simple filtering
- **Issue**: The benefits of asynchronous materialized views cannot offset the refresh costs

Scenarios Requiring Real-Time (1–5 Minute Freshness) Data

- **Scenario**: Business requirements demand the latest data
- **Issue**: Asynchronous materialized views introduce data latency

Small Source Tables

- **Scenario**: Base tables contain only a small number of records (e.g., a few hundred rows)
- **Issue**: The optimization effect of asynchronous materialized views is negligible

Refresh Strategy Recommendations

Asynchronous materialized views offer three primary refresh strategies, each suited to different business scenarios and data characteristics. Choosing the right strategy is critical for balancing data freshness and system performance.

Detailed Refresh Strategies

Manual Refresh

How It Works: - Triggered explicitly by user commands or external system scheduling

Applicable Scenarios: - Reporting systems with low real-time data requirements - Historical data analysis in data warehouses - Scenarios requiring synchronization with specific business processes - Large-scale data refreshes requiring coordinated system resources

Pros and Cons: - Pros: Full control over refresh timing, avoiding peak business hours - Cons: Requires additional scheduling management and fault tolerance to prevent external loops from continuously triggering

refreshes

Scheduled Refresh

How It Works: - Automatically refreshes at fixed intervals - Minimum time unit: minutes - Can specify the start time for the first task run

Applicable Scenarios: - Periodic business metric monitoring - Tiered data pipelines - Time-sensitive reporting systems - Source data with regular fluctuations

Pros and Cons: - Pros: Scheduled data processing with predictable data latency - Cons: Limited data freshness; refresh sequences for related views require manual orchestration

Configuration Constraints:
Avoid configuring all materialized views for high-frequency scheduled refreshes to achieve near-real-time results, as this may: - Continuously occupy system resources - Cause refresh jobs to compete for resources - Frequent partition/tablet operations may impose heavy pressure on BEs

Trigger-Based Refresh

How It Works: - Automatically triggers refreshes when base table data changes

Applicable Scenarios: - Upper-layer views in a multi-level materialized view architecture - Scenarios where base tables change infrequently

Pros and Cons: - Pros: High data freshness and automation - Cons: May cause refresh storms and unpredictable system load

Configuration Constraints:
Avoid using trigger-based refreshes for foundational materialized views unless: - Base table refresh frequency is known to be low (e.g., changes every few tens of minutes)

Combined Refresh Strategy Recommendations

Layered Strategy

- **Foundation Layer**: Scheduled refresh (e.g., hourly)
- **Intermediate Layer**: Scheduled or trigger-based refresh
- **Presentation Layer**: Trigger-based or manual refresh

Business Criticality Tiering

- **Critical Real-Time Business Data**: Not recommended for asynchronous materialized views
- **Regular Analytical Data**: Scheduled refresh (daily/hourly)
- **Historical/Archived Data**: Manual refresh

Data Change Frequency Adaptation

- **High-Frequency Changes**: Scheduled refresh (longer intervals) or manual refresh
- **Low-Frequency Changes**: Trigger-based refresh or short-interval scheduled refresh
- **Bulk Changes**: Manual refresh after changes

Refresh Frequency Recommendations

These are general guidelines; actual settings should consider system resources, the number of materialized views, and other business resource usage.

Actual Refresh Time	Recommended Refresh Frequency
< 15s	≥ 5 minutes
< 10 minutes	≥ 1 hour
< 1 hour	≥ 1 day

Key Considerations for Asynchronous Materialized Views

1. **Monitoring**: After deploying materialized views, monitor system performance via metrics. Additional metrics for asynchronous materialized views will be exposed in the future. Currently, use tasks to check task count, execution status, and duration.
2. **Planning**: Plan the number of materialized views, refresh frequency, and the cluster's maximum computational capacity. Avoid "creating materialized views without maintaining them"—they are essentially enhanced ETL computations and require maintenance like traditional ETL.
3. **Resource Isolation**: Materialized views are data computation tasks. Implement resource isolation as needed.

FAQ

Build and Refresh

Q1: How does Doris determine which partitions need to be refreshed for a materialized view?

Doris internally calculates the partition correspondence between the materialized view and the base tables and records the version of the base table partitions used by the materialized view after the last successful refresh. For example, if materialized view, mv1 is created from base tables t1 and t2 and is partitioned based on t1.

Assuming partition p202003 of mv1 corresponds to partitions p20200301 and p20200302 of base table t1, after refreshing p202003, Doris will record partitions p20200301 and p20200302, along with the current version of table t2.

During the next refresh, Doris checks if the versions of p20200301, p20200302, and t2 have changed. If any of them have changed, it indicates that p202003 needs to be refreshed.

Alternatively, if changes to t2 can be accepted without triggering a refresh of mv1, the `excluded_trigger_tables` property of the materialized view can be used to configure this.

Q2: What can be done if a materialized view consumes too many resources, impacting other business operations?

You can control the resources allocated to materialized view refresh tasks by specifying a workload_group through the materialized view's properties.

It's important to note that if the memory allocation is too small and the refresh of a single partition requires more memory, the task may fail. This trade-off should be carefully considered based on business requirements.

Q3: Can a new materialized view be created based on an existing materialized view?

Yes, this is supported starting from Doris 2.1.3. However, each materialized view employs its own refresh logic when updating data. For example, if mv2 is based on mv1, which in turn is based on t1, the synchronization between mv1 and t1 will not be considered during the refresh of mv2.

Q4: Which external tables are supported by Doris?

All external tables supported by Doris can be used to create materialized views. However, only Hive currently supports partition refreshes, with support for other types planned in the future.

Q5: The materialized view displays the same data as Hive, but in reality, they are inconsistent.

A materialized view guarantees consistency only with the results obtained through the Catalog. Since the Catalog contains metadata and data caching, to ensure that the materialized view and Hive data remain consistent, you may need to refresh the Catalog using methods such as `REFRESH CATALOG` to synchronize the Catalog data with Hive.

Q6: Does materialized view support schema change?

No, schema changes are not supported because the column attributes of a materialized view are derived from the SQL definition of the materialized view itself. Explicit custom modifications are not allowed.

Q7: Can the base tables used by materialized views undergo schema changes?

Yes, schema changes are allowed. However, after the change, the status of the materialized views that use this base table will change from NORMAL to SCHEMA_CHANGE, at which point the materialized view cannot be used for transparent rewriting but direct queries to the materialized view will not be affected. If the next refreshing task of the materialized view is successful, its status will change back to NORMAL.

Q8: Can tables with the primary key model be used to create materialized views?

There are no restrictions on the data model of the base tables for materialized views. However, the materialized view itself can only be of the detailed model.

Q9: Can indexes be created on materialized views?

Yes.

Q10: Does the materialized view lock tables during refresh?

Table locking occurs for a brief period during the refresh but does not continuously occupy table locks (almost equivalent to the locking time during data import).

Q11: Is the materialized view suitable for near-real-time scenarios?

Not particularly. The minimum unit for refreshing materialized views is the partition, which can consume significant resources for large data volumes and lacks real-time capabilities. Consider using synchronous materialized views or other methods instead.

Q12: Error encountered when building a partitioned materialized view

Error Message:

```
Unable to find a suitable base table for partitioning
```

This error typically indicates that the SQL definition of the materialized view and the choice of partitioning fields do not allow incremental partition updates, resulting in an error during the creation of the partitioned materialized view.

- For incremental partition updates, the materialized view's SQL definition and partitioning field selection must meet specific requirements. See Materialized View Refresh Modes for details.
- The latest code can indicate the reason for partition build failure, with error summaries and descriptions provided in Appendix 2.

Example:

```
CREATE TABLE IF NOT EXISTS orders (
  o_orderkey INTEGER NOT NULL,
  o_custkey INTEGER NOT NULL,
  o_orderstatus CHAR(1) NOT NULL,
  o_totalprice DECIMALV3(15, 2) NOT NULL,
  o_orderdate DATE NOT NULL,
  o_orderpriority CHAR(15) NOT NULL,
  o_clerk CHAR(15) NOT NULL,
  o_shippriority INTEGER NOT NULL,
  O_COMMENT VARCHAR(79) NOT NULL
) DUPLICATE KEY(o_orderkey, o_custkey) PARTITION BY RANGE(o_orderdate) (
  FROM
    ('2024-05-01') TO ('2024-06-30') INTERVAL 1 DAY
) DISTRIBUTED BY HASH(o_orderkey) BUCKETS 3 PROPERTIES ("replication_num" = "1");


CREATE TABLE IF NOT EXISTS lineitem (
  l_orderkey INTEGER NOT NULL,
  l_partkey INTEGER NOT NULL,
  l_suppkey INTEGER NOT NULL,
  l_linenumber INTEGER NOT NULL,
  l_quantity DECIMALV3(15, 2) NOT NULL,
  l_extendedprice DECIMALV3(15, 2) NOT NULL,
  l_discount DECIMALV3(15, 2) NOT NULL,
  l_tax DECIMALV3(15, 2) NOT NULL,
  l_returnflag CHAR(1) NOT NULL,
  l_linestatus CHAR(1) NOT NULL,
  l_shipdate DATE NOT NULL,
  l_commitdate DATE NOT NULL,
  l_receiptdate DATE NOT NULL,
  l_shipinstruct CHAR(25) NOT NULL,
  l_shipmode CHAR(10) NOT NULL,
  l_comment VARCHAR(44) NOT NULL
) DUPLICATE KEY(
  l_orderkey, l_partkey, l_suppkey,
  l_linenumber
) DISTRIBUTED BY HASH(l_orderkey) BUCKETS 3 PROPERTIES ("replication_num" = "1");
```

The materialized view definition below allows for incremental partition updates if `orders.o_orderdate` is chosen as the partitioning field for the materialized view. Conversely, using `lineitem.l_shipdate` would not enable incremental updates.

Reason:

1. `lineitem.l_shipdate` is not a partitioning column of the base table, and `lineitem` does not have a partitioning column defined.
2. `lineitem.l_shipdate` is the column that generates `null` values during the `outer join` operation.

```
CREATE MATERIALIZED VIEW mv_1 BUILD IMMEDIATE REFRESH AUTO ON MANUAL partition by(o_orderdate) DISTRIBUTED BY RANDOM BUCKETS 2
PROPERTIES ('replication_num' = '1') AS
SELECT
  l_linestatus,
  sum(
    l_extendedprice * (1 - l_discount)
  ) AS revenue,
  o_orderdate,
  o_shippriority
FROM
  orders
  LEFT JOIN lineitem ON l_orderkey = o_orderkey
```

```
WHERE
  o_orderdate <= DATE '2024-06-30'
  AND o_orderdate >= DATE '2024-05-01'
GROUP BY
  l_linestatus,
  o_orderdate,
  o_shippriority;
```

Q13: Error encountered when creating a materialized view

Error Message:

```
ERROR 1105 (HY000): errCode = 2, detailMessage = Syntax error in line 1:
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
```

Reasons may be:

1. The statement for creating an asynchronous materialized view is only supported by the new optimizer. Ensure you are using the new optimizer:

   ```
   SET enable_nereids_planner = true;
   ```

2. There may be a typographical error in the refresh keywords or a syntax error in the SQL definition of the materialized view. Check the SQL definition and creation statement for the materialized view for correctness.

Q14: After the materialized view is refreshed successfully, there is still no data

The materialized view determines whether the data needs to be updated based on its ability to retrieve version information from the base table or base table partitions.

When encountering data lakes that currently do not support retrieving version information, such as JDBC Catalog, the refresh process will assume that the materialized view does not need to be updated. Therefore, when creating or refreshing a materialized view, you should specify complete instead of auto.

For the progress of materialized view support for data lakes, please refer toData Lake Support Status.

Q15: Why is my partitioned materialized view always fully refreshed?

The incremental refresh of a materialized view's partitions depends on version information from the base table partitions. If data in the base table partitions changes since the last refresh, the materialized view will refresh those corresponding partitions. If your partitioned materialized view is being fully refreshed, the possible reasons are:

Changes occurred in non-partition-tracked tables referenced in the materialized view's definition SQL, making it impossible to determine which partitions need updating, thus forcing a full refresh. For example: This materialized view tracks the o_orderdate partition of the orders table, but if data in lineitem or partsupp changes, the system cannot determine which partitions need updating, resulting in a full refresh.

```
CREATE MATERIALIZED VIEW partition_mv
BUILD IMMEDIATE
REFRESH AUTO
ON SCHEDULE EVERY 1 DAY STARTS '2024-12-01 20:30:00'
PARTITION BY (DATE_TRUNC(o_orderdate, 'MONTH'))
DISTRIBUTED BY HASH (l_orderkey) BUCKETS 2
PROPERTIES
("replication_num" = "3")
AS
SELECT
o_orderdate,
l_orderkey,
l_partkey
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey
LEFT JOIN partsupp ON ps_partkey = l_partkey
and l_suppkey = ps_suppkey;
```

You can check which base tables the materialized view tracks by running

```
SELECT *
FROM mv_infos('database'='db_name')
WHERE Name = 'partition_mv' \G
```

The returned result shows partitionType=FOLLOW_BASE_TABLE in MvPartitionInfo, indicating the materialized view partitions follow the base table partitions. relatedCol shows o_orderdate, meaning the materialized view partitions are based on the o_orderdate column.

```
Id: 1752809156450
Name: partition_mv
JobName: inner_mtmv_1752809156450
State: NORMAL
SchemaChangeDetail:
RefreshState: SUCCESS
RefreshInfo: BUILD IMMEDIATE REFRESH AUTO ON SCHEDULE EVERY 1 DAY STARTS "2024-12-01 20:30:00"
QuerySql: SELECT
`internal`.`doc_db`.`orders`.`o_orderdate`,
`internal`.`doc_db`.`lineitem`.`l_orderkey`,
`internal`.`doc_db`.`lineitem`.`l_partkey`
FROM
`internal`.`doc_db`.`orders`
LEFT JOIN `internal`.`doc_db`.`lineitem` ON `internal`.`doc_db`.`lineitem`.`l_orderkey` = `internal`.`doc_db`.`orders`.`o_orderkey`
LEFT JOIN `internal`.`doc_db`.`partsupp` ON `internal`.`doc_db`.`partsupp`.`ps_partkey` = `internal`.`doc_db`.`lineitem`.`l_partkey`
and `internal`.`doc_db`.`lineitem`.`l_suppkey` = `internal`.`doc_db`.`partsupp`.`ps_suppkey`
MvPartitionInfo: MTMVPartitionInfo{partitionType=EXPR, relatedTable=orders, relatedCol='o_orderdate', partitionCol='o_orderdate'}
SyncWithBaseTables: 1
```

Solution:

If changes in lineitem or partsupp tables don't affect your materialized view, you can exclude these tables from triggering full refreshes by setting the `excluded_trigger_tables` property: `ALTER MATERIALIZED VIEW partition_mv set("excluded_trigger_tables"="lineitem,partsupp");`

Queries and Transparent Rewriting

Q1: How to confirm if a Materialized View hits, and how to find the reasons for Non-Hits?

You can use `explain query_sql` to view a summary of materialized view hits.

For example, consider the following materialized view:

```
CREATE MATERIALIZED VIEW mv11
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
partition by(l_shipdate)
DISTRIBUTED BY HASH(l_orderkey) BUCKETS 10
PROPERTIES ('replication_num' = '1')
AS
SELECT l_shipdate, l_orderkey, O_ORDERDATE, count(*)
FROM lineitem
LEFT OUTER JOIN orders on l_orderkey = o_orderkey
GROUP BY l_shipdate, l_orderkey, O_ORDERDATE;
```

The query can be:

```
explain
SELECT l_shipdate, l_orderkey, O_ORDERDATE, count(*)
FROM lineitem
LEFT OUTER JOIN orders on l_orderkey = o_orderkey
GROUP BY l_shipdate, l_orderkey, O_ORDERDATE;
```

- The materialized view hit information is at the end of the plan.
- **MaterializedViewRewriteSuccessAndChose:** Indicates that transparent rewriting was successful, and lists the names of the materialized views chosen by the Cost-Based Optimizer (CBO).
- **MaterializedViewRewriteSuccessButNotChose:** Indicates that transparent rewriting was successful, but lists the names of materialized views that were not chosen by the CBO. Not choosing them means the execution plan will not use these materialized views.
- **MaterializedViewRewriteFail:** Lists the failures and summaries of the reasons for transparent rewriting failures.
- If there is no `MaterializedView` information at the end of the `explain` output, it means the materialized view is in an unusable state and therefore cannot participate in transparent rewriting. (For details on when a materialized view becomes unusable, refer to the “Usage and Practice - Viewing Materialized View Status” section.)

Here’s an example output:

```
| MaterializedView                                                                |
| MaterializedViewRewriteSuccessAndChose:                                         |
| internal#regression_test_nereids_rules_p0_mv#mv11,                              |
|                                                                                 |
| MaterializedViewRewriteSuccessButNotChose:                                      |
|                                                                                 |
| MaterializedViewRewriteFail:                                                    |
+---------------------------------------------------------------------------------+
```

Q2: What Are the Reasons for a Materialized View Not Hitting?

First, to confirm if a materialized view hits, execute the following SQL (refer to Queries and Transparent Rewriting - Q1 for details):

```
explain
your_query_sql;
```

If there is no hit, the following reasons may apply:

- In Doris versions before 2.1.3, the transparent rewriting feature for materialized views is disabled by default. You need to enable the corresponding switch to achieve transparent rewriting. For specific switch values, refer to async-materialized view-related switches.
- The materialized view may be in an unusable state, preventing transparent rewriting from hitting it. To view the build status of the materialized view, refer to the section on viewing materialized view status.
- If, after checking the first two steps, the materialized view still does not hit, it may be because SQL defines the materialized view and the query SQL is outside the current rewriting capabilities of the materialized view. Refer to the Materialized View Transparent Rewriting Capabilities for details.
- For detailed information and explanations on failed hits, refer to Appendix 1.

Here's an example of a failed transparent rewriting for a materialized view:

Case 1:

Materialized view creation SQL:

```
CREATE MATERIALIZED VIEW mv11
BUILD IMMEDIATE REFRESH AUTO ON MANUAL
partition by(l_shipdate)
DISTRIBUTED BY HASH(l_orderkey) BUCKETS 10
PROPERTIES ('replication_num' = '1')
AS
SELECT l_shipdate, l_orderkey, O_ORDERDATE, count(*)
FROM lineitem
LEFT OUTER JOIN orders on l_orderkey = o_orderkey
GROUP BY l_shipdate, l_orderkey, O_ORDERDATE;
```

Query execution:

```
explain
SELECT l_shipdate, l_linestatus, O_ORDERDATE, count(*)
FROM orders
LEFT OUTER JOIN lineitem on l_orderkey = o_orderkey
GROUP BY l_shipdate, l_linestatus, O_ORDERDATE;
```

`Explain` output:

```
| MaterializedView                                                
```

```
|                                                
```

Q4: What If Direct Queries to a Materialized View Return No Data?

It's possible that the materialized view is still being built or that the building has failed.

You can check the status of the materialized view to confirm this. Refer to the section on viewing materialized view status for specific methods.

Q5: When the Data in the Base Table Used by a Materialized View Changes, but the Materialized View Has Not Yet Been Refreshed, What Is the Behavior of Transparent Rewriting?

There is a certain delay between the data in async-materialized views and the underlying tables.

1. For internal tables and external tables that can perceive data changes (like Hive): When the underlying table data changes, whether the materialized view is usable depends on the `grace_period` threshold.

`grace_period` is the time period that allows for data inconsistency between the materialized view and the underlying table. For example:

- If `grace_period` is set to 0, it means the materialized view must be consistent with the underlying table data for it to be used for transparent rewriting. For external tables (except Hive), since they cannot perceive data changes, materialized views that use them can still be used for transparent rewriting (but the data may be inconsistent).
- If `grace_period` is set to 10 seconds, it allows for up to 10 seconds of delay between the materialized view data and the underlying table data. If the delay is within 10 seconds, the materialized view can still be used for transparent rewriting.

2. For partitioned materialized views, if some partitions become invalid, there are two scenarios:

- If the query does not use data from invalid partitions, the materialized view is still usable.
- If the query uses data from invalid partitions, and the data delay is within the `grace_period`, the materialized view is still usable. If the delay exceeds the `grace_period`, the query can be responded to by unioning the original table and the materialized view. This requires enabling the `enable_materialized_view_union_rewrite` switch, which is on by default from version 2.1.5.

Reference

1 Materialized View-Related Configuration

Configuration	Description
SET enable_nereids_planner = true;	Enables the new optimizer required for materialized view rewriting.
SET enable_materialized_view_rewrite = true;	Enables or disables query rewriting. Default: Enabled.
SET materialized_view_rewrite_enable_contain_external_table = true;	Allows materialized views containing external tables to participate in rewriting. Default: Disabled.
SET materialized_view_rewrite_success_candidate_num = 3;	Maximum number of successful rewrite candidates considered by CBO. Default: 3.

Configuration	Description
SET enable_materialized_view_union_rewrite = true;	Allows UNION ALL between base tables and materialized views when data is insufficient. Default: Enabled.
SET enable_materialized_view_nest_rewrite = true;	Enables nested materialized view rewriting. Default: Disabled.
SET materialized_view_relation_mapping_max_count = 8;	Maximum number of relation mappings during rewriting. Default: 8.

2 Summary and Description of Transparent Rewriting Failures

Summary	Description
View struct info is invalid	The structure information of the materialized view is invalid. Currently supported SQL patterns for rewriting include joins in both queries and materialized views, and aggregations in queries with or without joins in materialized views. This error is often displayed during transparent rewriting, as each rewriting rule is responsible for a specific SQL pattern. If a rule is hit that does not match the required pattern, this error will occur, but it is generally not the primary cause of rewriting failure.
Materialized view rule exec fail	Typically indicates an exception during the execution of the transparent rewriting rule. To investigate, use EXPLAIN memo plan query_sql to view the specific exception stack.
Match mode is invalid	The number of tables in the query does not match the number of tables in the materialized view, and rewriting is not supported.
Query to view table mapping is null	Failed to generate the mapping between the query and materialized view tables.
queryToViewTableMappings are over the limit and be intercepted	Too many self-joined tables in the query led to excessive expansion of the rewriting space, stopping transparent rewriting.
Query to view slot mapping is null	Failed to map slots between the query and materialized view tables.
The graph logic between query and view is not consistent	The join types or joined tables between the query and materialized view are different.
Predicate compensate fail	Typically occurs when the query's condition range exceeds that of the materialized view, e.g., query is a > 10 but materialized view is a > 15.
Rewrite compensate predicate by view fail	Predicate compensation failed, usually because the query has additional conditions that need compensation, but the columns used in those conditions do not appear in the SELECT clause of the materialized view.
Calc invalid partitions fail	For partitioned materialized views, attempts to calculate whether partitions used by the query are valid failed.
mv can not offer any partition for query	Query only uses invalid partitions of materialized view (data changed since last refresh). Check partition validity via show partitions from mv_name (SyncWithBaseTables=false indicates need for refresh). Set grace_period (in seconds) to allow data latency.
Add filter to base table fail when union rewrite	The query used invalid partitions of the materialized view, and attempting to union all the materialized view and base table failed.

Summary	Description
RewrittenPlan output logical properties is different with target group	After rewriting, the output logical properties of the materialized view do not match those of the original query.
Rewrite expressions by view in join fail	In join rewriting, fields or expressions used in the query are not present in the materialized view.
Rewrite expressions by view in scan fail	In single-table rewriting, fields or expressions used in the query are not present in the materialized view.
Split view to top plan and agg fail, view doesn't not contain aggregate	During aggregation rewriting, the materialized view does not contain an aggregate function.
Split query to top plan and agg fail	During aggregation rewriting, the query does not contain an aggregate function.
rewritten expression contains aggregate functions when group equals aggregate rewrite	When the query and materialized view have the same GROUP BY, the rewritten expression contains aggregate functions.
Can not rewrite expression when no roll up	When the query and materialized view have the same GROUP BY, expression rewriting fails.
Query function roll up fail	During aggregation rewriting, the aggregation function roll-up fails.
View dimensions do not cover the query dimensions	The GROUP BY in the query uses dimensions not present in the GROUP BY of the materialized view.
View dimensions don't not cover the query dimensions in bottom agg	Similar to above, but specific to bottom-level aggregations.
View dimensions do not cover the query group set dimensions	The GROUP SETS in the query use dimensions not present in the GROUP BY of the materialized view.
The only one of query or view is scalar aggregate and can not rewrite expression meanwhile	The query has a GROUP BY but the materialized view does not.
Both query and view have group sets, or query doesn't have but view has, not supported	Unsupported transparent rewriting scenario involving GROUP SETS in both the query and materialized view, or only in the materialized view.

3 Reasons of Async-Materialized View Partition Building Failures

The refresh mechanism for partitioned materialized views relies on incremental partition updates:

- First, calculate whether the partition columns of the materialized view can be mapped to those of the base table.
- Second, determine the specific mapping relationship, whether it is 1:1 or 1:n.

Abstract	Description
Partition column cannot be found in the SQL SELECT column	The column used after PARTITION BY in the materialized view definition must appear in the SELECT clause of the SQL defining the materialized view.
Cannot find a valid partition track column, because %s	Unable to locate a suitable partition column; the specific reason follows "because".

Abstract	Description
Partition track does not support mark join	The column referenced by the partition field of the materialized view is a partition column of the input table in a mark join, which is currently not supported.
Partition column is in an unsupported join null generation side	The referenced column of the partition field in the materialized view is on the null-generating side of a join, such as the right side of a LEFT JOIN.
Relation should be LogicalCatalogRelation	The scan type of the partition base table referenced by the materialized view should be LogicalCatalogRelation; other types are currently not supported.
Self join does not support partition update	For SQL queries involving self-joins, constructing a materialized view is currently not supported.
Partition track already has a related base table column	The partition column referenced by the materialized view currently only supports referencing the partition column of a single base table.
Relation base table is not MTMVRelatedTableIf	The partition base table referenced by the materialized view does not inherit from MTMVRelatedTableIf, which indicates whether a table can be partitioned.
The related base table is not a partition table	The base table used by the materialized view is not a partition table.
The related base table partition column doesn’t contain the MV partition	The column referenced after PARTITION BY in the materialized view does not exist in the partition base table.
Group BY sets are empty, does not contain the target partition	The SQL defining the materialized view uses aggregation, but the GROUP BY clause is empty.
Window partition sets do not contain the target partition	Window functions are used, but the partition column referenced by the materialized view is not in the PARTITION BY clause.
Unsupported plan operation in track partition	The SQL defining the materialized view uses unsupported operations, such as ORDER BY.
Context partition column should be a slot from column	Window functions are used, and in the PARTITION BY clause, the partition column referenced by the materialized view is not a simple column but an expression.
Partition expressions use more than one slot reference	The partition column after GROUP BY or PARTITION BY is an expression containing multiple columns, rather than a simple column. For example, GROUP BY partition_col + other_col.
Column to check using invalid implicit expression	The partition column of the materialized view can only be used in date_trunc, and the expression using the partition column can only be date_trunc, etc.
Partition column time unit level should be greater than SQL SELECT column	In the materialized view, the time unit granularity in date_trunc after PARTITION BY is smaller than the time unit granularity appearing after SELECT in the SQL defining the materialized view. For example, the materialized view uses `PARTITION BY(date_trunc(col, 'day'))`, but the SQL defining the materialized view has `date_trunc(col, 'month')` after SELECT.

SQL Cache

Description

SQL Cache is a query optimization mechanism provided by Doris that can significantly enhance query performance. It reduces redundant computations by caching query results, making it suitable for scenarios where data update infrequently.

SQL Cache stores and retrieves caches based on the following key factors:

- SQL Text
- View Definitions
- Table and Partition Versions
- User Variables and Result Values
- Non-deterministic Functions and Result Values
- Row Policy Definitions
- Data Masking Definitions

The combination of these factors uniquely determines a cached dataset. If any of these factors change, such as variations in SQL, different query fields or conditions, or version changes after data updates, the cache will not be hit.

For queries involving multi-table joins, if one of the tables is updated, the partition ID or version number will differ, resulting in a cache miss.

SQL Cache is highly suitable for T+1 update scenarios. Data is updated early in the morning, the first query fetches results from the Backend (BE) and stores them in the cache, and subsequent queries of the same nature retrieve results directly from the cache. Real-time data updates can also use SQL Cache, but may face a lower cache hit rate.

Currently, SQL Cache supports both internal OlapTables and external Hive tables.

Usage Limitations

Non-Deterministic Functions

1. Non-deterministic functions refer to those whose computation results do not form a fixed relationship with their input parameters.
2. Take the common function `select now()` as an example. It returns the current date and time. Since this function returns different results when executed at different times, its return value is dynamically changing. The `now` function returns time at the second level, so SQL Cache from the previous second

can be reused within the same second; however, a new SQL Cache needs to be created for the next second.

3. To optimize cache utilization, it is recommended to convert such fine-grained time into coarse-grained time, such as using `select * from tbl where dt=date(now())`. In this case, queries within the same day can leverage the SQL Cache.

4. In contrast, the `random()` function is difficult to utilize cache because its results vary each time it is executed. Therefore, the use of such non-deterministic functions in queries should be avoided as much as possible.

Principles

BE Principle

In most cases, SQL Cache results are selected through a consistent hashing method to choose a BE and are stored in that BE's memory. These results are stored in a HashMap structure. When requests to read or write the cache arrive, the system uses a digest of metadata information, such as the SQL string, as a key to quickly retrieve and manipulate result data from the HashMap.

FE Principle

When the Frontend (FE) receives a query request, it first searches in its memory using the SQL string to determine if the same query has been executed before and attempts to retrieve the metadata information for that query. This information includes the versions of the tables and partitions involved in the query.

If these metadata remain unchanged, it indicates that the data in the corresponding tables has not been modified, allowing the reuse of the previous SQL Cache. In this case, the FE can skip the SQL parsing and optimization process, directly locate the corresponding BE based on the consistent hashing algorithm, and attempt to retrieve the query results from it.

- If the target BE contains a cached result for the query, the FE can quickly return the results to the client.
- Conversely, if the corresponding result cache is not found in the BE, the FE needs to execute the complete SQL parsing and optimization process and then transmit the query plan to the BE for computation and processing.

When the BE returns the computation results to the FE, the FE is responsible for storing these results in the corresponding BE and recording the metadata information of this query in its memory. This is done so that when the same query is received subsequently, the FE can directly retrieve the results from the BE, thereby improving query efficiency.

Additionally, if the SQL optimization phase determines that the query results contain only 0 or 1 row of data, the FE will choose to store these results in its memory to respond more quickly to potential future identical queries.

Get Started

Enabling and Disabling SQL Cache

```
-- Enable SQL Cache for the current session, default is disabled
set enable_sql_cache=true;
-- Disable SQL Cache for the current session
set enable_sql_cache=false;

-- Globally enable SQL Cache, default is disabled
set global enable_sql_cache=true;
-- Globally disable SQL Cache
set global enable_sql_cache=false;
```

Checking If a Query Hits SQL Cache

In Doris versions 2.1.3 and later, users can execute the `explain plan` statement to check if the current query successfully hits the SQL Cache.

As shown in the example, when the query plan tree contains `LogicalSqlCache` or `PhysicalSqlCache` nodes, it indicates that the query has hit the SQL Cache.

```
> explain plan select * from t2;

+----------------------------------------------------------------------------------------------------------------+
| Explain String (Nereids Planner)                                                                               |
+----------------------------------------------------------------------------------------------------------------+
| ========== PARSED PLAN (time: 28ms) ==========                                                                 |
| LogicalSqlCache[1] ( queryId=711dea740e4746e6-8bc11afe08f6542c )                                               |
| +--PhysicalResultSink[39] ( outputExprs=[id#0, name#1] )                                                       |
|    +--PhysicalOlapScan[t2]@0 ( stats=12 )                                                                      |
|                                                                                                                |
| ========== ANALYZED PLAN ==========                                                                            |
| LogicalSqlCache[1] ( queryId=711dea740e4746e6-8bc11afe08f6542c )                                               |
| +--PhysicalResultSink[39] ( outputExprs=[id#0, name#1] )                                                       |
|    +--PhysicalOlapScan[t2]@0 ( stats=12 )                                                                      |
|                                                                                                                |
| ========== REWRITTEN PLAN ==========                                                                           |
| LogicalSqlCache[1] ( queryId=711dea740e4746e6-8bc11afe08f6542c )                                               |
| +--PhysicalResultSink[39] ( outputExprs=[id#0, name#1] )                                                       |
|    +--PhysicalOlapScan[t2]@0 ( stats=12 )                                                                      |
|                                                                                                                |
| ========== OPTIMIZED PLAN ==========                                                                           |
| PhysicalSqlCache[3] ( queryId=711dea740e4746e6-8bc11afe08f6542c, backend=192.168.126.3:9051, rowCount=12 )     |
| +--PhysicalResultSink[39] ( outputExprs=[id#0, name#1] )                                                       |
|    +--PhysicalOlapScan[t2]@0 ( stats=12 )                                                                      |
+----------------------------------------------------------------------------------------------------------------+
```

For versions before Doris 2.1.3, users need to check the Profile information to confirm if the query hits the SQL Cache. In the Profile information, if the `Is Cached:` field displays `Yes`, it indicates that the query has successfully hit the SQL Cache.

```
Execution  Summary:
      -  Parse  SQL  Time:  18ms
      -  Nereids  Analysis  Time:  N/A
      -  Nereids  Rewrite  Time:  N/A
```

```
    - Nereids Optimize Time: N/A
    - Nereids Translate Time: N/A
    - Workload Group: normal
    - Analysis Time: N/A
    - Wait and Fetch Result Time: N/A
    - Fetch Result Time: 0ms
    - Write Result Time: 0ms
    - Doris Version: 915138e801
    - Is Nereids: Yes
    - Is Cached: Yes
    - Total Instances Num: 0
    - Instances Num Per BE:
    - Parallel Fragment Exec Instance Num: 1
    - Trace ID:
    - Transaction Commit Time: N/A
    - Nereids Distribute Time: N/A
```

Both methods provide effective means for users to verify whether queries utilize the SQL Cache, helping users better assess query performance and optimize query strategies.

Metrics and Monitor

1. The HTTP interface on the FE `http://${FE_IP}:${FE_HTTP_PORT}/metrics` **returns two relevant metrics:** This indicator counts the number of hits, which only increases and never decreases. When FE is restarted, the count starts from 0.

```
# Represents that 1 SQL has been written to the cache
doris_fe_cache_added{type="sql"} 1

# Represents that the SQL Cache has been hit twice
doris_fe_cache_hit{type="sql"} 2
```

2. The HTTP interface on the BE `http://${BE_IP}:${BE_HTTP_PORT}/metrics` **returns relevant information:** Since different caches may be stored in different BEs, it is necessary to collect metrics from all BEs to obtain complete information.

```
# Represents that there are 1205 caches in the memory of the current BE
doris_be_query_cache_sql_total_count 1205

# The current total memory occupied by all caches in the BE is 44k
doris_be_query_cache_memory_total_byte 44101
```

Memory Control

FE Memory Control

In FE, the metadata information of Cache is set to weak references. When FE memory is insufficient, the system will automatically release the least recently used Cache metadata. Additionally, users can further limit FE memory usage by executing the following SQL statements. This configuration takes effect in real-time and needs to be set for each FE. For persistent configuration, it should be saved in the fe.conf file.

```
-- Store up to 100 Cache metadata items, automatically releasing the least recently used ones when exceeded. The default value is
100.
```

```
ADMIN SET FRONTEND CONFIG ('sql_cache_manage_num'='100');

-- Automatically release Cache metadata after 300 seconds of inactivity. The default value is 300.
ADMIN SET FRONTEND CONFIG ('expire_sql_cache_in_fe_second'='300');
```

BE Memory Control

Modify the following configurations in the be.conf file, and the changes will take effect after restarting BE:

```
-- When the Cache memory exceeds query_cache_max_size_mb + query_cache_elasticity_size_mb,
-- release the least recently used Cache until the memory usage is below query_cache_max_size_mb.
query_cache_max_size_mb = 256
query_cache_elasticity_size_mb = 128
```

Furthermore, configurations can be set in FE to avoid creating SQL Cache when the result row count or size exceeds certain thresholds:

```
-- By default, do not create SQL Cache for results exceeding 3000 rows.
ADMIN SET FRONTEND CONFIG ('cache_result_max_row_count'='3000');

-- By default, do not create SQL Cache for results exceeding 30MB.
ADMIN SET FRONTEND CONFIG ('cache_result_max_data_size'='31457280');
```

Troubleshooting Cache Miss

The reasons for cache invalidation typically include the following:

1. Changes in table/view structure, such as executing `drop table`, `replace table`, `alter table`, or `alter view`.
2. Changes in table data, such as executing `insert`, `delete`, `update`, or `truncate`.
3. Removal of user privileges, such as executing `revoke`.
4. Use of non-deterministic functions with changing evaluation values, such as executing `select random()`.
5. Use of variables with changing values, such as executing `select * from tbl where dt = @dt_var`.
6. Changes in Row Policy or Data Masking, such as setting certain table data to be invisible to users.
7. The result row count exceeds the FE-configured `cache_result_max_row_count`, with a default value of 3000 rows.
8. The result size exceeds the FE-configured `cache_result_max_data_size`, with a default value of 30MB.

High-Concurrency Point Query Optimization

> **Tips**
>
> This feature is supported since the Apache Doris 2.0 version

Description

Doris is built on a columnar storage format engine. In high-concurrency service scenarios, users always want to retrieve entire rows of data from the system. However, when tables are wide, the columnar format greatly amplifies random read IO. Doris query engine and planner are too heavy for some simple queries, such as point queries. A short path needs to be planned in the FE's query plan to handle such queries. FE is the access layer service for SQL queries, written in Java. Parsing and analyzing SQL also leads to high CPU overhead for high-concurrency queries. To solve these problems, we have introduced row storage, short query path, and PreparedStatement in Doris. Below is a guide to enable these optimizations.

Row Store Format

We support a row format for olap table to reduce point lookup io cost, but to enable this format, you need to spend more disk space for row format store. Currently, we store row in an extra column called `row column` for simplicity. The Row Storage mode can only be turned on when creating a table. You need to specify the following properties in the property of the table creation statement:

```
"store_row_column" = "true"
```

Accelerate point query for unique model

The above row storage is used to enable the Merge-On-Write strategy under the Unique model to reduce the IO overhead during enumeration. When `enable_unique_key_merge_on_write` and `store_row_column` are enabled when creating a Unique table, the query of the primary key will take a short path to optimize SQL execution, and only one RPC is required to complete the query. The following is an example of enabling the Merge-On-Write strategy under the Unique model by combining the query and row existence:

```
CREATE TABLE `tbl_point_query` (
    `k1` int(11) NULL,
    `v1` decimal(27, 9) NULL,
    `v2` varchar(30) NULL,
    `v3` varchar(30) NULL,
    `v4` date NULL,
    `v5` datetime NULL,
    `v6` float NULL,
    `v7` datev2 NULL
) ENGINE=OLAP
UNIQUE KEY(`k1`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`k1`) BUCKETS 1
```

```
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1",
    "enable_unique_key_merge_on_write" = "true",
    "light_schema_change" = "true",
    "store_row_column" = "true"
);
```

Note: 1. `enable_unique_key_merge_on_write` should be enabled, since we need primary key for quick point lookup in storage engine

2. when condition only contains primary key like `select * from tbl_point_query where key = 123`, such query will go through the short fast path
3. `light_schema_change` should also been enabled since we rely on `column unique id` of each column when doing a point query.
4. It only supports equality queries on the key column of a single table and does not support joins or nested subqueries. The WHERE condition should consist of the key column alone and be an equality comparison. It can be considered as a type of key-value query.
5. Enabling rowstore may lead to space expansion and occupy more disk space. For scenarios where querying only specific columns is needed, starting from Doris 3.0, it is recommended to use `"row_store_columns"="k1,v1,v2"` to specify certain columns for rowstore storage. Queries can then selectively access these columns, for example:

   ```
   SELECT k1, v1, v2 FROM tbl_point_query WHERE k1 = 1
   ```

Using `PreparedStatement`

In order to reduce CPU cost for parsing query SQL and SQL expressions, we provide `PreparedStatement` feature in FE fully compatible with mysql protocol (currently only support point queries like above mentioned).Enable it will pre calculate PreparedStatement SQL and expressions and caches it in a session level memory buffer and will be reused later on.We could improve 4x+ performance by using `PreparedStatement` when CPU became bottleneck doing such queries.Bellow is an JDBC example of using `PreparedStatement`.

1. Setup JDBC url and enable server side prepared statement

   ```
   url = jdbc:mysql://127.0.0.1:9030/ycsb?useServerPrepStmts=true
   ```

2. Using `PreparedStatement`

   ```
   // use `?` for placement holders, readStatement should be reused
   PreparedStatement readStatement = conn.prepareStatement("select * from tbl_point_query where k1 = ?");
   ...
   readStatement.setInt(1,1234);
   ResultSet resultSet = readStatement.executeQuery();
   ...
   readStatement.setInt(1,1235);
   resultSet = readStatement.executeQuery();
   ...
   ```

Enable row cache

Doris has a page-level cache that stores data for a specific column in each page. Therefore, the page cache is a column-based cache. For the row storage mentioned earlier, a row contains data for multiple columns, and the cache may be evicted by large queries, which can reduce the hit rate. To increase the hit rate of the row cache, a separate row cache is introduced, which reuses the LRU cache mechanism in Doris to ensure memory usage. You can enable it by specifying the following BE configuration:

- `disable_storage_row_cache` : Whether to enable the row cache. It is not enabled by default.
- `row_cache_mem_limit` : Specifies the percentage of memory occupied by the row cache. The default is 20% of memory.

Performance Optimization

1. Generally, it is effective to improve query processing capabilities by increasing the number of Observers.
2. Query load balancing: During the enumeration, if it is found that the FE CPU that accepts enumeration requests is used too high, or the request response becomes slow, you can use jdbc load balance for load balancing, and distribute the requests to multiple nodes to share the pressure (and also You can use other methods for query load balancing configuration, such as Nginx, proxySQL)
3. By directing the query requests to the Observer role to share the request pressure of high-concurrency queries and reducing the number of query requests sent to the fe master, it can usually solve the problem of the time-consuming fluctuation of the Fe Master node query to obtain better performance and stability

FAQ

1. How to confirm that the configuration is correct and short path optimization using concurrent enumeration is used ?

A: explain sql, when SHORT-CIRCUIT appears in the execution plan, it proves that short path optimization is used

```
mysql> explain select * from tbl_point_query where k1 = -2147481418 ;
   +-----------------------------------------------------------------------------------------------+
   | Explain String(Old Planner)                                                                   |
   +-----------------------------------------------------------------------------------------------+
   | PLAN FRAGMENT 0                                                                               |
   |   OUTPUT EXPRS:                                                                               |
   |     `test`.`tbl_point_query`.`k1`                                                            |
   |     `test`.`tbl_point_query`.`v1`                                                             |
   |     `test`.`tbl_point_query`.`v2`                                                             |
   |     `test`.`tbl_point_query`.`v3`                                                             |
   |     `test`.`tbl_point_query`.`v4`                                                             |
   |     `test`.`tbl_point_query`.`v5`                                                             |
   |     `test`.`tbl_point_query`.`v6`                                                             |
   |     `test`.`tbl_point_query`.`v7`                                                             |
   |   PARTITION: UNPARTITIONED                                                                    |
   |                                                                                               |
   |   HAS_COLO_PLAN_NODE: false                                                                   |
   |                                                                                               |
```

```
|   VRESULT SINK                                                                                 |
|      MYSQL_PROTOCAL                                                                            |
|                                                                                                |
|   0:VOlapScanNode                                                                              |
|      TABLE: test.tbl_point_query(tbl_point_query), PREAGGREGATION: ON                          |
|      PREDICATES: `k1` = -2147481418 AND `test`.`tbl_point_query`.`__DORIS_DELETE_SIGN__` = 0 |
|      partitions=1/1 (tbl_point_query), tablets=1/1, tabletList=360065                          |
|      cardinality=9452868, avgRowSize=833.31323, numNodes=1                                     |
|      pushAggOp=NONE                                                                            |
|      SHORT-CIRCUIT                                                                             |
+------------------------------------------------------------------------------------------------+
```

2. How to confirm that prepared statement is effective ?

A: After sending the request to Doris, find the corresponding query request in fe.audit.log and find Stmt=EXECUTE(), indicating that prepared statement is effective

```
2024-01-02 11:15:51,248 [query]
|Client=192.168.1.82:53450|User=root|Db=test|State=EOF|ErrorCode=0|ErrorMessage=|Time(ms)=49|ScanBytes=0|ScanRows=0|ReturnRows=1|Stm
tId=51|QueryId=b63d30b908f04dad-ab4a

3ba21d2c776b|IsQuery=true|isNereids=false|feIp=10.16.10.6|Stmt=EXECUTE(-2147481418)|CpuTimeMS=0|SqlHash=eee20fa2ac13a4f93bd4503a8792
1024|peakMemoryBytes=0|SqlDigest=|TraceId=|WorkloadGroup=|FuzzyVaria
   bles=
```

3. Can non-primary key queries use special optimization of high-concurrency point lookups?

A: No, high-concurrency query only targets the equivalent query of the key column, and the query cannot contain join or nested subqueries.

4. Is useServerPrepStmts useful in ordinary queries?

A: Prepared Statement currently only takes effect when primary key is checked.

5. Does optimizer selection require global settings?

A: When using prepared statement for query, Doris will choose the query method with the best performance, and there is no need to manually set the optimizer.

6. What should we do when the FE becomes a bottleneck?

A: If the FE is consuming too much CPU (i.e., high %CPU usage), enable the following configuration in the JDBC URL:

```
jdbc:mysql:loadbalance://[host1][:port],[host2][:port][,[host3][:port]]/${tbl_name}?
useServerPrepStmts=true&cachePrepStmts=true&prepStmtCacheSize=500&prepStmtCacheSqlLimit=1024
```

- Enable loadbalance to ensure multiple FEs can serve requests, and the more FE instances, the better (deploy one per instance).
- Enable useServerPrepStmts to reduce parsing and planning overhead on the FE.

- Enable cachePrepStmts so the client caches prepared statements, reducing the need to frequently send prepare requests to the FE.
- Adjust prepStmtCacheSize to set the maximum number of cached query templates.
- Adjust prepStmtCacheSqlLimit to set the maximum length of a single cached SQL template.

7. How to optimize query performance under a compute-storage separation architecture?

A:

- `set global enable_snapshot_point_query = false`. Point queries require an additional RPC to the meta service to obtain the version, which can easily become a bottleneck under high QPS. Setting it to false can speed up queries but reduces data visibility (requires a trade-off between performance and consistency).
- Configure the BE parameter enable_file_cache_keep_base_compaction_output=1 so that the result data after base compaction is stored in the cache, avoiding query jitter caused by remote access.

BITMAP Precise Deduplication

BITMAP Precise Deduplication

This document explains how to achieve precise deduplication using the Bitmap type.

Bitmap is an efficient bitmap indexing technique that uses bits to indicate the presence of corresponding data. It is particularly suitable for scenarios requiring efficient set operations (e.g., union, intersection) and is highly memory-efficient. Using Bitmap for precise deduplication offers the following benefits over `COUNT DISTINCT`:

- Improved query speed.
- Reduced memory/disk usage.

Implementation of Count Distinct

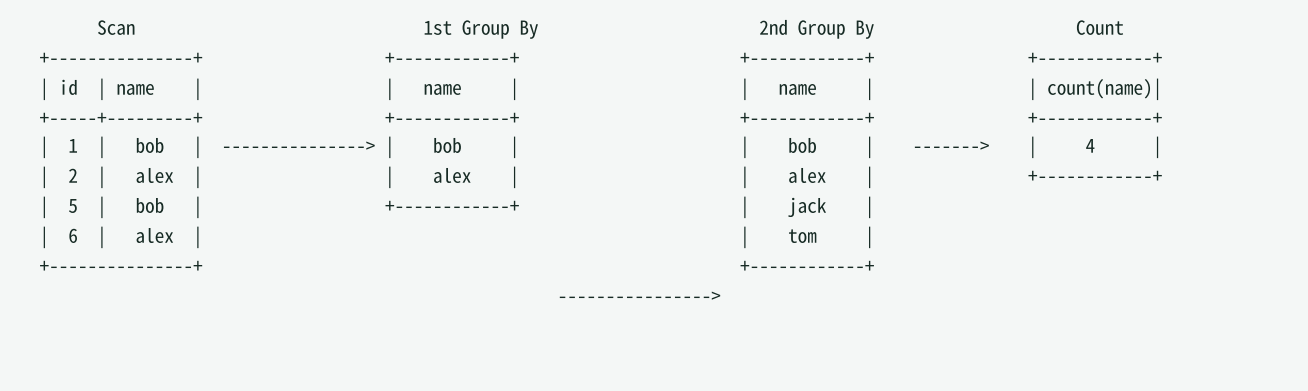
Traditional precise deduplication relies on `COUNT DISTINCT`. Consider the following example where deduplication is performed on the `name` column:

id	name
1	bob
2	alex
3	jack
4	tom
5	bob
6	alex

When Doris executes the query `SELECT COUNT(DISTINCT name) FROM t`, the process involves:

1. Grouping by the `name` column for stage-one deduplication.
2. Shuffling the grouped data.
3. Performing stage-two deduplication and finally counting the distinct names.

The process can be visualized as follows:



```
  +---------------+                  +------------+
  | id  | name    |                  |   name     |
  +-----+---------+  ---------------> +------------+
  |  3  |  jack   |                  |    jack    |
  |  4  |   tom   |                  |    tom     |
  +-----+---------+                  +------------+
```

Since `COUNT DISTINCT` requires storing detailed data and performing shuffling, query performance slows down as the dataset grows. Using Bitmap for precise deduplication addresses the performance issues of `COUNT DISTINCT` in large datasets.

Use Cases

In large-scale data scenarios, the cost of deduplication using `COUNT DISTINCT` increases significantly, resulting in slower queries. Bitmap-based precise deduplication addresses these performance bottlenecks by mapping detailed data to bits. While sacrificing the flexibility of raw data, Bitmap greatly enhances computational efficiency. Consider using Bitmap in the following scenarios:

- **Query Acceleration**: Bitmap utilizes bitwise operations for computation, offering excellent performance.
- **Storage Compression**: Bitmap compresses detailed data into bits, significantly reducing resource consumption on disk and in memory.

However, Bitmap can only perform precise deduplication on data types such as `TINYINT`, `SMALLINT`, `INT`, and `BIGINT`. For other data types, a global dictionary must be constructed. Doris implements precise deduplication using `RoaringBitmap`. For more details, refer to RoaringBitmap.

Use BITMAP for Precise Deduplication

Table Creation

1. When using Bitmap for deduplication, set the target column type to `BITMAP` in the table creation statement and specify `BITMAP_UNION` as the aggregate function.
2. Columns of type Bitmap cannot be used as key columns.

Create an aggregate table `test_bitmap`. The `id` column represents the user ID, and the `uv` column is of type `BITMAP`, using the aggregate function `BITMAP_UNION`:

```
CREATE TABLE test_bitmap(
        dt DATE,
        id INT,
        name CHAR(10),
        province CHAR(10),
        os CHAR(10),
        uv BITMAP BITMAP_UNION
)
AGGREGATE KEY (dt, id, name, province, os)
DISTRIBUTED BY HASH(id) BUCKETS 10;
```

Data Import

Here is a sample dataset (`test_bitmap.csv`) that can be imported using Stream Load:

```
2022-05-05,10001,Test 01,Beijing,windows
2022-05-05,10002,Test 01,Beijing,linux
2022-05-05,10003,Test 01,Beijing,macos
2022-05-05,10004,Test 01,Hebei,windows
2022-05-06,10001,Test 01,Shanghai,windows
2022-05-06,10002,Test 01,Shanghai,linux
2022-05-06,10003,Test 01,Jiangsu,macos
2022-05-06,10004,Test 01,Shaanxi,windows
```

Stream Load Command:

```
curl --location-trusted -u root: -H "label:label_test_bitmap_load" \
    -H "column_separator:," \
    -H "columns:dt,id,name,province,os, uv=to_bitmap(id)" -T test_bitmap.csv http://fe_IP:8030/api/demo/test_bitmap/_stream_load
```

Querying Data

Bitmap columns cannot directly return raw values. Instead, use the `BITMAP_UNION_COUNT` aggregate function for queries.

Total UV Calculation:

```
SELECT BITMAP_UNION_COUNT(uv) FROM test_bitmap;
+---------------------+
| BITMAP_UNION_COUNT(`uv`) |
+---------------------+
|                   4 |
+---------------------+
```

Equivalent to:

```
SELECT COUNT(DISTINCT id) FROM test_bitmap;
+----------------------+
| COUNT(DISTINCT `id`) |
+----------------------+
|                    4 |
+----------------------+
```

Daily UV Calculation:

```
SELECT dt, BITMAP_UNION_COUNT(uv) FROM test_bitmap GROUP BY dt;
+------------+---------------------+
| dt         | BITMAP_UNION_COUNT |
+------------+---------------------+
| 2022-05-05 |                   4 |
| 2022-05-06 |                   4 |
+------------+---------------------+
```

HLL Approximate Deduplication

HLL Approximate Deduplication

Use Cases

In real-world business scenarios, as the volume of business data grows, the pressure of deduplication also increases. When the data reaches a certain scale, the cost of precise deduplication becomes increasingly high. **HLL** (HyperLogLog) stands out for its excellent space complexity of O(m·loglogn) time complexity of O(n), and a controlled error rate of 1%–2%, depending on the dataset size and the hash function used.

When acceptable to the business, using approximate algorithms for fast deduplication is an effective way to reduce computational pressure.

What is HyperLogLog

HyperLogLog (HLL) is an enhanced version of the LogLog algorithm. It is used for approximate distinct counting and is mathematically based on **Bernoulli trials**.

Explanation:

Imagine flipping a coin with heads and tails. Each flip has a 50% probability of landing on either side. Keep flipping the coin until it shows heads, and record the number of flips as one trial.

For multiple Bernoulli trials:

- Let n be the number of heads obtained after n trials.
- Let k be the number of flips required in each trial. For example, if it took 12 flips to get heads in a trial, k_max would be 12 for this set of trials.

Bernoulli trials yield the following conclusions:

1. In n trials, no trial will require more than k_max flips.
2. At least one trial will require exactly k_max flips.

By applying maximum likelihood estimation, it can be derived that:

n = 2 ^ k_max

Thus, by recording only k_max, the total number of unique items (cardinality) can be estimated.

Use HLL for Approximate Deduplication

Creating a Table

1. When using HLL for deduplication:

    - The target column type must be set to `HLL`.
    - The aggregation function must be set to `HLL_UNION`.
2. HLL-type columns cannot be used as key columns.
3. Users do not need to specify the length or default value. The system internally manages the length based on data aggregation levels.

Example table creation:

```
CREATE TABLE test_hll(
        dt DATE,
        id INT,
        name CHAR(10),
        province CHAR(10),
        os CHAR(10),
        uv HLL HLL_UNION
)
AGGREGATE KEY (dt, id, name, province, os)
DISTRIBUTED BY HASH(id) BUCKETS 10
PROPERTIES(
        "replication_num" = "1",
        "in_memory"="false"
);
```

Importing Data

Here is sample data (`test_hll.csv`) that can be imported using Stream Load:

```
2022-05-05,10001,Test 01,Beijing,windows
2022-05-05,10002,Test 01,Beijing,linux
2022-05-05,10003,Test 01,Beijing,macos
2022-05-05,10004,Test 01,Hebei,windows
2022-05-06,10001,Test 01,Shanghai,windows
2022-05-06,10002,Test 01,Shanghai,linux
2022-05-06,10003,Test 01,Jiangsu,macos
2022-05-06,10004,Test 01,Shaanxi,windows
```

Stream Load Command:

```
curl --location-trusted -u root: -H "label:label_test_hll_load" \
    -H "column_separator:," \
    -H "columns:dt,id,name,province,os, uv=hll_hash(id)" -T test_hll.csv http://fe_IP:8030/api/demo/test_hll/_stream_load
```

Result:

```
{
    "TxnId": 693,
    "Label": "label_test_hll_load",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 8,
    "NumberLoadedRows": 8,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 320,
```

```
    "LoadTimeMs": 23,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 9,
    "CommitAndPublishTimeMs": 11
}
```

Querying Data

HLL columns cannot return raw values directly. Instead, HLL aggregate functions must be used for queries.

Total UV Calculation:

```
SELECT HLL_UNION_AGG(uv) FROM test_hll;
+----------------------+
| hll_union_agg(`uv`) |
+----------------------+
|                    4 |
+----------------------+
```

Equivalent to:

```
SELECT COUNT(DISTINCT id) FROM test_hll;
+-----------------------+
| count(DISTINCT `id`) |
+-----------------------+
|                     4 |
+-----------------------+
```

Daily UV Calculation:

```
SELECT dt, HLL_UNION_AGG(uv) FROM test_hll GROUP BY dt;
+------------+----------------------+
| dt         | hll_union_agg       |
+------------+----------------------+
| 2022-05-05 |                    4 |
| 2022-05-06 |                    4 |
+------------+----------------------+
```

Related Functions

- **HLL_UNION_AGG(hll)**: An aggregate function to estimate the cardinality of all data meeting the conditions.
- **HLL_CARDINALITY(hll)**: A function to calculate the cardinality of a single HLL column.
- **HLL_HASH(column_name)**: Generates an HLL column type, used during insert or data import (as shown above).
- **HLL_EMPTY()**: Generates an empty HLL column for default values during `INSERT` or data import.

Colocation Join

Colocation Join

Colocation Join provides local optimization for some Join queries to reduce data transmission time between nodes and accelerating query execution.

Note: This property will not be synchronized by CCR. If this table is copied by CCR, that is, PROPERTIES contains `is_being_synced = true`, this property will be erased in this table.

Noun Interpretation

- FE: Frontend, the front-end node of Doris. Responsible for metadata management and request access.
- BE: Backend, Doris's back-end node. Responsible for query execution and data storage.
- Colocation Group (CG): A CG contains one or more tables. Tables within the same group have the same Colocation Group Schema and the same data fragmentation distribution.
- Colocation Group Schema (CGS): Used to describe tables in a CG and general schema information related to Colocation. Including bucket column type, bucket number and copy number.

Principle

The Colocation Join function is to make a CG of a set of tables with the same CGS. Ensure that the corresponding data fragments of these tables will fall on the same BE node. When tables in CG perform Join operations on bucket columns, local data Join can be directly performed to reduce data transmission time between nodes.

The data of a table will eventually fall into a barrel according to the barrel column value Hash and the number of barrels modeled. Assuming that the number of buckets in a table is 8, there are eight buckets `[0, 1, 2, 3, 4, 5, 6, 7]`Buckets'. We call such a sequence a `Buckets Sequence`. Each Bucket has one or more Tablets. When a table is a single partitioned table, there is only one Tablet in a Bucket. If it is a multi-partition table, there will be more than one.

In order for a table to have the same data distribution, the table in the same CG must ensure the following attributes are the same:

1. Bucket column and number of buckets

Bucket column, that is, the column specified in `DISTRIBUTED BY HASH (col1, col2,...)` in the table building statement. Bucket columns determine which column values are used to Hash data from a table into different Tablets. Tables in the same CG must ensure that the type and number of barrel columns are identical, and the number of barrels is identical, so that the data fragmentation of multiple tables can be controlled one by one.

2. Number of copies

The number of copies of all partitions of all tables in the same CG must be the same. If inconsistent, there may be a copy of a Tablet, and there is no corresponding copy of other table fragments on the same BE.

Tables in the same CG do not require consistency in the number, scope, and type of partition columns.

After fixing the number of bucket columns and buckets, the tables in the same CG will have the same Buckets Sequence. The number of replicas determines the number of replicas of Tablets in each bucket, which BE they are stored on. Suppose that Buckets Sequence is `[0, 1, 2, 3, 4, 5, 6, 7]`, and that BE nodes have `[A, B, C, D]`4. A possible distribution of data is as follows:

```
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 |
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| A | | B | | C | | D | | A | | B | | C | | D |
|   | |   | |   | |   | |   | |   | |   | |   |
| B | | C | | D | | A | | B | | C | | D | | A |
|   | |   | |   | |   | |   | |   | |   | |   |
| C | | D | | A | | B | | C | | D | | A | | B |
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
```

The data of all tables in CG will be uniformly distributed according to the above rules, which ensures that the data with the same barrel column value are on the same BE node, and local data Join can be carried out.

Usage

Establishment of tables

When creating a table, you can specify the attribute `"colocate_with"="group_name"` in `PROPERTIES`, which means that the table is a Colocation Join table and belongs to a specified Colocation Group.

Examples:

```
CREATE TABLE tbl (k1 int, v1 int sum)
DISTRIBUTED BY HASH(k1)
BUCKETS 8
PROPERTIES(
  "colocate_with" = "group1"
);
```

If the specified group does not exist, Doris automatically creates a group that contains only the current table. If the Group already exists, Doris checks whether the current table satisfies the Colocation Group Schema. If satisfied, the table is created and added to the Group. At the same time, tables create fragments and replicas based on existing data distribution rules in Groups. Group belongs to a database, and its name is unique in a database. Internal storage is the full name of Group `dbId_groupName`, but users only perceive groupName.

In version 2.0, Doris supports cross-Database Group. When creating a table, you need to use the keyword `__global__` as a prefix of the Group name. like:

```
CREATE TABLE tbl (k1 int, v1 int sum)
DISTRIBUTED BY HASH(k1)
BUCKETS 8
PROPERTIES(
     "colocate_with" = "__global__group1"
);
```

The Group prefixed with `__global__` no longer belongs to a Database, and its name is also globally unique.

Cross-Database Colocate Join can be realized by creating a Global Group.

Delete table

When the last table in Group is deleted completely (deleting completely means deleting from the recycle bin). Usually, when a table is deleted by the `DROP TABLE` command, it will be deleted after the default one-day stay in the recycle bin, and the group will be deleted automatically.

View Group

The following command allows you to view the existing Group information in the cluster.

```
SHOW PROC '/colocation_group';

+-------------+--------------+--------------+------------+----------------+----------+----------+
| GroupId     | GroupName    | TableIds     | BucketsNum | ReplicationNum | DistCols | IsStable |
+-------------+--------------+--------------+------------+----------------+----------+----------+
| 10005.10008 | 10005_group1 | 10007, 10040 | 10         | 3              | int(11)  | true     |
+-------------+--------------+--------------+------------+----------------+----------+----------+
```

- GroupId: The unique identity of a group's entire cluster, with DB ID in the first half and group ID in the second half.
- GroupName: The full name of Group.
- TabletIds: The group contains a list of Tables'ID.
- Buckets Num: Number of barrels.
- Replication Num: Number of copies.
- DistCols: Distribution columns,
- IsStable: Is the group stable (for the definition of stability, see section `Collocation replica balancing and repair').

You can further view the data distribution of a group by following commands:

```
SHOW PROC '/colocation_group/10005.10008';

+-------------+---------------------+
| BucketIndex | BackendIds          |
+-------------+---------------------+
| 0           | 10004, 10002, 10001 |
| 1           | 10003, 10002, 10004 |
| 2           | 10002, 10004, 10001 |
| 3           | 10003, 10002, 10004 |
| 4           | 10002, 10004, 10003 |
| 5           | 10003, 10002, 10001 |
| 6           | 10003, 10004, 10001 |
| 7           | 10003, 10004, 10002 |
+-------------+---------------------+
```

- BucketIndex: Subscript to the bucket sequence.
- Backend Ids: A list of BE node IDs where data fragments are located in buckets.

> The above commands require ADMIN privileges. Normal user view is not supported at this time.

Modify Colocate Group

You can modify the Colocation Group property of a table that has been created. Examples:

```
ALTER TABLE tbl SET ("colocate_with" = "group2");
```

- If the table has not previously specified a Group, the command checks the Schema and adds the table to the Group (if the Group does not exist, it will be created).
- If other groups are specified before the table, the command first removes the table from the original group and adds a new group (if the group does not exist, it will be created).

You can also delete the Colocation attribute of a table by following commands:

```
ALTER TABLE tbl SET ("colocate_with" = "");
```

Other related operations

When an ADD PARTITION is added to a table with a Colocation attribute and the number of copies is modified, Doris checks whether the modification violates the Colocation Group Schema and rejects it if it does.

Colocation Duplicate Balancing and Repair

Copy distribution of Colocation tables needs to follow the distribution specified in Group, so it is different from common fragmentation in replica repair and balancing.

Group itself has a Stable attribute, when Stable is true, which indicates that all fragments of the table in the current Group are not changing, and the Colocation feature can be used normally. When Stable is false, it indicates that some tables in Group are being repaired or migrated. At this time, Colocation Join of related tables will degenerate into ordinary Join.

Replica Repair

Copies can only be stored on specified BE nodes. So when a BE is unavailable (downtime, Decommission, etc.), a new BE is needed to replace it. Doris will first look for the BE with the lowest load to replace it. After replacement, all data fragments on the old BE in the Bucket will be repaired. During the migration process, Group is marked Unstable.

Replica Balancing

Doris will try to distribute the fragments of the Collocation table evenly across all BE nodes. For the replica balancing of common tables, the granularity is single replica, that is to say, it is enough to find BE nodes with lower load for each replica alone. The equilibrium of the Colocation table is at the Bucket level, where all replicas within a Bucket migrate together. We adopt a simple equalization algorithm, which distributes Buckets Sequence evenly on all BEs, regardless of the actual size of the replicas, but only according to the number of replicas. Specific algorithms can be referred to the code annotations in `ColocateTableBalancer.java`.

> Note 1: Current Colocation replica balancing and repair algorithms may not work well for heterogeneously deployed Doris clusters. The so-called heterogeneous deployment, that is, the BE node's disk capacity, number, disk type (SSD and HDD) is inconsistent. In the case of heterogeneous deployment, small BE nodes and large BE nodes may store the same number of replicas.
>
> Note 2: When a group is in an Unstable state, the Join of the table in it will degenerate into a normal Join. At this time, the query performance of the cluster may be greatly reduced. If you do not want the system to balance automatically, you can set the FE configuration item `disable_colocate_balance` to prohibit automatic balancing. Then open it at the right time. (See Section `Advanced Operations` for details)

Query

The Colocation table is queried in the same way as ordinary tables, and users do not need to perceive Colocation attributes. If the Group in which the Colocation table is located is in an Unstable state, it will automatically degenerate to a normal Join.

Examples are given to illustrate:

Table 1:

```
CREATE TABLE `tbl1` (
    `k1` date NOT NULL COMMENT "",
    `k2` int(11) NOT NULL COMMENT "",
    `v1` int(11) SUM NOT NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
PARTITION BY RANGE(`k1`)
(
    PARTITION p1 VALUES LESS THAN ('2019-05-31'),
    PARTITION p2 VALUES LESS THAN ('2019-06-30')
)
DISTRIBUTED BY HASH(`k2`) BUCKETS 8
PROPERTIES (
    "colocate_with" = "group1"
);
```

Table 2:

```
CREATE TABLE `tbl2` (
    `k1` datetime NOT NULL COMMENT "",
    `k2` int(11) NOT NULL COMMENT "",
    `v1` double SUM NOT NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
DISTRIBUTED BY HASH(`k2`) BUCKETS 8
PROPERTIES (
    "colocate_with" = "group1"
);
```

View the query plan:

```
DESC SELECT * FROM tbl1 INNER JOIN tbl2 ON (tbl1.k2 = tbl2.k2);

+-----------------------------------------------------+
| Explain String                                      |
+-----------------------------------------------------+
| PLAN FRAGMENT 0                                     |
|  OUTPUT EXPRS:`tbl1`.`k1` |                         |
|   PARTITION: RANDOM                                 |
|                                                     |
|   RESULT SINK                                       |
|                                                     |
|   2:HASH JOIN                                       |
|   |  join op: INNER JOIN                            |
|   |  hash predicates:                               |
|   |  colocate: true                                 |
|   |    `tbl1`.`k2` = `tbl2`.`k2`                    |
|   |  tuple ids: 0 1                                 |
|   |                                                 |
|   |----1:OlapScanNode                               |
|   |       TABLE: tbl2                               |
|   |       PREAGGREGATION: OFF. Reason: null         |
|   |       partitions=0/1                            |
|   |       rollup: null                              |
|   |       buckets=0/0                               |
|   |       cardinality=-1                            |
|   |       avgRowSize=0.0                            |
|   |       numNodes=0                                |
|   |       tuple ids: 1                              |
|   |                                                 |
|   0:OlapScanNode                                    |
|      TABLE: tbl1                                    |
|      PREAGGREGATION: OFF. Reason: No AggregateInfo  |
|      partitions=0/2                                 |
|      rollup: null                                   |
|      buckets=0/0                                    |
|      cardinality=-1                                 |
|      avgRowSize=0.0                                 |
|      numNodes=0                                     |
|      tuple ids: 0                                   |
+-----------------------------------------------------+
```

If Colocation Join works, the Hash Join Node will show `colocate: true`.

If not, the query plan is as follows:

```
+-----------------------------------------------------+
| Explain String                                      |
+-----------------------------------------------------+
| PLAN FRAGMENT 0                                     |
|  OUTPUT EXPRS:`tbl1`.`k1` |                         |
|   PARTITION: RANDOM                                 |
|                                                     |
|   RESULT SINK                                       |
|                                                     |
|   2:HASH JOIN                                       |
|   |  join op: INNER JOIN (BROADCAST)                |
|   |  hash predicates:                               |
|   |  colocate: false, reason: group is not stable   |
|   |    `tbl1`.`k2` = `tbl2`.`k2`                    |
|   |  tuple ids: 0 1                                 |
|   |                                                 |
|   |----3:EXCHANGE                                   |
```

```
|   |      tuple ids: 1                                |
|   |                                                  |
|   0:OlapScanNode                                     |
|      TABLE: tbl1                                     |
|      PREAGGREGATION: OFF. Reason: No AggregateInfo |
|      partitions=0/2                                  |
|      rollup: null                                    |
|      buckets=0/0                                     |
|      cardinality=-1                                  |
|      avgRowSize=0.0                                  |
|      numNodes=0                                      |
|      tuple ids: 0                                    |
|                                                      |
| PLAN FRAGMENT 1                                      |
|  OUTPUT EXPRS:                                       |
|   PARTITION: RANDOM                                  |
|                                                      |
|   STREAM DATA SINK                                   |
|     EXCHANGE ID: 03                                  |
|     UNPARTITIONED                                    |
|                                                      |
|   1:OlapScanNode                                     |
|      TABLE: tbl2                                     |
|      PREAGGREGATION: OFF. Reason: null               |
|      partitions=0/1                                  |
|      rollup: null                                    |
|      buckets=0/0                                     |
|      cardinality=-1                                  |
|      avgRowSize=0.0                                  |
|      numNodes=0                                      |
|      tuple ids: 1                                    |
+------------------------------------------------------+
```

The HASH JOIN node displays the corresponding reason: `colocate: false, reason: group is not stable`. At the same time, an EXCHANGE node will be generated.

Advanced Operations

FE Configuration Item

- disable_colocate_relocate

Whether to close Doris's automatic Colocation replica repair. The default is false, i.e. not closed. This parameter only affects the replica repair of the Colocation table, but does not affect the normal table.

- disable_colocate_balance

Whether to turn off automatic Colocation replica balancing for Doris. The default is false, i.e. not closed. This parameter only affects the replica balance of the Collocation table, but does not affect the common table.

User can set these configurations at runtime. See `HELP ADMIN SHOW CONFIG;` and `HELP ADMIN SET CONFIG;`.

- disable_colocate_join

Whether to turn off the Colocation Join function or not. In 0.10 and previous versions, the default is true, that is, closed. In a later version, it will default to false, that is, open.

- use_new_tablet_scheduler

In 0.10 and previous versions, the new replica scheduling logic is incompatible with the Colocation Join function, so in 0.10 and previous versions, if `disable_colocate_join = false`, you need to set `use_new_tablet_scheduler = false`, that is, close the new replica scheduler. In later versions, `use_new_tablet_scheduler` will be equal to true.

HTTP RESTful API

Doris provides several HTTP RESTful APIs related to Colocation Join for viewing and modifying Colocation Group.

The API is implemented on the FE side and accessed using `fe_host: fe_http_port`. ADMIN privileges are required.

1. View all Colocation information for the cluster

```
GET /api/colocate

Return the internal Colocation info in JSON format:

{
    "msg": "success",
  "code": 0,
  "data": {
    "infos": [
      ["10003.12002", "10003_group1", "10037, 10043", "1", "1", "int(11)", "true"]
    ],
    "unstableGroupIds": [],
    "allGroupIds": [{
      "dbId": 10003,
      "grpId": 12002
    }]
  },
  "count": 0
}
```

2. Mark Group as Stable or Unstable

- Mark as Stable

```
```
DELETE /api/colocate/group_stable?db_id=10005&group_id=10008

Returns: 200
```
```

- Mark as Unstable

```
```
POST /api/colocate/group_stable?db_id=10005&group_id=10008

Returns: 200
```
```

3. Setting Data Distribution for Group

The interface can force the bucket sequence distribution of a group.

```
POST /api/colocate/bucketseq?db_id=10005&group_id=10008

Body:
[[10004,10002],[10003,10002],[10002,10004],[10003,10002],[10002,10004],[10003,10002],[10003,10004],[10003,10004],[10003,10004],
[10002,10004]]

Returns: 200
```

Body is a Buckets Sequence represented by a nested array and the ID of the BE where the fragments are distributed in each Bucket.

Note that using this command, you may need to set the FE configuration `disable_colocate_relocate` and `disable_colocate_balance` to true. “This will prevent the system from automatically repairing or balancing Colocation replicas. Otherwise, it may be automatically reset by the system after modification.

Overview of Hints

Database Hints are query optimization techniques used to guide the database query optimizer on how to generate a specific plan. By providing Hints, users can fine-tune the default behavior of the query optimizer in hopes of achieving better performance or meeting specific requirements.

> **Note**
>
> Currently, Doris possesses excellent out-of-the-box capabilities. In most scenarios, Doris adaptively optimizes performance across various situations without requiring users to manually control hints for business tuning. The content presented in this chapter is primarily intended for professional tuning personnel. Business users can have a brief understanding of it.

Hint Classification

Doris currently supports several types of hints, including leading hint, ordered hint, and distribute hint:

- Leading Hint：Specifies the join order according to the order provided in the leading hint.
- Ordered Hint：A specific type of leading hint that specifies the join order as the original text sequence.
- Distribute Hint：Specifies the data distribution method for joins as either shuffle or broadcast.

Hint Example

Imagine a table with a large amount of data. In certain specific cases, you may know that the join order of the tables can affect query performance. In such situations, the Leading Hint allows you to specify the table join order you want the optimizer to follow.

Take the following SQL query as an example. If the execution efficiency is not ideal, you may want to adjust the join order without changing the original SQL to avoid impacting the user's original scenario and achieve tuning goals.

```
mysql> explain shape plan select * from t1 join t2 on t1.c1 = c2;
+----------------------------------------+
| Explain String                         |
+----------------------------------------+
| PhysicalResultSink                     |
| --PhysicalDistribute                   |
| ----PhysicalProject                    |
| ------hashJoin[INNER_JOIN](t1.c1 = t2.c2) |
| --------PhysicalOlapScan[t2]           |
| --------PhysicalDistribute             |
| ----------PhysicalOlapScan[t1]         |
+----------------------------------------+
```

In this case, we can use the Leading Hint to arbitrarily change the join order of t1 and t2. For example:

```
mysql> explain shape plan select  /*+ leading(t2 t1) */ * from t1 join t2 on t1.c1 = c2;
+-----------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                     |
+-----------------------------------------------------------------------------------------------------+
| PhysicalResultSink                                                                                  |
| --PhysicalDistribute                                                                                |
| ----PhysicalProject                                                                                 |
| ------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() build RFs:RF0 c1->[c2] |
| --------PhysicalOlapScan[t2] apply RFs: RF0                                                         |
| --------PhysicalDistribute                                                                          |
| ----------PhysicalOlapScan[t1]                                                                      |
|                                                                                                     |
| Hint log:                                                                                           |
| Used: leading(t2 t1)                                                                                |
| UnUsed:                                                                                             |
| SyntaxError:                                                                                        |
+-----------------------------------------------------------------------------------------------------+
```

In this example, the Leading Hint `/*+ leading(t2 t1) */` is used. The Leading Hint informs the optimizer to use the specified table (t2) as the driving table and place it before (t1) in the execution plan.

Hint Log

The Hint Log is primarily used to display whether the hint is effective when executing `EXPLAIN`. It is usually located at the bottom of the `EXPLAIN` output.

Hint Log has three statuses:

```
+---------------------------------+
| Hint log:                       |
| Used:                           |
| UnUsed:                         |
| SyntaxError:                    |
+---------------------------------+
```

- `Used`：Indicates that the hint is effective.
- `UnUsed` 和 `SyntaxError`：Both indicate that the hint is not effective. SyntaxError indicates that there is a syntax error in using the hint or the syntax is not supported, and additional information about the unsupported reason will be provided.

Users can view the effectiveness and reasons for non-effectiveness through the Hint log, facilitating adjustments and verification.

Summary

Hints are powerful tools for manually managing execution plans. Currently, Doris supports leading hint, ordered hint, distribute hint, etc., enabling users to manually manage join order, shuffle methods, and other variable configurations, providing users with more convenient and effective operational capabilities.

Leading Hint

Leading Hint is a powerful query optimization technique that allows users to guide the Doris optimizer to determine the table join order in the query plan. Correct use of Leading Hint can significantly improve the performance of complex queries. This article will describe in detail how to use Leading Hint to control the join order in Doris.

Regular Leading Hint

Syntax

Leading Hint allows specifying the table join order that the optimizer should follow. In Doris, the basic syntax of Leading Hint is as follows:

```
SELECT /*+ LEADING(tablespec [tablespec]...) */ ...
```

It should be noted that:

- Leading Hint is surrounded by `/*+` and `*/` and placed after the SELECT keyword in the SQL statement.
- `tablespec` is the table name or table alias, and at least two tables need to be specified.
- Multiple tables are separated by spaces or ‘,’.
- You can use curly braces `{}` to explicitly specify the shape of the Join Tree.

For example:

```
mysql> explain shape plan select /*+ leading(t2 t1) */ * from t1 join t2 on c1 = c2;
+------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                              |
+------------------------------------------------------------------------------+
| PhysicalResultSink                                                           |
| --PhysicalDistribute[DistributionSpecGather]                                 |
| ----PhysicalProject                                                          |
| ------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| --------PhysicalOlapScan[t2]                                                 |
| --------PhysicalDistribute[DistributionSpecHash]                             |
| ----------PhysicalOlapScan[t1]                                               |
|                                                                              |
| Hint log:                                                                    |
| Used: leading(t2 t1)                                                         |
| UnUsed:                                                                      |
| SyntaxError:                                                                 |
+------------------------------------------------------------------------------+
```

When Leading Hint is not effective, the normal process will be used to generate the plan. EXPLAIN will display whether the used Hint is effective, mainly divided into three types:

Status	Description
`Used`	Leading Hint is effective normally.

Status	Description
`Unused`	The unsupported cases here include that the join order specified by Leading Hint is not equivalent to the original SQL or the feature is not supported in this version (see limitations for details).
`SyntaxError`	Indicates a syntax error in Leading Hint, such as the inability to find the corresponding table.

1. The default syntax of Leading Hint constructs a left-deep tree: `sql mysql> explain shape plan select /*+ leading(t1 t2 t3) */ * from t1 join t2 on c1 = c2 join t3 on c2=c3; +---------- ---------- ----------+ | Explain String(Nereids Planner) | +---------- ---------- ----------+ | PhysicalResultSink | | -- PhysicalDistribute[DistributionSpecGather] | | ----PhysicalProject | | ------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=() | | --------hashJoin[INNER_JOIN] hashCondition= ((t1.c1 = t2.c2)) otherCondition=() | | ----------PhysicalOlapScan[t1] | | ---------- PhysicalDistribute[DistributionSpecHash] | | ------------PhysicalOlapScan[t2] | | -------- PhysicalDistribute[DistributionSpecHash] | | ----------PhysicalOlapScan[t3] | | | | Hint log: | | Used: leading(t1 t2 t3) | | UnUsed: | | SyntaxError: | +---------- ---------- ----------+`

2. At the same time, curly braces can be used to specify the shape of the Join tree: `sql mysql> explain shape plan select /*+ leading(t1 {t2 t3}) */ * from t1 join t2 on c1 = c2 join t3 on c2=c3; +--------- ---------- ----------+ | Explain String(Nereids Planner) | +---------- ---------- ----------+ | PhysicalResultSink | | --PhysicalDistribute[DistributionSpecGather] | | ----PhysicalProject | | ------ hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() | | -------- PhysicalOlapScan[t1] | | --------PhysicalDistribute[DistributionSpecHash] | | ---------- hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=() | | ------------ PhysicalOlapScan[t2] | | ------------PhysicalDistribute[DistributionSpecHash] | | -------------- PhysicalOlapScan[t3] | | | | Hint log: | | Used: leading(t1 { t2 t3 }) | | UnUsed: | | SyntaxError: | +---------- ---------- ----------+`

3. When a View is used as an alias to participate in JoinReorder, the corresponding View can be specified as a parameter of Leading Hint. For example:

```
mysql> explain shape plan select /*+ leading(alias t1) */ count(*) from t1 join (select c2 from t2 join t3 on t2.c2 =
t3.c3) as alias on t1.c1 = alias.c2;
  +---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------+
  | Explain String(Nereids Planner)                                                                              |
  +---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------+
  | PhysicalResultSink                                                                                           |
  | --hashAgg[GLOBAL]                                                                                            |
  | ----PhysicalDistribute[DistributionSpecGather]                                                               |
  | ------hashAgg[LOCAL]                                                                                         |
  | --------PhysicalProject                                                                                      |
  | ----------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = alias.c2)) otherCondition=()  |
  | ------------PhysicalProject                                                                                  |
  | --------------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=() |
  | ----------------PhysicalProject                                                                             |
  | ------------------PhysicalOlapScan[t2]                                                                      |
  | ----------------PhysicalDistribute[DistributionSpecHash]                                                    |
  | ------------------PhysicalProject                                                                           |
  | --------------------PhysicalOlapScan[t3]                                                                    |
  | ------------PhysicalDistribute[DistributionSpecHash]                                                        |
  | --------------PhysicalProject                                                                               |
```

```
  | ---------------PhysicalOlapScan[t1]                                                  |
  |                                                                                      |
  | Hint log:                                                                            |
  | Used: leading(alias t1)                                                              |
  | UnUsed:                                                                              |
  | SyntaxError:                                                                         |
  +--------------------------------------------------------------------------------------+
```

Case

Basic Scenario

1. The table creation statements are as follows:

```
CREATE DATABASE testleading;
USE testleading;

create table t1 (c1 int, c11 int) distributed by hash(c1) buckets 3 properties('replication_num' = '1');
create table t2 (c2 int, c22 int) distributed by hash(c2) buckets 3 properties('replication_num' = '1');
create table t3 (c3 int, c33 int) distributed by hash(c3) buckets 3 properties('replication_num' = '1');
create table t4 (c4 int, c44 int) distributed by hash(c4) buckets 3 properties('replication_num' = '1');
```

2. The original plan:

```
mysql> explain shape plan select * from t1 join t2 on t1.c1 = c2;
+-----------------------------------------+
| Explain String                          |
+-----------------------------------------+
| PhysicalResultSink                      |
| --PhysicalDistribute                    |
| ----PhysicalProject                     |
| ------hashJoin[INNER_JOIN](t1.c1 = t2.c2) |
| --------PhysicalOlapScan[t2]            |
| --------PhysicalDistribute              |
| ----------PhysicalOlapScan[t1]          |
+-----------------------------------------+
```

3. When we need to exchange the join order of t1 and t2, we only need to add `leading(t2 t1)` in front. When executing `explain`, it will show whether this hint is used. The following is the Leading plan: `Used` indicates that the Hint is effective normally.

```
mysql> explain shape plan select /*+ leading(t2 t1) */ * from t1 join t2 on c1 = c2;
+------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                              |
+------------------------------------------------------------------------------+
| PhysicalResultSink                                                           |
| --PhysicalDistribute[DistributionSpecGather]                                 |
| ----PhysicalProject                                                          |
| ------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| --------PhysicalOlapScan[t2]                                                 |
| --------PhysicalDistribute[DistributionSpecHash]                             |
| ----------PhysicalOlapScan[t1]                                               |
|                                                                              |
| Hint log:                                                                    |
| Used: leading(t2 t1)                                                         |
| UnUsed:                                                                      |
```

```
| SyntaxError:                                                                |
+-----------------------------------------------------------------------------+
```

4. If there is a syntax error in Leading Hint, when executing `explain`, the corresponding information will be displayed in `SyntaxError`, but the plan can still be generated normally, just without using Leading. For example:

```
mysql> explain shape plan select /*+ leading(t2 t3) */ * from t1 join t2 on t1.c1 = c2;
+----------------------------------------------------+
| Explain String                                     |
+----------------------------------------------------+
| PhysicalResultSink                                 |
| --PhysicalDistribute                               |
| ----PhysicalProject                                |
| ------hashJoin[INNER_JOIN](t1.c1 = t2.c2)          |
| --------PhysicalOlapScan[t1]                       |
| --------PhysicalDistribute                         |
| ----------PhysicalOlapScan[t2]                     |
|                                                    |
| Used:                                              |
| UnUsed:                                            |
| SyntaxError: leading(t2 t3) Msg:can not find table: t3 |
+----------------------------------------------------+
```

Extended Scenario

1. Left-Deep Tree

   As mentioned above, when the query statement in Doris does not use any parentheses, Leading will default to generating a left-deep tree.

```
mysql> explain shape plan select /*+ leading(t1 t2 t3) */ * from t1 join t2 on t1.c1 = c2 join t3 on c2 = c3;
+-------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                               |
+-------------------------------------------------------------------------------+
| PhysicalResultSink                                                            |
| --PhysicalDistribute[DistributionSpecGather]                                  |
| ----PhysicalProject                                                           |
| ------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=()  |
| --------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| ----------PhysicalOlapScan[t1]                                                |
| ----------PhysicalDistribute[DistributionSpecHash]                            |
| ------------PhysicalOlapScan[t2]                                              |
| --------PhysicalDistribute[DistributionSpecHash]                              |
| ----------PhysicalOlapScan[t3]                                                |
|                                                                               |
| Hint log:                                                                     |
| Used: leading(t1 t2 t3)                                                       |
| UnUsed:                                                                       |
| SyntaxError:                                                                  |
+-------------------------------------------------------------------------------+
```

2. Right-Deep Tree

   When you need to make the shape of the plan a right-deep tree, Bushy tree, or zig-zag tree, you only need to add curly braces to limit the shape of the plan, without the need to use swap like Oracle to adjust step by step from the left-deep tree.

```
mysql> explain shape plan select /*+ leading(t1 {t2 t3}) */ * from t1 join t2 on t1.c1 = c2 join t3 on c2 = c3;
+------------------------------------------------+
| Explain String                                 |
+------------------------------------------------+
| PhysicalResultSink                             |
| --PhysicalDistribute                           |
| ----PhysicalProject                            |
| ------hashJoin[INNER_JOIN](t1.c1 = t2.c2)      |
| --------PhysicalOlapScan[t1]                   |
| --------PhysicalDistribute                     |
| ----------hashJoin[INNER_JOIN](t2.c2 = t3.c3)  |
| ------------PhysicalOlapScan[t2]               |
| ------------PhysicalDistribute                 |
| --------------PhysicalOlapScan[t3]             |
|                                                |
| Used: leading(t1 { t2 t3 })                    |
| UnUsed:                                        |
| SyntaxError:                                   |
+------------------------------------------------+
```

3. Bushy Tree

```
mysql> explain shape plan select /*+ leading({t1 t2} {t3 t4}) */ * from t1 join t2 on t1.c1 = c2 join t3 on c2 = c3 join t4
on c3 = c4;
+------------------------------------------------+
| Explain String                                 |
+------------------------------------------------+
| PhysicalResultSink                             |
| --PhysicalDistribute                           |
| ----PhysicalProject                            |
| ------hashJoin[INNER_JOIN](t2.c2 = t3.c3)      |
| --------hashJoin[INNER_JOIN](t1.c1 = t2.c2)    |
| ----------PhysicalOlapScan[t1]                 |
| ----------PhysicalDistribute                   |
| ------------PhysicalOlapScan[t2]               |
| --------PhysicalDistribute                     |
| ----------hashJoin[INNER_JOIN](t3.c3 = t4.c4)  |
| ------------PhysicalOlapScan[t3]               |
| ------------PhysicalDistribute                 |
| --------------PhysicalOlapScan[t4]             |
|                                                |
| Used: leading({ t1 t2 } { t3 t4 })             |
| UnUsed:                                        |
| SyntaxError:                                   |
+------------------------------------------------+
```

4. zig-zag tree

```
mysql> explain shape plan select /*+ leading(t1 {t2 t3} t4) */ * from t1 join t2 on t1.c1 = c2 join t3 on c2 = c3 join t4
on c3 = c4;
+-----------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                         |
+-----------------------------------------------------------------------------------------+
| PhysicalResultSink                                                                      |
| --PhysicalDistribute[DistributionSpecGather]                                            |
| ----PhysicalProject                                                                     |
| ------hashJoin[INNER_JOIN] hashCondition=((t3.c3 = t4.c4)) otherCondition=()            |
| --------PhysicalDistribute[DistributionSpecHash]                                       |
| ----------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=()        |
| ------------PhysicalOlapScan[t1]                                                        |
| ------------PhysicalDistribute[DistributionSpecHash]                                   |
```

```
| --------------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=() |
| ----------------PhysicalOlapScan[t2]                                                |
| ----------------PhysicalDistribute[DistributionSpecHash]                           |
| ------------------PhysicalOlapScan[t3]                                             |
| --------PhysicalDistribute[DistributionSpecHash]                                   |
| ----------PhysicalOlapScan[t4]                                                     |
|                                                                                     |
| Hint log:                                                                           |
| Used: leading(t1 { t2 t3 } t4)                                                      |
| UnUsed:                                                                             |
| SyntaxError:                                                                        |
+-------------------------------------------------------------------------------------+
```

5. Non-inner Join

   When encountering a non-inner join (such as Outer Join or Semi/Anti Join), Leading Hint will automatically derive the join method of each join according to the original SQL semantics. If Leading Hint is different from the original SQL semantics or cannot be generated, it will be placed in UnUsed, but this does not affect the normal generation of the plan. The following is an example that cannot be swapped:

```
-------- test outer join which can not swap
-- t1 leftjoin (t2 join t3 on (P23)) on (P12) != (t1 leftjoin t2 on (P12)) join t3 on (P23)
mysql> explain shape plan select /*+ leading(t1 {t2 t3}) */ * from t1 left join t2 on c1 = c2 join t3 on c2 = c3;
+-------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                               |
+-------------------------------------------------------------------------------+
| PhysicalResultSink                                                            |
| --PhysicalDistribute[DistributionSpecGather]                                  |
| ----PhysicalProject                                                           |
| ------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=()   |
| --------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| ----------PhysicalOlapScan[t1]                                                |
| ----------PhysicalDistribute[DistributionSpecHash]                            |
| ------------PhysicalOlapScan[t2]                                              |
| --------PhysicalDistribute[DistributionSpecHash]                              |
| ----------PhysicalOlapScan[t3]                                                |
|                                                                               |
| Hint log:                                                                     |
| Used:                                                                         |
| UnUsed: leading(t1 { t2 t3 })                                                 |
| SyntaxError:                                                                  |
+-------------------------------------------------------------------------------+
```

   Here are some examples of Outer, Semi, and Anti Joins with Leading Hint for readers to verify on their own.

```
-------- test outer join which can swap
-- (t1 leftjoin t2  on (P12)) innerjoin t3 on (P13) = (t1 innerjoin t3 on (P13)) leftjoin t2  on (P12)
explain shape plan select * from t1 left join t2 on c1 = c2 join t3 on c1 = c3;
explain shape plan select /*+ leading(t1 t3 t2) */ * from t1 left join t2 on c1 = c2 join t3 on c1 = c3;

-- (t1 leftjoin t2  on (P12)) leftjoin t3 on (P13) = (t1 leftjoin t3 on (P13)) leftjoin t2  on (P12)
explain shape plan select * from t1 left join t2 on c1 = c2 left join t3 on c1 = c3;
explain shape plan select /*+ leading(t1 t3 t2) */ * from t1 left join t2 on c1 = c2 left join t3 on c1 = c3;

-- (t1 leftjoin t2  on (P12)) leftjoin t3 on (P23) = t1 leftjoin (t2  leftjoin t3 on (P23)) on (P12)
select /*+ leading(t2 t3 t1) SWAP_INPUT(t1) */ * from t1 left join t2 on c1 = c2 left join t3 on c2 = c3;
explain shape plan select /*+ leading(t1 {t2 t3}) */ * from t1 left join t2 on c1 = c2 left join t3 on c2 = c3;
explain shape plan select /*+ leading(t1 {t2 t3}) */ * from t1 left join t2 on c1 = c2 left join t3 on c2 = c3;
```

```
-------- test outer join which can not swap
--  t1 leftjoin (t2  join t3 on (P23)) on (P12) != (t1 leftjoin t2  on (P12)) join t3 on (P23)
-- eliminated to inner join
explain shape plan select /*+ leading(t1 {t2 t3}) */ * from t1 left join t2 on c1 = c2 join t3 on c2 = c3;
explain graph select /*+ leading(t1 t2 t3) */ * from t1 left join (select * from t2 join t3 on c2 = c3) on c1 = c2;

-- test semi join
explain shape plan select * from t1 where c1 in (select c2 from t2);
explain shape plan select /*+ leading(t2 t1) */ * from t1 where c1 in (select c2 from t2);

-- test anti join
explain shape plan select * from t1 where exists (select c2 from t2);
```

6. View

   In the case of involving aliases (Alias), the alias can be specified as a complete and independent subtree, and the join order is generated according to the text order

```
mysql>  explain shape plan select /*+ leading(alias t1) */ count(*) from t1 join (select c2 from t2 join t3 on t2.c2 =
t3.c3) as alias on t1.c1 = alias.c2;
+-----------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                         |
+-----------------------------------------------------------------------------------------+
| PhysicalResultSink                                                                      |
| --hashAgg[GLOBAL]                                                                       |
| ----PhysicalDistribute[DistributionSpecGather]                                          |
| ------hashAgg[LOCAL]                                                                    |
| --------PhysicalProject                                                                 |
| ----------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = alias.c2)) otherCondition=()  |
| ------------PhysicalProject                                                             |
| --------------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=() |
| ----------------PhysicalProject                                                        |
| ------------------PhysicalOlapScan[t2]                                                 |
| ----------------PhysicalDistribute[DistributionSpecHash]                               |
| ------------------PhysicalProject                                                      |
| --------------------PhysicalOlapScan[t3]                                               |
| ------------PhysicalDistribute[DistributionSpecHash]                                   |
| --------------PhysicalProject                                                          |
| ----------------PhysicalOlapScan[t1]                                                   |
|                                                                                         |
| Hint log:                                                                               |
| Used: leading(alias t1)                                                                 |
| UnUsed:                                                                                 |
| SyntaxError:                                                                            |
+-----------------------------------------------------------------------------------------+
```

Ordered Hint

Ordered hint can be regarded as a special case of leading hint and is used to control the join order as the text order.

Syntax

The syntax of Ordered Hint is `/*+ ORDERED */`, which is placed after the `SELECT` keyword in the `SELECT` statement, followed immediately by the rest of the query.

Case

The following is an example of using Ordered Hint:

```
mysql> explain shape plan select /*+ ORDERED */ t1.c1 from t2 join t1 on t1.c1 = t2.c2 join t3 on c2 = c3;
+------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                              |
+------------------------------------------------------------------------------+
| PhysicalResultSink                                                           |
| --PhysicalDistribute[DistributionSpecGather]                                 |
| ----PhysicalProject                                                          |
| ------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=()  |
| --------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| ----------PhysicalProject                                                    |
| ------------PhysicalOlapScan[t2]                                            |
| ----------PhysicalDistribute[DistributionSpecHash]                          |
| ------------PhysicalProject                                                  |
| --------------PhysicalOlapScan[t1]                                          |
| --------PhysicalDistribute[DistributionSpecHash]                            |
| ----------PhysicalProject                                                    |
| ------------PhysicalOlapScan[t3]                                            |
|                                                                              |
| Hint log:                                                                    |
| Used: ORDERED                                                                |
| UnUsed:                                                                      |
| SyntaxError:                                                                 |
+------------------------------------------------------------------------------+
```

Relationship with Leading Hint

When Ordered Hint and Leading Hint are used simultaneously, Ordered Hint will take precedence over Leading Hint. This means that even if Leading Hint is specified, if Ordered Hint also exists, the query plan will be executed according to the rules of Ordered Hint, and Leading Hint will be ignored. The following is an example showing the situation when both are used simultaneously:

```
mysql> explain shape plan select /*+ ORDERED LEADING(t1 t2 t3) */ t1.c1 from t2 join t1 on t1.c1 = t2.c2 join t3 on c2 = c3;
  +------------------------------------------------------------------------------+
  | Explain String(Nereids Planner)                                              |
  +------------------------------------------------------------------------------+
  | PhysicalResultSink                                                           |
  | --PhysicalDistribute[DistributionSpecGather]                                 |
  | ----PhysicalProject                                                          |
  | ------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=()  |
  | --------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
  | ----------PhysicalProject                                                    |
  | ------------PhysicalOlapScan[t2]                                            |
  | ----------PhysicalDistribute[DistributionSpecHash]                          |
  | ------------PhysicalProject                                                  |
  | --------------PhysicalOlapScan[t1]                                          |
  | --------PhysicalDistribute[DistributionSpecHash]                            |
  | ----------PhysicalProject                                                    |
  | ------------PhysicalOlapScan[t3]                                            |
  |                                                                              |
  | Hint log:                                                                    |
  | Used: ORDERED                                                                |
  | UnUsed: leading(t1 t2 t3)                                                    |
  | SyntaxError:                                                                 |
  +------------------------------------------------------------------------------+
```

Summary

Leading Hint is a powerful feature for manually controlling the join order and is widely used in production business tuning. Using leading hint well can meet the tuning requirements for the join order on site and increase the flexibility of system control

Distribute Hint

Overview

The Distribute hint is used to control the shuffle method for joins.

Syntax

- Supports specifying the Distribute Type for the right table, which can be either `[shuffle]` or `[broadcast]`, and should be written before the right table in the Join.
- Supports an arbitrary number of Distribute Hints.
- When encountering a Distribute Hint that cannot correctly generate a plan, the system will not display an error. It will make the best effort to apply the hint, and the final Distribute method will be shown in the EXPLAIN output.

Examples

Used in Combination with Ordered Hint

Fix the Join order to the textual sequence, and then specify the expected Distribute method for the Join. For example:

Before using:

```
mysql> explain shape plan select count(*) from t1 join t2 on t1.c1 = t2.c2;
  +------------------------------------------------------------------------------------+
  | Explain String(Nereids Planner)                                                    |
  +------------------------------------------------------------------------------------+
  | PhysicalResultSink                                                                 |
  | --hashAgg[GLOBAL]                                                                  |
  | ----PhysicalDistribute[DistributionSpecGather]                                     |
  | ------hashAgg[LOCAL]                                                               |
  | --------PhysicalProject                                                            |
  | ----------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
  | ------------PhysicalProject                                                        |
  | --------------PhysicalOlapScan[t1]                                                 |
  | ------------PhysicalDistribute[DistributionSpecHash]                              |
  | --------------PhysicalProject                                                      |
  | ----------------PhysicalOlapScan[t2]                                              |
  +------------------------------------------------------------------------------------+
```

After using:

```
mysql> explain shape plan select /*+ ordered */ count(*) from t2 join[broadcast] t1 on t1.c1 = t2.c2;
+------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                    |
+------------------------------------------------------------------------------------+
| PhysicalResultSink                                                                 |
| --hashAgg[GLOBAL]                                                                  |
| ----PhysicalDistribute[DistributionSpecGather]                                     |
| ------hashAgg[LOCAL]                                                               |
```

```
| --------PhysicalProject                                                                |
| ----------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| ------------PhysicalProject                                                            |
| --------------PhysicalOlapScan[t2]                                                     |
| ------------PhysicalDistribute[DistributionSpecReplicated]                            |
| --------------PhysicalProject                                                          |
| ----------------PhysicalOlapScan[t1]                                                   |
|                                                                                        |
| Hint log:                                                                              |
| Used: ORDERED                                                                          |
| UnUsed:                                                                                |
| SyntaxError:                                                                           |
+----------------------------------------------------------------------------------------+
```

The Explain Shape Plan will display information related to the Distribute operator. Specifically:

- `DistributionSpecReplicated` indicates that the corresponding data will be replicated to all BE nodes.
- `DistributionSpecGather` indicates that the data will be gathered to the FE node.
- `DistributionSpecHash` indicates that the data will be distributed to different BE nodes based on a specific hashKey and algorithm. **Used in Combination with Leading Hint**

When writing SQL queries, you can specify the corresponding `DISTRIBUTE` method for each `JOIN` operation while using the `LEADING` hint. Below is a specific example demonstrating how to mix `Distribute Hint` and `Leading Hint` in an SQL query.

```
explain shape plan
    select
        nation,
        o_year,
        sum(amount) as sum_profit
    from
        (
            select
                /*+ leading(orders shuffle {lineitem shuffle part} shuffle {supplier broadcast nation} shuffle partsupp) */
                n_name as nation,
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
            from
                part,
                supplier,
                lineitem,
                partsupp,
                orders,
                nation
            where
                s_suppkey = l_suppkey
                and ps_suppkey = l_suppkey
                and ps_partkey = l_partkey
                and p_partkey = l_partkey
                and o_orderkey = l_orderkey
                and s_nationkey = n_nationkey
                and p_name like '%green%'
        ) as profit
    group by
        nation,
        o_year
    order by
        nation,
        o_year desc;
```

Summary

The Distribute hint is a commonly used hint for controlling the join shuffle method, allowing manual specification of shuffle or broadcast distribution methods. Proper use of the Distribute hint can meet on-site tuning needs for join shuffle methods, increasing the flexibility of system control.

Optimizing Table Schema Design

Overview

In Schema design and tuning, table Schema design is a crucial part, encompassing table engine selection, partition and bucket column selection, partition and bucket size settings, key column and field type optimization, etc. Systems lacking proper Schema design may encounter issues such as data skew, failing to fully leverage system parallelism and sorting features, thereby hindering the Doris system from realizing its true performance advantages within business systems.

Detailed design principles can be found in the Data Table Design section for further information. This chapter, from the perspective of practical cases, will showcase performance bottlenecks caused by Schema design issues in several typical scenarios and provide optimization suggestions for business tuning reference.

Case 1: Table Engine Selection

Doris supports three table models: Duplicate, Unique, and Aggregate. Among them, Unique can be further divided into Merge-On-Read (MOR) and Merge-On-Write (MOW).

The query performance of these table models, from best to worst, is: Duplicate > MOW > MOR == Aggregate. Therefore, under normal circumstances, if there are no special requirements, the Duplicate table is recommended for better query performance.

> **When the business has no data update requirements but high demands for query performance, the Duplicate table is recommended.**

Case 2: Bucket Column Selection

Doris supports bucketing data, which means distributing data based on bucket keys in the Schema to form data buckets.

Selecting appropriate bucket columns is vital for the reasonable distribution of raw data, effectively preventing performance issues caused by data skew. Meanwhile, it maximizes the utilization of Doris's Colocate Join and Bucket Shuffle Join features, significantly enhancing the performance of Join operations.

Taking the table creation statement of table t1 as an example, the current bucket column is set as c2. However, during the actual data import process, if all values of column c2 are defaulted to null, even if 64 buckets are set, only one bucket will contain all the data. This extreme case leads to severe data skew, resulting in performance bottlenecks.

```
CREATE TABLE `t1` (
  `c1` INT NULL,
  `c2` INT NULL
) ENGINE=OLAP
DUPLICATE KEY(`c1`)
DISTRIBUTED BY HASH(`c2`) BUCKETS 64
```

```
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
) ;
insert into t1 select number, null from numbers ('number'='10000000');
```

In response to the above situation, we can change the bucket column from c2 to c1 to achieve adequate data hashing and maximize the system's parallel processing capabilities, thereby achieving tuning purposes.

Therefore, during the Schema design phase, business personnel need to design reasonable bucket columns beforehand based on business characteristics. For instance, if it is known beforehand that the business meaning of column c2 may contain a large number of skewed values, such as null or certain specific values, these fields should be avoided as bucket columns. Conversely, fields with adequate hashing characteristics in business meaning, such as user ID, should be selected as bucket columns. During the performance issue troubleshooting phase, the following SQL statement can be used to confirm whether the bucket field has data skew and make subsequent optimization adjustments accordingly.

```
select c2, count(*) cnt from t1 group by c2 order by cnt desc limit 10;
```

> **Check whether the bucket column has data skew issues. If so, replace it with a field that has adequate hashing characteristics in business meaning as the bucket column.**

It is clear that good prior design can significantly reduce the cost of locating and correcting issues when they occur. Therefore, it is strongly recommended that business personnel conduct rigorous design and checks during the Schema design phase to avoid introducing unnecessary costs.

Case 3: Key Column Optimization

Among the three table models, if the table creation Schema explicitly specifies a Duplicate Key, Unique Key, or Aggregate Key, Doris will ensure that data is sorted based on the Key column at the storage level. This feature provides new ideas for data query performance optimization. Specifically, during the Schema design phase, if columns frequently used for equality or range queries in business queries can be defined as Key columns, it will significantly increase the execution speed of such queries, thereby enhancing overall performance.

Here is a set of examples of business query requirements:

```
select * from t1 where t1.c1 = 1;
select * from t1 where t1.c1 > 1 and t1.c1 < 10;
select * from t1 where t1.c1 in (1, 2, 3);
```

For the above business requirements and the Schema design and later optimization of table t1, considering setting column c1 as the Key column to accelerate the query process is advisable. Here is an example:

```
CREATE TABLE `t1` (
  `c1` INT NULL,
  `c2` INT NULL
) ENGINE=OLAP
DUPLICATE KEY(`c1`)
DISTRIBUTED BY HASH(`c2`) BUCKETS 10
PROPERTIES (
```

```
"replication_allocation" = "tag.location.default: 1"
) ;
```

> **Set columns frequently used in business queries as key columns to accelerate the query process.**

Case 4: Field Type Optimization

In database systems, the complexity of processing different types of data can vary significantly. For example, processing variable-length types is much more complex than fixed-length types; similarly, processing high-precision types is more complex than low-precision types.

This characteristic provides important insights into the design and later optimization of business system Schemas:

1. While meeting the expression and computation needs of business systems, priority should be given to fixed-length types, avoiding the use of variable-length types;
2. At the same time, low-precision types should be adopted instead of high-precision types. Specific practices include using BIGINT to replace VARCHAR or STRING type fields and using FLOAT / INT / BIGINT to replace DECIMAL type fields. Reasonable design and optimization of such field types will greatly enhance business computation efficiency, thereby improving system performance.

> **When defining Schema types, follow the principle of prioritizing fixed-length and low-precision types.**

Summary

In summary, a well-designed Schema can maximize the utilization of Doris's features, thereby significantly enhancing business performance. Conversely, a non-optimized Schema design may have a global negative impact on the business, such as causing data skew. Therefore, the initial Schema design optimization work is particularly important.

Optimizing Table Index Design

Overview

Doris currently supports two types of indexes:

1. Built-in Indexes: These include prefix indexes, ZoneMap indexes, etc.
2. Secondary Indexes: These include inverted indexes, Bloom filter indexes, N-Gram Bloom filter indexes, and Bitmap indexes, etc.

In the process of business optimization, fully analyzing business characteristics and make effective use of indexes can greatly enhance the effectiveness of queries and analyses, thereby achieving the purpose of performance tuning.

For a detailed introduction to various indexes, please refer to the Table Index section. This chapter will demonstrate index usage techniques in several typical scenarios from the perspective of actual cases and summarize optimization suggestions for reference in business tuning.

Case 1: Optimizing the Order of Key Columns to Leverage Prefix Indexes for Accelerated Queries

In optimizing table schema design, we have introduced how to select appropriate fields as key fields and utilize Doris's key column sorting feature to accelerate queries. This case will further expand on this scenario.

Due to Doris's built-in prefix index function, it automatically takes the first 36 bytes of the table's Key as a prefix index when creating the table. When query conditions match the prefix of the prefix index, it can significantly speed up the query. Below is an example of a table definition:

```
CREATE TABLE `t1` (
  `c1` VARCHAR(10) NULL,
  `c2` VARCHAR(10) NULL
) ENGINE=OLAP
DUPLICATE KEY(`c1`)
DISTRIBUTED BY HASH(`c2`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

The corresponding business SQL pattern is as follows:

```
select * from t1 where t1.c2 = '1';
select * from t1 where t1.c2 in ('1', '2', '3');
```

In the above schema definition, `c1` comes before `c2`. However, the queries use the `c2` field for filtering. In this case, the acceleration function of the prefix index cannot be utilized. To optimize, we can adjust the definition order of `c1` and `c2`, placing the `c2` column in the first field position to leverage the acceleration function of the prefix index.

The adjusted schema is as follows:

```
CREATE TABLE `t1` (
  `c2` VARCHAR(10) NULL,
  `c1` VARCHAR(10) NULL
) ENGINE=OLAP
DUPLICATE KEY(`c2`)
DISTRIBUTED BY HASH(`c1`) BUCKETS 10
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

> **When defining the schema column order, reference the high-frequency and high-priority columns in business query filtering to fully leverage Doris's prefix index acceleration function.**

Case 2: Using Inverted Indexes to Accelerate Queries

Doris supports inverted indexes as secondary indexes to accelerate business scenarios such as equal value, range, and full-text search of text types. The creation and management of inverted indexes are independent, allowing for convenient business performance optimization without affecting the original table schema and without the need to re-import table data.

For typical usage scenarios, syntax, and cases, please refer to the Table Index - Inverted Index section for a detailed introduction, so this chapter will not repeat the explanation.

> **For full-text searches of text types and equal value or range queries on string, numeric, and datetime type fields, inverted indexes can be utilized to accelerate queries. Especially in certain situations, such as when the original table structure and key definition are not convenient to optimize, or the cost of re-importing table data is high, inverted indexes provide a flexible acceleration solution to optimize business execution performance.**

Summary

In schema tuning, apart from table-level schema optimization, index optimization also occupies an important position. Doris provides multiple index types, including built-in indexes such as prefix index, as well as secondary indexes such as inverted indexes, which provide strong support for performance acceleration. By reasonably utilizing these indexes, we can significantly improve the speed of business queries and analyses in multiple scenarios, which is of great significance for multi-scenario business queries and analyses.

Optimizing Table Scanning

Overview

Doris, as a high-performance real-time analytic data warehouse, offers a powerful partition pruning feature that can significantly enhance query performance.

Partition pruning is a query optimization technique that intelligently identifies partitions relevant to a query by analyzing its conditions, and scans only the data within these partitions, thereby avoiding unnecessary scans of irrelevant partitions. This approach can greatly reduce I/O operations and computational load, thus accelerating query execution.

Case

Here is a usage case to demonstrate Doris's partition pruning feature.

Suppose we have a sales data table named `sales`, which is partitioned by date, with each day's data stored in a separate partition. The table structure is defined as follows:

```
CREATE TABLE sales (
    date DATE,
    product VARCHAR(50),
    amount DECIMAL(10, 2)
)
PARTITION BY RANGE(date) (
    PARTITION p1 VALUES LESS THAN ('2023-01-01'),
    PARTITION p2 VALUES LESS THAN ('2023-02-01'),
    PARTITION p3 VALUES LESS THAN ('2023-03-01'),
    PARTITION p4 VALUES LESS THAN ('2023-04-01')
)
DISTRIBUTED BY HASH(date) BUCKETS 16
PROPERTIES
(
    "replication_num" = "1"
);
```

Now, we need to query the total sales amount between January 15, 2023 and February 15, 2023. The query statement is as follows:

```
SELECT SUM(amount) AS total_amount
FROM sales
WHERE date BETWEEN '2023-01-15' AND '2023-02-15';
```

For the above query, Doris's partition pruning optimization process is as follows:

1. Doris intelligently analyzes the partition column `date` in the query conditions and identifies the date range of the query as being between '2023-01-15' and '2023-02-15'.
2. By comparing the query conditions with the partition definitions, Doris precisely locates the range of partitions that need to be scanned. In this example, only partitions `p2` and `p3` need to be scanned, as their date ranges fully cover the query conditions.

3. Doris automatically skips partitions unrelated to the query conditions, such as `p1` and `p4`, avoiding unnecessary data scans and thereby reducing I/O overhead.
4. Finally, Doris performs data scanning and aggregation computations only within partitions `p2` and `p3`, quickly obtaining the query results.

By using the `EXPLAIN` command, we can view the query execution plan and confirm that Doris's partition pruning optimization has taken effect. In the execution plan, the `partition` attribute of the `OlapScanNode` node will display the actually scanned partitions as `p2` and `p3`.

```
|   0:VOlapScanNode(212)                                                 |
|      TABLE: cir.sales(sales), PREAGGREGATION: ON                       |
|      PREDICATES: (date[#0] >= '2023-01-15') AND (date[#0] <= '2023-02-15') |
|      partitions=2/4 (p2,p3)                                            |
```

Summary

In summary, Doris's partition pruning feature can intelligently identify the relevance between query conditions and partitions, automatically prune irrelevant partitions, and scan only necessary data, thereby significantly enhancing query performance. Reasonable utilization of the partition pruning feature can help users build efficient real-time analytics systems and easily handle massive data query demands.

Transparent Rewriting with Sync-Materialized View

Overview

The Sync-Materialized View is a special kind of table that pre-computes and stores data according to a predefined SELECT statement. Its main purpose is to meet users' needs for analyzing raw detailed data from any dimension and also enable quick analysis and queries on fixed dimensions.

The applicable scenarios for synchronous materialized views are as follows:

1. The analysis requirements cover both detailed data queries and fixed-dimension queries.
2. The queries involve *only* a small number of columns or rows in the table.
3. The queries include time-consuming processing operations, such as long-duration aggregation operations.
4. The queries need to match different prefix indexes.

For queries that frequently and repeatedly use the results of the same subqueries, synchronous materialized views can significantly improve performance. Doris will automatically maintain the data of materialized views to ensure data consistency between the base table and the materialized view table, without requiring additional manual maintenance costs. During a query, the system will automatically match the optimal materialized view and directly read data from it.

> **Precautions**
>
> - In Doris 2.0 and *later* versions, materialized views have some enhanced functions. It is recommended that users confirm in the test environment whether the expected queries can hit the materialized views they want to create before using materialized views in the formal production environment.
> - It is not recommended to create multiple materialized views with similar forms on the same table, as this may lead to conflicts among multiple materialized views and thus cause query hit failures. :::
>
> ## Case
>
> The following uses a specific example to demonstrate the process of using synchronous materialized views to accelerate queries:
>
> Suppose we have a detailed sales record table named `sales_records`, which records various pieces of information for each transaction in detail, including the transaction ID, salesperson ID, selling store ID, sales date, and transaction amount. Now, we often need to conduct analysis and queries on the sales volume of different stores.
>
> To optimize the performance of these queries, we can create a materialized view named `store_amt`. This view groups by the selling store and sums up the sales amounts of the same store. The specific steps are as follows:

Create a Sync-Materialized View

Firstly, we use the following SQL statement to create the materialized view `store_amt`:

```
CREATE MATERIALIZED VIEW store_amt AS
SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
```

After submitting the creation task, Doris will build this synchronous materialized view asynchronously in the background. We can use the following command to check the creation progress of the materialized view:

```
SHOW ALTER TABLE MATERIALIZED VIEW FROM db_name;
```

When the `State` field becomes `FINISHED`, it means that the `store_amt` materialized view has been successfully created.

Transparent Rewriting

After the materialized view is created, when we query the sales volume of different stores, Doris will automatically match the `store_amt` materialized view and directly read the pre-aggregated data from it, thus significantly improving the query efficiency. The query statement is as follows:

```
SELECT store_id, SUM(sale_amt) FROM sales_records GROUP BY store_id;
```

We can also use the `EXPLAIN` command to check whether the query has successfully hit the materialized view:

```
EXPLAIN SELECT store_id, SUM(sale_amt) FROM sales_records GROUP BY store_id;
```

At the end of the execution plan, if something like the following is displayed, it means that the query has successfully hit the `store_amt` materialized view:

```
TABLE: default_cluster:test.sales_records(store_amt), PREAGGREGATION: ON
```

Through the above steps, we can use synchronous materialized views to optimize query performance and improve the efficiency of data analysis.

Summary

By creating synchronous materialized views, we can significantly improve the query speed for relevant aggregation analysis. Single-table materialized views not only enable us to conduct statistical analysis quickly but also flexibly support the query needs for detailed data, which is a very powerful feature in Doris.

Transparent Rewriting by Async-Materialized View

Overview

The Async-materialized view adopts a transparent rewriting algorithm based on the SPJG (SELECT-PROJECT-JOIN-GROUP-BY) pattern. This algorithm can analyze the structural information of the query SQL, automatically find the appropriate materialized views, and attempt to perform transparent rewriting to express the query SQL using the optimal materialized views. By using the pre-computed results of materialized views, the query performance can be significantly improved and the computing cost can be reduced.

Case

Next, an example will be used to demonstrate in detail how to utilize async-materialized views to accelerate queries.

Create Base Tables

Firstly, create the tpch database and then create two tables, namely `orders` and `lineitem`, within it, and insert corresponding data.

```
CREATE DATABASE IF NOT EXISTS tpch;
USE tpch;

CREATE TABLE IF NOT EXISTS orders (
    o_orderkey       integer not null,
    o_custkey        integer not null,
    o_orderstatus    char(1) not null,
    o_totalprice     decimalv3(15,2) not null,
    o_orderdate      date not null,
    o_orderpriority  char(15) not null,
    o_clerk          char(15) not null,
    o_shippriority   integer not null,
    o_comment        varchar(79) not null
)
DUPLICATE KEY(o_orderkey, o_custkey)
PARTITION BY RANGE(o_orderdate)(
    FROM ('2023-10-17') TO ('2023-10-20') INTERVAL 1 DAY
)
DISTRIBUTED BY HASH(o_orderkey) BUCKETS 3
PROPERTIES ("replication_num" = "1");

INSERT INTO orders VALUES
    (1, 1, 'o', 99.5, '2023-10-17', 'a', 'b', 1, 'yy'),
    (2, 2, 'o', 109.2, '2023-10-18', 'c','d',2, 'mm'),
    (3, 3, 'o', 99.5, '2023-10-19', 'a', 'b', 1, 'yy');

CREATE TABLE IF NOT EXISTS lineitem (
    l_orderkey    integer not null,
    l_partkey     integer not null,
    l_suppkey     integer not null,
    l_linenumber  integer not null,
```

```
    l_quantity    decimalv3(15,2) not null,
    l_extendedprice  decimalv3(15,2) not null,
    l_discount    decimalv3(15,2) not null,
    l_tax         decimalv3(15,2) not null,
    l_returnflag  char(1) not null,
    l_linestatus  char(1) not null,
    l_shipdate    date not null,
    l_commitdate  date not null,
    l_receiptdate date not null,
    l_shipinstruct char(25) not null,
    l_shipmode     char(10) not null,
    l_comment      varchar(44) not null
)
DUPLICATE KEY(l_orderkey, l_partkey, l_suppkey, l_linenumber)
PARTITION BY RANGE(l_shipdate)
(FROM ('2023-10-17') TO ('2023-10-20') INTERVAL 1 DAY)
DISTRIBUTED BY HASH(l_orderkey) BUCKETS 3
PROPERTIES ("replication_num" = "1");

INSERT INTO lineitem VALUES
    (1, 2, 3, 4, 5.5, 6.5, 7.5, 8.5, 'o', 'k', '2023-10-17', '2023-10-17', '2023-10-17', 'a', 'b', 'yyyyyyyyy'),
    (2, 2, 3, 4, 5.5, 6.5, 7.5, 8.5, 'o', 'k', '2023-10-18', '2023-10-18', '2023-10-18', 'a', 'b', 'yyyyyyyyy'),
    (3, 2, 3, 6, 7.5, 8.5, 9.5, 10.5, 'k', 'o', '2023-10-19', '2023-10-19', '2023-10-19', 'c', 'd', 'xxxxxxxxx');
```

Create an Asynchronous Materialized View

Based on several original tables in the tpch benchmark, create an asynchronous materialized view named `mv1`.

```
CREATE MATERIALIZED VIEW mv1
BUILD IMMEDIATE REFRESH COMPLETE ON MANUAL
PARTITION BY(l_shipdate)
DISTRIBUTED BY RANDOM BUCKETS 2
PROPERTIES ('replication_num' = '1')
AS
SELECT l_shipdate, o_orderdate, l_partkey, l_suppkey, SUM(o_totalprice) AS sum_total
FROM lineitem
LEFT JOIN orders ON lineitem.l_orderkey = orders.o_orderkey AND l_shipdate = o_orderdate
GROUP BY
l_shipdate,
o_orderdate,
l_partkey,
l_suppkey;
```

Use the Materialized View for Transparent Rewriting

```
mysql> explain shape plan SELECT l_shipdate, SUM(o_totalprice) AS total_price
    -> FROM lineitem
    -> LEFT JOIN orders ON lineitem.l_orderkey = orders.o_orderkey AND l_shipdate = o_orderdate
    -> WHERE l_partkey = 2 AND l_suppkey = 3
    -> GROUP BY l_shipdate;
+------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                        |
+------------------------------------------------------------------------+
| PhysicalResultSink                                                     |
| --PhysicalDistribute[DistributionSpecGather]                           |
| ----PhysicalProject                                                    |
| ------hashAgg[GLOBAL]                                                  |
| --------PhysicalDistribute[DistributionSpecHash]                       |
| ----------hashAgg[LOCAL]                                               |
| ------------PhysicalProject                                            |
```

```
| --------------filter((mv1.l_partkey = 2) and (mv1.l_suppkey = 3)) |
| ---------------PhysicalOlapScan[mv1]                             |
+----------------------------------------------------------------------+
```

It can be seen from the `explain shape plan` that the plan after being transparently rewritten by `mv1` has already hit `mv1`. You can also use `explain` to view the current state of the plan after being rewritten by the materialized view, including whether it has hit and which materialized view has been hit, etc., as shown below:

```
| ========== MATERIALIZATIONS ==========                                  |
|                                                                         |
| MaterializedView                                                        |
| MaterializedViewRewriteSuccessAndChose:                                 |
|   internal.tpch.mv1 chose,                                              |
|                                                                         |
| MaterializedViewRewriteSuccessButNotChose:                              |
|   not chose: none,                                                      |
|                                                                         |
| MaterializedViewRewriteFail:                                            |
```

Summary

By using async-materialized views, the query performance can be significantly improved, especially for complex join and aggregation queries. When using them, the following points need to be noted:

Usage Suggestions

- Pre-computed Results: Materialized views pre-compute and store the query results, avoiding the overhead of repeated computations for each query. This is especially effective for complex queries that need to be executed frequently.
- Reducing Join Operations: Materialized views can combine the data of multiple tables into one view, reducing the join operations during queries and thus improving query efficiency.
- Automatic Updates: When the data in the base tables changes, materialized views can be updated automatically to maintain data consistency. This ensures that the query results always reflect the latest data status.
- Space Overhead: Materialized views require additional storage space to save the pre-computed results. When creating materialized views, it is necessary to balance the improvement in query performance and the consumption of storage space.
- Maintenance Cost: The maintenance of materialized views requires certain system resources and time. Base tables that are updated frequently may lead to relatively high update overheads for materialized views. Therefore, it is necessary to choose an appropriate refresh strategy according to the actual situation.
- Applicable Scenarios: Materialized views are suitable for scenarios where the data change frequency is low and the query frequency is high. For frequently changing data, real-time computation may be more appropriate. :::

Reasonable utilization of async-materialized views can significantly improve the query performance of the database, especially in the case of complex queries and large data volumes. Meanwhile, factors such as storage and maintenance also need to be considered comprehensively to achieve a balance between performance and cost.

Optimizing Join with Colocate Group

Defining colocate group is an efficient way of Join. It allows the execution engine to effectively avoid the data transmission overhead typically associated with Join operations (for an introduction to Colocate Group, see Colocation Join)

However, in some use cases, even if a Colocate Group has been successfully established, the execution plan may still show as Shuffle Join or Bucket Shuffle Join. This situation typically occurs when Doris is organizing data. For instance, it may be migrating tablets between BEs to ensure a more balanced data distribution across multiple BEs.

You can view the Colocate Group status using the command `SHOW PROC "/colocation_group";`. As shown in the figure below, if `IsStable` is `false`, it indicates that there are unavailable Colocate Group instances.

```
mysql> show proc "/colocation_group";
+------------+----------------------+--------------------------+-------------+--------------------------+-----------+----------+----------+
| GroupId    | GroupName            | TableIds                 | BucketsNum  | ReplicaAllocation        | DistCols  | IsStable | ErrorMsg |
+------------+----------------------+--------------------------+-------------+--------------------------+-----------+----------+----------+
| 36023.36619| 36023_part_partsupp  | 36425,36621              | 96          | tag.location.default: 1 | int(11)    | true     |          |
| 32022.34353| 32022_lineitem_orders| 32815,34355              | 768         | tag.location.default: 1 | bigint(20) | true     |          |
| 10644.11087| 10644_part_partsupp  | 11037,11089              | 24          | tag.location.default: 1 | int(11)    | true     |          |
| 83282.83478| 83282_lineitem_orders| 8328483480               | 96          | tag.location.default: 1 | higint(20) | true     |          |
| 47947.50278| 47947_lineitem_orders| 48740,50280              | 768         | tag.location.default: 1 | bigint(20) | false    |          |
| 29082.30133| 29082_part_partsupp  | 30083,30135              | 24          | tag.location.default: 1 | int(11)    | false    |          |
| 85801.85997| 85801_tineitem_oraers| 85803,85999              | 96          | tag.location.default: 1 | bigint(20) | true     |          |
| 10644.10840| 10644_lineitem_orders| 10646,10842              | 96          | tag.location.default: 1 | int(11)    | true     |          |
| 47947.48543| 47947_part_partsupp  | 48349,48545              | 96          | tag.location.default: 1 | bigint(20) | true     |          |
| 29082.29866| 29082_lineitem_orders| 29672,29888              | 96          | tag.location.default: 1 | int(11)    | true     |          |
| 32022.32618| 32022_part_partsupp  | 32424,32620              | 96          | tag.location.default: 1 | int(11)    | true     |          |
| 44074.46405| 44074_lineitem_orders| 44867,46407              | 768         | tag.location.default: 1 | bigint(20) | true     |          |
| 83282.83725| 83282_part_partsupp  | 83675,83727              | 24          | tag.location.default: 1 | int(11)    | true     |          |
| 85801.86244| 85801_part_partsupp  | 86194,86246              | 96          | tag.location.default: 1 | int(11)    | true     |          |
| 40021.40617| 40021_part_partsupp  | 40423,40619              | 96          | tag.location.default: 1 | int(11)    | true     |          |
| 36023.38354| 36023_lineitem_orders| 36816, 38356             | 768         | tag.location.default: 1 | bigint(20) | true     |          |
| 44074.44670| 44074_part_partsupp  | 44476, 44672             | 96          | tag.location.default: 1 | int(11)    | true     |          |
| 40021.42352| 40021_lineitem_orders| 40814, 42354             | 768         | tag.location.default: 1 | bigint(20) | true     |          |
+------------+----------------------+--------------------------+-------------+--------------------------+-----------+----------+----------+
18 rows in set (0.00 sec)
```

Optimizing Join with Colocate Group

Adjusting Join Shuffle Mode with Hint

Overview

Doris supports the use of hints to adjust the type of data shuffle in Join operations, thereby optimizing query performance. This section provides detailed instructions on how to specify the Join Shuffle type in Doris using hints.

> **Note**
>
> Currently, Doris has good out-of-the-box capabilities. This means that in most scenarios, Doris will adaptively optimize performance in various scenarios, and users do not need to manually control hints for performance tuning. The content introduced in this chapter is mainly for professional tuners, and business personnel only need a simple understanding.

Currently, Doris supports two independent Distribute Hint, `[shuffle]` and `[broadcast]`, to specify the Distribute Type for the right table in a Join. The Distribute Type should be placed before the right table in the Join, enclosed in square brackets `[]`. Additionally, Doris can specify the shuffle mode by using the Leading Hint in conjunction with the Distribute Hint (for more details, refer to Reordering Join With Leading Hint).

Examples are as follows:

```
SELECT COUNT(*) FROM t2 JOIN [broadcast] t1 ON t1.c1 = t2.c2;
SELECT COUNT(*) FROM t2 JOIN [shuffle] t1 ON t1.c1 = t2.c2;
```

Case

Next, we will demonstrate the usage of Distribute Hints through an example:

```
EXPLAIN SHAPE PLAN SELECT COUNT(*) FROM t1 JOIN t2 ON t1.c1 = t2.c2;
```

The plan for the original SQL is as follows, showing that the join between t1 and t2 uses the hash distribute method, indicated by `DistributionSpecHash`.

```
+----------------------------------------------------------------------------------+
| Explain String (Nereids Planner)                                                 |
+----------------------------------------------------------------------------------+
| PhysicalResultSink                                                               |
| --hashAgg [GLOBAL]                                                               |
| ----PhysicalDistribute [DistributionSpecGather]                                  |
| ------hashAgg [LOCAL]                                                            |
| --------PhysicalProject                                                          |
| ----------hashJoin [INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=()|
| ------------PhysicalProject                                                      |
| --------------PhysicalOlapScan [t1]                                             |
| ------------PhysicalDistribute [DistributionSpecHash]                           |
| --------------PhysicalProject                                                    |
```

```
| ---------------PhysicalOlapScan [t2]                                              |
+-----------------------------------------------------------------------------------+
```

After adding the [broadcast] hint:

```
EXPLAIN SHAPE PLAN SELECT COUNT(*) FROM t1 JOIN [broadcast] t2 ON t1.c1 = t2.c2;
```

It can be seen that the distribution method for the join between t1 and t2 has been changed to the broadcast method, indicated by `DistributionSpecReplicated`.

```
+-----------------------------------------------------------------------------------+
| Explain String (Nereids Planner)                                                  |
+-----------------------------------------------------------------------------------+
| PhysicalResultSink                                                                |
| --hashAgg [GLOBAL]                                                                |
| ----PhysicalDistribute [DistributionSpecGather]                                   |
| ------hashAgg [LOCAL]                                                             |
| --------PhysicalProject                                                           |
| ----------hashJoin [INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=()|
| ------------PhysicalProject                                                       |
| --------------PhysicalOlapScan [t1]                                              |
| ------------PhysicalDistribute [DistributionSpecReplicated]                      |
| --------------PhysicalProject                                                     |
| ----------------PhysicalOlapScan [t2]                                            |
+-----------------------------------------------------------------------------------+
```

Summary

By appropriately using Distribute Hints, you can optimize the shuffle mode for Join operations and improve query performance. In practice, it is recommended to first analyze the query execution plan using EXPLAIN and then specify the appropriate shuffle type based on the actual situation.

Control CBO Rules With Hint

Overview

The query optimizer applies a series of rules when generating execution plans. These rules are mainly categorized into two types: Rule-Based Optimizer (RBO) and Cost-Based Optimizer (CBO).

- RBO: This type of optimization improves query plans by applying a set of predefined heuristic rules without considering specific data statistics. Strategies such as predicate pushdown and projection pushdown fall into this category.
- CBO: This type of optimization leverages data statistics to estimate the cost of different execution plans and selects the plan with the lowest cost for execution. This includes choices of access paths and join algorithms.

In some cases, database administrators or developers may need more granular control over the query optimization process. Based on this, this document will introduce how to use query hints to manage CBO rules.

> **Note**
>
> Currently, Doris has good out-of-the-box capabilities. This means that in most scenarios, Doris will adaptively optimize performance in various scenarios, and users do not need to manually control hints for performance tuning. The content introduced in this chapter is mainly for professional tuners, and business personnel only need a simple understanding.

The basic syntax for CBO rule control hints is as follows:

```
SELECT /*+ USE_CBO_RULE(rule1, rule2, ...) */ ...
```

This hint immediately follows the `SELECT` keyword and specifies the names of the rules to be enabled within parentheses (rule names are case-insensitive).

Currently, the Doris optimizer supports several cost-based rewrites, which can be explicitly enabled using the `USE_CBO_RULE` hint, such as:

- PUSH_DOWN_AGG_THROUGH_JOIN
- PUSH_DOWN_AGG_THROUGH_JOIN_ONE_SIDE
- PUSH_DOWN_DISTINCT_THROUGH_JOIN

Case

Here is a query example:

```
explain shape plan
    select /*+ USE_CBO_RULE(push_down_agg_through_join_one_side) */
            a.event_id,
            b.group_id,
```

```
        COUNT(a.event_id)
from a
join b on
        a.device_id = b.device_id
group by
        a.event_id,
        b.group_id
;
```

In this example, a CBO rule for aggregation pushdown is enabled. This operation allows table a to be aggregated before the join operation, reducing the cost of the join and speeding up the query. The plan after pushdown is as follows:

```
PhysicalResultSink
--hashAgg[GLOBAL]
----hashAgg[LOCAL]
------hashJoin[INNER_JOIN] hashCondition=((a.device_id = b.device_id)) otherCondition=()
--------hashAgg[LOCAL]
----------PhysicalOlapScan[a]
--------filter((cast(experiment_id as DOUBLE) = 73.0))
----------PhysicalOlapScan[b]
```

Summary

Proper use of the `USE_CBO_RULE` hint can help manually enable certain advanced CBO optimization rules, optimizing performance in specific scenarios. However, using CBO optimization rules requires a deep understanding of the query optimization process and data characteristics. In most cases, relying on the automatic decisions of the Doris optimizer is still the best choice.

Reordering Join With Leading Hint

Overview

The Leading Hint feature allows users to manually specify the join order of tables in a query, optimizing the performance of complex queries in specific scenarios. This article will describe in detail how to use Leading Hint to control the join order in Doris. For detailed usage instructions, please refer to the leading hint document.

> **Note**
>
> Currently, Doris has good out-of-the-box capabilities. This means that in most scenarios, Doris will adaptively optimize performance in various scenarios, and users do not need to manually control hints for performance tuning. The content introduced in this chapter is mainly for professional tuners, and business personnel only need a simple understanding.

Case 1: Adjusting the Left and Right Table Order

For the following query:

```
mysql> explain shape plan select * from t1 join t2 on t1.c1 = t2.c2;
+------------------------------------------------------------------------------+
| _Explain_ String(Nereids Planner)                                            |
+------------------------------------------------------------------------------+
| PhysicalResultSink                                                           |
| --PhysicalDistribute[DistributionSpecGather]                                 |
| ----PhysicalProject                                                          |
| ------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| --------PhysicalOlapScan[t1]                                                 |
| --------PhysicalDistribute[DistributionSpecHash]                             |
| ----------PhysicalOlapScan[t2]                                               |
+------------------------------------------------------------------------------+
```

You can use Leading Hint to force the join order to be t2 join t1 and adjust the original join order.

```
mysql> explain shape plan select /*+ leading(t2 t1) */ * from t1 join t2 on t1.c1 = t2.c2;
+------------------------------------------------------------------------------+
| _Explain_ String(Nereids Planner)                                            |
+------------------------------------------------------------------------------+
| PhysicalResultSink                                                           |
| --PhysicalDistribute[DistributionSpecGather]                                 |
| ----PhysicalProject                                                          |
| ------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=() |
| --------PhysicalOlapScan[t2]                                                 |
| --------PhysicalDistribute[DistributionSpecHash]                             |
| ----------PhysicalOlapScan[t1]                                               |
|                                                                              |
| Hint log:                                                                    |
| Used: leading(t2 t1)                                                         |
| UnUsed:                                                                      |
```

```
| SyntaxError:                                                                |
+-----------------------------------------------------------------------------+
```

The Hint log shows the successfully applied hint: Used: `leading(t2 t1)`.

Case 2: Forcing the Generation of a Left-Deep Tree

```
mysql> explain shape plan select /*+ leading(t1 t2 t3) */ * from t1 join t2 on t1.c1 = t2.c2 join t3 on t2.c2 = t3.c3;
+-------------------------------------------------------------------------------------+
| _Explain_ String(Nereids Planner)                                                   |
+-------------------------------------------------------------------------------------+
| PhysicalResultSink                                                                  |
| --PhysicalDistribute[DistributionSpecGather]                                        |
| ----PhysicalProject                                                                 |
| ------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=()        |
| --------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=()      |
| ----------PhysicalOlapScan[t1]                                                      |
| ----------PhysicalDistribute[DistributionSpecHash]                                  |
| ------------PhysicalOlapScan[t2]                                                    |
| --------PhysicalDistribute[DistributionSpecHash]                                    |
| ----------PhysicalOlapScan[t3]                                                      |
|                                                                                     |
| Hint log:                                                                           |
| Used: leading(t1 t2 t3)                                                             |
| UnUsed:                                                                             |
| SyntaxError:                                                                        |
+-------------------------------------------------------------------------------------+
```

Similarly, the Hint log shows the successfully applied hint: `Used: leading(t1 t2 t3)`.

Case 3: Forcing the Generation of a Right-Deep Tree

```
mysql> explain shape plan select /*+ leading(t1 {t2 t3}) */ * from t1 join t2 on t1.c1 = t2.c2 join t3 on t2.c2 = t3.c3;
+-------------------------------------------------------------------------------------+
| _Explain_ String(Nereids Planner)                                                   |
+-------------------------------------------------------------------------------------+
| PhysicalResultSink                                                                  |
| --PhysicalDistribute[DistributionSpecGather]                                        |
| ----PhysicalProject                                                                 |
| ------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = t2.c2)) otherCondition=()        |
| --------PhysicalOlapScan[t1]                                                       |
| --------PhysicalDistribute[DistributionSpecHash]                                   |
| ----------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=()    |
| ------------PhysicalOlapScan[t2]                                                    |
| ------------PhysicalDistribute[DistributionSpecHash]                               |
| --------------PhysicalOlapScan[t3]                                                 |
|                                                                                     |
| Hint log:                                                                           |
| Used: leading(t1 { t2 t3 })                                                         |
| UnUsed:                                                                             |
| SyntaxError:                                                                        |
+-------------------------------------------------------------------------------------+
```

Similarly, the Hint log shows the successfully applied hint: `Used: leading(t1 { t2 t3 })`.

Case 4: Forcing the Generation of a Bushy Tree

```
mysql> explain shape plan select /*+ leading({t1 t2} {t3 t4}) */ * from t1 join t2 on t1.c1 = t2.c2 join t3 on t2.c2 = t3.c3 join t4
on t3.c3 = t4.c4;
+------------------------------------------------+
| _Explain_ String                                |
+------------------------------------------------+
| PhysicalResultSink                             |
| --PhysicalDistribute                           |
| ----PhysicalProject                            |
| ------hashJoin[INNER_JOIN](t2.c2 = t3.c3)      |
| --------hashJoin[INNER_JOIN](t1.c1 = t2.c2)    |
| ----------PhysicalOlapScan[t1]                 |
| ----------PhysicalDistribute                   |
| ------------PhysicalOlapScan[t2]               |
| --------PhysicalDistribute                     |
| ----------hashJoin[INNER_JOIN](t3.c3 = t4.c4)  |
| ------------PhysicalOlapScan[t3]               |
| ------------PhysicalDistribute                 |
| --------------PhysicalOlapScan[t4]             |
|                                                |
| Used: leading({ t1 t2 } { t3 t4 })             |
| UnUsed:                                        |
| SyntaxError:                                   |
+------------------------------------------------+
```

Similarly, the Hint log shows the successfully applied hint: `Used: leading({ t1 t2 } { t3 t4 })`.

Case 5: View Participating in the Join as a Whole

```
mysql>  explain shape plan select /*+ leading(alias t1) */ count(*) from t1 join (select c2 from t2 join t3 on t2.c2 = t3.c3) as
alias on t1.c1 = alias.c2;
+-----------------------------------------------------------------------------------------+
| _Explain_ String(Nereids Planner)                                                        |
+-----------------------------------------------------------------------------------------+
| PhysicalResultSink                                                                      |
| --hashAgg[GLOBAL]                                                                       |
| ----PhysicalDistribute[DistributionSpecGather]                                          |
| ------hashAgg[LOCAL]                                                                    |
| --------PhysicalProject                                                                 |
| ----------hashJoin[INNER_JOIN] hashCondition=((t1.c1 = alias.c2)) otherCondition=()     |
| ------------PhysicalProject                                                             |
| --------------hashJoin[INNER_JOIN] hashCondition=((t2.c2 = t3.c3)) otherCondition=()    |
| ----------------PhysicalProject                                                        |
| ------------------PhysicalOlapScan[t2]                                                 |
| ----------------PhysicalDistribute[DistributionSpecHash]                               |
| ------------------PhysicalProject                                                      |
| --------------------PhysicalOlapScan[t3]                                               |
| ------------PhysicalDistribute[DistributionSpecHash]                                   |
| --------------PhysicalProject                                                          |
| ----------------PhysicalOlapScan[t1]                                                   |
|                                                                                         |
| Hint log:                                                                               |
| Used: leading(alias t1)                                                                 |
| UnUsed:                                                                                 |
| SyntaxError:                                                                            |
+-----------------------------------------------------------------------------------------+
```

Similarly, the Hint log shows the successfully applied hint: `Used: leading(alias t1)`.

Case 6: Mixing DistributeHint and LeadingHint

```
explain shape plan
    select
        nation,
        o_year,
        sum(amount) as sum_profit
    from
        (
            select
                /*+ leading(orders shuffle {lineitem shuffle part} shuffle {supplier broadcast nation} shuffle partsupp) */
                n_name as nation,
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
            from
                part,
                supplier,
                lineitem,
                partsupp,
                orders,
                nation
            where
                s_suppkey = l_suppkey
                and ps_suppkey = l_suppkey
                and ps_partkey = l_partkey
                and p_partkey = l_partkey
                and o_orderkey = l_orderkey
                and s_nationkey = n_nationkey
                and p_name like '%green%'
        ) as profit
    group by
        nation,
        o_year
    order by
        nation,
        o_year desc;
```

The above hint specification `/*+ leading(orders shuffle {lineitem shuffle part} shuffle {supplier broadcast nation} shuffle partsupp) */` mixes the two formats of leading and distribute hint. Leading is used to control the relative join order among the overall tables, while shuffle and broadcast are used to specify the shuffle method for specific joins. By combining the two, the connection order and connection method can be flexibly controlled, making it convenient to manually control the expected plan behavior of the user.

> **Usage Suggestions**
>
> - It is recommended to use EXPLAIN to carefully analyze the execution plan to ensure that the Leading Hint can achieve the expected effect.
> - When the Doris version is upgraded or the business data changes, the effect of the Leading Hint should be re-evaluated, and timely recording and adjustment should be made.

Summary

Leading Hint is a powerful function that can manually control the connection order. At the same time, it can also be combined with the shuffle hint to control the join distribution method at the same time, thereby optimizing the

query performance. Note that this advanced feature should be used with caution based on a full understanding of the query characteristics and data distribution.

Accelerating Queries with SQL Cache

Overview

For the detailed implementation principle of SQL Cache, please refer to the chapter SQL Cache.

Case

For detailed cases, please refer to the chapter SQL Cache.

Summary

SQL Cache is a query optimization mechanism provided by Doris, which can significantly improve query performance. When using it, the following points should be noted:

> **Note**
>
> - SQL Cache is not suitable for queries containing functions that generate random values (such as `random()`), as this will cause the query results to lose randomness.
> - Currently, it does not support using the cached results of some metrics to meet the needs of querying more metrics. For example, the cache for previously queried two metrics cannot be used for the situation of querying three metrics.
> - By reasonably using SQL Cache, the query performance of Doris can be significantly improved, especially in scenarios with a low data update frequency. In practical applications, cache parameters need to be adjusted according to specific data characteristics and query patterns to achieve the best performance improvement. :::

DML Tuning Plan

For DML plan tuning, it is first necessary to identify whether the performance bottleneck is caused by the import process or the query section. For the troubleshooting and tuning of performance bottlenecks in the query section, please refer to other subsections in Plan Tuning for details.

Doris supports importing data from multiple data sources. By flexibly utilizing the various import functions provided by Doris, data from various sources can be efficiently imported into Doris for analysis. For details of best practices, please refer to Import Overview.

Adjustment of RuntimeFilter Wait Time

Adjustment of RuntimeFilter Wait Time

Overview

In actual production scenarios, there may be performance issues caused by unreasonable RuntimeFilter wait times. RuntimeFilter is a query optimization technique that generates filter conditions at runtime, thereby avoiding scanning irrelevant data. This optimization method can significantly reduce I/O operations and computational load, thus accelerating query execution. The following sections introduce several common cases to help with optimization in data skew scenarios.

Case: Too Short RuntimeFilter Wait Time

Refer to the following Profile information:

```
OLAP_SCAN_OPERATOR (id=22. nereids_id=1764. table name = test_doris(test_doris)):(ExecTime: 62.870ms)
               - RuntimeFilters: : RuntimeFilter: (id = 6, type = minmax, need_local_merge: true, is_broadcast: false,
build_bf_cardinality: false, RuntimeFilter: (id = 7, type = in_or_bloomfilter, need_local_merge: true, is_broadcast: false,
build_bf_cardinality: false,
               - PushDownPredicates: []
               - KeyRanges: ScanKeys:ScanKey=[null(-9223372036854775808) : 9223372036854775807]
               - TabletIds: [1732763414173, 1732763414187, 1732763414201, 1732763414215]
               - UseSpecificThreadToken: False
               - AcquireRuntimeFilterTime: 969ns
               - BlocksProduced: 1.829K (1829)
               - CloseTime: 0ns
               - ExecTime: 62.870ms
               - InitTime: 75.703us
               - KeyRangesNum: 0
               - MaxScannerThreadNum: 32
               - MemoryUsage:
                 - PeakMemoryUsage: 0.00
               - NumScanners: 32
               - OpenTime: 19.276ms
               - ProcessConjunctTime: 30.360us
               - ProjectionTime: 0ns
               - RowsProduced: 7.433056M (7433056)
               - RowsRead: 0
               - RuntimeFilterInfo:
               - ScannerWorkerWaitTime: 0ns
               - TabletNum: 4
               - TotalReadThroughput: 0
               - WaitForDependency[OLAP_SCAN_OPERATOR_DEPENDENCY]Time: 0ns
               - WaitForRuntimeFilter: 1000ms
             RuntimeFilter: (id = 6, type = minmax):
                - Info: [IsPushDown = false, RuntimeFilterState = NOT_READY, HasRemoteTarget = true, HasLocalTarget = false,
Ignored = false]
             RuntimeFilter: (id = 7, type = in_or_bloomfilter):
                - Info: [IsPushDown = false, RuntimeFilterState = NOT_READY, HasRemoteTarget = true, HasLocalTarget = false,
Ignored = false]
```

From the Profile, we can see that `WaitForRuntimeFilter: 1000ms`. Here, the RuntimeFilter waited for 1000ms, but this ScanOperator did not receive the corresponding RuntimeFilter, and `RuntimeFilterState = NOT_READY`.

```
 RuntimeFilter: (id = 6, type = minmax):
                 - Info: [IsPushDown = false, RuntimeFilterState = NOT_READY, HasRemoteTarget = true, HasLocalTarget = false,
Ignored = false]
 RuntimeFilter: (id = 7, type = in_or_bloomfilter):
                 - Info: [IsPushDown = false, RuntimeFilterState = NOT_READY, HasRemoteTarget = true, HasLocalTarget = false,
Ignored = false]
```

So, the corresponding RuntimeFilters with ids 6 and 7 were not received. By locating the Join that generates the RuntimeFilter through the Profile, we found that the Join took time:

```
  HASH_JOIN_OPERATOR (id=26, nereids_id=37948):
                - PlanInfo
                   - join op: RIGHT OUTER JOIN(PARTITIONED)[]
                   - equal join conjunct: (id = ID)
                   - runtime filters: RF006[min_max] <- ID(6418/8192/1048576), RF007[in_or_bloom] <- ID(6418/8192/1048576)
                   - cardinality=6,418
                   - vec output tuple id: 27
                   - output tuple id: 27
                   - vIntermediate tuple ids: 25
                   - hash output slot ids: 396 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 447
                   - projections: USER_ID
                   - project output tuple id: 27
                - BlocksProduced: sum 1, avg 1, max 1, min 1
                - CloseTime: avg 10.111us, max 10.111us, min 10.111us
                - ExecTime: avg 364.497us, max 364.497us, min 364.497us
                - InitTime: avg 26.653us, max 26.653us, min 26.653us
                - MemoryUsage: sum, avg, max, min
                  - PeakMemoryUsage: sum 0.00, avg 0.00, max 0.00, min 0.00
                  - ProbeKeyArena: sum 0.00, avg 0.00, max 0.00, min 0.00
                - OpenTime: avg 45.985us, max 45.985us, min 45.985us
                - ProbeRows: sum 0, avg 0, max 0, min 0
                - ProjectionTime: avg 211.930us, max 211.930us, min 211.930us
                - RowsProduced: sum 1, avg 1, max 1, min 1
                - WaitForDependency[HASH_JOIN_OPERATOR_DEPENDENCY]Time: avg 1sec780ms, max 1sec780ms, min 1sec780ms
```

It can be seen that this Join took approximately `1sec780ms`, so the RuntimeFilter did not wait in 1s. Therefore, the RuntimeFilter wait time was adjusted:

```
set runtime_filter_wait_time_ms = 3000;
```

After the adjustment, the query time was reduced from 5s to 2s.

Summary

The wait time of RuntimeFilter needs to be defined according to the scenario. Doris is undergoing some adaptive optimization and transformation. Use the EXPLAIN and PROFILE tools to observe the execution bottleneck, locate the corresponding problem, and modify the RuntimeFilter wait time through SQL Hint to avoid the impact of the corresponding problem on performance.

Data Skew Handling

Data Skew Handling

Overview

Doris is an MPP database that relies on data shuffle for parallel computing acceleration. However, in actual production scenarios, performance bottlenecks in single threads of query parallelism are often encountered due to data skew. The following sections introduce how to identify such problems and provide some general solutions.

Case 1: Bucket Data Skew Leading to Suboptimal Shuffle Method

When data skew occurs on the Join Key of a table, the data will be unevenly distributed among different BE instances, resulting in a single-point execution bottleneck and thus slowing down the overall query execution time.

```
HASH_JOIN_OPERATOR  (id=27):
      -  PlanInfo
            -  join  op: INNER  JOIN(PARTITIONED)[]
            -  equal  join  conjunct:  (customer_number  =  customer_number)
            -  runtime  filters:  RF001[bloom]  <-  customer_number(200/256/2048)
            -  cardinality=200
            -  vec  output  tuple  id:  28
            -  output  tuple  id:  28
            -  vIntermediate  tuple  ids:  27
            -  hash  output  slot  ids:  192  193  194  195  196  197  198  199  200  201  174  175  240  176  177  178  179  180
181  182  183  184  185  186  187  188  189  190  191
            -  project  output  tuple  id:  28
      -  BlocksProduced:  sum  4.883K  (4883),  avg  33,  max  39,  min  29
      -  CloseTime:  avg  37.28us,  max  132.653us,  min  13.945us
      -  ExecTime:  avg  166.206ms,  max  10s947.344ms,  min  8.845ms
      -  InitTime:  avg  0ns,  max  0ns,  min  0ns
      -  MemoryUsage:  sum ,  avg ,  max ,  min
          -  PeakMemoryUsage:  sum  11.81  MB,  avg  84.00  KB,  max  84.00  KB,  min  84.00  KB
          -  ProbeKeyArena:  sum  11.81  MB,  avg  84.00  KB,  max  84.00  KB,  min  84.00  KB
      -  OpenTime:  avg  194.970us,  max  497.685us,  min  93.738us
      -  ProbeRows:  sum  23.884018M  (23884018),  avg  165.861K  (165861),  max  219.346276M  (219346276),  min  1984  (1984)
      -  ProjectionTime:  avg  7.336ms,  max  33.540ms,  min  3.760ms
      -  RowsProduced:  sum  28.8K  (28800),  avg  200,  max  200,  min  200
```

From the max indicators in the above Join's Profile, there is an obvious skew in the execution time and ProbeRows.

```
ExecTime:  avg  166.206ms,  max  10s947.344ms,  min  8.845ms
ProbeRows:  sum  23.884018M  (23884018),  avg  165.861K  (165861),  max  219.346276M  (219346276),  min  1984  (1984)
```

However, due to the uneven distribution of data after shuffling based on the join key, one thread may process 200 million rows of data while another thread only processes a few thousand rows. In the ideal case of the above scenario, each thread should process approximately the same amount of data. But due to the data skew

problem of the Join column, a large amount of computational work may be completed by a single thread. To solve this performance bottleneck, refer to the tuning techniques mentioned in the “Using Hint to Control Join Shuffle Method” section and specify the broadcast join hint as follows to prevent the left table from shuffling data, thus effectively avoiding the performance bottleneck caused by data skew on the Join column.

```
SELECT COUNT(*) FROM orders o JOIN [broadcast] customer c ON o.customer_number = c.customer_number;
```

Case 2: Column Data Skew Leading to Reversed Join Sides

The current Doris optimizer estimates the selectivity based on the assumption of uniform data distribution. Large deviations in the estimated number of rows after filtering can affect the operator’s plan selection. Take the following SQL as an example:

```
select count(*)
from orders, customer
where o_custkey = c_custkey
and o_orderdate < '1920-01-02';
```

Under the assumption of uniform distribution, the optimizer may think that the number of rows output after filtering by `o_orderdate < '1920-01-02'` will be less than the number of rows in the `customer` table. Therefore, it may choose the join order of `customer` join `orders`. However, if the actual data is skewed and the number of rows in the `orders` table that satisfy the condition is greater than that in the `customer` table, then a more reasonable join order should be `orders` join `customer`. To solve this performance problem, refer to the tuning techniques mentioned in the “Using Leading Hint to Control Join Order” section and specify the leading hint as follows to force the generation of the join order of `customer` join `orders`.

Rewrite the SQL as follows:

```
select /*+leading(orders customer)*/ count(*)
from orders, customer
where o_custkey = c_custkey
and o_orderdate < '1920-01-02'
```

Summary

Data skew is a common performance problem in production scenarios. By observing the plan and execution bottlenecks through the EXPLAIN and PROFILE tool outputs, locating the cause of the skew, and then using the Hint tool to make corresponding plan adjustments, the impact of data skew on performance can be avoided.

Parallelism Tuning

Overview

Doris queries are executed in an MPP (Massively Parallel Processing) framework, where each query is executed in parallel across multiple BEs (Backend Executors). Meanwhile, within a single BE, a multi-threaded parallel approach is adopted to enhance query execution efficiency. Currently, all types of statements, including queries, DML (Data Manipulation Language), and DDL (Data Definition Language), support parallel execution.

The control parameter for parallelism within a single BE is `parallel_pipeline_task_num`, which refers to the number of working tasks used by a single Fragment during execution. In actual production scenarios, performance issues may arise due to improper parallelism settings. The following examples illustrate cases of optimizing parallelism.

Principles of Parallelism Tuning

The purpose of setting `parallel_pipeline_task_num` is to fully utilize multi-core resources and reduce query latency. However, to enable multi-core parallel execution, some data shuffle operators and synchronization logic between multiple threads are usually introduced, which may also lead to unnecessary resource wastage.

The default value in Doris is 0, which is half the number of CPU cores of the BE. This value takes into account the resource utilization of both single queries and concurrent operations, and usually does not require user intervention for adjustment. When there is a performance bottleneck, refer to the following examples for necessary adjustments. Doris is continuously improving its adaptive strategy, and it is usually recommended to make necessary adjustments in specific scenarios or at the SQL level.

Suppose the BE has 16 CPU cores:

1. For simple operations on a single table (such as single-table point queries, `WHERE` clause scans to retrieve a small amount of data, `LIMIT` a small amount of data, or hitting a materialized view), **the parallelism can be set to 1**.

   Explanation: Simple operations on a single table involve only one Fragment. The bottleneck of such queries usually lies in data scanning and processing. The data scanning thread and the query execution thread are separated, and the data scanning thread will perform parallel scanning adaptively. Here, the bottleneck is not the query thread, so the parallelism can be directly set to 1.

2. For queries involving two-table `JOIN` or aggregation queries, if the data volume is large and it is confirmed to be a CPU-bound query, **the parallelism can be set to 16**.

   Explanation: For two-table `JOIN` or aggregation queries, which are data computation-intensive queries, if the CPU is not fully utilized, consider increasing the parallelism on the basis of the default value to take advantage of the parallel capabilities of the Pipeline execution engine and fully utilize CPU resources for computation. It cannot be guaranteed that each PipelineTask can utilize the allocated CPU resources to the fullest. Therefore, the parallelism can be adjusted appropriately, for example, set to 16, to make better use of the CPU. However, the parallelism should not be increased indefinitely. Setting it to 48 will

not bring substantial benefits and will instead increase thread scheduling overhead and framework scheduling overhead.

3. In a stress testing scenario, where the multiple queries in the stress test can fully utilize the CPU, **the parallelism can be set to 1**.

   Explanation: In a stress testing scenario, there are sufficient query tasks. Excessive parallelism also brings thread scheduling overhead and framework scheduling overhead. Setting it to 1 is more reasonable in this case.

4. For complex queries, the parallelism should be adjusted flexibly based on the Profile and machine load. Here, it is recommended to use the default value. If it is not suitable, a stepwise adjustment of 4-2-1 can be tried, and the query performance and machine load should be observed.

Methods of Parallelism Tuning

Doris allows users to manually specify the parallelism of a query to adjust the parallel execution efficiency during query execution.

SQL Level Adjustment

Use SQL HINT to specify the parallelism of a single SQL statement. This allows for flexible control of the parallelism of different SQL statements to achieve the best execution results.

```
select /*+SET_VAR("parallel_pipeline_task_num=8")*/ * from nation, lineitem where lineitem.l_suppkey = nation.n_nationkey
select /*+SET_VAR("parallel_pipeline_task_num=8,runtime_filter_mode=global")*/ * from nation, lineitem where lineitem.l_suppkey =
nation.n_nationkey
```

Session Level Adjustment

Adjust the parallelism at the session level through session variables. All query statements in the session will be executed with the specified parallelism. Please note that even single-line SQL queries will use this parallelism, which may lead to performance degradation.

```
set parallel_pipeline_task_num = 8;
```

Global Adjustment

If global adjustment is required, usually involving CPU utilization adjustment, the parallelism can be set globally.

```
set global parallel_pipeline_task_num = 8;
```

Tablets and parallelism

Since version 2.1, Doris supports decoupling parallelism from the number of tablets.

In previous versions, parallelism could not exceed the number of tablets involved in the query. For example, if a query involved 5 tablets, the maximum scan concurrency was only 5. This could prevent some large tablets

from being read concurrently.

In the new version, Doris supports concurrent reads within a shard. This feature is enabled automatically and requires no user configuration.

Note that this feature only supports the Duplicate and Unique Key Merge-On-Write table models. It does not apply to the Aggregate and Unique Key Merge-On-Read models. For these two models, query parallelism is still constrained by the number of tablets.

Best Practice

Case 1: High Parallelism Leading to High CPU Usage in a High-Concurrency Pressure Scenario

When observing high CPU usage online, which affects the performance of some low-latency queries, consider adjusting the query parallelism to reduce CPU usage. Since Doris's design philosophy is to prioritize using more resources to obtain query results as quickly as possible, in some scenarios with tight online resources, this may lead to poor performance. Therefore, appropriate adjustment of parallelism can improve the overall stability and efficiency of queries under limited resources.

Set the parallelism from the default value of 0 (half the number of CPU cores) to 4:

```
set global parallel_pipeline_task_num = 4;
```

After the global setting, it takes effect for the current connection and new connections. Existing other connections are not affected. If immediate global effect is required, the FE (Frontend) can be restarted. After the adjustment, the CPU usage is reduced to 60% of the previous peak value, reducing the impact on some low-latency queries.

Case 2: Increasing Parallelism to Further Utilize the CPU for Query Acceleration

The current default parallelism in Doris is half the number of CPU cores, and some computation-intensive scenarios cannot fully utilize the CPU for query acceleration.

```
select sum(if(t2.value is null, 0, 1)) exist_value, sum(if(t2.value is null, 1, 0)) no_exist_value
from  t1 left join  t2 on t1.key = t2.key;
```

In a scenario with 2 billion rows in the left table and 5 million rows in the right table, the above SQL takes 28 seconds to execute. Observe the Profile:

```
HASH_JOIN_OPERATOR (id=3, nereids_id=448):
                - PlanInfo
                   - join op: LEFT OUTER JOIN(BROADCAST)[]
                   - equal join conjunct: (value = value)
                   - cardinality=2,462,330,332
                   - vec output tuple id: 5
                   - output tuple id: 5
                   - vIntermediate tuple ids: 4
                   - hash output slot ids: 16
                   - projections: value
                   - project output tuple id: 5
```

```
                - BlocksProduced: sum 360.099K (360099), avg 45.012K (45012), max 45.014K (45014), min 45.011K (45011)
                - CloseTime: avg 8.44us, max 13.327us, min 5.574us
                - ExecTime: avg 26sec153ms, max 26sec261ms, min 26sec33ms
                - InitTime: avg 7.122us, max 13.395us, min 4.541us
                - MemoryUsage: sum, avg, max, min
                  - PeakMemoryUsage: sum 1.16 MB, avg 148.00 KB, max 148.00 KB, min 148.00 KB
                  - ProbeKeyArena: sum 1.16 MB, avg 148.00 KB, max 148.00 KB, min 148.00 KB
                - OpenTime: avg 2.967us, max 4.120us, min 1.562us
                - ProbeRows: sum 1.462330332B (1462330332), avg 182.791291M (182791291), max 182.811875M (182811875), min
182.782658M (182782658)
                - ProjectionTime: avg 165.392ms, max 169.762ms, min 161.727ms
                - RowsProduced: sum 1.462330332B (1462330332), avg 182.791291M (182791291), max 182.811875M (182811875), min
182.782658M (182782658)
```

The main time-consuming part here: `ExecTime: avg 26sec153ms, max 26sec261ms, min 26sec33ms` all occurs in the Join operator, and the total amount of data processed: `ProbeRows: sum 1.4662330332B` is 1.4 billion, which is a typical CPU-intensive computation scenario. Observing the machine monitoring, it is found that the CPU resources are not fully utilized, with a CPU utilization rate of 60%. At this time, consider increasing the parallelism to further utilize the idle CPU resources for acceleration.

Set the parallelism as follows:

```
set parallel_pipeline_task_num = 16;
```

The query execution time is reduced from 28 seconds to 19 seconds, and the CPU utilization rate is increased from 60% to 90%.

Summary

Usually, users do not need to adjust the query parallelism. If adjustment is required, the following points should be noted:

1. It is recommended to start from the CPU utilization. Observe whether it is a CPU bottleneck through the PROFILE tool output and try to make reasonable modifications to the parallelism.
2. Adjusting a single SQL is relatively safe. Try not to make overly aggressive global modifications.

Common Tuning Parameters

Parameter	Description	Default Value	Usage Scenario
enable_nereids_planner	Whether to enable the new optimizer	TRUE	For scenarios such as low-version upgrades, initially set to false; after upgrading, it can be set to true
enable_nereids_dml	Whether to enable DML support for the new optimizer	TRUE	For scenarios such as low-version upgrades, initially set to false; after upgrading, it can be set to true
parallel_pipeline_task_num	Pipeline parallelism	0	For scenarios such as low-version upgrades, this value was previously set to a fixed value; after upgrading, it can be set to 0, indicating that the system's adaptive strategy determines the parallelism
runtime_filter_mode	Runtime Filter type	GLOBAL	For scenarios such as low-version upgrades, this value was NONE, indicating that Runtime Filter was not enabled; after upgrading, it can be set to GLOBAL, indicating that Runtime Filter is enabled by default

Query Optimizers

In the current information technology landscape, query optimizers face multiple challenges: on the one hand, they need to handle increasingly complex query statements and diverse query scenarios from users; on the other hand, users have increasingly strict demands for query real-time performance, desiring instant access to required results. Furthermore, to address emerging new requirements, query optimizers must possess the capabilities of rapid iteration and flexible adaptation.

Based on this background, Doris embarked on the development of a brand-new query optimizer. Leveraging a modern optimizer architecture, this optimizer aims to more efficiently tackle query requests in the current Doris scenarios while providing exceptional scalability, laying a solid foundation for potentially more complex future demands.

Advantages of the Optimizer

Smarter

The optimizer clearly presents each optimization point of RBO (Rule-Based Optimization) and CBO (Cost-Based Optimization) in the form of rules. For each rule, the optimizer provides a set of patterns that describe the shape of the query plan, enabling precise matching of optimizable query plans. Therefore, the optimizer can better support more complex query statements such as nested multi-layer subqueries.

Meanwhile, the optimizer's CBO is based on the advanced Cascades framework, fully utilizing rich statistical data, data characteristic information, and a meticulously tuned cost model. This empowers the optimizer to handle complex queries like multi-table joins with ease and proficiency.

More Stable

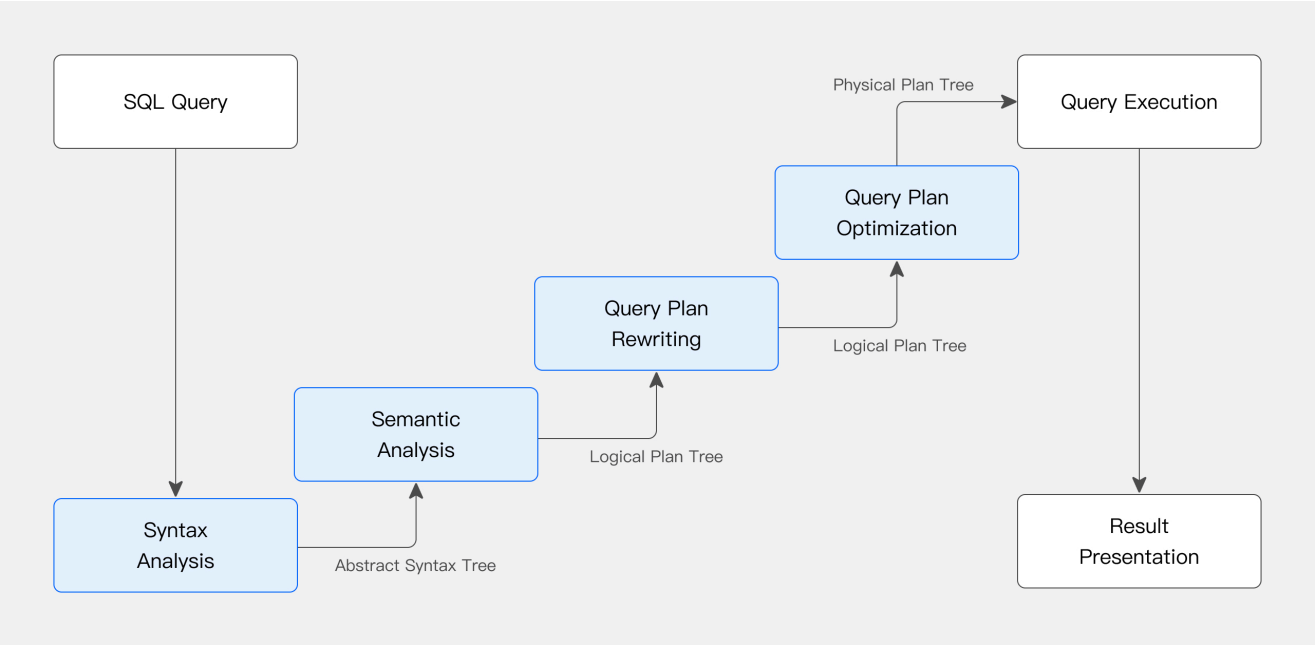
All optimization rules of the optimizer are executed on the logical execution plan tree. After parsing the query syntax and semantics, the query is transformed into a tree structure. Compared to the old optimizer, the new optimizer's internal data structure is more reasonable and unified.

Taking subquery processing as an example, the new optimizer, based on its new data structure, avoids the separate handling of subqueries by numerous rules in the old optimizer, thereby reducing the likelihood of logical errors in optimization rules.

More Flexible

The optimizer's architecture is reasonably and modernly designed, making it very convenient to extend optimization rules and processing stages. Therefore, we can swiftly add new functionalities to meet evolving new requirements.

Principles of the Optimizer



Principles of the Optimizer

The execution process of the optimizer is divided into the following steps:

1. **Syntax Analysis**: The optimizer attempts to convert the SQL text into an Abstract Syntax Tree (AST). If the SQL text is valid, it proceeds to the next steps; if invalid, it reports an error and terminates execution.
2. **Semantic Analysis**: The optimizer performs semantic analysis on the elements in the AST. This step checks whether tables, columns, functions, etc., in the SQL query exist and whether their usage complies with syntax and semantic rules. If the semantics are valid, execution continues; if invalid, it reports an error and terminates execution.
3. **Rewrite Query Plan (RBO):** After syntax and semantic analysis, the optimizer performs Rule-Based Optimization (RBO). This step rewrites the query plan through a series of predefined rules to deterministically optimize execution speed. Common optimization techniques include column pruning, predicate pushdown, partition pruning, etc.
4. **Optimize Query Plan (CBO)**: Finally, the optimizer performs Cost-Based Optimization (CBO). In this step, the optimizer enumerates equivalent plan sets in the search space and evaluates their execution costs. By comparing the execution costs of different plans, the optimizer selects the plan with the lowest cost as the final execution plan. This step aims to ensure that queries are executed in the most efficient manner, thereby providing optimal performance.

Session Variables

1. Set Planning Timeout `nereids_timeout_second`

- This variable is used to set the maximum allowed time for query planning. When the planning time exceeds this set value, query planning will be terminated, and an error message will be returned. During the process of planning query statements, the system obtains read locks for all tables involved in the

SQL, primarily to maintain cluster stability and prevent excessive resource occupation and lock conflicts caused by excessively long planning times.

- Default value: 30s
- Applicable scenarios: When queries involve a large number of external tables or particularly complex query statements, this value can be appropriately increased to ensure that queries can proceed normally.

Parallel Execution

The parallel execution model of Doris is a Pipeline execution model, primarily inspired by the implementation described in the Hyper paper. The Pipeline execution model fully leverages the computational power of multi-core CPUs while limiting the number of query threads in Doris, addressing the issue of thread explosion during execution. For details on its design, implementation, and effectiveness, refer to DSIP-027 and DSIP-035.

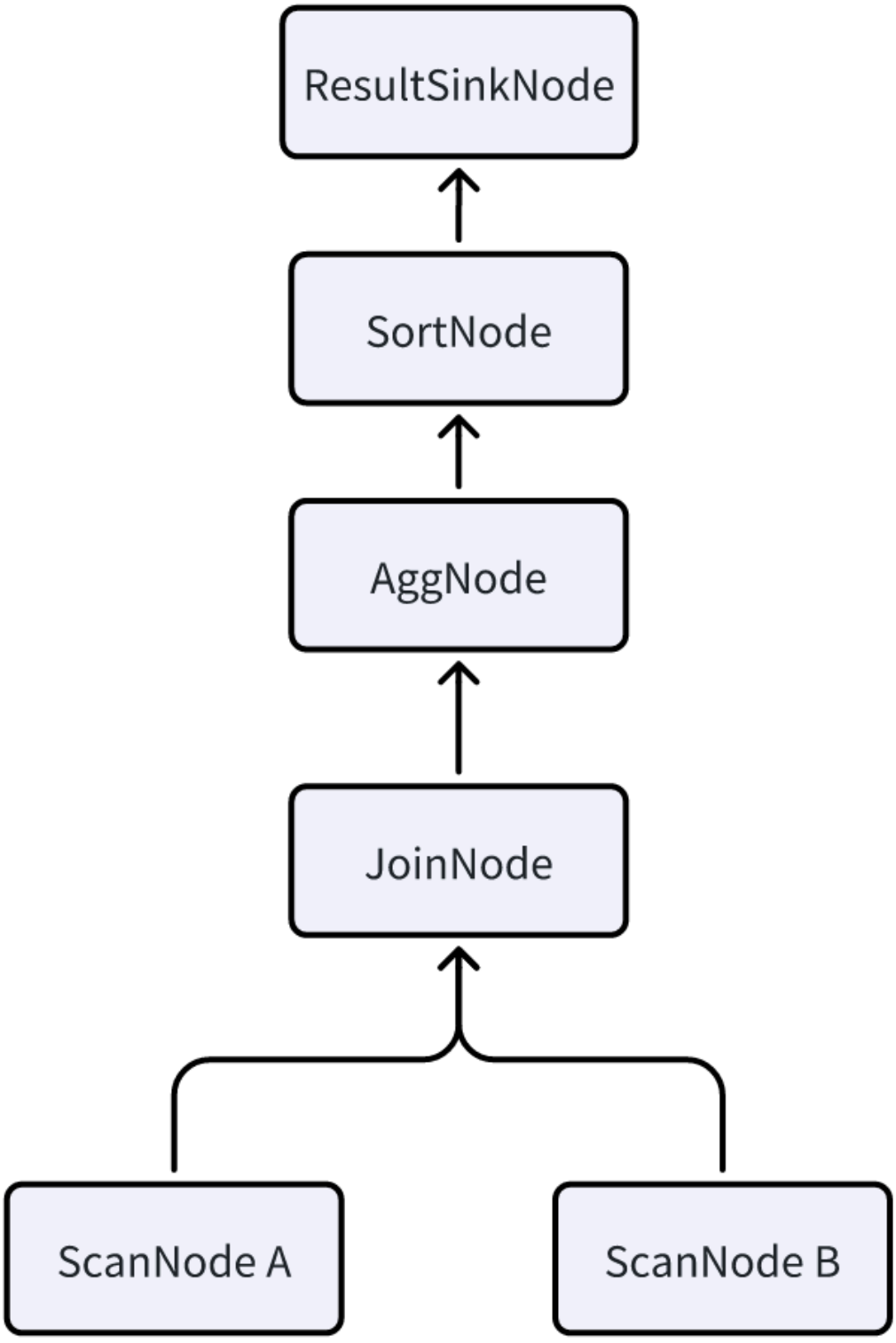
Starting from Doris 3.0, the Pipeline execution model has completely replaced the original Volcano model. Based on the Pipeline execution model, Doris supports the parallel processing of Query, DDL, and DML statements.

Physical Plan

To better understand the Pipeline execution model, it is first necessary to introduce two important concepts in the physical query plan: PlanFragment and PlanNode. We will use the following SQL statement as an example:

```
SELECT k1, SUM(v1) FROM A,B WHERE A.k2 = B.k2 GROUP BY k1 ORDER BY SUM(v1);
```

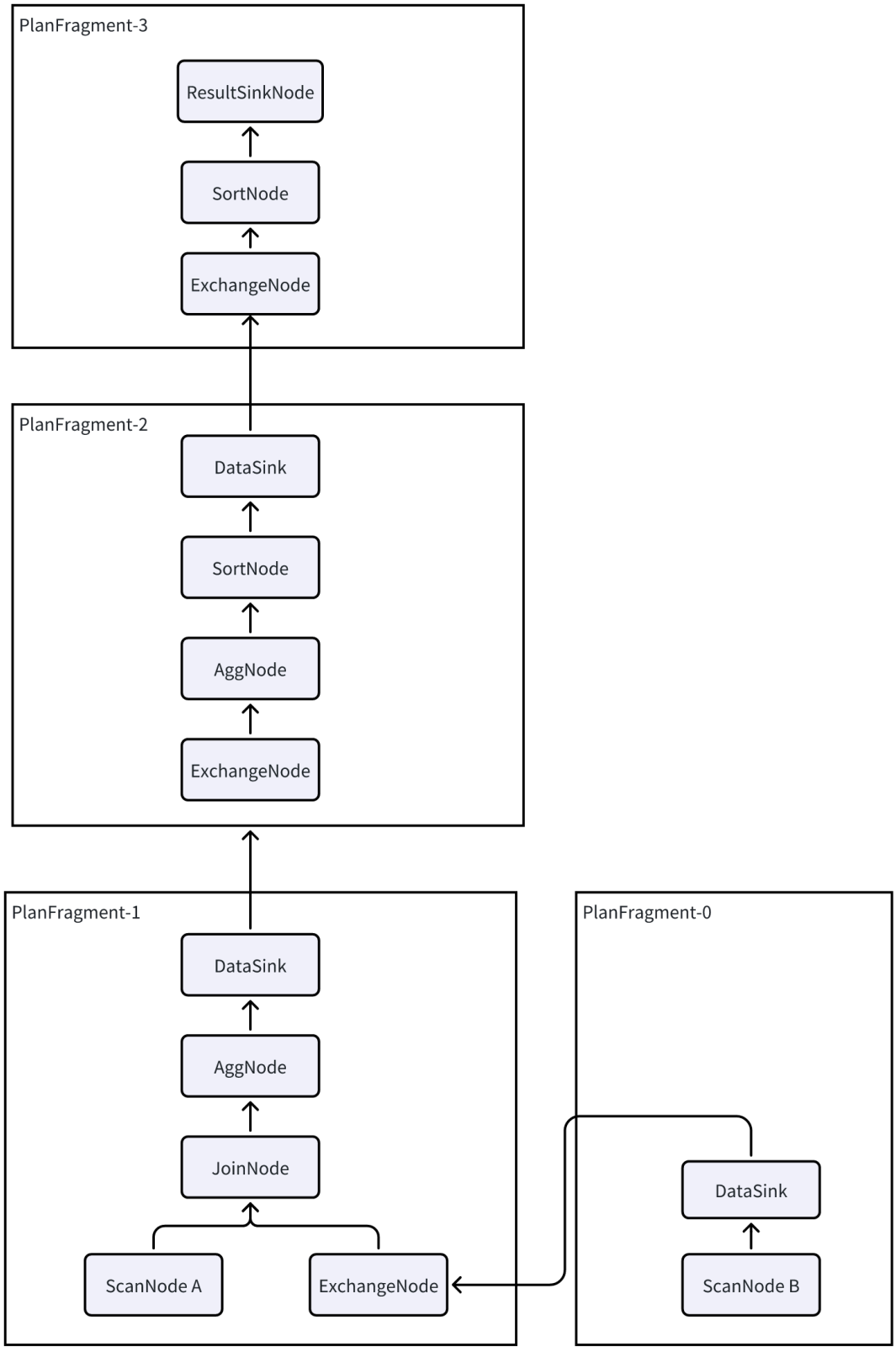
FE will first translate it into the following logical plan, each node represents a PlanNode. The detail meaning of each node type can be found in the introduction of physical plan.



pip_exec_1

Since Doris is built on an MPP architecture, each query aims to involve all BEs in parallel execution as much as possible to reduce query latency. Therefore, the logical plan must be transformed into a physical plan. The transformation essentially involves inserting DataSink and ExchangeNode into the logical plan. These two nodes facilitate the shuffling of data across multiple BEs.

After the transformation, each PlanFragment corresponds to a portion of the PlanNode and can be sent as an independent task to a BE. Each BE processes the PlanNode contained within the PlanFragment and then uses the DataSink and ExchangeNode operators to shuffle data to other BEs for subsequent computation.



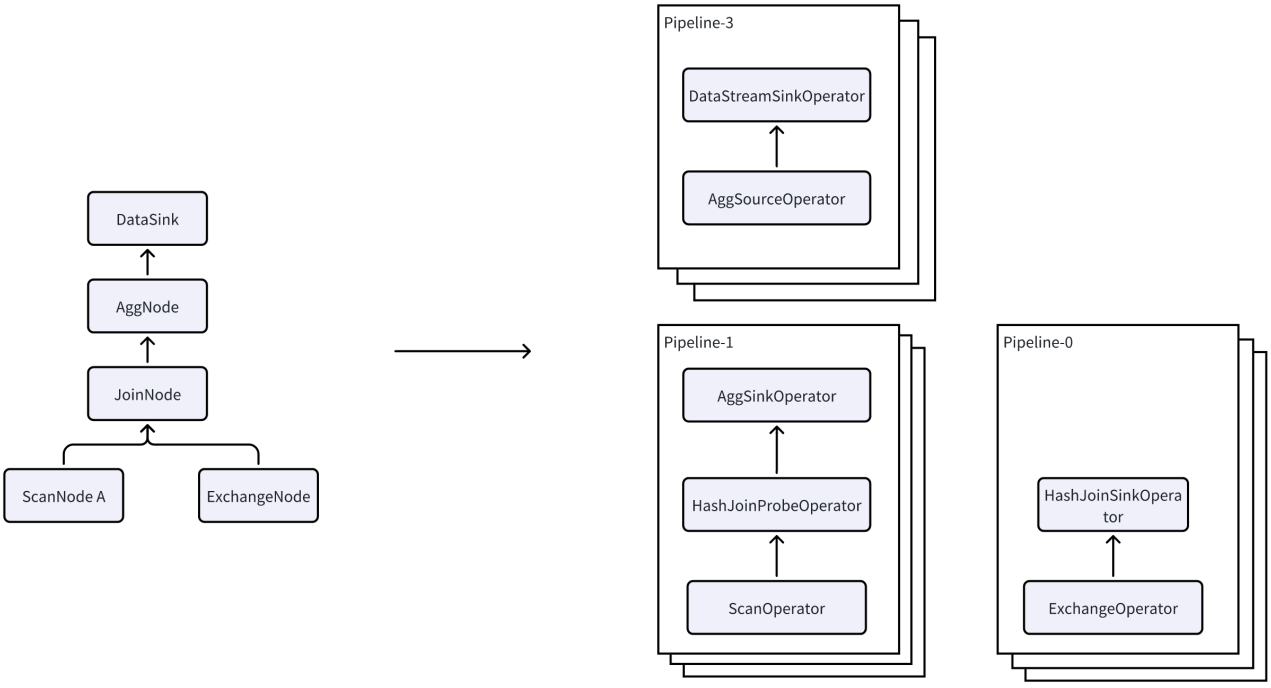
pip_exec_2

Doris's plan is divided into three layers:

- PLAN: The execution plan. A SQL statement is translated by the query planner into an execution plan, which is then provided to the execution engine for execution.
- FRAGMENT: Since Doris is a distributed execution engine, a complete execution plan is divided into multiple single-machine execution fragments. A FRAGMENT represents a complete single-machine execution fragment. Multiple fragments combine to form a complete PLAN.
- PLAN NODE: Operators, which are the smallest units of the execution plan. A FRAGMENT consists of multiple operators, each responsible for a specific execution logic, such as aggregation or join operations.

Pipeline Execution

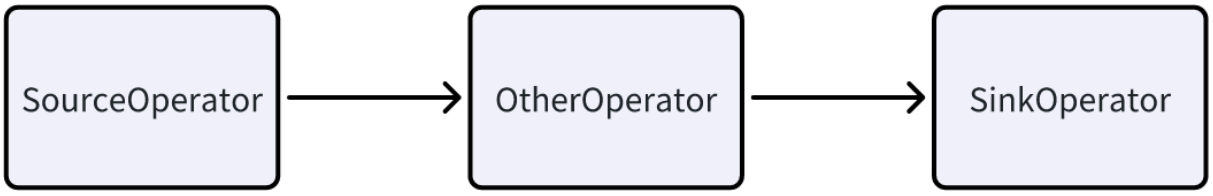
A PlanFragment is the smallest unit of a task sent by the FE to the BE for execution. A BE may receive multiple different PlanFragments for the same query, and each PlanFragment is processed independently. Upon receiving a PlanFragment, the BE splits it into multiple Pipelines and then starts multiple PipelineTasks to achieve parallel execution, thereby improving query efficiency.



pip_exec_3

Pipeline

Pipeline consists of a SourceOperator, a SinkOperator, and several intermediate operators. The SourceOperator represents reading data from an external source, which can be a table (e.g., OlapTable) or a buffer (e.g., Exchange). The SinkOperator represents the data output, which can either be shuffled to other nodes over the network (e.g., DataStreamSinkOperator) or output to a hash table (e.g., aggregation operators, join build hash tables, etc.).



pip_exec_4

Multiple Pipelines are actually interdependent. Take the JoinNode as an example—it is split into two Pipelines. Pipeline-0 reads data from Exchange to build the hash table, while Pipeline-1 reads data from the table to perform the probe operation. These two Pipelines are connected by a dependency relationship, meaning Pipeline-1 can only execute after Pipeline-0 has completed. This dependency relationship is referred to as a Dependency. Once Pipeline-0 finishes execution, it calls the set_ready method of the Dependency to notify Pipeline-1 that it is ready to execute.

PipelineTask

Pipeline is actually a logical concept; it is not an executable entity. Once a Pipeline is defined, it needs to be further instantiated into multiple PipelineTasks. The data that needs to be read is then distributed to different PipelineTasks, ultimately achieving parallel processing. The operators within the multiple PipelineTasks of the same Pipeline are identical, but they differ in their states. For example, they might read different data or build different hash tables. These differing states are referred to as LocalState.

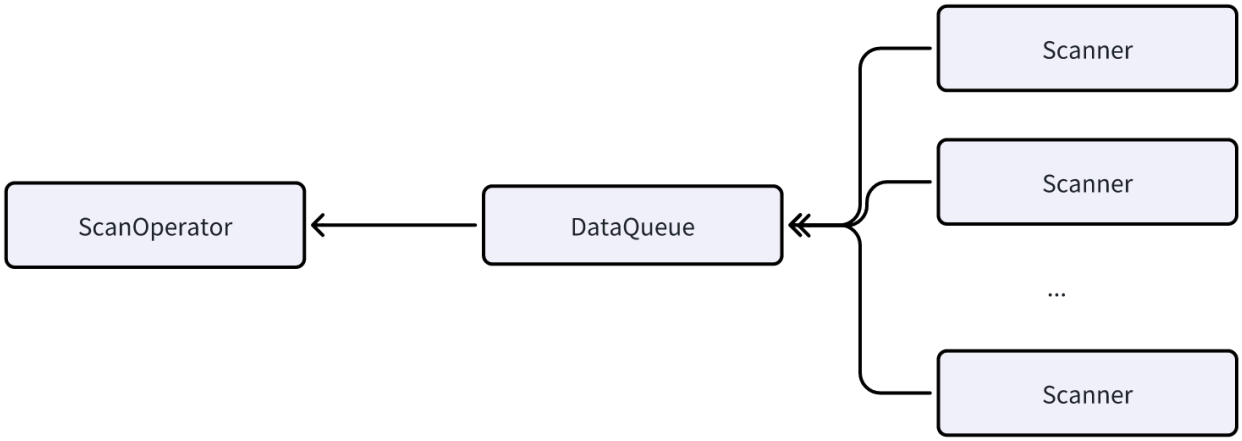
Each PipelineTask is eventually submitted to a thread pool to be executed as an independent task. With the Dependency trigger mechanism, this approach allows better utilization of multi-core CPUs and achieves full parallelism.

Operator

In most cases, each operator in a Pipeline corresponds to a PlanNode, but there are some special operators with exceptions: * JoinNode is split into JoinBuildOperator and JoinProbeOperator. * AggNode is split into AggSinkOperator and AggSourceOperator. * SortNode is split into SortSinkOperator and SortSourceOperator. The basic principle is that for certain “breaking” operators (those that need to collect all the data before performing computation), the data ingestion part is split into a Sink, while the part that retrieves data from the operator is referred to as the Source.

Parallel Scan

Scanning data is a very heavy I/O operation, as it requires reading large amounts of data from local disks (or from HDFS or S3 in the case of data lake scenarios, which introduces even longer latency), consuming a significant amount of time. Therefore, we have introduced parallel scanning technology in the ScanOperator. The ScanOperator dynamically generates multiple Scanners, each of which scans around 1 to 2 million rows of data. While performing the scan, each Scanner handles tasks such as data decompression, filtering, and other calculations, and then sends the data to a DataQueue for the ScanOperator to read.



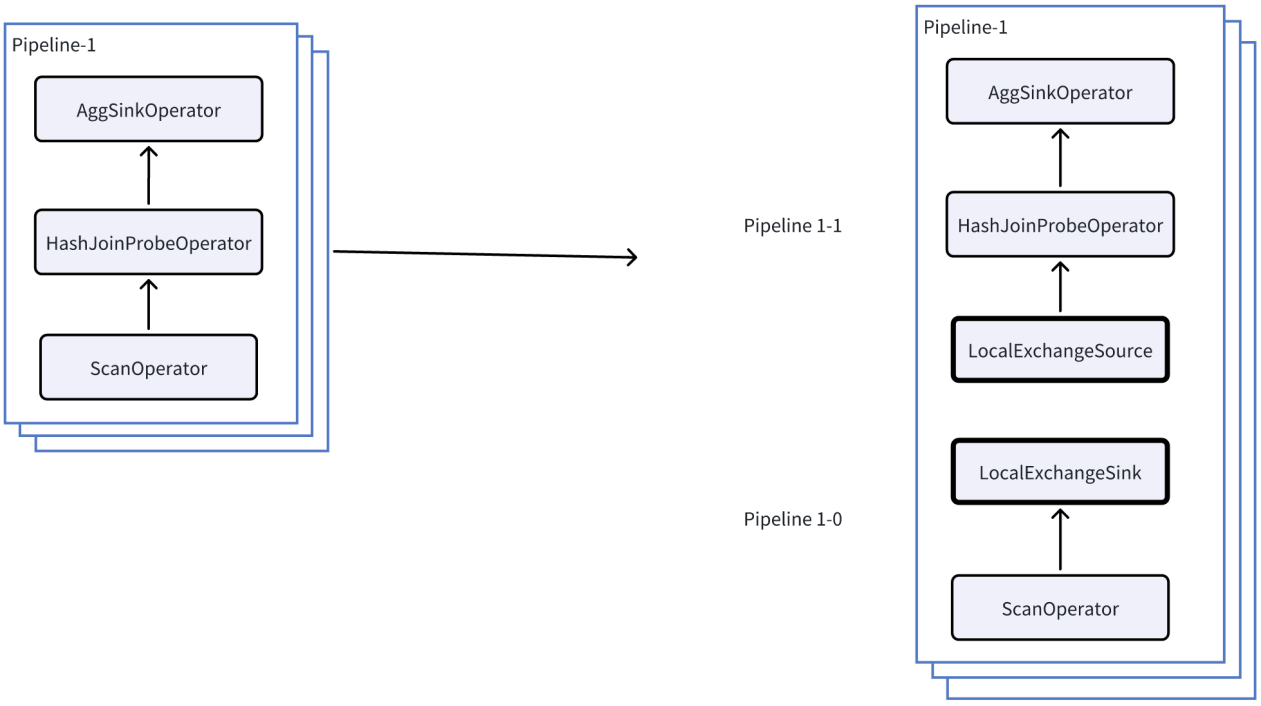
pip_exec_5

By using parallel scanning technology, we can effectively avoid issues where certain ScanOperators take an excessively long time due to improper bucketing or data skew, which would otherwise slow down the entire query latency.

Local Shuffle

In the Pipeline execution model, Local Shuffle acts as a Pipeline Breaker, a technique that redistributes data locally across different execution tasks. It evenly distributes all the data output by the upstream Pipeline to all the tasks in the downstream Pipeline using methods like HASH or Round Robin. This helps solve the problem of data skew during execution, ensuring that the execution model is no longer limited by data storage or the query plan. Let's now provide an example to illustrate how Local Exchange works.

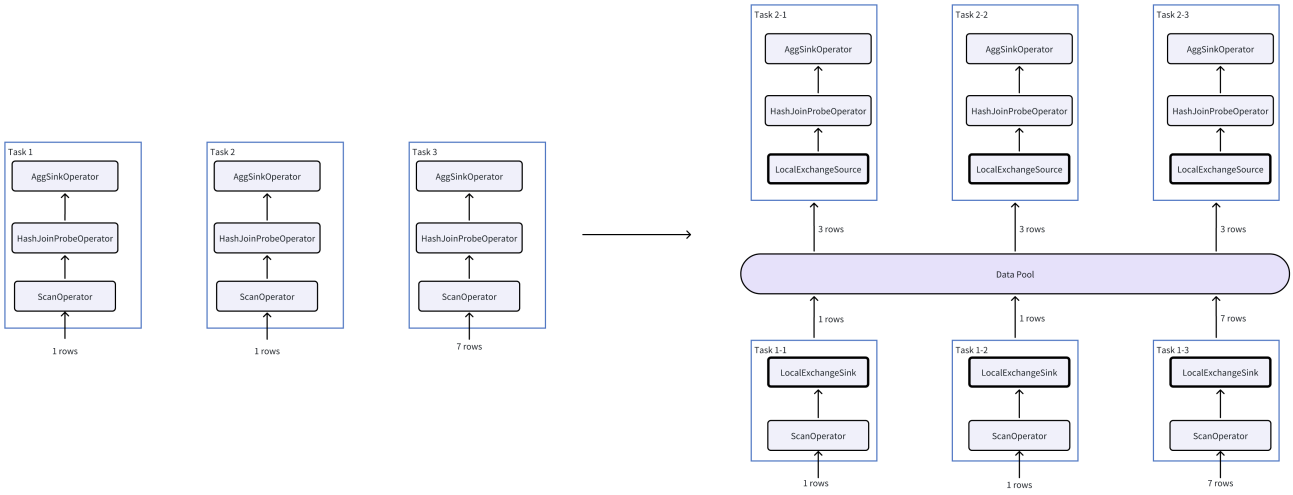
We will further explain how Local Exchange can prevent data skew using Pipeline-1 from the previous example.



pip_exec_6

As shown in the figure above, by inserting a Local Exchange in Pipeline-1, we further split Pipeline-1 into Pipeline-1-0 and Pipeline-1-1.

Now, let’s assume the current concurrency level is 3 (each Pipeline has 3 tasks), and each task reads one bucket from the storage layer. The number of rows in the three buckets is 1, 1, and 7, respectively. The execution before and after inserting the Local Exchange changes as follows:



pip_exec_7

As can be seen from the figure on the right, the amount of data that the HashJoin and Agg operators need to process changes from (1, 1, 7) to (3, 3, 3), thereby avoiding data skew.

Local Shuffle is planned based on a series of rules. For example, when a query involves time-consuming operators like Join, Aggregation, or Window Functions, Local Shuffle is used to minimize data skew as much as possible.

Runtime Filter

Runtime Filter mainly consists of two types: Join Runtime Filter and TopN Runtime Filter. This article will provide a detailed introduction to the working principles, usage guidelines, and tuning methods of these two types of Runtime Filters.

Join Runtime Filter

Join Runtime Filter (hereinafter referred to as JRF) is an optimization technique that dynamically generates filters at the Join node based on runtime data, leveraging the Join condition. This technique not only reduces the size of the Join Probe but also effectively minimizes data I/O and network transmission.

Principles

Let's illustrate the working principle of JRF using a Join operation similar to that found in the TPC-H Schema.

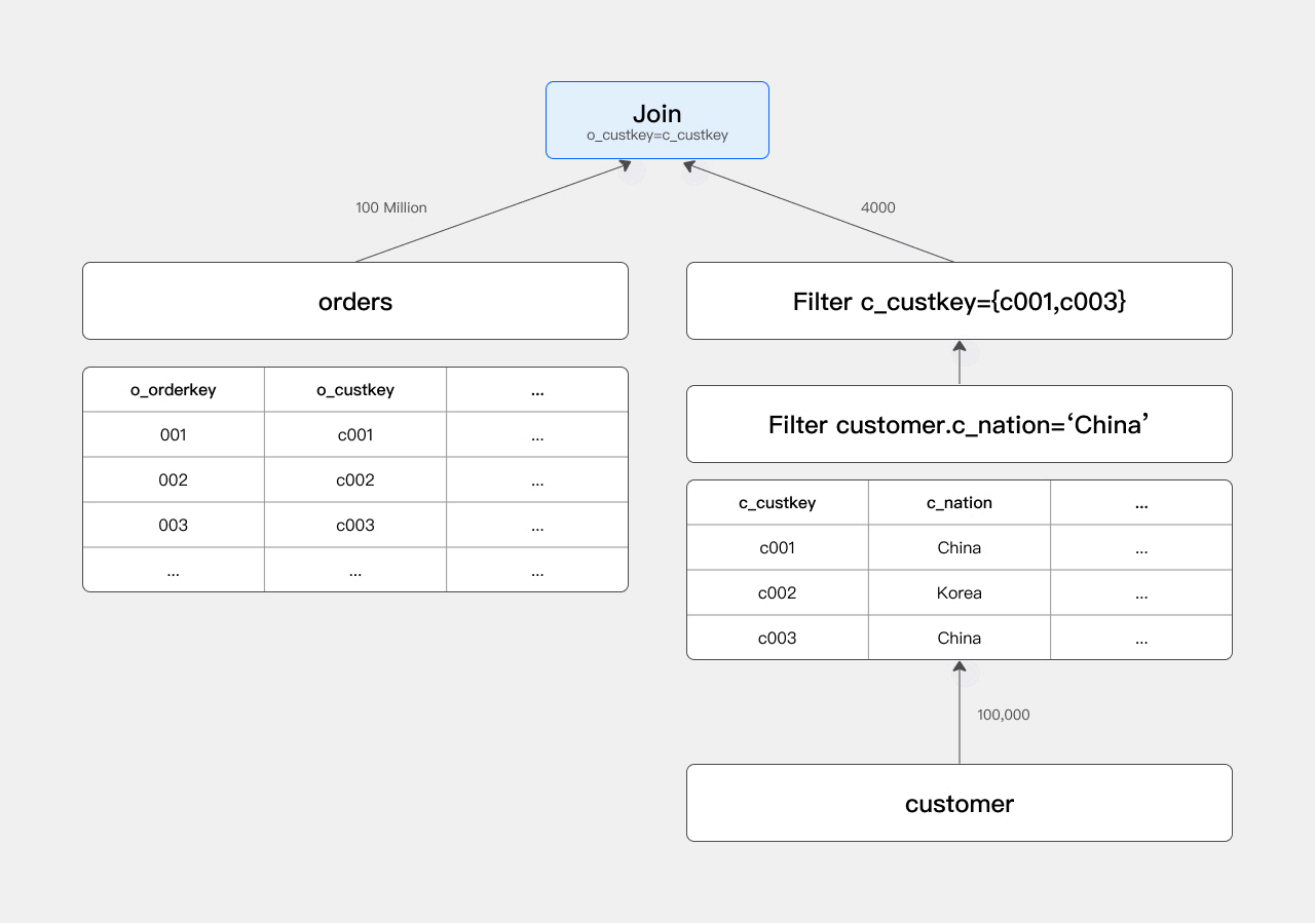
Assume there are two tables in the database:

- Orders Table: Contains 100 million rows of data, recording order keys (`o_orderkey`), customer keys (`o_custkey`), and other order information.
- Customer Table: Contains 100,000 rows of data, recording customer keys (`c_custkey`), customer nations (`c_nation`), and other customer information. This table records customers from 25 countries, with approximately 4,000 customers per country.

To count the number of orders from customers in China, the query statement would be:

```
select count(*)
from orders join customer on o_custkey = c_custkey
where c_nation = "china"
```

The main component of the execution plan for this query is a Join, as illustrated below:



Join Runtime Filter

Without JRF: The Scan node scans the orders table, reading 100 million rows of data. The Join node then performs a Hash Probe on these 100 million rows to generate the Join result.

1. Optimization

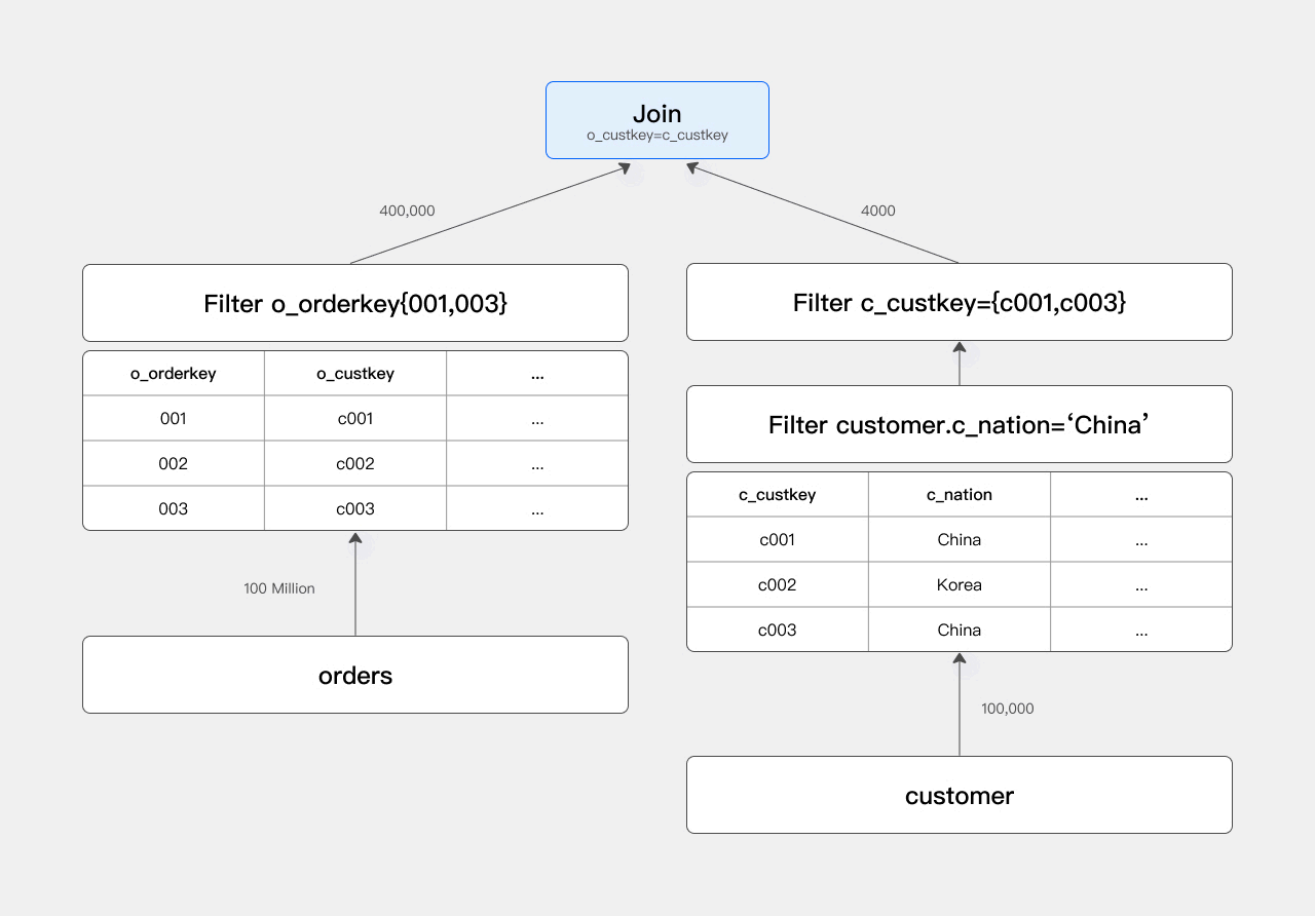
The filter condition `c_nation = 'china'` filters out all non-Chinese customers, so only a portion (approximately 1/25) of the customer table is involved in the Join. Given the subsequent Join condition `o_custkey = c_custkey`, we need to focus on the `c_custkey` values selected in the filtered result. Let's denote the filtered `c_custkey` values as set A. In the following text, we use set A specifically to refer to the `c_custkey` set participating in the Join.

If set A is pushed down to the orders table as an IN condition, the Scan node for the orders table can filter the orders accordingly. This is similar to adding a filter condition `o_custkey IN (c001, c003)`.

Based on this optimization concept, SQL can be optimized to:

```
select count(*)
from orders join customer on o_custkey = c_custkey
where c_nation = "china" and o_custkey in (c001, c003)
```

The optimized execution plan is illustrated below:



join-runtime-filter-2

By adding a filter condition on the orders table, the actual number of orders participating in the Join is reduced from 100 million to 400,000, significantly improving query speed.

2. Implementation

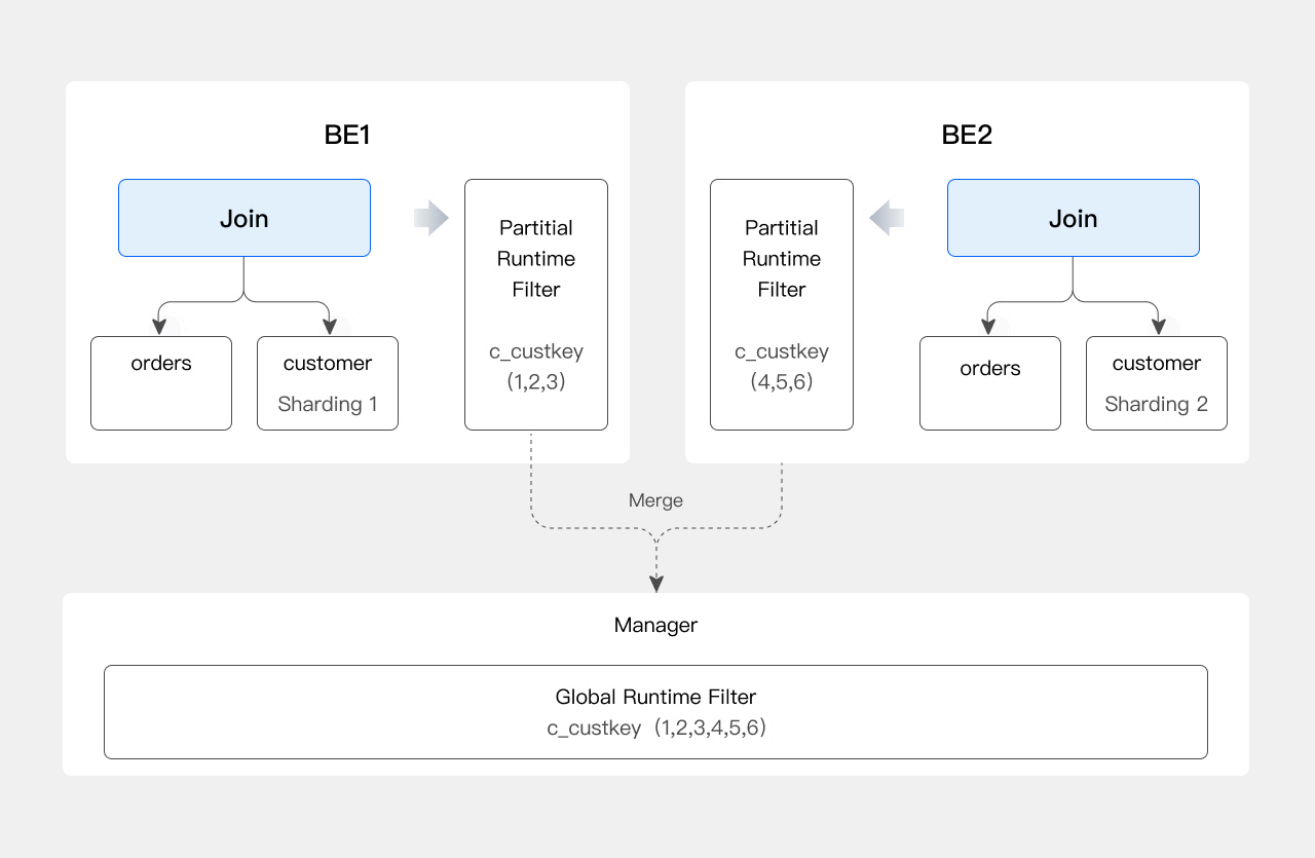
While the optimization described above is significant, the optimizer does not know the actual `c_custkey` values selected (set A) and thus cannot statically generate a fixed in-predicate filter operator during the optimization phase.

In practical applications, we collect the right-side data at the Join node, generate set A at runtime, and push down set A to the Scan node of the orders table. We typically denote this JRF as: `RF(c_custkey -> [o_custkey])`.

As Doris is a distributed database, JRF requires an additional merging step to cater to distributed scenarios. Assuming the Join in the example is a Shuffle Join, multiple instances of this Join handle individual shards of the orders and customer tables. Consequently, each Join instance only obtains a portion of set A.

In the current version of Doris, we select a node to serve as the Runtime Filter Manager. Each Join instance generates a Partial JRF based on the `c_custkey` values in its shard and sends it to the Manager. The Manager collects all Partial JRFs, merges them into a Global JRF, and then sends the Global JRF to all Scan instances of the orders table.

The process of generating the Global JRF is illustrated below:



Global JRF

Filter Types

There are various data structures that can be employed to implement JRF (Join Runtime Filter), each with varying efficiencies in generation, merging, transmission, and application, making them suitable for different scenarios.

1. In Filter

The simplest approach to implementing JRF is through the use of an In Filter. Taking the previous example, when using an In Filter, the execution engine generates a predicate `o_custkey in (...list of elements in Set A...)` on the left table. This In filter condition can then be applied to filter the orders table. When the number of elements in Set A is small, the In Filter is efficient.

However, using an In Filter becomes problematic when the number of elements in Set A is large:

1. Firstly, the cost of generating an In Filter is high, especially when JRF merging is required. Values collected from Join nodes corresponding to different data partitions may contain duplicates. For instance, if `c_custkey` is not the primary key of the table, values like `c001` and `c003` could appear multiple times, necessitating a time-consuming deduplication process.

2. Secondly, when Set A contains many elements, the cost of transmitting data between the Join node and the Scan node of the orders table is significant.

3. Lastly, executing the In predicate at the Scan node of the orders table also takes time.

Considering these factors, we introduce the Bloom Filter.

2. Bloom Filter

For those unfamiliar with Bloom Filters, they can be thought of as a set of superimposed hash tables. Using a Bloom Filter (or hash table) for filtering leverages the following property:

- A hash table T is generated based on Set A. If an element is not in hash table T, it can be definitively concluded that the element is not in Set A. However, the reverse is not true.

  Therefore, if an `o_orderkey` is filtered out by the Bloom Filter, it can be concluded that there is no matching `c_custkey` on the right side of the Join. Nevertheless, due to hash collisions, some `o_custkey`s may pass through the Bloom Filter even if there is no matching `c_custkey`.

  While a Bloom Filter cannot achieve precise filtering, it still provides a certain level of filtering effectiveness.

- The number of buckets in the hash table determines the accuracy of filtering. The larger the number of buckets, the larger and more accurate the Filter becomes, but at the cost of increased computational overhead in generation, transmission, and usage.

  Therefore, the size of the Bloom Filter must strike a balance between filtering effectiveness and usage costs. To this end, we have set a configurable range for the Bloom Filter's size, defined by `RUNTIME_BLOOM_FILTER_MIN_SIZE` and `RUNTIME_BLOOM_FILTER_MAX_SIZE`.

3. Min/Max Filter

Apart from the Bloom Filter, the Min-Max Filter can also be used for approximate filtering. If the data column is ordered, the Min-Max Filter can achieve excellent filtering results. Additionally, the costs of generating, merging, and using a Min-Max Filter are significantly lower than those of an In Filter or Bloom Filter.

For non-equi Joins, both In Filters and Bloom Filters become ineffective, but the Min-Max Filter can still function. Suppose we modify the query from the previous example to:

```
select count(*)
from orders join customer on o_custkey > c_custkey
where c_name = "China"
```

In this case, we can select the maximum filtered `c_custkey`, denote it as n, and pass it to the Scan node of the orders table. The Scan node will then only output rows where `o_custkey > n`.

Viewing Join Runtime Filter

To see which JRFs (Join Runtime Filters) have been generated for a specific query, you can use the `explain`, `explain shape plan`, or `explain physical plan` commands.

Using the TPC-H Schema as an example, we will detail how to view JRFs using these three commands.

```
select count(*) from orders join customer on o_custkey=c_custkey;
```

1. Explain

In traditional Explain output, JRF information is typically displayed in Join and Scan nodes, as shown in the following example:

```
4: VHASH JOIN(258)
| join op: INNER JOIN(PARTITIONED)[]
|  equal join conjunct: (o_custkey[#10] = c_custkey[#0])
|  runtime filters: RF000[bloom] <- c_custkey[#0] (150000000/134217728/16777216)
|  cardinality=1,500,000,000
|  vec output tuple id: 3
|  output tuple id: 3
|  vIntermediate tuple ids: 2
|  hash output slot ids: 10
|  final projections: o_custkey[#17]
|  final project output tuple id: 3
|  distribute expr lists: o_custkey[#10]
|  distribute expr lists: c_custkey[#0]
|
|---1: VEXCHANGE
|      offset: 0
|      distribute expr lists: c_custkey[#0]
3: VEXCHANGE
|  offset: 0
|  distribute expr lists:

PLAN FRAGMENT 2
| PARTITION: HASH_PARTITIONED: o_orderkey[#8]
| HAS_COLO_PLAN_NODE: false
| STREAM DATA SINK
|   EXCHANGE ID: 03
|   HASH_PARTITIONED: o_custkey[#10]

2: VOlapScanNode(242)
|  TABLE: regression_test_nereids_tpch_shape_sf1000_p0.orders(orders)
|  PREAGGREGATION: ON
|  runtime filters: RF000[bloom] -> o_custkey[#10]
|  partitions=1/1 (orders)
|  tablets=96/96, tabletList=54990,54992,54994 ...
|  cardinality=0, avgRowSize=0.0, numNodes=1
|  pushAggOp=NONE
```

- Join Side: `runtime filters: RF000[bloom] <- c_custkey[#0] (150000000/134217728/16777216)`

  This indicates that a Bloom Filter with ID 000 has been generated, using `c_custkey` as input to create the JRF. The three numbers following are related to Bloom Filter size calculations and can be ignored for now.

- Scan Side: `runtime filters: RF000[bloom] -> o_custkey[#10]`

  This indicates that JRF 000 will be applied to the Scan node of the orders table, filtering the `o_custkey` field.

2. Explain Shape Plan

In the Explain Plan series, we'll use Shape Plan as an example to show how to view JRFs.

```
mysql> explain shape plan select count(*) from orders join customer on o_custkey=c_custkey where c_nationkey=5;
+------------------------------------------------------------------------------------------------------------------------+
Explain String(Nereids Planner)                                                                                          |
+------------------------------------------------------------------------------------------------------------------------+
```

```
PhysicalResultSink                                                                                                                                  |
--hashAgg[GLOBAL]                                                                                                                                   |
----PhysicalDistribute[DistributionSpecGather]                                                                                                      |
------hashAgg[LOCAL]                                                                                                                                |
--------PhysicalProject                                                                                                                             |
----------hashJoin[INNER_JOIN shuffle]                                                                                                              |
------------hashCondition=((orders.o_custkey=customer.c_custkey)) otherCondition=() buildRFs:RF0 c_custkey->[o_custkey]                               |
--------------PhysicalProject                                                                                                                      |
----------------PhysicalOlapScan[orders] apply RFs: RF0                                                                                           |
--------------PhysicalProject                                                                                                                      |
----------------filter((customer.c_nationkey=5))                                                                                                  |
------------------PhysicalOlapScan[customer]                                                                                                      |
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
11 rows in set (0.02 sec)
```

As shown above:

- Join Side: `build RFs: RF0 c_custkey -> [o_custkey]` indicates that JRF 0 is generated using `c_custkey` data and applied to `o_custkey`.
- Scan Side: `PhysicalOlapScan[orders] apply RFs: RF0` indicates that orders table is filtered by JRF 0.

3. Profile

During actual execution, BE outputs JRF usage details to the Profile (requires `set enable_profile=true`). Using the same SQL query as an example, we can view JRF execution details in the Profile.

- Join Side

  ```
  HASH_JOIN_SINK_OPERATOR  (id=3  ,  nereids_id=367):(ExecTime:  703.905us)
        -  JoinType:  INNER_JOIN
        ◦◦◦
        -  BuildRows:  617
        ◦◦◦
        -  RuntimeFilterComputeTime:  70.741us
        -  RuntimeFilterInitTime:  10.882us
  ```

  This is the Build side Profile for the Join. In this example, generating the JRF took 70.741us with 617 rows of input data. The JRF size and type are shown on the Scan side.

- Scan Side

  ```
  OLAP_SCAN_OPERATOR  (id=2.  nereids_id=351.  table  name  =  orders(orders)):(ExecTime:  13.32ms)
             -  RuntimeFilters:  :  RuntimeFilter:  (id  =  0,  type  =  bloomfilter,  need_local_merge:  false,
  is_broadcast:  true,  build_bf_cardinality:  false,
             ◦◦◦
             -  RuntimeFilterInfo:
                 -  filter  id  =  0  filtered:  714.761K  (714761)
                 -  filter  id  =  0  input:  747.862K  (747862)
             ◦◦◦
             -  WaitForRuntimeFilter:  6.317ms
           RuntimeFilter:  (id  =  0,  type  =  bloomfilter):
                 -  Info:  [IsPushDown  =  true,  RuntimeFilterState  =  READY,  HasRemoteTarget  =  false,  HasLocalTarget
  =  true,  Ignored  =  false]
                 -  RealRuntimeFilterType:  bloomfilter
                 -  BloomFilterSize:  1024
  ```

  Note:

1. Lines 5-6 show the input rows and the number of filtered rows. A higher number of filtered rows indicates better JRF effectiveness.
2. Line 10, `IsPushDown = true`, indicates that JRF computation has been pushed down to the storage layer, which can help reduce IO through delayed materialization.
3. Line 10, `RuntimeFilterState = READY`, indicates whether the Scan node has applied the JRF. Since JRF uses a try-best mechanism, if JRF generation takes too long, the Scan node may start scanning data after a waiting period, potentially outputting unfiltered data.
4. Line 12, `BloomFilterSize: 1024`, shows the size of the Bloom Filter in bytes.

Tuning

For Join Runtime Filter tuning, in most cases, the function is adaptive, and users do not need to manually tune it. However, there are a few adjustments that can be made to optimize performance.

1. Enable or Disable JRF

The session variable `runtime_filter_mode` controls whether JRFs are generated.

- To enable JRF: `set runtime_filter_mode = GLOBAL`
- To disable JRF: `set runtime_filter_mode = OFF`

2. Set JRF Type

The session variable `runtime_filter_type` controls the type of JRFs, including:

- `IN(1)`
- `BLOOM(2)`
- `MIN_MAX(4)`
- `IN_OR_BLOOM(8)`

The `IN_OR_BLOOM` Filter allows BE to adaptively choose between generating an `IN` Filter or a `BLOOM` Filter based on the actual number of rows of data.

Multiple JRF types can be generated for a single Join condition by setting `runtime_filter_type` to the sum of the corresponding enumeration values.

For example:

- To generate both a `BLOOM` Filter and a `MIN_MAX` Filter for each Join condition: `set runtime_filter_type = 6`
- In version 2.1, the default value of `runtime_filter_type` is 12, which generates both a `MIN_MAX` Filter and an `IN_OR_BLOOM` Filter.

The integers in parentheses represent the enumeration values for Runtime Filter Types.

3. Set Wait Time

As mentioned earlier, JRF uses a Try-best mechanism, where Scan nodes wait for JRFs before starting. Doris calculates the wait time based on runtime conditions. However, in some cases, the calculated wait time may not be sufficient, resulting in JRFs not being fully effective, and the Scan nodes may output more rows than expected. As discussed in the Profile section, if `RuntimeFilterState = NOT_READY` in the Scan node's Profile, users can manually set a longer wait time.

The session variable `runtime_filter_wait_time_ms` controls the wait time for Scan nodes to wait for JRFs. The default value is 1000 milliseconds.

4. Pruning JRF

In some cases, JRFs may not provide filtering benefits. For example, if the `orders` and `customer` tables have a primary-foreign key relationship, but there are no filtering conditions on the `customer` table, the input to the JRF would be all `custkeys`, allowing all rows in the `orders` table to pass through the JRF. The optimizer prunes ineffective JRFs based on column statistics.

The session variable `enable_runtime_filter_prune = true/false` controls whether pruning is performed. The default value is `true`.

TopN Runtime Filter

Principles

In Doris, data is processed in a block-streaming manner. Therefore, when an SQL statement includes a `topN` operator, Doris does not compute all results but instead generates a dynamic filter to pre-filter the data early on.

Consider the following SQL statement as an example:

```
select o_orderkey from orders order by o_orderdate limit 5;
```

The execution plan for this SQL statement is illustrated below:

```
mysql> explain select o_orderkey from orders order by o_orderdate limit 5;
+-----------------------------------------------------+
| Explain String(Nereids Planner)                     |
+-----------------------------------------------------+
| PLAN FRAGMENT 0                                     |
|   OUTPUT EXPRS:                                     |
|     o_orderkey[#11]                                 |
|   PARTITION: UNPARTITIONED                          |
|                                                     |
|   HAS_COLO_PLAN_NODE: false                         |
|                                                     |
|   VRESULT SINK                                      |
|      MYSQL_PROTOCAL                                 |
|                                                     |
|   2:VMERGING-EXCHANGE                               |
|      offset: 0                                      |
|      limit: 5                                       |
|      final projections: o_orderkey[#9]              |
|      final project output tuple id: 2               |
|      distribute expr lists:                         |
|                                                     |
| PLAN FRAGMENT 1                                     |
|                                                     |
```

```
|   PARTITION: HASH_PARTITIONED: O_ORDERKEY[#0]       |
|                                                     |
|   HAS_COLO_PLAN_NODE: false                         |
|                                                     |
|   STREAM DATA SINK                                  |
|     EXCHANGE ID: 02                                 |
|     UNPARTITIONED                                   |
|                                                     |
|   1:VTOP-N(119)                                     |
|   |  order by: o_orderdate[#10] ASC                 |
|   |  TOPN OPT                                       |
|   |  offset: 0                                      |
|   |  limit: 5                                       |
|   |  distribute expr lists: O_ORDERKEY[#0]          |
|   |                                                 |
|   0:VOlapScanNode(113)                              |
|      TABLE: tpch.orders(orders), PREAGGREGATION: ON |
|      TOPN OPT:1                                     |
|      partitions=1/1 (orders)                        |
|      tablets=3/3, tabletList=135112,135114,135116   |
|      cardinality=150000, avgRowSize=0.0, numNodes=1 |
|      pushAggOp=NONE                                 |
+-----------------------------------------------------+
41 rows in set (0.06 sec)
```

Without a `topn filter`, the scan node would sequentially read each data block from the `orders` table and pass them to the TopN node. The TopN node maintains the current top 5 rows from the `orders` table through heap sorting.

Since a data block typically contains around 1024 rows, the TopN node can identify the 5th ranked row within the first data block after processing it.

Assuming this `o_orderdate` is `1995-01-01`, the scan node can then use `1995-01-01` as a filter condition when outputting the second data block, eliminating the need to send rows with `o_orderdate` greater than `1995-01-01` to the TopN node for further processing.

This threshold is dynamically updated. For instance, if the TopN node discovers a smaller `o_orderdate` when processing the second filtered data block, it updates the threshold to the fifth-ranked `o_orderdate` among the first two data blocks.

Viewing TopN Runtime Filter

Using the Explain command, we can inspect the TopN Runtime Filter planned by the optimizer.

```
1:VTOP-N(119)
| order by: o_orderdate[#10] ASC
| TOPN OPT
| offset: 0
| limit: 5
| distribute expr lists: O_ORDERKEY[#0]
|

0:VLapScanNode[113]
    TABLE: regression_test_nereids_tpch_p0.(orders), PREAGGREGATION: ON
    TOPN OPT: 1
    partitions=1/1 (orders)
    tablets=3/3, tabletList=135112,135114,135116
```

```
    cardinality=150000, avgRowSize=0.0, numNodes=1
    pushAggOp: NONE
```

As shown in the example above:

1. The TopN node displays `TOPN OPT`, indicating that this TopN node generates a TopN Runtime Filter.
2. The Scan node indicates which TopN node generates the TopN Runtime Filter it uses. For instance, in the example, line 11 indicates that the Scan node for the `orders` table will use the Runtime Filter generated by TopN node 1, shown as `TOPN OPT: 1` in the plan.

As a distributed database, Doris considers the physical machines where TopN and Scan nodes actually run. Due to the high cost of cross-BE communication, BEs adaptively decide whether and to what extent to use TopN Runtime Filters. Currently, we have implemented BE-level TopN Runtime Filters, where TopN and Scan reside on the same BE. This is because updating TopN Runtime Filter thresholds only requires inter-thread communication, which is relatively inexpensive.

Tuning

The session variable `topn_filter_ratio` controls whether to generate a TopN Runtime Filter.

The fewer rows specified in the SQL's `limit` clause, the stronger the filtering effect of the TopN Runtime Filter. Therefore, by default, Doris enables the generation of corresponding TopN Runtime Filters only when the `limit` number is less than half of the data in the table.

For example, setting `set topn_filter_ratio=0` would prevent the generation of a TopN Runtime Filter for the following query:

```
select o_orderkey from orders order by o_orderdate limit 20;
```

TOPN Query Optimization

TOPN queries refer to queries that involve ORDER BY LIMIT operations, which are common in log retrieval and other detailed query scenarios. Doris automatically optimizes this type of query.

```
SELECT * FROM tablex WHERE xxx ORDER BY c1,c2 ... LIMIT n
```

Advantages of TOPN

1. During execution, dynamic range filters are built for the sorting columns (e.g., c1 >= 1000), which automatically apply the preceding conditions when reading data, leveraging zonemap indexes to filter out some rows or even entire files.
2. If the sorting fields c1, c2 are exactly the prefix of the table key, further optimization is applied. When reading data, only the header or tail of the data files is read, reducing the amount of data read to just the n rows needed.
3. SELECT * deferred materialization, during the data reading and sorting process, only the sorting columns are read, not the other columns. After obtaining the row numbers that meet the conditions, the entire data of those n rows needed is read, significantly reducing the amount of data read and sorted.

Limitations

1. It only applies to DUP and MOW tables, not to MOR and AGG tables.
2. Due to the high memory consumption on very large `n`, it will not take effect if n is greater than `topn_opt_limit_threshold`.

Configuration and Query Analysis

The following two parameters are session variables that can be set for a specific SQL or globally.

1. `topn_opt_limit_threshold`: This session variable determines whether TOPN optimization is applied. It defaults to 1024, and setting it to 0 disables the optimization.
2. `enable_two_phase_read_optimization`: This session variable determines whether to enable this optimization. It defaults to true, and setting it to false disables the optimization.
3. `topn_filter_ratio`, the ratio between LIMIT n and the total data in the table, the default value is 0.5, which means that if the number of LIMIT is more than half of the data in the table, no filter will be generated.

Checking if TOPN Query Optimization is Enabled

To confirm if TOPN query optimization is enabled for a particular SQL, you can use the `EXPLAIN` statement to get the query plan. An example is as follows:

- `TOPN OPT` indicates that optimization point 1 is applied.
- `VOlapScanNode` with `SORT LIMIT` indicates optimization point 2 is applied.
- `OPT TWO PHASE` indicates optimization point 3 is applied.

```
1:VTOP-N(137)
|   order by: @timestamp18 DESC
|   TOPN OPT
|   OPT TWO PHASE
|   offset: 0
|   limit: 10
|   distribute expr lists: applicationName5
|
0:VOlapScanNode(106)
   TABLE: log_db.log_core_all_no_index(log_core_all_no_index), PREAGGREGATION: ON
   SORT INFO:
        @timestamp18
   SORT LIMIT: 10
   TOPN OPT:1
   PREDICATES: ZYCFC-TRACE-ID4 like '%flowId-1720055220933%'
   partitions=1/8 (p20240704), tablets=250/250, tabletList=1727094,1727096,1727098 ...
   cardinality=345472780, avgRowSize=0.0, numNodes=1
   pushAggOp=NONE
```

Checking the Effectiveness of TOPN Query Optimization During Execution

First, set `topn_opt_limit_threshold` to 0 to disable TOPN query optimization and compare the execution time of the SQL with and without optimization enabled.

After enabling TOPN query optimization, search for `RuntimePredicate` in the query profile and focus on the following metrics:

- `RowsZonemapRuntimePredicateFiltered`: The number of rows filtered out, the higher the better.
- `NumSegmentFiltered`: The number of data files filtered out, the higher the better.
- `BlockConditionsFilteredZonemapRuntimePredicateTime`: The time taken to filter data, the lower the better.

Before version 2.0.3, the `RuntimePredicate` metrics were not separated out, and the `Zonemap` metrics can be used as a rough guide.

```
 SegmentIterator:
       -  BitmapIndexFilterTimer:  46.54us
       -  BlockConditionsFilteredBloomFilterTime:  10.352us
       -  BlockConditionsFilteredDictTime:  7.299us
       -  BlockConditionsFilteredTime:  202.23ms
       -  BlockConditionsFilteredZonemapRuntimePredicateTime:  0ns
       -  BlockConditionsFilteredZonemapTime:  402.917ms
       -  BlockInitSeekCount:  399
       -  BlockInitSeekTime:  11.309ms
       -  BlockInitTime:  215.59ms
       -  BlockLoadTime:  7s567ms
       -  BlocksLoad:  392.97K  (392970)
       -  CachedPagesNum:  0
       -  CollectIteratorMergeTime:  0ns
       -  CollectIteratorNormalTime:  0ns
       -  CompressedBytesRead:  29.76  MB
```

```
          -  DecompressorTimer:  427.713ms
          -  ExprFilterEvalTime:  3s930ms
          -  FirstReadSeekCount:  392.921K  (392921)
          -  FirstReadSeekTime:  528.287ms
          -  FirstReadTime:  1s134ms
          -  IOTimer:  51.286ms
          -  InvertedIndexFilterTime:  49.457us
          -  InvertedIndexQueryBitmapCopyTime:  0ns
          -  InvertedIndexQueryBitmapOpTime:  0ns
          -  InvertedIndexQueryCacheHit:  0
          -  InvertedIndexQueryCacheMiss:  0
          -  InvertedIndexQueryTime:  0ns
          -  InvertedIndexSearcherOpenTime:  0ns
          -  InvertedIndexSearcherSearchTime:  0ns
          -  LazyReadSeekCount:  0
          -  LazyReadSeekTime:  0ns
          -  LazyReadTime:  106.952us
          -  NumSegmentFiltered:  0
          -  NumSegmentTotal:  50
          -  OutputColumnTime:  61.987ms
          -  OutputIndexResultColumnTimer:  12.345ms
          -  RawRowsRead:  3.929151M  (3929151)
          -  RowsBitmapIndexFiltered:  0
          -  RowsBloomFilterFiltered:  0
          -  RowsConditionsFiltered:  6.38976M  (6389760)
          -  RowsDictFiltered:  0
          -  RowsInvertedIndexFiltered:  0
          -  RowsKeyRangeFiltered:  0
          -  RowsShortCircuitPredFiltered:  0
          -  RowsShortCircuitPredInput:  0
          -  RowsStatsFiltered:  6.38976M  (6389760)
          -  RowsVectorPredFiltered:  0
          -  RowsVectorPredInput:  0
          -  RowsZonemapRuntimePredicateFiltered:  6.38976M  (6389760)
          -  SecondReadTime:  0ns
          -  ShortPredEvalTime:  0ns
          -  TotalPagesNum:  2.301K  (2301)
          -  UncompressedBytesRead:  137.99  MB
          -  VectorPredEvalTime:  0ns
```

Statistics | Optimization Technology Principle

Statistics

Starting from version 2.0, Doris integrated Cost-Based Optimization (CBO) capabilities into its optimizer. Statistics are the cornerstone of CBO, and their accuracy directly determines the accuracy of cost estimation, which is crucial for selecting the optimal execution plan. This document serves as a guide to statistical usage for unreleased development version, focusing on the collection and management methods, relevant configuration options, and frequently asked questions.

Collection of Statistics

Doris enables the automatic sampling collection of internal tables by default. Therefore, in most cases, users don’t need to pay attention to the collection of statistical information. Doris collects statistics at the column level for each table. The information collected includes:

Info of Statistics	Description
row_count	Total number of rows
data_size	Total data size of the column
avg_size_byte	Average data size per row for the column
ndv	Number of distinct values
min	Minimum value
max	Maximum value
null_count	Number of null values

Currently, the system only supports collecting statistics for columns of basic data types, including BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DATE, DATETIME, STRING, VARCHAR, TEXT, among others.

Columns of complex types, such as JSONB, VARIANT, MAP, STRUCT, ARRAY, HLL, BITMAP, TIME, TIMEV2, are skipped.

Statistics can be collected manually or automatically, and the results are stored in the `internal.__internal_schema.column_statistics` table. The following sections detail these two collection methods.

Manual Collection

Doris allows users to manually trigger the collection and update of statistics by submitting an ANALYZE statement.

1. Syntax

```
ANALYZE < TABLE table_name > | < DATABASE db_name >
    [ (column_name [, ...]) ]
    [ [ WITH SYNC ] [ WITH SAMPLE PERCENT | ROWS ] ];
```

Parameters Explanation:

- `table_name`: Specifies the target table for which statistics will be collected.
- `column_name`: Specifies the target columns for which statistics will be collected. These columns must exist in `table_name`, and multiple column names are separated by commas. If no column names are specified, statistics will be collected for all columns in the table.
- `sync`: Optional parameter to collect statistics synchronously. If specified, the result will be returned after collection is complete; if not specified, the operation will be performed asynchronously, and a JOB ID will be returned, which can be used to check the status of the collection task.
- `sample percent | rows`: Optional parameter for sampling during statistics collection. Allows specifying a sampling percentage or number of rows. If `WITH SAMPLE` is not specified, a full table scan will be performed. For large tables (e.g., over 5 GiB), sampling is generally recommended from a cluster resource utilization perspective. To ensure the accuracy of statistics, it is recommended to sample at least 4 million rows.

2. Examples

Collect statistics for all columns in the `lineitem` table:

```
ANALYZE TABLE lineitem;
```

Collect statistics for all columns in all tables in the `tpch100` database:

```
ANALYZE DATABASE tpch100;
```

Collect statistics for the `l_orderkey` and `l_linenumber` columns in the `lineitem` table by sampling 100,000 rows (note: the correct syntax should be used `WITH SAMPLE ROWS` or `WITH SAMPLE PERCENT`):

```
ANALYZE TABLE lineitem WITH SAMPLE ROWS 100000;
```

Automatic Collection

Automatic collection is supported from version 2.0.3 onwards and is enabled by default throughout the day. Users can control the feature's activation or deactivation by setting the `ENABLE_AUTO_ANALYZE` variable:

```
SET GLOBAL ENABLE_AUTO_ANALYZE = TRUE; // Enable automatic collection
SET GLOBAL ENABLE_AUTO_ANALYZE = FALSE; // Disable automatic collection
```

When enabled, a background thread periodically scans all tables in the `InternalCatalog` within the cluster. For tables requiring statistics collection, the system automatically creates and executes collection jobs without manual intervention.

To avoid excessive resource usage for collection wide tables' statistics, tables with more than 300 columns are not automatically collected by default. Users can adjust this threshold by modifying the session variable `auto_analyze_table_width_threshold`:

```
SET GLOBAL auto_analyze_table_width_threshold = 350;
```

The default polling interval for automatic collection is 5 minutes (adjustable via the `auto_check_statistics_in_minutes` configuration in `fe.conf`). The first iteration starts 5 minutes after cluster startup. After all tables requiring collection are processed, the background thread sleeps for 5 minutes before starting the next iteration. Therefore, there is no guarantee that a table will have its statistics collected within 5 minutes, as the time to iterate through all tables can vary based on the number and size of tables.

When a table is polled, the system first determines if statistical collection is required. If so, a collection job is created and executed; otherwise, the table is skipped. Statistics collection is required if:

1. The table has columns without statistics.
2. The table's health is below the threshold (default 90, adjustable via `table_stats_health_threshold`). Health indicates the percentage of data that has remained unchanged since the last statistics collection: 100 indicates no change; 0 indicates all changes; a health below 90 indicates significant deviation in current statistics, necessitating re-collection.

To reduce background job overhead and improve collection speed, automatic collection uses sampling by default, sampling 4,194,304 (`2^22`) rows. Users can adjust the sampling size by modifying `huge_table_default_sample_rows` for more accurate data distribution information.

To prevent automatic collection jobs from interfering with business operations, users can specify the execution window for automatic collection based on their requirements by setting `auto_analyze_start_time` and `auto_analyze_end_time`:

```
SET GLOBAL auto_analyze_start_time = "03:00:00"; // Set the start time to 3 AM
SET GLOBAL auto_analyze_end_time = "14:00:00"; // Set the end time to 2 PM
```

External Table Collection

External tables typically include Hive, Iceberg, JDBC, and other types.

- Manual Collection: Hive, Iceberg, and JDBC tables support manual statistics collection. Hive tables support both full and sampled collection, while Iceberg and JDBC tables only support full collection. Other external table types do not support manual collection.
- Automatic Collection: Currently, only Hive tables are supported.

External Catalogs do not participate in automatic collection by default because they often contain large amounts of historical data, which could consume excessive resources during automatic collection. However, you can enable or disable automatic collection for an external Catalog by setting its properties:

```
ALTER CATALOG external_catalog SET PROPERTIES ('enable.auto.analyze'='true'); // Enable automatic collection
ALTER CATALOG external_catalog SET PROPERTIES ('enable.auto.analyze'='false'); // Disable automatic collection
```

External tables do not have the concept of health. When automatic collection is enabled for a Catalog, the system defaults to collecting statistics for an external table only once every 24 hours to avoid frequent collection. You can adjust the minimum collection interval for external tables using the `external_table_auto_analyze_interval_in_millis` variable.

By default, external tables do not collect statistics, but for Hive and Iceberg tables, the system attempts to obtain row count information through the Hive Metastore and Iceberg API.

1. For Hive Tables:

Doris first attempts to retrieve `numRows` or `totalSize` information from the Hive table's Parameters:

- If `numRows` is found, its value is used as the table's row count.
- If `numRows` is not found but `totalSize` is available, the row count is estimated based on the table's schema and `totalSize`.
- If `totalSize` is also unavailable, by default, the system will estimate the number of rows based on the file size corresponding to the Hive table and its Schema. If there are concerns that obtaining the file size may consume excessive resources, this function can be disabled by setting the following variables.

  ```
  SET GLOBAL enable_get_row_count_from_file_list = FALSE
  ```

2. For Iceberg Tables:

Doris calls the Iceberg snapshot API to retrieve `total-records` and `total-position-deletes` information to calculate the table's row count.

3. For Paimon Tables:

Doris calls Paimon's scan API to obtain the number of rows contained in each Split and calculate the row count of the table by summing up the rows of the Splits.

4. For JDBC Tables:

Doris sends SQL of reading table statistics to remote database to get table row count. This can only be achieved when the remote database has collected the row count information of the table. Currently, Doris supports retrieving the row count of tables in MySQL, Oracle, PostgreSQL and SQLServer.

5. For Other External Tables:

Automatic row count acquisition and estimation are currently not supported.

Users can view the estimated row count for external tables using the following command (see `Viewing Table Statistics Overview` for more detail):

```
SHOW table stats table_name;
```

- If `row_count` displays as `-1`, row count information could not be obtained or the table is empty.

Statistics Job Management

Viewing Statistics Jobs

Use `SHOW ANALYZE` to view information about statistics collection jobs. Currently, the system retains information for only 20,000 historical jobs. Note that only information for asynchronous jobs can be viewed using this command; synchronous jobs (using `WITH SYNC`) do not retain historical job information.

1. Syntax:

```
SHOW [AUTO] ANALYZE < table_name | job_id >
    [ WHERE STATE = < "PENDING" | "RUNNING" | "FINISHED" | "FAILED" > ];
```

- `AUTO`: Displays information about historical automatic collection jobs. If unspecified, displays information about manual `ANALYZE` historical jobs.
- `table_name`: Table name, specifying which allows you to view statistics job information for that table. Can be in the form `db_name.table_name`. If unspecified, returns information about all statistics jobs.
- `job_id`: Statistics job ID, obtained when executing an asynchronous `ANALYZE` collection. If unspecified, the command returns information about all statistics jobs.

2. Output:

Includes the following columns:

Column Name	Description
job_id	Statistics job ID
catalog_name	Catalog name
db_name	Database name
tbl_name	Table name
col_name	List of column names (index_name:column_name)
job_type	Job type
analysis_type	Statistics type
message	Job information
state	Job state
progress	Job progress
schedule_type	Scheduling type
start_time	Job start time
end_time	Job end time

3. Example:

```
mysql show analyze 245073\G;
*************************** 1. row ***************************
```

```
              job_id: 93021
       catalog_name: internal
            db_name: tpch
           tbl_name: region
           col_name: [region:r_regionkey,region:r_comment,region:r_name]
           job_type: MANUAL
      analysis_type: FUNDAMENTALS
            message:
              state: FINISHED
           progress: 3 Finished  |  0 Failed  |  0 In Progress  |  3 Total
      schedule_type: ONCE
         start_time: 2024-07-11 15:15:00
           end_time: 2024-07-11 15:15:33
```

Viewing Statistics Tasks

Each collection job can contain one or more tasks, with each task corresponding to the collection of a single column. Users can view the completion status of statistics collection for each column using the following command.

1. Syntax:

```
SHOW ANALYZE TASK STATUS [job_id]
```

2. Example:

```
mysql> show analyze task status 93021;
+---------+-------------+------------+---------+-----------------------+-----------------+----------+
| task_id | col_name    | index_name | message | last_state_change_time | time_cost_in_ms | state    |
+---------+-------------+------------+---------+-----------------------+-----------------+----------+
| 93022   | r_regionkey | region     |         | 2024-07-11 15:15:33    | 32883           | FINISHED |
| 93023   | r_comment   | region     |         | 2024-07-11 15:15:33    | 32883           | FINISHED |
| 93024   | r_name      | region     |         | 2024-07-11 15:15:33    | 32883           | FINISHED |
+---------+-------------+------------+---------+-----------------------+-----------------+----------+
```

Viewing Statistics

Users can view collected column statistics using the `SHOW COLUMN STATS` command.

1. Syntax:

```
SHOW COLUMN [cached] STATS table_name [ (column_name [, ...]) ];
```

Where:

- `cached`: Displays statistics currently cached in the FE memory.
- `table_name`: Target table for which statistics were collected, can be in the form `db_name.table_name`.
- `column_name`: Specified target column, must exist in `table_name`, multiple column names separated by commas. If unspecified, displays information for all columns.

2. Example:

```
mysql> show column stats region (r_regionkey)\G
*************************** 1. row ***************************
  column_name: r_regionkey
   index_name: region
        count: 5.0
          ndv: 5.0
     num_null: 0.0
    data_size: 20.0
avg_size_byte: 4.0
          min: 0
          max: 4
       method: FULL
         type: FUNDAMENTALS
      trigger: MANUAL
  query_times: 0
 updated_time: 2024-07-11 15:15:33
1 row in set (0.36 sec)
```

Viewing Table Statistics Overview

Use `SHOW TABLE STATS` to view an overview of table statistics collection.

1. Syntax:

```
SHOW TABLE STATS table_name;
```

Where: `table_name`: Target table name, can be in the form `db_name.table_name`.

2. Output:

Includes the following columns:

Column Name	Description
updated_rows	Number of rows updated in the table since the last ANALYZE
query_times	Reserved column, for recording the number of queries on the table in future versions
row_count	Number of rows in the table (may not reflect the exact count at command execution)
updated_time	Time of the last statistics update
columns	Columns for which statistics have been collected
trigger	Method by which statistics were triggered
new_partition	Whether there are new partitions with first-time data imports
user_inject	Whether statistics were manually injected by the user

3. Example:

```
mysql> show column stats region (r_regionkey)\G
*************************** 1. row ***************************
  column_name: r_regionkey
   index_name: region
        count: 5.0
          ndv: 5.0
     num_null: 0.0
```

```
    data_size: 20.0
avg_size_byte: 4.0
          min: 0
          max: 4
       method: FULL
         type: FUNDAMENTALS
      trigger: MANUAL
  query_times: 0
 updated_time: 2024-07-11 15:15:33
1 row in set (0.36 sec)
```

Killing Statistics Jobs

Use `KILL ANALYZE` to terminate a currently running asynchronous statistics job.

1. Syntax:

```
KILL ANALYZE job_id;
```

Where: `job_id`: The ID of the statistics job. This is the value returned when executing an asynchronous statistics collection with `ANALYZE` or obtained using the `SHOW ANALYZE` statement.

2. Example:

Terminate the statistics job with ID 52357.

```
mysql> KILL ANALYZE 52357;
```

Deleting Statistics

If a Catalog, Database, or Table is deleted, users do not need to manually delete its statistics as the background process will periodically clean up this information.

However, for tables that still exist, the system does not automatically clear their statistics. In this case, users need to manually delete them using the following syntax:

```
DROP STATS table_name
```

Session Variables and Configuration Options

Session Variables

Session Variable	Description	Default Value
auto_analyze_start_time	Start time for automatic statistics collection	0:00:00
auto_analyze_end_time	End time for automatic statistics collection	23:59:59
enable_auto_analyze	Whether to enable automatic collection functionality	TRUE
huge_table_default_sample_rows	Number of rows to sample for large tables	4194304

Session Variable	Description	Default Value
table_stats_health_threshold	Value range 0-100, indicating the percentage of data updated since the last statistics collection (100 - table_stats_health_threshold)% at which statistics are considered outdated	90
auto_analyze_table_width_threshold	Controls the maximum table width for automatic statistics collection, tables exceeding this column count do not participate in automatic statistics collection	300
enable_get_row_count_from_file_list	Whether to estimate row counts for Hive tables based on file sizes	FALSE (TRUE by default after 2.1.5)

FE Configuration

> **Note**
>
> The following FE configuration options typically do not require special attention.

FE Configuration Option	Description	Default Value
analyze_record_limit	Controls the number of persistent rows for statistics job execution records	20000
stats_cache_size	Number of statistics entries cached on the FE side	500000
statistics_simultaneously_running_task_num	Number of asynchronous statistics jobs that can run simultaneously	3
statistics_sql_mem_limit_in_bytes	Controls the amount of BE memory each statistics SQL can occupy	2L * 1024 * 1024 (2GiB)

FAQs

Q1: How can I check if statistics have been collected for a table and if the content is correct?

First, execute `show column stats table_name` to see if there are any statistical outputs.

Next, execute `show column cached stats table_name` to check if the statistics for the table are loaded into the cache.

```
mysql> show column stats test_table\G
Empty set (0.02 sec)

mysql> show column cached stats test_table\G
Empty set (0.00 sec)
```

The empty result indicates that there are currently no statistics for the `test_table`. If statistics exist, the result will be similar to the following:

```
mysql> show column cached stats mvTestDup;
+-------------+------------+-------+------+----------+-----------+---------------+------+------+--------+--------------+---------+--
```

```
-----------+--------------------+
| column_name | index_name | count | ndv  | num_null | data_size | avg_size_byte | min  | max  | method | type         | trigger |
query_times | updated_time        |
+-------------+------------+-------+------+----------+-----------+---------------+------+------+--------+--------------+---------+--
-----------+--------------------+
| key1        | mvTestDup  | 6.0   | 4.0  | 0.0      | 48.0      | 8.0           | 1    | 1001 | FULL   | FUNDAMENTALS | MANUAL  | 0
| 2024-07-22 10:53:25 |
| key2        | mvTestDup  | 6.0   | 4.0  | 0.0      | 48.0      | 8.0           | 2    | 2001 | FULL   | FUNDAMENTALS | MANUAL  | 0
| 2024-07-22 10:53:25 |
| value2      | mvTestDup  | 6.0   | 4.0  | 0.0      | 24.0      | 4.0           | 4    | 4001 | FULL   | FUNDAMENTALS | MANUAL  | 0
| 2024-07-22 10:53:25 |
| value1      | mvTestDup  | 6.0   | 4.0  | 0.0      | 24.0      | 4.0           | 3    | 3001 | FULL   | FUNDAMENTALS | MANUAL  | 0
| 2024-07-22 10:53:25 |
| mv_key1     | mv1        | 6.0   | 4.0  | 0.0      | 48.0      | 8.0           | 1    | 1001 | FULL   | FUNDAMENTALS | MANUAL  | 0
| 2024-07-22 10:53:25 |
| value3      | mvTestDup  | 6.0   | 4.0  | 0.0      | 24.0      | 4.0           | 5    | 5001 | FULL   | FUNDAMENTALS | MANUAL  | 0
| 2024-07-22 10:53:25 |
+-------------+------------+-------+------+----------+-----------+---------------+------+------+--------+--------------+---------+--
-----------+--------------------+
6 rows in set (0.00 sec)
```

If statistics exist, you can manually execute SQL queries to verify their accuracy.

```
Select count(1), ndv(col1), min(col1), max(col1) from table
```

If the errors in `count` and `ndv` are within an order of magnitude, the accuracy is generally acceptable.

Q2: Why are statistics not being automatically collected for a table?

First, check if automatic collection is enabled:

```
Show variables like "enable_auto_analyze";  // If false, set it to true:

Set global enable_auto_analyze = true
```

If it's already true, check the number of columns in the table. If it exceeds the `auto_analyze_table_width_threshold`, the table will not participate in automatic collection. Modify this value to be greater than the current number of columns in the table:

```
Show variables like "auto_analyze_table_width_threshold"

// If the value is less than the width of the table, you can modify it:

Set global auto_analyze_table_width_threshold=350
```

If the number of columns does not exceed the threshold, execute `show auto analyze` to check if there are other collection tasks running (in the running state). Since automatic collection is executed serially by a single thread, the execution cycle may be long as it polls all databases and tables.

Q3: Why are statistics not available for some columns?

Currently, the system only supports collecting statistics for columns of basic data types. For complex types such as JSONB, VARIANT, MAP, STRUCT, ARRAY, HLL, BITMAP, TIME, and TIMEV2, the system skips them.

Q4: Error: “Stats table not available, please make sure your cluster status is normal”

This error typically indicates that the internal statistics table is in an unhealthy state.

First, check if all BEs (Backend) in the cluster are in a normal state and ensure they are all functioning correctly.

Next, execute the following statement to retrieve all `tabletId`s (first column of the output):

```
show tablets from internal.__internal_schema.column_statistics;
```

Then, check each tablet’s status using its `tablet_id`:

```
ADMIN DIAGNOSE TABLET tablet_id
```

If any tablets are found to be abnormal, repair them first before re-collecting statistics.

Q5: How can I address the issue of untimely statistics collection?

The interval for automatic collection is uncertain and depends on the number and size of tables in the system. In urgent cases, manually execute an `analyze` operation on the table.

If automatic collection is not triggered after importing large amounts of data, consider adjusting the `table_stats_health_threshold` parameter. Its default value is 90, meaning that automatic collection is triggered when more than 10% (100 - 90) of the table’s data changes. You can increase this value, for example, to 95, so that statistics are recollected when more than 5% of the table’s data changes.

Q6: How can I address excessive resource usage during automatic collection?

Automatic collection uses sampling and does not require full table scans, and the tasks are executed serially by a single thread. Usually, system resource usage is manageable and does not impact normal query tasks.

For some special tables, such as those with many partitions or large individual tablets, memory usage may be higher.

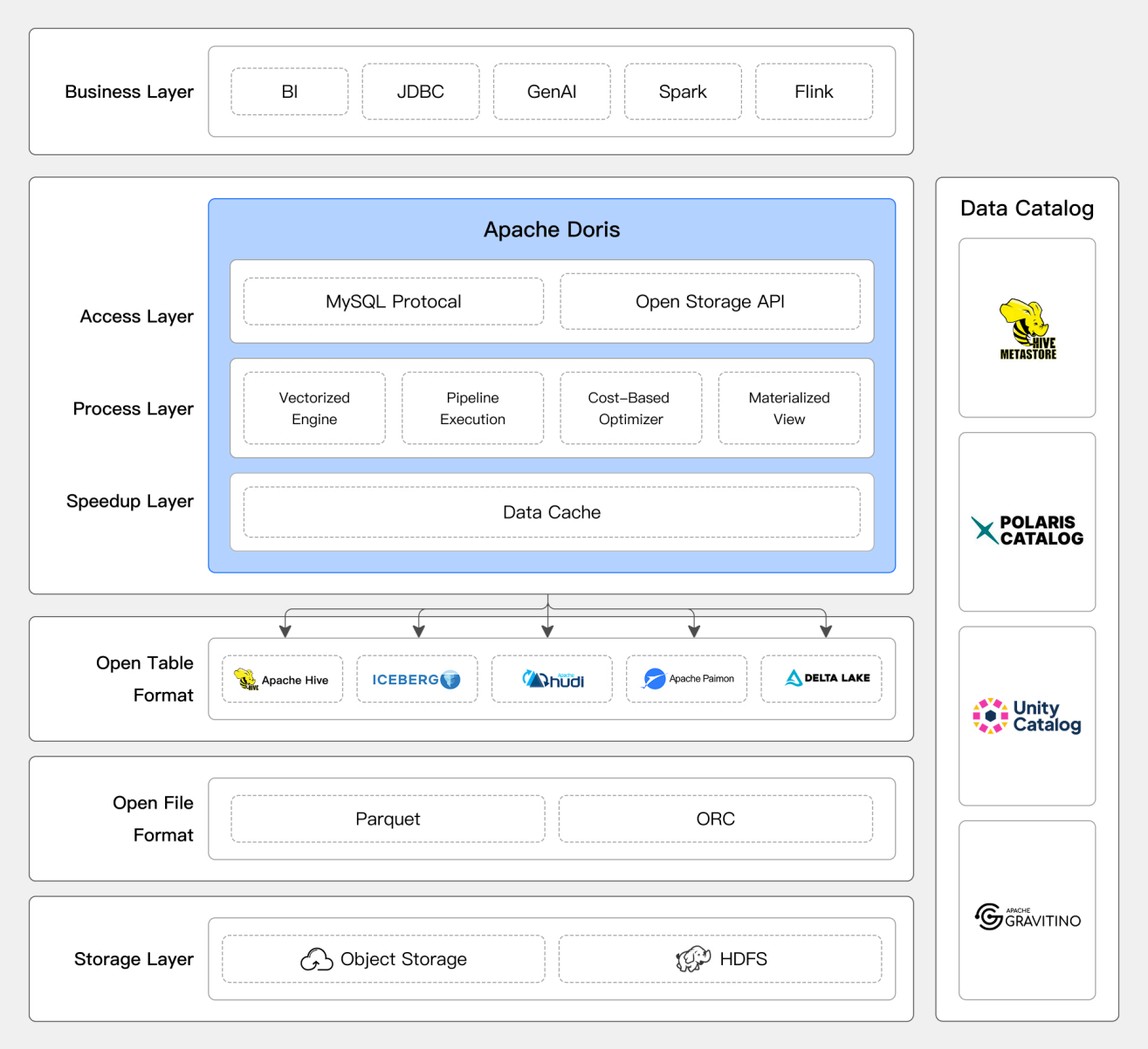
It is recommended to plan the number of tablets reasonably when creating tables to avoid creating oversized tablets. If the tablet structure is not easily adjustable, consider enabling automatic collection or manually collecting statistics for large tables during off-peak hours to avoid impacting business operations. In the Doris 3.x series, we will optimize for such scenarios.

Lakehouse Overview

The lakehouse is a modern big data solution that combines the advantages of data lakes and data warehouses. It integrates the low cost and high scalability of data lakes with the high performance and strong data governance capabilities of data warehouses, enabling efficient, secure, and quality-controlled storage and processing analysis of various data in the big data era. Through standardized open data formats and metadata management, it unifies **real-time** and **historical** data, **batch processing**, and **stream processing**, gradually becoming the new standard for enterprise big data solutions.

Doris Lakehouse Solution

Doris provides an excellent lakehouse solution for users through an extensible connector framework, a compute-storage decoupled architecture, a high-performance data processing engine, and data ecosystem openness.



doris lakehouse architecture

Flexible Data Access

Doris supports mainstream data systems and data format access through an extensible connector framework and provides unified data analysis capabilities based on SQL, allowing users to easily perform cross-platform data queries and analysis without moving existing data. For details, refer to Catalog Overview

Data Source Connectors

Whether it's Hive, Iceberg, Hudi, Paimon, or database systems supporting the JDBC protocol, Doris can easily connect and efficiently access data.

For lakehouse systems, Doris can obtain the structure and distribution information of data tables from metadata services such as Hive Metastore, AWS Glue, and Unity Catalog, perform reasonable query planning, and utilize the MPP architecture for distributed computing.

For details, refer to each catalog document, such as Iceberg Catalog

Extensible Connector Framework

Doris provides a good extensibility framework to help developers quickly connect to unique data sources within enterprises, achieving fast data interoperability.

Doris defines three levels of standard Catalog, Database, and Table, allowing developers to easily map to the required data source levels. Doris also provides standard interfaces for metadata service and storage service accessing, and developers only need to implement the corresponding interface to complete the data source connection.

Doris is compatible with the Trino Connector plugin, allowing the Trino plugin package to be directly deployed to the Doris cluster, and with minimal configuration, the corresponding data source can be accessed. Doris has already completed connections to data sources such as Kudu, BigQuery, and Delta Lake. You can also adapt new plugins yourself.

Convenient Cross-Source Data Processing

Doris supports creating multiple data catalogs at runtime and using SQL to perform federated queries on these data sources. For example, users can associate query fact table data in Hive with dimension table data in MySQL:

```
SELECT h.id, m.name
FROM hive.db.hive_table h JOIN mysql.db.mysql_table m
ON h.id = m.id;
```

Combined with Doris's built-in job scheduling capabilities, you can also create scheduled tasks to further simplify system complexity. For example, users can set the result of the above query as a routine task executed every hour and write each result into an Iceberg table:

```
CREATE JOB schedule_load
ON SCHEDULE EVERY 1 HOUR DO
INSERT INTO iceberg.db.ice_table
SELECT h.id, m.name
FROM hive.db.hive_table h JOIN mysql.db.mysql_table m
ON h.id = m.id;
```

High-Performance Data Processing

As an analytical data warehouse, Doris has made numerous optimizations in lakehouse data processing and computation and provides rich query acceleration features:

- Execution Engine

  The Doris execution engine is based on the MPP execution framework and Pipeline data processing model, capable of quickly processing massive data in a multi-machine, multi-core distributed environment. Thanks to fully vectorized execution operators, Doris leads in computing performance in standard benchmark datasets like TPC-DS.

- Query Optimizer

Doris can automatically optimize and process complex SQL requests through the query optimizer. The query optimizer deeply optimizes various complex SQL operators such as multi-table joins, aggregation, sorting, and pagination, fully utilizing cost models and relational algebra transformations to automatically obtain better or optimal logical and physical execution plans, greatly reducing the difficulty of writing SQL and improving usability and performance.

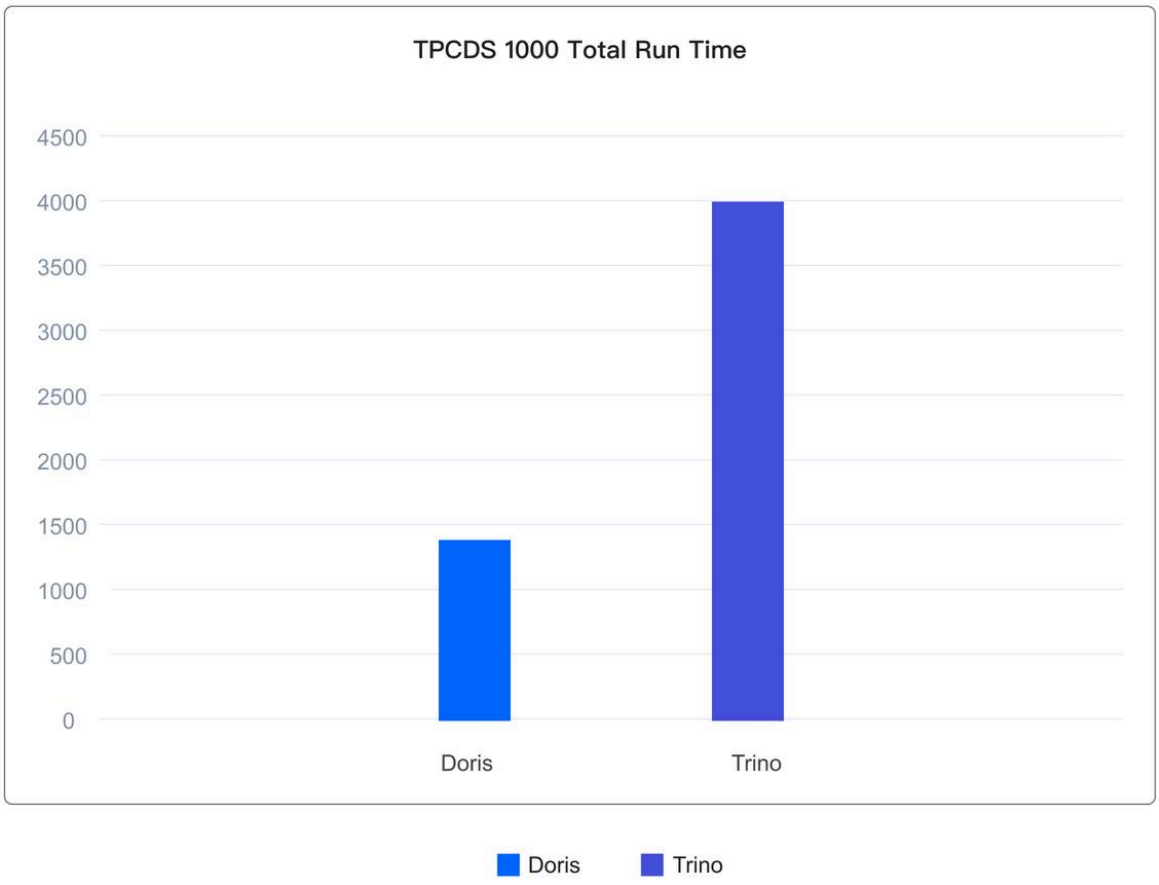
- Data Cache and IO Optimization

  Access to external data sources is usually network access, which can have high latency and poor stability. Apache Doris provides rich caching mechanisms and has made numerous optimizations in cache types, timeliness, and strategies, fully utilizing memory and local high-speed disks to enhance the analysis performance of hot data. Additionally, Doris has made targeted optimizations for network IO characteristics such as high throughput, low IOPS, and high latency, providing external data source access performance comparable to local data.

- Materialized Views and Transparent Acceleration

  Doris provides rich materialized view update strategies, supporting full and partition-level incremental refresh to reduce construction costs and improve timeliness. In addition to manual refresh, Doris also supports scheduled refresh and data-driven refresh, further reducing maintenance costs and improving data consistency. Materialized views also have transparent acceleration capabilities, allowing the query optimizer to automatically route to appropriate materialized views for seamless query acceleration. Additionally, Doris's materialized views use high-performance storage formats, providing efficient data access capabilities through column storage, compression, and intelligent indexing technologies, serving as an alternative to data caching and improving query efficiency.

As shown below, on a 1TB TPCDS standard test set based on the Iceberg table format, Doris's overall execution of 99 queries is only 1/3 of Trino's.



doris-tpcds

In actual user scenarios, Doris reduces average query latency by 20% and 95th percentile latency by 50% compared to Presto while using half the resources, significantly reducing resource costs while enhancing user experience.



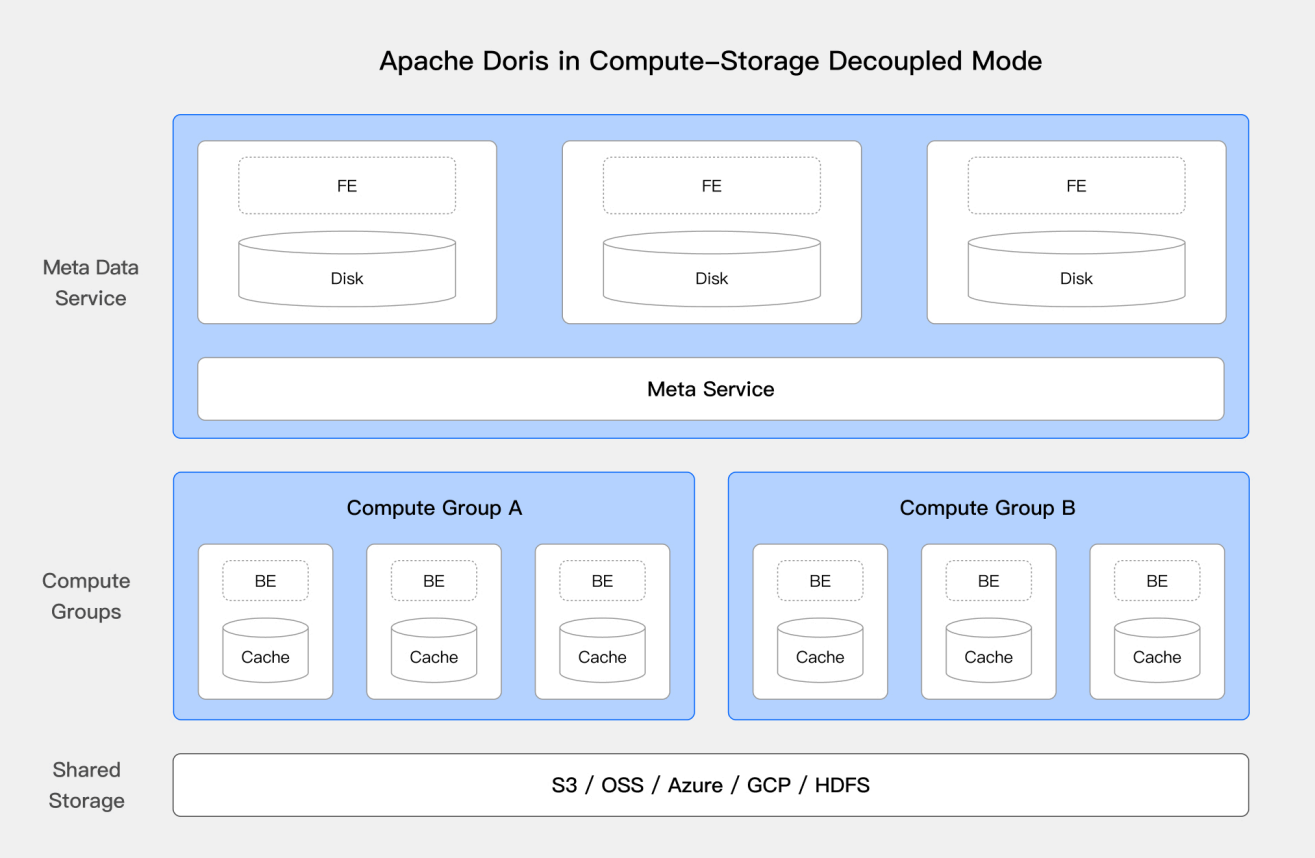
doris-performance

Convenient Service Migration

In the process of integrating multiple data sources and achieving lakehouse transformation, migrating SQL queries to Doris is a challenge due to differences in SQL dialects across systems in terms of syntax and function support. Without a suitable migration plan, the business side may need significant modifications to adapt to the new system's SQL syntax.

Modern Deployment Architecture

Since version 3.0, Doris supports a cloud-native compute-storage separation architecture. This architecture, with its low cost and high elasticity, effectively improves resource utilization and enables independent scaling of compute and storage.



compute-storage-decouple

The above diagram shows the system architecture of Doris's compute-storage separation, decoupling compute and storage. Compute nodes no longer store primary data, and the underlying shared storage layer (HDFS and object storage) serves as the unified primary data storage space, supporting independent scaling of compute and storage resources. The compute-storage separation architecture brings significant advantages to the lakehouse solution:

- **Low-Cost Storage**: Storage and compute resources can be independently scaled, allowing enterprises to increase storage capacity without increasing compute resources. Additionally, by using cloud object storage, enterprises can enjoy lower storage costs and higher availability, while still using local high-speed disks for caching relatively low-proportion hot data.
- **Single Source of Truth**: All data is stored in a unified storage layer, allowing the same data to be accessed and processed by different compute clusters, ensuring data consistency and integrity, and reducing the complexity of data synchronization and duplicate storage.
- **Workload Diversity**: Users can dynamically allocate compute resources based on different workload needs, supporting various application scenarios such as batch processing, real-time analysis, and machine learning. By separating storage and compute, enterprises can more flexibly optimize resource usage, ensuring efficient operation under different loads.

In addition, under the storage-computing coupled architecture, elastic computing nodes can still be used to provide elastic computing capabilities in lake warehouse data query scenarios.

Openness

Doris not only supports access to open lake table formats but also has good openness for its own stored data. Doris provides an open storage API and implements a high-speed data link based on the Arrow Flight SQL protocol, offering the speed advantages of Arrow Flight and the ease of use of JDBC/ODBC. Based on this interface, users can access data stored in Doris using Python/Java/Spark/Flink's ABDC clients.

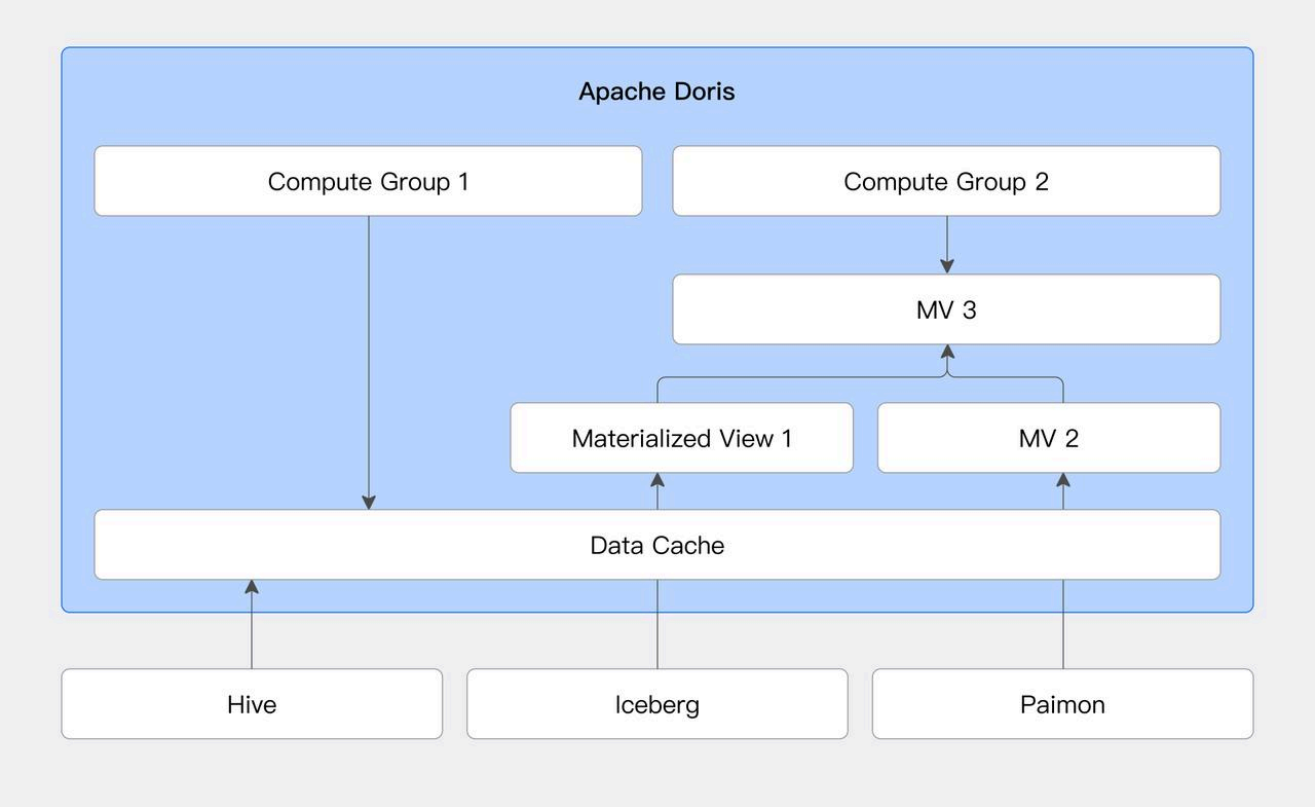
Compared to open file formats, the open storage API abstracts the specific implementation of the underlying file format, allowing Doris to accelerate data access through advanced features in its storage format, such as rich indexing mechanisms. Additionally, upper-layer compute engines do not need to adapt to changes or new features in the underlying storage format, allowing all supported compute engines to simultaneously benefit from new features.

Lakehouse Best Practices

In the lakehouse solution, Doris is mainly used for **lakehouse query acceleration**, **multi-source federated analysis**, and **lakehouse data processing**.

Lakehouse Query Acceleration

In this scenario, Doris acts as a **compute engine**, accelerating query analysis on lakehouse data.



lakehouse query acceleration

Cache Acceleration

For lakehouse systems like Hive and Iceberg, users can configure local disk caching. Local disk caching automatically stores query-designed data files in local cache directories and manages cache eviction using the

LRU strategy. For details, refer to the Data Cache document.

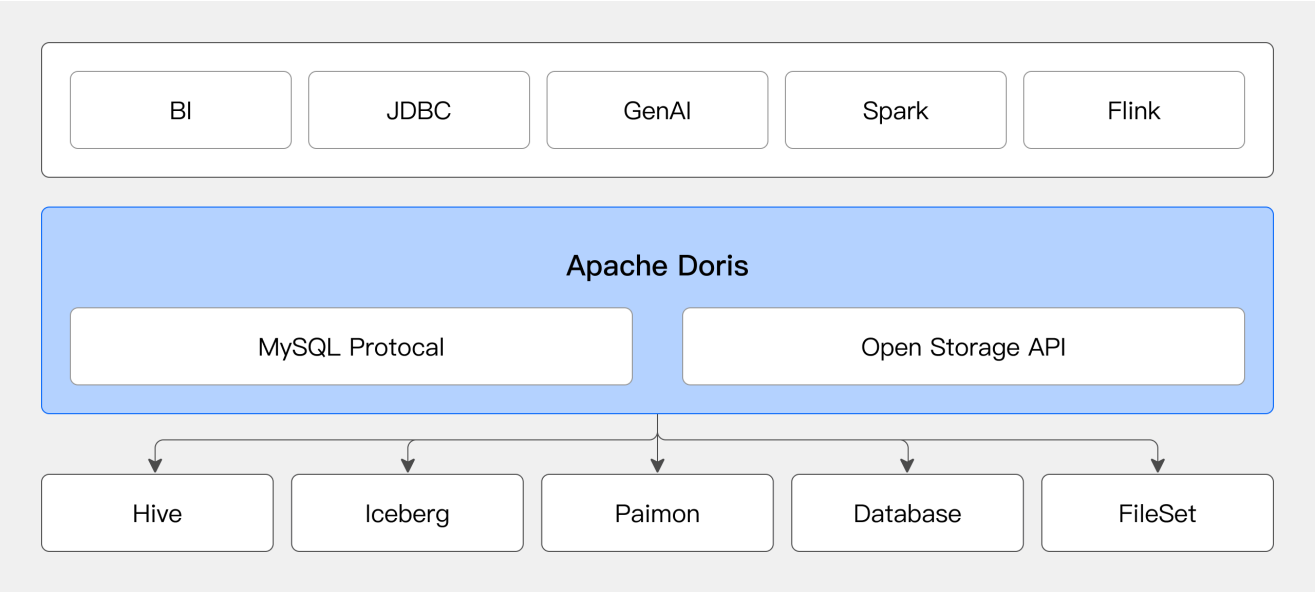
Materialized Views and Transparent Rewrite

Doris supports creating materialized views for external data sources. Materialized views store pre-computed results as Doris internal table formats based on SQL definition statements. Additionally, Doris's query optimizer supports a transparent rewrite algorithm based on the SPJG (SELECT-PROJECT-JOIN-GROUP-BY) pattern. This algorithm can analyze the structure information of SQL, automatically find suitable materialized views for transparent rewrite, and select the optimal materialized view to respond to query SQL.

This feature can significantly improve query performance by reducing runtime computation. It also allows access to data in materialized views through transparent rewrite without business awareness. For details, refer to the Materialized Views document.

Multi-Source Federated Analysis

Doris can act as a **unified SQL query engine**, connecting different data sources for federated analysis, solving data silos.

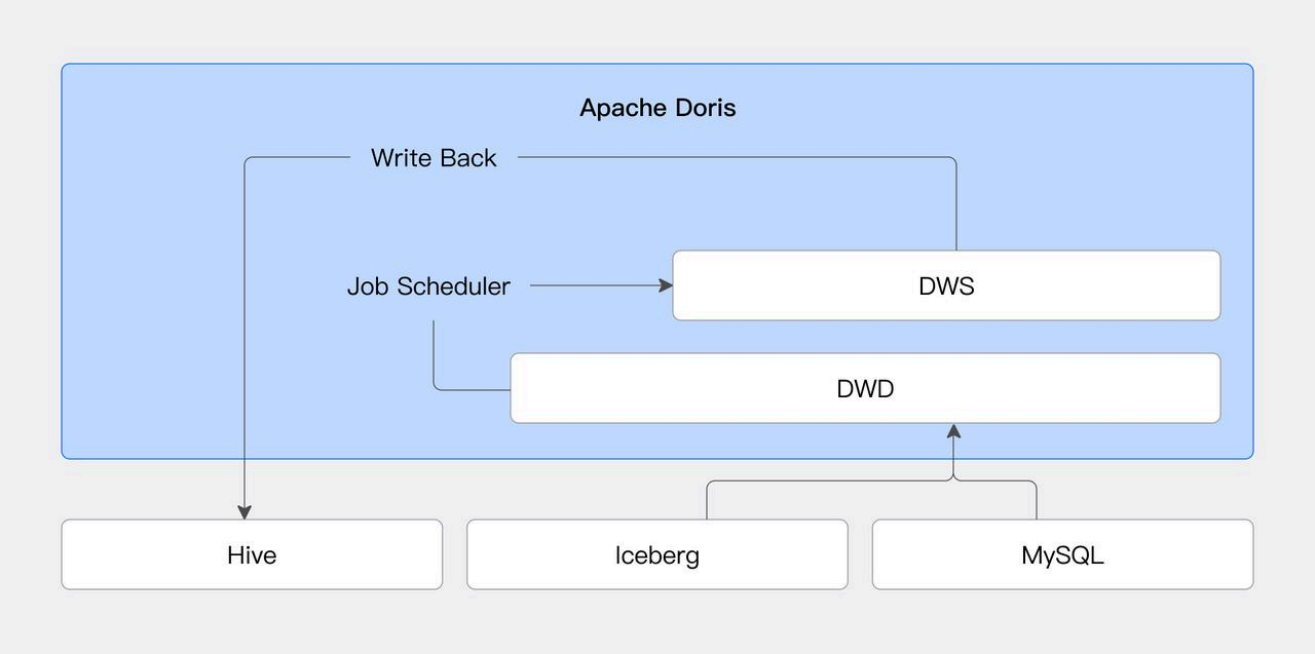


federated analysis

Users can dynamically create multiple catalogs in Doris to connect different data sources. They can use SQL statements to perform arbitrary join queries on data from different data sources. For details, refer to the Catalog Overview.

Lakehouse Data Processing

In this scenario, **Doris acts as a data processing engine**, processing lakehouse data.



lakehouse data processing

Task Scheduling

Doris introduces the Job Scheduler feature, enabling efficient and flexible task scheduling, reducing dependency on external systems. Combined with data source connectors, users can achieve periodic processing and storage of external data. For details, refer to the Job Scheduler.

Data Modeling

User typically use data lakes to store raw data and perform layered data processing on this basis, making different layers of data available to different business needs. Doris's materialized view feature supports creating materialized views for external data sources and supports further processing based on materialized views, reducing system complexity and improving data processing efficiency.

Data Write-Back

The data write-back feature forms a closed loop of Doris's lakehouse data processing capabilities. Users can directly create databases and tables in external data sources through Doris and write data. Currently, JDBC, Hive, and Iceberg data sources are supported, with more data sources to be added in the future. For details, refer to the documentation of the corresponding data source.

Data Catalog Overview

A Data Catalog is used to describe the attributes of a data source.

In Doris, multiple catalogs can be created to point to different data sources (such as Hive, Iceberg, MySQL). Doris will automatically obtain the databases, tables, columns, partitions, data locations, etc. of the corresponding data sources through the catalogs. Users can access these catalogs for data analysis through standard SQL statements and can conduct join queries on the data from multiple catalogs.

There are two types of catalogs in Doris:

Type	Description
Internal Catalog	The built-in catalog, named `internal`, used to store Doris internal table data. It cannot be created, modified, or deleted.
External Catalog	External catalogs refer to all catalogs other than the Internal Catalog. Users can create, modify, and delete external catalogs.

Catalogs are mainly applicable to the following three scenarios, but different catalogs are suitable for different scenarios. For details, see the documentation for the corresponding catalog.

Scenario	Description
Query Acceleration	Direct query acceleration for data lakes such as Hive, Iceberg, Paimon, etc.
Data Integration	ZeroETL solution, directly accessing different data sources to generate result data, or facilitating data flow between different data sources.
Data Write-back	After data processing via Doris, write back to external data sources.

This document uses Iceberg Catalog as an example to focus on the basic operations of catalogs. For detailed descriptions of different catalogs, please refer to the documentation of the corresponding catalog.

Creating Catalog

Create an Iceberg Catalog using the `CREATE CATALOG` statement.

```
CREATE CATALOG iceberg_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 's3://bucket/dir/key',
    's3.endpoint' = 'https://s3.us-east-1.amazonaws.com',
    's3.access_key' = 'ak',
    's3.secret_key' = 'sk'
);
```

Essentially, a catalog created in Doris acts as a “proxy” to access the metadata services (such as Hive Metastore) and storage services (such as HDFS/S3) of the corresponding data source. Doris only stores connection properties and other information of the catalog, not the actual metadata and data of the corresponding data source.

Common Properties

In addition to the set of properties specific to each catalog, here are the common properties for all catalogs `{CommonProperties}`.

Property Name	Description	Example
`include_database_list`	Supports synchronizing only specified Databases, separated by `,`. By default, all Databases are synchronized. Database names are case-sensitive. Use this parameter when there are many Databases in the external data source but only a few need to be accessed, to avoid synchronizing a large amount of metadata.	`'include_database_list' = 'db1,db2'`
`exclude_database_list`	Supports specifying multiple Databases that do not need to be synchronized, separated by `,`. By default, no filtering is applied, and all Databases are synchronized. Database names are case-sensitive. This is used in the same scenario as above, to exclude databases that do not need to be accessed. If there is a conflict, `exclude` takes precedence over `include`.	`'exclude_database_list' = 'db1,db2'`

Column Type Mapping

After a user creates a catalog, Doris automatically synchronizes the databases, tables, and columns of the catalog. For column type mapping rules of different catalogs, please refer to the documentation of the corresponding catalog.

For external data types that cannot currently be mapped to Doris column types, such as `UNION`, `INTERVAL`, etc., Doris will map the column type to `UNSUPPORTED`. For queries involving `UNSUPPORTED` types, see the example below:

Assume the synchronized table schema is:

```
k1 INT,
k2 INT,
k3 UNSUPPORTED,
k4 INT
```

The query behavior is as follows:

```
SELECT * FROM table;               -- Error: Unsupported type 'UNSUPPORTED_TYPE' in 'k3'
SELECT * EXCEPT(k3) FROM table;    -- Query OK.
SELECT k1, k3 FROM table;          -- Error: Unsupported type 'UNSUPPORTED_TYPE' in 'k3'
SELECT k1, k4 FROM table;          -- Query OK.
```

Nullable Attribute

Doris currently has special restrictions on the Nullable attribute support for external table columns, with specific behaviors as follows:

Source Type	Doris Read Behavior	Doris Write Behavior
Nullable	Nullable	Allow writing Null values
Not Null	Nullable, i.e., still treated as columns that allow NULL during reading	Allow writing Null values, i.e., no strict checking for Null values. Users need to ensure data integrity and consistency themselves.

Using Catalog

Viewing Catalog

After creation, you can view the catalog using the `SHOW CATALOGS` command:

```
mysql> SHOW CATALOGS;
+-----------+-----------------+----------+-----------+-------------------------+---------------------+-----------------------+
| CatalogId | CatalogName     | Type     | IsCurrent | CreateTime              | LastUpdateTime      | Comment               |
+-----------+-----------------+----------+-----------+-------------------------+---------------------+-----------------------+
|     10024 | iceberg_catalog | iceberg  | yes       | 2023-12-25 16:11:41.687 | 2023-12-25 20:43:18 | NULL                  |
|         0 | internal        | internal |           | NULL                    | NULL                | Doris internal catalog |
+-----------+-----------------+----------+-----------+-------------------------+---------------------+-----------------------+
```

You can view the statement to create a catalog using SHOW CREATE CATALOG.

Switching Catalog

Doris provides the `SWITCH` statement to switch the connection session context to the corresponding catalog. This is similar to using the `USE` statement to switch databases.

After switching to a catalog, you can use the `USE` statement to continue switching to a specified database, or use `SHOW DATABASES` to view the databases under the current catalog.

```
SWITCH iceberg_catalog;

SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
| iceberg_db         |
+--------------------+

USE iceberg_db;
```

You can also use the `USE` statement with the fully qualified name `catalog_name.database_name` to switch directly to a specified database within a specified catalog:

```
USE iceberg_catalog.iceberg_db;
```

Fully qualified names can also be used in MySQL command line or JDBC connection strings to be compatible with the MySQL connection protocol.

```
# Command line tool
mysql -h host -P9030 -uroot -Diceberg_catalog.iceberg_db

# JDBC url
jdbc:mysql://host:9030/iceberg_catalog.iceberg_db
```

The fixed name for the built-in catalog is `internal`. The switching method is the same as for external catalogs.

Default Catalog

The user attribute `default_init_catalog` is used to set the default catalog for a specific user. Once set, when the specified user connects to Doris, they will automatically switch to the set catalog.

```
SET PROPERTY default_init_catalog=hive_catalog;
```

Note 1: If the catalog has been explicitly specified in the MySQL command line or JDBC connection strings, then the specified catalog will be used, and the `default_init_catalog` user attribute will not take effect. Note 2: If the catalog set by the user attribute `default_init_catalog` no longer exists, it will automatically switch to the default `internal` catalog. Note 3: This feature takes effect starting from version v3.1.x.

Simple Query

You can query tables in external catalogs using any SQL statement supported by Doris.

```
SELECT id, SUM(cost) FROM iceberg_db.table1
GROUP BY id ORDER BY id;
```

Cross-Catalog Query

Doris supports join queries across different catalogs.

Here, let's create another MySQL Catalog:

```
CREATE CATALOG mysql_catalog properties(
    'type' = 'jdbc',
    'user' = 'root',
    'password' = '123456',
    'jdbc_url' = 'jdbc:mysql://host:3306/mysql_db',
    'driver_url' = 'mysql-connector-java-8.0.25.jar',
    'driver_class' = 'com.mysql.cj.jdbc.Driver'
);
```

Then, perform a join query between Iceberg tables and MySQL tables using SQL:

```
SELECT * FROM
iceberg_catalog.iceberg_db.table1 tbl1 JOIN mysql_catalog.mysql_db.dim_table tbl2
ON tbl1.id = tbl2.id;
```

Data Import

You can import data from a data source into Doris using the `INSERT` command.

```
INSERT INTO internal.doris_db.tbl1
SELECT * FROM iceberg_catalog.iceberg_db.table1;
```

You can also use the `CTAS (Create Table As Select)` statement to create an internal Doris table from an external data source and import the data:

```
CREATE TABLE internal.doris_db.tbl1
PROPERTIES('replication_num' = '1')
AS
SELECT * FROM iceberg_catalog.iceberg_db.table1;
```

Data Write-Back

Doris supports writing data back to external data sources using the `INSERT` statement. For more details, refer to:

- Hive Catalog
- Iceberg Catalog
- JDBC Catalog

Refreshing Catalog

Catalogs created in Doris act as “proxy” to access the metadata services of corresponding data sources. Doris caches some metadata to improve access performance and reduce frequent cross-network requests. However, the cache has a validity period, and without refreshing, you cannot access the latest metadata. Therefore, Doris provides several ways to refresh catalogs.

```
-- Refresh catalog
REFRESH CATALOG catalog_name;

-- Refresh specified database
REFRESH DATABASE catalog_name.db_name;

-- Refresh specified table
REFRESH TABLE catalog_name.db_name.table_name;
```

Doris also supports disabling metadata caching to access the latest metadata in real-time.

For detailed information and configuration of metadata caching, please refer to: Metadata Cache

Modifying Catalog

You can modify the properties or name of a catalog using the `ALTER CATALOG` statement:

```
-- Rename a catalog
ALTER CATALOG iceberg_catalog RENAME iceberg_catalog2;

-- Modify properties of a catalog
```

```
ALTER CATALOG iceberg_catalog SET PROPERTIES ('key1' = 'value1' [, 'key' = 'value2']);

-- Modify the comment of a catalog
ALTER CATALOG iceberg_catalog MODIFY COMMENT 'my iceberg catalog';
```

Deleting Catalog

You can delete a specified external catalog using the `DROP CATALOG` statement.

```
DROP CATALOG [IF EXISTS] iceberg_catalog;
```

Deleting an external catalog from Doris does not remove the actual data; it only deletes the mapping relationship stored in Doris.

Permission Management

The permission management for databases and tables in an external catalog is the same as for internal tables. For details, refer to the Authentication and Authorization documentation.

Hive Catalog

By connecting to Hive Metastore or metadata services compatible with Hive Metastore, Doris can automatically retrieve Hive database and table information for data querying.

In addition to Hive, many other systems use Hive Metastore to store metadata. Therefore, through the Hive Catalog, we can access not only Hive tables but also other table formats that use Hive Metastore for metadata storage, such as Iceberg and Hudi.

> **Writing to Hive through Doris is supported from version 2.1.3. Earlier 2.1.x releases only support querying.**

Applicable Scenarios

Scenario	Description
Query Acceleration	Use Doris's distributed computing engine to directly access Hive data for query acceleration.
Data Integration	Read Hive data and write it to Doris internal tables, or perform ZeroETL operations using the Doris computing engine.
Data Write-back	Process data from any source supported by Doris and write it back to Hive tables.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name PROPERTIES (
    'type'='hms', -- required
    'hive.metastore.type' = '<hive_metastore_type>', -- optional
    'hive.version' = '<hive_version>', -- optional
    'fs.defaultFS' = '<fs_defaultfs>', -- required for write, optional for read-only
    {MetaStoreProperties},
    {StorageProperties},
    {HiveProperties},
    {CommonProperties}
);
```

- `<hive_metastore_type>`

  Specify the type of Hive Metastore.

  - `hms`: Standard Hive Metastore service.
  - `glue`: Access AWS Glue metadata service using Hive Metastore compatible interface.
  - `dlf`: Access Alibaba Cloud DLF metadata service using Hive Metastore compatible interface.

- `<fs_defaultfs>`

This parameter is required when writing data from Doris to tables in this Hive Catalog. Example:

`'fs.defaultFS' = 'hdfs://namenode:port'`

- `{MetaStoreProperties}`

  The MetaStoreProperties section is for entering connection and authentication information for the Metastore metadata service. Refer to the “Supported Metadata Services” section for details.

- `{StorageProperties}`

  The StorageProperties section is for entering connection and authentication information related to the storage system. Refer to the “Supported Storage Systems” section for details.

- `{HiveProperties}`

  HiveProperties section is for entering properties related to Hive Catalog.

  - `get_schema_from_table`: The default value is false. By default, Doris will obtain the table schema information from the Hive Metastore. However, in some cases, compatibility issues may occur, such as the error `Storage schema reading not supported`. In this case, you can set this parameter to true, and the table schema will be obtained directly from the Table object. But please note that this method will cause the default value information of the column to be ignored. This property is supported since version 2.1.10 and 3.0.6.

  - `hive.recursive_directories`: Whether to recurse into subdirectories when listing partition directories. This parameter has been supported since version 3.0.2. Before version 4.0, this parameter defaulted to `false`; in later versions, it defaults to `true`. The partition paths of some Hive tables may not match the partition information in the table schema. Setting this parameter to true is necessary to retrieve data files in subdirectories.

  - `hive.ignore_absent_partitions`: Whether to ignore non-existent partitions. Defaults to `true`. If set to `false`, the query will report an error when encountering non-existent partitions. This parameter has been supported since version 3.0.2.

- `{CommonProperties}`

  The CommonProperties section is for entering common attributes. Please see the “Common Properties” section in the Catalog Overview.

Supported Hive Versions

Supports Hive 1.x, 2.x, 3.x, and 4.x.

Hive transactional tables are supported from version 3.x onwards. For details, refer to the “Hive Transactional Tables” section.

Supported Metadata Services

- Hive Metastore
- AWS Glue
- Aliyun DLF

> Note: The service types and parameters supported by different Doris versions are slightly different. Please refer to the Examples section.

Hive Catalog Feature Support Matrix

Metadata Service	Table Query	View Query	DDL Operations	Data Writeback
Hive	✅	✅	✅	✅
AWS Glue	✅	✅	❌	❌
DLF	✅	✅	✅	✅

Supported Storage Systems

- HDFS
- AWS S3
- Google Cloud Storage
- Azure Blob
- Alibaba Cloud OSS
- Tencent Cloud COS
- Huawei Cloud OBS
- MINIO

> To create Hive tables and write data through Doris, you need to explicitly add the `fs.defaultFS` property in the Catalog attributes. If the Catalog is created only for querying, this parameter can be omitted.
>
> The service types and parameters supported by different Doris versions are slightly different. Please refer to the Examples section.

Supported Data Formats

- Hive
  - Parquet
  - ORC
  - Text/CSV/JSON
- Hudi
  - Parquet
  - ORC
- Iceberg

- Parquet
- ORC

Column Type Mapping

Hive Type	Doris Type	Comment
boolean	boolean	
tinyint	tinyint	
smallint	smallint	
int	int	
bigint	bigint	
date	date	
timestamp	datetime(6)	Mapped to datetime with precision 6
float	float	
double	double	
decimal(P, S)	decimal(P, S)	Defaults to decimal(9, 0) if precision not specified
char(N)	char(N)	
varchar(N)	varchar(N)	
string	string	
binary	string/varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
array	array	
map	map	
struct	struct	
other	unsupported	

Examples

Hive Metastore

Version 3.1+

HDFS

````
Access HMS and HDFS services without Kerberos authentication
```sql
CREATE CATALOG hive_hms_hdfs_test_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
````
````

```
            'hadoop.username' = 'doris'
        );
        ```
        Access HMS and HDFS services with Kerberos authentication enabled
        ```sql
        CREATE CATALOG hive_hms_hdfs_kerberos_test_catalog PROPERTIES (
            'type' = 'hms',
            'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
            'hive.metastore.client.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
            'hive.metastore.client.keytab' = '/keytabs/hive-presto-master.keytab',
            'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
            'hive.metastore.sasl.enabled' = 'true',
            'hive.metastore.authentication.type' = 'kerberos',
            'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
            'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
            RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
            RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
            DEFAULT',
            'hadoop.security.authentication' = 'kerberos',
            'hadoop.kerberos.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
            'hadoop.kerberos.keytab' = '/keytabs/hive-presto-master.keytab'
        );
        ```
```

## ViewFs

```
        ```sql
        CREATE CATALOG hive_viewfs PROPERTIES (
            'type' = 'hms',
            'hive.metastore.uris' = 'thrift://172.0.0.1:9083',
            'fs.defaultFS' = 'viewfs://your-cluster',
            'fs.viewfs.mounttable.your-cluster.link./ns1' = 'hdfs://your-nameservice/',
            'fs.viewfs.mounttable.your-cluster.homedir' = '/ns1',
            'dfs.nameservices' = 'your-nameservice',
            'dfs.ha.namenodes.your-nameservice' = 'nn1,nn2',
            'dfs.namenode.rpc-address.your-nameservice.nn1' = '172.21.0.2:8088',
            'dfs.namenode.rpc-address.your-nameservice.nn2' = '172.21.0.3:8088',
            'dfs.client.failover.proxy.provider.your-nameservice' =
'org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider'
        );
        ```
```

## S3

```
        Access HMS without Kerberos authentication
        ```sql
        CREATE CATALOG hive_hms_s3_test_catalog PROPERTIES (
            'type' = 'hms',
            'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
            's3.region' = 'ap-east-1',
            's3.access_key' = '<ak>',
            's3.secret_key' = '<sk>'
        );
        ```
        Obtaining S3 access credentials using an IAM Assumed Role (3.1.2+)
        ```sql
        CREATE CATALOG hive_hms_on_s3_iamrole PROPERTIES (
            'type' = 'hms',
```

```
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.region'   = 'us-east-1',
    's3.role_arn' = 'arn:aws:iam::543815668950:role/role'
);
```

OSS

Access HMS without Kerberos authentication
```sql
CREATE CATALOG hive_hms_on_oss_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```
Access HMS with Kerberos authentication enabled
```sql
CREATE CATALOG hive_hms_on_oss_kerberos_old_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.client.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/keytabs/hive-presto-master.keytab',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hive.metastore.authentication.type' = 'kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```

COS

Access HMS without Kerberos authentication
```sql
CREATE CATALOG hive_hms_cos_test_region_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```
Access HMS with Kerberos authentication enabled
```sql
CREATE CATALOG hive_hms_on_cos_kerberos_new_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
```

```
    'hive.metastore.client.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/keytabs/hive-presto-master.keytab',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hive.metastore.authentication.type' = 'kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```
```

## OBS

```
Access HMS without Kerberos authentication
```sql
CREATE CATALOG test_hive_on_hms_obs_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```
Access HMS with Kerberos authentication enabled
```sql
CREATE CATALOG hive_hms_on_obs_kerberos_new_catalog PROPERTIES (
    'type' = 'hms',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hive.metastore.client.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/keytabs/hive-presto-master.keytab',
    'hive.metastore.authentication.type' = 'kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```
```

## GCS

```
Access HMS without Kerberos authentication
```sql
CREATE CATALOG `hive_hms_on_gcs_new_catalog` PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
```

```
    'gs.access_key' = '<ak>',
    'gs.secret_key' = '<sk>',
    'fs.gcs.support'='true'
);
```

Azure Blob

Supported since 3.1.3

```sql
CREATE CATALOG `hive_hms_on_azure_blob_new_catalog` PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'azure.account_name' = '<account_name>',
    'azure.account_key' = '<account_key>',
    'azure.endpoint' = 'https://<account_name>.blob.core.windows.net',
    'fs.azure.support'='true'
);
```

MinIO

```sql
CREATE CATALOG test_hive_on_hms_minio_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'fs.minio.support' = 'true',
    'minio.use_path_style' = 'true',
    'minio.endpoint' = 'http://127.0.0.1:19001',
    'minio.access_key' = '<ak>',
    'minio.secret_key' = '<sk>'
);
```

Version 2.1 & 3.0

HDFS

Access HMS without Kerberos authentication

```sql
CREATE CATALOG hive_hms_hdfs_test_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'hadoop.username' = 'doris',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8320'
);

```

Access HMS with Kerberos authentication enabled

```sql
CREATE CATALOG test_two_hive_kerberos PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.sasl.enabled' = 'true',
```

```
    'hive.metastore.kerberos.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
    'hadoop.kerberos.min.seconds.before.relogin' = '5',
    'hadoop.security.authentication' = 'kerberos',
    'hadoop.kerberos.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hadoop.kerberos.keytab' = '/keytabs/hive-presto-master.keytab',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT'
);
```

S3

```sql
CREATE CATALOG hms_on_s3_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.region' = 'ap-east-1',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```

OSS

```sql
CREATE CATALOG hive_hms_on_oss_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```

COS

```sql
CREATE CATALOG hive_hms_cos_test_region_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```

OBS

````
```sql
CREATE CATALOG test_hive_on_hms_obs_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```
````

GCS

````
```sql
CREATE CATALOG hive_hms_on_gcs_s3 PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 's3://bucket/gcs/paimon_warehouse',
    'gs.access_key' = '<ak>',
    'gs.secret_key' = '<sk>',
    'gs.endpoint' = 'storage.googleapis.com'
);
```
````

MinIO

````
```sql
CREATE CATALOG test_hive_on_hms_minio_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.endpoint' = 'http://127.0.0.1:19000',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.path.style.access' = 'true'
);
```
````

AWS Glue

Version 3.1+

S3

````
AWS Glue and S3 storage services share the same authentication information.
```sql
CREATE CATALOG hive_glue_on_s3_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.type' = 'glue',
    'glue.region' = 'ap-east-1',
    'glue.endpoint' = 'https://glue.ap-east-1.amazonaws.com',
````
````

```
            'glue.access_key' = '<ak>',
            'glue.secret_key' = '<sk>'
        );
        ```

        When Glue service authentication information differs from S3 authentication information, you can specify S3 authentication information separately in the following way.
        ```sql
        CREATE CATALOG hive_glue_on_s3_catalog PROPERTIES (
            'type' = 'hms',
            'hive.metastore.type' = 'glue',
            'glue.region' = 'ap-east-1',
            'glue.endpoint' = 'https://glue.ap-east-1.amazonaws.com',
            'glue.access_key' = '<ak>',
            'glue.secret_key' = '<sk>',
            's3.region' = 'ap-east-1',
            's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com/',
            's3.access_key' = '<ak>',
            's3.secret_key' = '<sk>'
        );
        ```

        Using IAM Assumed Role to obtain S3 access credentials (Since 3.1.2+)
        ```sql
        CREATE CATALOG `glue_hive_iamrole` PROPERTIES (
            'type' = 'hms',
            'hive.metastore.type' = 'glue',
            'glue.region' = 'us-east-1',
            'glue.endpoint' = 'https://glue.us-east-1.amazonaws.com',
            'glue.role_arn' = '<role_arn>'
        );
        ```
```

### Version 2.1 & 3.0

## S3

```
        AWS Glue and S3 storage services share the same authentication credentials.

        In non-EC2 environments, you need to use [aws configure](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html) to configure Credentials information and generate a credentials file in the ~/.aws directory.

        Alternatively, you can explicitly specify the credentials provider factory class
        when creating the catalog:

        "aws.catalog.credentials.provider.factory.class" = "com.amazonaws.glue.catalog.credentials.ConfigurationAWSCredentialsProviderFactory"
        ```sql
        create catalog hive_glue PROPERTIES(
            'type' = 'hms',
            'hive.metastore.type' = 'glue',
            'glue.endpoint' = 'https://glue.ap-northeast-1.amazonaws.com',
            'glue.region' = 'ap-northeast-1',
            'glue.access_key' = '<ak>',
            'glue.secret_key' = '<sk>'
        );
        ```
```
```

Aliyun DLF

Version 3.1+

DLF 1.0

````
```sql
CREATE CATALOG hive_dlf_oss_test_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.type' = 'dlf',
    'dlf.uid' = '203225413946383283',
    'dlf.catalog_id' = 'p2_regression_case',
    'dlf.endpoint' = 'dlf.cn-beijing.aliyuncs.com',
    'dlf.region' = 'cn-beijing',
    'dlf.access_key' = '<ak>',
    'dlf.secret_key' = '<sk>'
);
```
````

Version 2.1 & 3.0

DLF 1.0

````
```sql
CREATE CATALOG hive_dlf_oss_test_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.type' = 'dlf',
    'dlf.uid' = '203225413946383283',
    'dlf.catalog.id' = 'p2_regression_case',
    'dlf.endpoint' = 'dlf.cn-beijing.aliyuncs.com',
    'dlf.region' = 'cn-beijing',
    'dlf.access_key' = '<ak>',
    'dlf.secret_key' = '<sk>'
);
```
````

Query Operations

Basic Query

After configuring the Catalog, you can query the table data within the Catalog using the following method:

```
-- 1. switch to catalog, use database and query
SWITCH hive_ctl;
USE hive_db;
SELECT * FROM hive_tbl LIMIT 10;

-- 2. use hive database directly
USE hive_ctl.hive_db;
SELECT * FROM hive_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM hive_ctl.hive_db.hive_tbl LIMIT 10;
```

Querying Hive Partitions

You can query Hive partition information using the following two methods:

- `SHOW PARTITIONS FROM [catalog.][db.]hive_table`

  This statement lists all partitions and their values for the specified Hive table.

```
SHOW PARTITIONS FROM hive_table;

+-------------------------------+
| Partition                     |
+-------------------------------+
| pt1=2024-10-10/pt2=beijing    |
| pt1=2024-10-10/pt2=shanghai   |
| pt1=2024-10-11/pt2=beijing    |
| pt1=2024-10-11/pt2=shanghai   |
| pt1=2024-10-12/pt2=nanjing    |
+-------------------------------+
```

- Using the `table$partitions` Metadata Table

  Starting from versions 2.1.7 and 3.0.3, you can query Hive partition information through the `table$partitions` metadata table. This table is essentially relational, with each partition column represented as a column, allowing it to be used in any SELECT statement.

```
SELECT * FROM hive_table$partitions;

+------------+-------------+
| pt1        | pt2         |
+------------+-------------+
| 2024-10-10 | beijing     |
| 2024-10-10 | shanghai    |
| 2024-10-12 | nanjing     |
| 2024-10-11 | beijing     |
| 2024-10-11 | shanghai    |
+------------+-------------+
```

Querying Hive Transactional Tables

Hive Transactional tables support ACID semantics. For more details, see Hive Transactions.

- Support for Hive Transactional Tables

Table Type	Supported Operations in Hive	Hive Table Properties	Supported Hive Versions
Full-ACID Transactional Table	Supports Insert, Update, Delete	`'transactional'='true'`	4.x, 3.x, 2.x (2.x requires Major Compaction in Hive to read)
Insert-Only Transactional Table	Supports Insert only	`'transactional'='true'`, `'transactional_properties'='insert_only'`	4.x, 3.x, 2.x (specify `hive.version` when creating the catalog)

- Current Limitations

Original Files scenarios are not supported. When a table is converted to a Transactional table, new data files will use the Hive Transactional table schema, but existing data files will not be converted. These files are referred to as Original Files.

Querying Hive Views

You can query Hive Views, but there are some limitations:

- The Hive View definition (HiveQL) must be compatible with SQL statements supported by Doris. Otherwise, a parsing error will occur.
- Some functions supported by HiveQL may have the same name as those in Doris but behave differently. This could lead to discrepancies between the results obtained from Hive and Doris. If you encounter such issues, please report them to the community.

Write Operations

Data can be written to Hive tables using the INSERT statement. This is supported for Hive tables created by Doris or existing Hive tables with compatible formats.

For partitioned tables, data will automatically be written to the corresponding partition or a new partition will be created based on the data. Currently, specifying a partition for writing is not supported.

INSERT INTO

The INSERT operation appends data to the target table. Specifying a partition for writing is currently not supported.

```
INSERT INTO hive_tbl VALUES (val1, val2, val3, val4);
INSERT INTO hive_ctl.hive_db.hive_tbl SELECT col1, col2 FROM internal.db1.tbl1;

INSERT INTO hive_tbl(col1, col2) VALUES (val1, val2);
INSERT INTO hive_tbl(col1, col2, partition_col1, partition_col2) VALUES (1, 2, "beijing", "2023-12-12");
```

INSERT OVERWRITE

INSERT OVERWRITE completely replaces the existing data in the table with new data. Specifying a partition for writing is currently not supported.

```
INSERT OVERWRITE TABLE hive_tbl VALUES (val1, val2, val3, val4);
INSERT OVERWRITE TABLE hive_ctl.hive_db.hive_tbl(col1, col2) SELECT col1, col2 FROM internal.db1.tbl1;
```

The semantics of INSERT OVERWRITE are consistent with Hive, with the following behaviors:

- If the target table is partitioned and the source table is empty, the operation has no effect. The target table remains unchanged.
- If the target table is non-partitioned and the source table is empty, the target table will be cleared.

- Since specifying a partition for writing is not supported, INSERT OVERWRITE automatically handles the relevant partitions in the target table based on the source table values. If the target table is partitioned, only the affected partitions will be overwritten; unaffected partitions remain unchanged.

CTAS

You can create a Hive table and insert data using the `CTAS (CREATE TABLE AS SELECT)` statement:

```
CREATE TABLE hive_ctas ENGINE=hive AS SELECT * FROM other_table;
```

CTAS supports specifying file formats, partitioning methods, and more, as shown below:

```
CREATE TABLE hive_ctas ENGINE=hive
PARTITION BY LIST (pt1, pt2) ()
AS SELECT col1, pt1, pt2 FROM part_ctas_src WHERE col1 > 0;

CREATE TABLE hive_ctl.hive_db.hive_ctas (col1, col2, pt1) ENGINE=hive
PARTITION BY LIST (pt1) ()
PROPERTIES (
    "file_format"="parquet",
    "compression"="zstd"
)
AS SELECT col1, pt1 AS col2, pt2 AS pt1 FROM test_ctas.part_ctas_src WHERE col1 > 0;
```

Related Parameters

- Session variables

Parameter name	Default value	Desciption	Since version
`hive_parquet_use_column_names`	`true`	When Doris reads the Parquet data type of the Hive table, it will find the column with the same name from the Parquet file to read the data according to the column name of the Hive table by default. When this variable is `false`, Doris will read data from the Parquet file according to the column order in the Hive table, regardless of the column name. Similar to the `parquet.column.index.access` variable in Hive. This parameter only applies to the top-level column name and is invalid inside the Struct.	2.1.6+, 3.0.3+

Parameter name	Default value	Desciption	Since version
`hive_orc_use_column_names`	`true`	Similar to `hive_parquet_use_column_names`, it is for the Hive table ORC data type. Similar to the `orc.force.positional.evolution` variable in Hive.	2.1.6+, 3.0.3+

- BE

Parameter Name	Default Value	Description
hive_sink_max_file_size	1GB	Maximum data file size. When the data size exceeds this limit, the current file will be closed, and a new file will be created for further writing.

Parameter Name	Default Value	Description
table_sink_partition_write_max_partition_nums_per_writer	128	Maximum number of partitions that each instance can write to on a BE node.

Parameter Name	Default Value	Description
table_sink_non_partition_write_scaling_data_processed_threshold	25MB	Data volume threshold for starting scaling-write for non-partitioned tables. A new writer (instance) will be used for every additional table_sink_non_partition_write_scaling_data_processe

Parameter Name	Default Value	Description
		d_threshold of data. This mechanism adjusts the number of writers (instances) based on data volume to enhance concurrent write throughput, saving resources and minimizing

Parameter Name	Default Value	Description
		file numbers for smaller data volumes.

Parameter Name	Default Value	Description
table_sink_partition_write_min_data_processed_rebalance_threshold	25MB	Minimum data volume threshold to trigger rebalancing for partitioned tables. Rebalancing is triggered if current accumulated data volume - data volume since last rebalancing or initial accu

Parameter Name	Default Value	Description
		mulation >= table_sink_partition_write_min_data_processed_rebalance_threshold. If the final file size varies significantly, reduce this threshold to improve balance. However, a lower threshold may

Parameter Name	Default Value	Description
		increase rebalancing costs, potentially affecting performance.

Parameter Name	Default Value	Description
table_sink_partition_write_min_partition_data_processed_rebalance_threshold	15MB	Minimum partition data volume threshold to trigger rebalancing. Rebalancing occurs if current partition data volume >= threshold * number of tasks already allocated to the partition. If

Parameter Name	Default Value	Description
		the final file size varies significantly, reduce this threshold to improve balance. However, a lower threshold may increase rebalancing costs, potentially affecting performance.

Database and Table Management

Users can create and delete databases and tables in the Hive Metastore through Doris. Note that Doris only calls the Hive Metastore API for these operations and does not store or persist any Hive metadata itself.

Creating and Dropping Databases

You can switch to the appropriate Catalog using the `SWITCH` statement and execute the `CREATE DATABASE` statement:

```
SWITCH hive_ctl;
CREATE DATABASE [IF NOT EXISTS] hive_db;
```

You can also create a database using a fully qualified name or specify a location, such as:

```
CREATE DATABASE [IF NOT EXISTS] hive_ctl.hive_db;

CREATE DATABASE [IF NOT EXISTS] hive_ctl.hive_db
PROPERTIES ('location'='hdfs://172.21.16.47:4007/path/to/db/');
```

You can view the location information of the Database using the `SHOW CREATE DATABASE` command:

```
mysql> SHOW CREATE DATABASE hive_db;
+----------+-----------------------------------------------------------------------------------------------+
| Database | Create Database                                                                               |
+----------+-----------------------------------------------------------------------------------------------+
| hive_db  | CREATE DATABASE hive_db LOCATION 'hdfs://172.21.16.47:4007/usr/hive/warehouse/hive_db.db'   |
+----------+-----------------------------------------------------------------------------------------------+
```

To drop a database:

```
DROP DATABASE [IF EXISTS] hive_ctl.hive_db;
```

> **For a Hive Database, you must first delete all tables under that Database before you can delete the Database itself; otherwise, an error will occur. This operation will also delete the corresponding Database in Hive.**

Creating and Dropping Tables

- **Creating Tables**

  Doris supports creating both partitioned and non-partitioned tables in Hive.

  ```
  -- Create unpartitioned hive table
  CREATE TABLE unpartitioned_table (
    `col1` BOOLEAN COMMENT 'col1',
    `col2` INT COMMENT 'col2',
    `col3` BIGINT COMMENT 'col3',
    `col4` CHAR(10) COMMENT 'col4',
  ```

```
  `col5` FLOAT COMMENT 'col5',
  `col6` DOUBLE COMMENT 'col6',
  `col7` DECIMAL(9,4) COMMENT 'col7',
  `col8` VARCHAR(11) COMMENT 'col8',
  `col9` STRING COMMENT 'col9'
)  ENGINE=hive
PROPERTIES (
  'file_format'='parquet'
);

-- Create partitioned hive table
-- The partition columns must be in table's column definition list
CREATE TABLE partition_table (
  `col1` BOOLEAN COMMENT 'col1',
  `col2` INT COMMENT 'col2',
  `col3` BIGINT COMMENT 'col3',
  `col4` DECIMAL(2,1) COMMENT 'col4',
  `pt1` VARCHAR COMMENT 'pt1',
  `pt2` VARCHAR COMMENT 'pt2'
)  ENGINE=hive
PARTITION BY LIST (pt1, pt2) ()
PROPERTIES (
  'file_format'='orc',
  'compression'='zlib'
);

-- Create text format table(Since 2.1.7 & 3.0.3)
CREATE TABLE text_table (
    `id` INT,
    `name` STRING
) PROPERTIES (
    'file_format'='text',
    'compression'='gzip',
    'field.delim'='\t',
    'line.delim'='\n',
    'collection.delim'=';',
    'mapkey.delim'=':',
    'serialization.null.format'='\\N',
    'escape.delim'='\\'
);
```

After creating a table, you can view the Hive table creation statement using the `SHOW CREATE TABLE` command.

Note that unlike Hive's table creation syntax, when creating a partitioned table in Doris, partition columns must be included in the table schema. Additionally, partition columns must be placed at the end of the schema and maintain the same order.

For Hive clusters where ACID transaction features are enabled by default, tables created by Doris will have the `transactional` property set to `true`. Since Doris only supports certain features of Hive transactional tables, this may lead to issues where Doris cannot read the Hive tables it creates. To avoid this, explicitly set `"transactional" = "false"` in the table properties to create non-transactional Hive tables:

```
CREATE TABLE non_acid_table(
  `col1` BOOLEAN COMMENT 'col1',
  `col2` INT COMMENT 'col2',
  `col3` BIGINT COMMENT 'col3'
) ENGINE=hive
PROPERTIES (
```

```
  'transactional'='false'
);
```

- **Dropping Tables**

  You can delete a Hive table using the `DROP TABLE` statement. When a table is deleted, all data, including partition data, is also removed.

- **Column Type Mapping**

  Refer to the Column Type Mapping section for details. Note the following restrictions:

  - Columns must be of the default nullable type; `NOT NULL` is not supported.
  - Hive 3.0 supports setting default values. To set default values, explicitly add `"hive.version" = "3.0.0"` in the catalog properties.
  - If inserted data types are incompatible (e.g., inserting `'abc'` into a numeric type), the value will be converted to `null`.

- **Partitioning**

  In Hive, partition types correspond to List partitions in Doris. Therefore, when creating a Hive partitioned table in Doris, use the List partition syntax, but there is no need to explicitly enumerate each partition. Doris will automatically create the corresponding Hive partition based on data values during data insertion. Single-column or multi-column partitioned tables are supported.

- **File Formats**

  - ORC (default)
  - Parquet

    Note that when the DATETIME type is written to a Parquet file, the physical type used is INT96 instead of INT64. This is to be compatible with the logic of Hive versions prior to 4.0.

  - Text (supported from versions 2.1.7 and 3.0.3)

    Text format supports the following table properties:

    ```
    - `field.delim`: Column delimiter. Default is `\1`.
    - `line.delim`: Line delimiter. Default is `\n`.
    - `collection.delim`: Delimiter for elements in complex types. Default is `\2`.
    - `mapkey.delim`: Delimiter for map key-value pairs. Default is `\3`.
    - `serialization.null.format`: Format for storing `NULL` values. Default is `\N`.
    - `escape.delim`: Escape character. Default is `\`.
    ```

- **Compression Formats**

  - Parquet: snappy (default), zstd, plain (no compression)
  - ORC: snappy, zlib (default), zstd, plain (no compression)
  - Text: gzip, deflate, bzip2, zstd, lz4, lzo, snappy, plain (default, no compression)

- **Storage Medium**

  - HDFS

- Object Storage

Subscribing to Hive Metastore Events

By having the FE nodes periodically read Notification Events from the HMS, Doris can detect real-time changes in Hive table metadata, improving metadata timeliness. Currently, the following events are supported:

Event	Action and Corresponding Behavior
CREATE DATABASE	Creates a database in the corresponding data directory.
DROP DATABASE	Deletes a database in the corresponding data directory.
ALTER DATABASE	Mainly affects changes to database properties, comments, and default storage locations. These changes do not affect Doris's ability to query external data directories, so this event is currently ignored.
CREATE TABLE	Creates a table in the corresponding database.
DROP TABLE	Deletes a table in the corresponding database and invalidates the table cache.
ALTER TABLE	If renamed, deletes the old table and creates a new one with the new name; otherwise, invalidates the table cache.
ADD PARTITION	Adds a partition to the cached partition list of the corresponding table.
DROP PARTITION	Removes a partition from the cached partition list and invalidates the partition cache.
ALTER PARTITION	If renamed, deletes the old partition and creates a new one with the new name; otherwise, invalidates the partition cache.

> **1. When data import causes file changes, partitioned tables trigger an `ALTER PARTITION` event, while non-partitioned tables trigger an `ALTER TABLE` event.**
>
> 2. If you bypass HMS and directly manipulate the file system, HMS will not generate corresponding events, and Doris will not detect metadata changes.

The following parameters in `fe.conf` are related to this feature:

1. `enable_hms_events_incremental_sync`: Enables automatic incremental metadata synchronization. Disabled by default.
2. `hms_events_polling_interval_ms`: Interval for reading events, default is 10000 milliseconds.
3. `hms_events_batch_size_per_rpc`: Maximum number of events to read per RPC, default is 500.

To use this feature (excluding Huawei Cloud MRS), you need to modify the `hive-site.xml` of HMS and restart both HMS and HiveServer2.

```
<property>
    <name>hive.metastore.event.db.notification.api.auth</name>
    <value>false</value>
</property>
<property>
    <name>hive.metastore.dml.events</name>
    <value>true</value>
</property>
```

```
<property>
    <name>hive.metastore.transactional.event.listeners</name>
    <value>org.apache.hive.hcatalog.listener.DbNotificationListener</value>
</property>
```

For Huawei Cloud MRS, you need to modify the `hivemetastore-site.xml` and restart both HMS and HiveServer2.

```
<property>
    <name>metastore.transactional.event.listeners</name>
    <value>org.apache.hive.hcatalog.listener.DbNotificationListener</value>
</property>
```

Appendix

Transaction Mechanism

Write operations to Hive are placed in a separate transaction. Before the transaction is committed, the data is not visible externally. Only after the transaction is committed do the related table operations become visible to others.

Transactions ensure the atomicity of operations, meaning all operations within a transaction either succeed together or fail together.

Transactions cannot fully guarantee isolation of operations, but they strive to minimize inconsistencies by separating file system operations from Hive Metastore metadata operations.

For example, in a transaction that requires modifying multiple partitions of a Hive table, if the task is divided into two batches, the first batch might be visible externally before the second batch is completed. This means the first batch of partitions can be read, but the second batch cannot.

If any exceptions occur during the transaction commit process, the transaction will be rolled back completely, including modifications to HDFS files and Hive Metastore metadata, without requiring any additional user intervention.

Concurrent Writing Mechanism

Apache Doris currently supports concurrent writing using multiple insert statements. However, users need to ensure that concurrent writes do not result in potential conflicts.

Since regular non-transactional Hive tables lack a complete transaction mechanism, the Apache Doris transaction mechanism aims to minimize inconsistency windows but cannot guarantee true ACID properties. Therefore, concurrent writes to Hive tables in Apache Doris may lead to data consistency issues.

1. **Concurrent `INSERT` Operations**
2. `INSERT` operations append data and do not conflict when executed concurrently, producing the expected results.
3. **Concurrent `INSERT OVERWRITE` Operations**

4. Concurrent `INSERT OVERWRITE` operations on the same table or partition may lead to data loss or corruption, resulting in unpredictable outcomes.
5. Common solutions include:
    - For partitioned tables, write data to different partitions. Concurrent operations on different partitions do not conflict.
    - For non-partitioned tables, use `INSERT` instead of `INSERT OVERWRITE` to avoid conflicts.
    - For operations that may conflict, ensure that only one write operation occurs at a time on the business side.

HDFS File Operations

For Hive table data on HDFS, data is typically first written to a temporary directory, then finalized using file system operations like `rename`. Below is a detailed explanation of the specific file operations on HDFS for different data operations.

The temporary directory format for data is: `/tmp/.doris_staging/<username>/<uuid>`

The format for the written data file names is: `<query-id>_<uuid>-<index>.<compress-type>.<file-type>`

Here are examples of file operations under various scenarios:

1. **Non-Partitioned Table**
    - **Append (Add Data)**
        - Target table directory: `hdfs://ns/usr/hive/warehouse/example.db/table1`
        - Temporary file: `hdfs://ns/tmp/.doris_staging/root/f02247cb662846038baae272af5eeb05/b35fdbcea3a4e39-86d1f36987ef1492_7e3985bf-9de9-4fc7-b84e-adf11aa08756-0.orc`
        - During the commit phase, all temporary files are moved to the target table directory.
    - **Overwrite (Replace Data)**
        - Target table directory: `hdfs://ns/usr/hive/warehouse/example.db/table1`
        - Temporary file: `hdfs://ns/tmp/.doris_staging/root/f02247cb662846038baae272af5eeb05/b35fdbcea3a4e39-86d1f36987ef1492_7e3985bf-9de9-4fc7-b84e-adf11aa08756-0.orc`
        - Commit phase steps:
            1. Rename the target table directory to a temporary directory: `hdfs://ns/usr/hive/warehouse/example.db/_temp_b35fdbcea3a4e39-86d1f36987ef1492_table1`
            2. Rename the temporary directory to the target table directory.
            3. Delete the temporary target table directory.

2. **Partitioned Table**

    - **Add (Add to New Partition)**

        - Target table directory: `hdfs://ns/usr/hive/warehouse/example.db/table2/part_col=2024-01-01`
        - Temporary file: `hdfs://ns/tmp/.doris_staging/root/a7eac7505d7a42fdb06cb9ef1ea3e912/par1=a/d678a74d232345e0-b659e2fb58e86ffd_549ad677-ee75-4fa1-b8a6-3e821e1dae61-0.orc`
        - During the commit phase, the temporary directory is renamed to the target table directory.

    - **Append (Add Data to Existing Partition)**

        - Target table directory: `hdfs://ns/usr/hive/warehouse/example.db/table2/part_col=2024-01-01`
        - Temporary file: `hdfs://ns/tmp/.doris_staging/root/a7eac7505d7a42fdb06cb9ef1ea3e912/par1=a/d678a74d232345e0-b659e2fb58e86ffd_549ad677-ee75-4fa1-b8a6-3e821e1dae61-0.orc`
        - During the commit phase, files from the temporary directory are moved to the target table directory.

    - **Overwrite (Replace Existing Partition)**

        - Target table directory: `hdfs://ns/usr/hive/warehouse/example.db/table2/part_col=2024-01-01`
        - Temporary file: `hdfs://ns/tmp/.doris_staging/root/a7eac7505d7a42fdb06cb9ef1ea3e912/par1=a/d678a74d232345e0-b659e2fb58e86ffd_549ad677-ee75-4fa1-b8a6-3e821e1dae61-0.orc`
        - Commit phase steps:

            1. Rename the target partition directory to a temporary partition directory: `hdfs://ns/usr/hive/warehouse/example.db/table2/_temp_d678a74d232345e0-b659e2fb58e86ffd_part_col=2024-01-01`
            2. Rename the temporary partition directory to the target partition directory.
            3. Delete the temporary target partition directory.

Change Log

Doris Version	Feature Support
2.1.6	Support for writing back to Hive tables
3.0.4	Support for Hive tables in JsonSerDe format. Support for transactional tables in Hive4.

Iceberg Catalog

Doris supports accessing Iceberg table data through various metadata services. In addition to reading data, Doris also supports writing to Iceberg tables.

Quick start with Apache Doris and Apache Iceberg.

> **Users can access Iceberg tables using Hive Metastore as metadata through the Hive Catalog. However, it is still recommended to use the Iceberg Catalog directly to avoid some compatibility issues.**

Applicable Scenarios

Scenario	Description
Query Acceleration	Use Doris's distributed computing engine to directly access Iceberg data for faster queries.
Data Integration	Read Iceberg data and write it to internal Doris tables, or perform ZeroETL operations using the Doris computing engine.
Data Write-back	Process data from any Doris-supported data source and write it back to Iceberg table storage.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = '<iceberg_catalog_type>',
    'warehouse' = '<warehouse>', --optional
    {MetaStoreProperties},
    {StorageProperties},
    {IcebergProperties},
    {CommonProperties}
);
```

- `<iceberg_catalog_type>`

  The type of Iceberg Catalog, supporting the following options:

  - `hms`: Uses Hive Metastore as the metadata service.
  - `rest`: Metadata service compatible with the Iceberg Rest Catalog interface.
  - `hadoop`: Directly accesses metadata stored on the file system.
  - `glue`: Uses AWS Glue as the metadata service.

    - `dlf`: Uses Alibaba Cloud DLF as the metadata service.
    - `s3tables`: Uses AWS S3 Tables Catalog to visit S3 Table Bucket.

- `<warehouse>`

  The warehouse path for Iceberg. This parameter must be specified when `<iceberg_catalog_type>` is `hadoop`.

  The `warehouse` path must point to the level above the `Database` path. For example, if your table path is `s3://bucket/path/to/db1/table1`, then the `warehouse` should be `s3://bucket/path/to/`.

- `{MetaStoreProperties}`

  The MetaStoreProperties section is for entering connection and authentication information for the Metastore metadata service. Refer to the section on Supported Metadata Services.

- `{StorageProperties}`

  The StorageProperties section is for entering connection and authentication information related to the storage system. Refer to the section on Supported Storage Systems.

- `{IcebergProperties}`

  The IcebergProperties section is used to fill in parameters specific to Iceberg Catalog.

    - `list-all-tables`

      For Iceberg Catalog that uses Hive Metastore as the metadata service. Default is `true`. By default, the `SHOW TABLES` operation will list all types of tables in the current Database (Hive Metastore may store non-Iceberg type tables). This approach has the best performance.

      If set to `false`, Doris will check the type of each table one by one and only return Iceberg type tables. This mode will have poor performance when there are many tables.

- `{CommonProperties}`

  The CommonProperties section is for entering general properties. See the Catalog Overview for details on common properties.

Supported Iceberg Versions

Doris Version	Iceberg SDK Version
2.1	1.6.1
3.0	1.6.1
3.1	1.9.1
4.0	1.9.1

Supported Iceberg Formats

- Supports Iceberg V1/V2 formats.

- Supports reading Position Delete and Equality Delete.
- Supports reading Deletion Vector (Since 4.1.0).

Supported Metadata Services

- Hive Metastore
- AWS Glue
- Aliyun DLF
- Iceberg Rest Catalog
- FileSystem

> Note: The service types and parameters supported by different Doris versions are slightly different. Please refer to the Examples section.

Iceberg Catalog Feature Support Matrix

Metadata Service	Table Query	View Query	DDL Operations	Data Writeback	System Tables (Supported since 3.1+)
Hive MetaStore	✅	✅	✅	✅	✅
AWS Glue	✅	✅	❌	❌	✅
Aliyun DLF	✅	✅	❌	❌	❌
Iceberg Rest Catalog	✅	✅	Depends on the service provider	Depends on the service provider	Depends on the service provider
FileSystem	✅	✅	✅	✅	✅

Supported Storage Systems

- HDFS
- AWS S3
- Google Cloud Storage
- Azure Blob
- Aliyun OSS
- Tencent COS
- Huawei OBS
- MINIO

> Note: The service types and parameters supported by different Doris versions are slightly different. Please refer to the Examples section.

Supported Data Formats

- Parquet

- ORC

Column Type Mapping

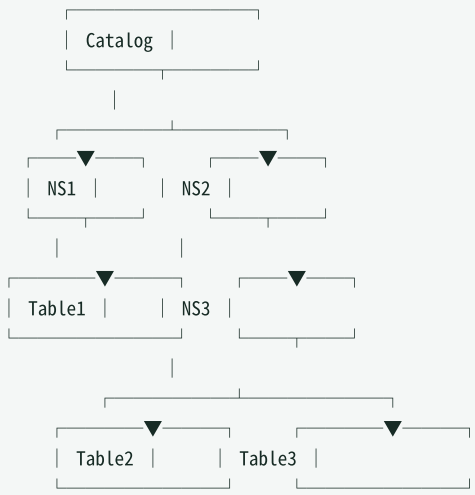
Iceberg Type	Doris Type	Comment
boolean	boolean	
integer	int	
long	bigint	
float	float	
double	double	
decimal(P, S)	decimal(P, S)	
date	date	
timestamp (Timestamp without timezone)	datetime(6)	Mapped to datetime with precision 6
timestamptz (Timestamp with timezone)	datetime(6)	Mapped to datetime with precision 6
fixed(N)	char(N)	
string	string	
binary	string/varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
uuid	string/varbinary	version < 4.0.2 maps to `string` type; version == 4.0.2 maps to `varbinary` type. For version > 4.0.2, it is controlled by the `enable.mapping.varbinary` property in the properties. The default is `false`, which maps to `string`.
struct	struct (supported from version 2.1.3)	
map	map (supported from version 2.1.3)	
list	array	
other	UNSUPPORTED	

> Note:
>
> Doris currently does not support `Timestamp` types with timezone. All `timestamp` and `timestamptz` will be uniformly mapped to `datetime(N)` type. However, during reading and writing, Doris will correctly handle timezones based on the actual source type. For example, after specifying a timezone with `SET time_zone=<tz>`, it will affect the reading and writing results of `timestamptz` columns.

> You can check whether the source type has timezone information in the Extra column of the `DESCRIBE table_name` statement. If it shows `WITH_TIMEZONE`, it indicates that the source type is a timezone-aware type. (Supported since 3.1.0).

Namespace Mapping

Iceberg's metadata hierarchy is Catalog -> Namespace -> Table. Namespace can have multiple levels (Nested Namespace).



Starting from version 3.1.2, for Iceberg Rest Catalog, Doris supports mapping of Nested Namespace.

In the above example, tables will be mapped to Doris metadata according to the following logic:

Catalog	Database	Table
Catalog	NS1	Table1
Catalog	NS2.NS3	Table2
Catalog	NS2.NS3	Table3

Support for Nested Namespace needs to be explicitly enabled. For details, please refer to Iceberg Rest Catalog

Examples

Hive Metastore

3.1+ Version

HDFS

````
Access HMS and HDFS services without Kerberos authentication
```sql
CREATE CATALOG iceberg_hms_on_hdfs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
````
````

````
    'warehouse' = 'hdfs://127.0.0.1:8320/user/iceberg/warehouse/iceberg-hms-hdfs-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8320',
    'hadoop.username' = 'doris'
);
```
Access HMS and HDFS services with Kerberos authentication enabled
```sql
CREATE CATALOG iceberg_hms_on_hdfs_kerberos_hdfs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'hdfs://127.0.0.1:8520/iceberg-hms-hdfs-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.client.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/keytabs/hive-presto-master.keytab',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hive.metastore.authentication.type' = 'kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
    'hadoop.security.authentication' = 'kerberos',
    'hadoop.kerberos.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hadoop.kerberos.keytab' = '/keytabs/hive-presto-master.keytab'
);
```
````

S3

````
Access HMS without Kerberos authentication
```sql
CREATE CATALOG iceberg_hms_on_s3_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 's3://selectdb-qa-datalake-test-hk/refactor-test/iceberg-hms-s3-warehouse',
    's3.region' = 'ap-east-1',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```
Obtaining S3 access credentials using an IAM Assumed Role (3.1.2+)
```sql
CREATE CATALOG iceberg_hms_on_s3_iamrole PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 's3://bucket/warehouse',
    's3.region' = 'us-east-1',
    's3.role_arn' = 'arn:aws:iam::543815668950:role/role'
);
```
````
````

OSS

Access HMS without Kerberos authentication

```sql
CREATE CATALOG iceberg_hms_on_oss_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'oss://bucket/iceberg/iceberg-hms-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9083',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```

Access HMS with Kerberos authentication enabled

```sql
CREATE CATALOG iceberg_hms_on_oss_kerberos_new_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'oss://bucket/iceberg/iceberg-hms-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.client.principal' = 'presto-server/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/keytabs/presto-server.keytab',
    'hive.metastore.authentication.type' = 'kerberos',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```

COS

Access HMS without Kerberos authentication

```sql
CREATE CATALOG `iceberg_hms_on_cos_catalog` PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'cos://bucket/iceberg-hms-cos-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.secret_key' = '<sk>',
    'cos.access_key' = '<ak>'
);
```

Access HMS with Kerberos authentication enabled

```sql
CREATE CATALOG `iceberg_hms_on_cos_kerberos_new_catalog` PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'cos://bucket/iceberg-hms-cos-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
```

```
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hive.metastore.client.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/keytabs/hive-presto-master.keytab',
    'hadoop.security.auth_to_local' = 'RULE:[2:$1@$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:$1@$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:$1@$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.secret_key' = '<sk>',
    'cos.access_key' = '<ak>'
);
```
```

## OBS

````
Access HMS without Kerberos authentication
```sql
CREATE CATALOG test_iceberg_on_hms_obs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'obs://bucket/regression/iceberg',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```
Access HMS with Kerberos authentication enabled
```sql
CREATE CATALOG obs PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'obs://bucket/regression/iceberg1',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hive.metastore.client.principal' = 'hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/keytabs/hive-presto-master.keytab',
    'hive.metastore.authentication.type' = 'kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```
````

## GCS

````
```sql
CREATE CATALOG `iceberg_hms_on_gcs_kerberos_new_catalog` PROPERTIES (
    'type' = 'iceberg',
````

```
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 'gs://bucket/iceberg_warehouse',
    'gs.access_key' = '<ak>',
    'gs.secret_key' = '<sk>',
    'fs.gcs.support'='true'
);
```

Azure Blob

Supported since 3.1.3

```sql
CREATE CATALOG iceberg_fs_on_azure_blob_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'abfs://<container>@<account_name>.dfs.core.windows.net/iceberg-warehouse',
    'azure.account_name' = '<account_name>',
    'azure.account_key' = '<account_key>',
    'azure.endpoint' = 'https://<account_name>.blob.core.windows.net',
    'fs.azure.support'='true'
);
```

MinIO

```sql
CREATE CATALOG test_iceberg_on_hms_minio_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 's3://warehouse/wh',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'fs.minio.support' = 'true',
    'minio.use_path_style' = 'true',
    'minio.endpoint' = 'http://127.0.0.1:19001',
    'minio.access_key' = '<ak>',
    'minio.secret_key' = '<sk>'
);
```

2.1 & 3.0 Version

HDFS

Access HMS and HDFS services without Kerberos authentication

```sql
CREATE CATALOG iceberg_hms_on_hdfs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 'hdfs://127.0.0.1:8320/user/iceberg/warehouse/iceberg-hms-hdfs-warehouse',
    'hadoop.username' = 'doris',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8320'
);
```

```
Access HMS and HDFS services with Kerberos authentication enabled
```sql
CREATE CATALOG iceberg_hms_on_hdfs_kerberos_hdfs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'hdfs://127.0.0.1:8520/iceberg-hms-hdfs-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.kerberos.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'hive.metastore.authentication.type' = 'kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
    'hadoop.security.authentication' = 'kerberos',
    'hadoop.kerberos.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hadoop.kerberos.keytab' = '/keytabs/hive-presto-master.keytab'
);
```
```

S3

```
```sql
CREATE CATALOG iceberg_hms_on_s3_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 's3://selectdb-qa-datalake-test-hk/refactor-test/iceberg-hms-s3-warehouse',
    's3.region' = 'ap-east-1',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```
```

OSS

```
```sql
CREATE CATALOG iceberg_hms_on_oss_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9083',
    'warehouse' = 'oss://bucket/iceberg/iceberg-hms-warehouse',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```
```

COS

```sql
CREATE CATALOG `iceberg_hms_on_cos_catalog` PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'warehouse' = 'cos://bucket/iceberg-hms-cos-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.secret_key' = '<sk>',
    'cos.access_key' = '<ak>'
);
```

OBS

```sql
CREATE CATALOG test_iceberg_on_hms_obs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 'obs://bucket/regression/iceberg',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```

MinIO

```sql
CREATE CATALOG test_iceberg_on_hms_minio_catalog PROPERTIES (
'type' = 'iceberg',
'iceberg.catalog.type' = 'hms',
'warehouse' = 's3://warehouse/iceberg',
'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
's3.endpoint' = 'http://127.0.0.1:19001',
's3.access_key' = '<ak>',
's3.secret_key' = '<sk>'
    );
    ```

AWS Glue

3.1+ Version

S3

AWS Glue and S3 storage services share the same authentication credentials.
```sql
```

```
CREATE CATALOG `iceberg_glue_on_s3_catalog_` PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'glue',
    'warehouse' = 's3://bucket/iceberg-glue-s3-warehouse',
    'glue.region' = 'ap-northeast-1',
    'glue.endpoint' = 'https://glue.ap-northeast-1.amazonaws.com',
    'glue.access_key' = '<ak>',
    'glue.secret_key' = '<sk>'
);
```

When Glue service authentication credentials differ from S3 authentication credentials, you can specify S3 authentication credentials separately using the following method.

```sql
CREATE CATALOG `iceberg_glue_on_s3_catalog_` PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'glue',
    'warehouse' = 's3://selectdb-qa-datalake-test/iceberg-glue-s3-warehouse',
    'glue.region' = 'ap-northeast-1',
    'glue.endpoint' = 'https://glue.ap-northeast-1.amazonaws.com',
    'glue.access_key' = '<ak>',
    'glue.secret_key' = '<sk>',
    's3.endpoint' = 'https://s3.ap-northeast-1.amazonaws.com',
    's3.region' = 'ap-northeast-1',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```

Using IAM Assumed Role to obtain S3 access credentials (Since 3.1.2+)

```sql
CREATE CATALOG `glue_iceberg_iamrole` PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'glue',
    'warehouse' = 's3://bucket/warehouse',
    'glue.region' = 'us-east-1',
    'glue.endpoint' = 'https://glue.us-east-1.amazonaws.com',
    'glue.role_arn' = '<role_arn>'
);
```

2.1 & 3.0 Version

S3

AWS Glue and S3 storage services share the same authentication credentials.

In non-EC2 environments, you need to use [aws configure](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html) to configure Credentials information and generate a credentials file in the ~/.aws directory.

```sql
CREATE CATALOG glue PROPERTIES (
    'type'='iceberg',
    'iceberg.catalog.type' = 'glue',
    'glue.endpoint' = 'https://glue.us-east-1.amazonaws.com',
    'glue.access_key' = '<ak>',
    'glue.secret_key' = '<sk>'
);
```

Aliyun DLF

3.1+ Version

DLF 1.0

````
```sql
CREATE CATALOG iceberg_dlf_catalog_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type'='dlf',
    'warehouse' = 'oss://bucket/iceberg-dlf-oss-warehouse',
    'dlf.uid' = '203225413946383283',
    'dlf.catalog_id' = 'p2_regression_case',
    'dlf.region' = 'cn-beijing',
    'dlf.endpoint' = 'datalake.cn-beijing.aliyuncs.com',
    'dlf.access_key' = '<ak>',
    'dlf.secret_key' = '<sk>'
);
```
````

2.1 & 3.0 Version

DLF 1.0

````
```sql
CREATE CATALOG iceberg_dlf_catalog_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type'='dlf',
    'warehouse' = 'oss://bucket/iceberg-dlf-oss-warehouse',
    'dlf.uid' = '203225413946383283',
    'dlf.catalog.id' = 'catalog_id',
    'dlf.region' = 'cn-beijing',
    'dlf.access_key' = '<ak>',
    'dlf.secret_key' = '<sk>'
);
```
````

Iceberg Rest Catalog

3.1+ Version

S3

No Authentication

````
```sql
CREATE CATALOG iceberg_static_s3 PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 's3://warehouse',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
    's3.access_key' = '<ak>',
````
````

```
    's3.secret_key' = '<sk>',
    's3.region' = 'ap-east-1'
);
```

OAuth2 Authentication

```sql
CREATE CATALOG polaris_oauth2_credential_static_s3 PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 'doris_test',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181/api/catalog',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = 'root:secret123',
    'iceberg.rest.oauth2.server-uri' = 'http://127.0.0.1:8181/api/catalog/v1/oauth/tokens',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:ALL',
    's3.endpoint' = 'https://s3.us-west-2.amazonaws.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.region' = 'us-west-2'
);
```

Vended Credentials

```sql
CREATE CATALOG gravitino_vended_s3 PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 'warehouse',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:9001/iceberg/',
    'iceberg.rest.vended-credentials-enabled' = 'true'
);
```

OSS

No Authentication

```sql
CREATE CATALOG iceberg_static_oss PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 's3://warehouse',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181',
    'oss.endpoint' = 'https://oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>',
    'oss.region' = 'cn-beijing'
);
```

OAuth2 Authentication

```sql
CREATE CATALOG polaris_oauth2_credential_static_oss PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 'doris_test',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181/api/catalog',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = 'root:secret123',
    'iceberg.rest.oauth2.server-uri' = 'http://127.0.0.1:8181/api/catalog/v1/oauth/tokens',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:ALL',
    'oss.endpoint' = 'https://oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>',
    'oss.region' = 'cn-beijing'
);
```

COS

No Authentication

```sql
CREATE CATALOG iceberg_static_cos PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 's3://warehouse',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181',
    'cos.endpoint' = 'https://cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>',
    'cos.region' = 'ap-beijing'
);
```

OAuth2 Authentication

```sql
CREATE CATALOG polaris_oauth2_credential_static_cos PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 'doris_test',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181/api/catalog',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = 'root:secret123',
    'iceberg.rest.oauth2.server-uri' = 'http://127.0.0.1:8181/api/catalog/v1/oauth/tokens',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:ALL',
    'cos.endpoint' = 'https://cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>',
    'cos.region' = 'ap-beijing'
);
```

OBS

No Authentication

```sql
CREATE CATALOG iceberg_static_obs PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 's3://warehouse',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>',
    'obs.region' = 'cn-north-4'
);
```

OAuth2 Authentication

```sql
CREATE CATALOG polaris_oauth2_credential_static_obs PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 'doris_test',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181/api/catalog',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = 'root:secret123',
    'iceberg.rest.oauth2.server-uri' = 'http://127.0.0.1:8181/api/catalog/v1/oauth/tokens',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:ALL',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>',
    'obs.region' = 'cn-north-4'
);
```

GCS

No Authentication

```sql
CREATE CATALOG iceberg_static_gcs PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 'gs://warehouse',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:8181',
    'gs.endpoint' = 'https://storage.googleapis.com',
    'gs.access_key' = '<ak>',
    'gs.secret_key' = '<sk>'
);
```

OAuth2 Authentication

````
        ```sql
        CREATE CATALOG polaris_oauth2_credential_static_gcs PROPERTIES (
            'type' = 'iceberg',
            'warehouse' = 'doris_test',
            'iceberg.catalog.type' = 'rest',
            'iceberg.rest.uri' = 'http://127.0.0.1:8181/api/catalog',
            'iceberg.rest.security.type' = 'oauth2',
            'iceberg.rest.oauth2.credential' = 'root:secret123',
            'iceberg.rest.oauth2.server-uri' = 'http://127.0.0.1:8181/api/catalog/v1/oauth/tokens',
            'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:ALL',
            'gs.endpoint' = 'https://storage.googleapis.com',
            'gs.access_key' = '<ak>',
            'gs.secret_key' = '<sk>'

        );
        ```
````

HDFS

````
    ```sql
    CREATE CATALOG iceberg_static_hdfs PROPERTIES (
        'type' = 'iceberg',
        'warehouse' = 'hdfs://127.0.0.1:8020/user/iceberg/warehouse',
        'iceberg.catalog.type' = 'rest',
        'iceberg.rest.uri' = 'http://127.0.0.1:8181',
        'fs.defaultFS' = 'hdfs://127.0.0.1:8020'
    );
    ```
````

2.1 & 3.0 Version

S3

````
    ```sql
    CREATE CATALOG iceberg_static_s3 PROPERTIES (
        'type' = 'iceberg',
        'warehouse' = 's3://warehouse',
        'iceberg.catalog.type' = 'rest',
        'uri' = 'http://127.0.0.1:8181',
        's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
        's3.access_key' = '<ak>',
        's3.secret_key' = '<sk>',
        's3.region' = 'ap-east-1'
    );
    ```
````

OSS

````
    ```sql
    CREATE CATALOG iceberg_static_oss PROPERTIES (
        'type' = 'iceberg',
````
````

```
    'warehouse' = 's3://warehouse',
    'iceberg.catalog.type' = 'rest',
    'uri' = 'http://127.0.0.1:8181',
    's3.endpoint' = 'https://oss-cn-beijing.aliyuncs.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.region' = 'cn-beijing'
);
```

COS

```sql
CREATE CATALOG iceberg_static_cos PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 's3://warehouse',
    'iceberg.catalog.type' = 'rest',
    'uri' = 'http://127.0.0.1:8181',
    's3.endpoint' = 'https://cos.ap-beijing.myqcloud.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.region' = 'ap-beijing'
);
```

OBS

```sql
CREATE CATALOG iceberg_static_obs PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 's3://warehouse',
    'iceberg.catalog.type' = 'rest',
    'uri' = 'http://127.0.0.1:8181',
    's3.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.region' = 'cn-north-4'
);
```

HDFS

```sql
CREATE CATALOG iceberg_static_hdfs PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = 'hdfs://127.0.0.1:8020/user/iceberg/warehouse',
    'iceberg.catalog.type' = 'rest',
    'uri' = 'http://127.0.0.1:8181',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8020'
);
```

FileSystem

3.1+ Version

HDFS

````
Access HDFS service without Kerberos authentication
```sql
CREATE CATALOG iceberg_fs_on_hdfs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'hdfs://127.0.0.1:8320/user/iceberg/warehouse/iceberg-fs-hdfs-warehouse',
    'hadoop.username' = 'doris',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8320'
);
```
Access HDFS service with Kerberos authentication enabled
```sql
CREATE CATALOG iceberg_fs_on_hdfs_kerberos_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'hdfs://127.0.0.1:8520/iceberg-fs-hdfs-warehouse',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
    'hadoop.security.authentication' = 'kerberos',
    'hadoop.kerberos.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hadoop.kerberos.keytab' = '/keytabs/hive-presto-master.keytab'
);

```
````

S3

````
```sql
CREATE CATALOG iceberg_fs_on_glue_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 's3://selectdb-qa-datalake-test-hk/refactor-test/iceberg-fs-s3-warehouse',
    's3.region' = 'ap-east-1',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```
Obtaining S3 access credentials using an IAM Assumed Role (3.1.2+)
```sql
CREATE CATALOG iceberg_fs_on_s3_iamrole PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 's3://bucket/warehouse',
    's3.region' = 'us-east-1',
    's3.role_arn' = 'arn:aws:iam::543815668950:role/role'
);
```
````

COS

````
```sql
CREATE CATALOG iceberg_fs_on_cos_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'cos://doris-build-1308700295/iceberg-fs-cos-warehouse',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);

```
````

OSS

````
```sql
CREATE CATALOG iceberg_fs_on_oss_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'oss://doris-regression-bj/refactor-test/iceberg-fs-oss-warehouse',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```
````

OBS

````
```sql
CREATE CATALOG test_fs_on_obs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'obs://bucket/regression/iceberg',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```
````

GCS

````
```sql
CREATE CATALOG iceberg_fs_on_gcs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'gs://bucket/iceberg_warehouse',
    'fs.gcs.support'='true',
    'gs.access_key' = '<ak>',
    'gs.secret_key' = '<sk>'
````
````

````
);
```
````

MinIO

````
```sql
CREATE CATALOG test_iceberg_fs_on_minio PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type'='hadoop',
    'warehouse' = 's3://warehouse/wh',
    'fs.minio.support'='true',
    'minio.endpoint' = 'http://127.0.0.1:19001',
    'minio.access_key' = 'admin',
    'minio.secret_key' = 'password',
    'minio.use_path_style' = 'true'
);
```
````

2.1 & 3.0 Version

HDFS

````
Access HDFS service without Kerberos authentication
```sql
CREATE CATALOG iceberg_fs_on_hdfs_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'hdfs://127.0.0.1:8320/user/iceberg/warehouse/iceberg-fs-hdfs-warehouse',
    'hadoop.username' = 'doris',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8320'
);
```
Access HDFS service with Kerberos authentication enabled
```sql
CREATE CATALOG iceberg_fs_on_hdfs_kerberos_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'hdfs://127.0.0.1:8520/iceberg-fs-hdfs-warehouse',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
    'hadoop.security.authentication' = 'kerberos',
    'hadoop.kerberos.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hadoop.kerberos.keytab' = '/keytabs/hive-presto-master.keytab'
);
```
````

S3

````
```sql
CREATE CATALOG iceberg_fs_on_s3_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 's3://selectdb-qa-datalake-test-hk/refactor-test/iceberg-fs-s3-warehouse',
    's3.region' = 'ap-east-1',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
````

```
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```

COS

```sql
CREATE CATALOG iceberg_fs_on_cos_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'cos://doris-build-1308700295/iceberg-fs-cos-warehouse',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```

OSS

```sql
CREATE CATALOG iceberg_fs_on_oss_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hadoop',
    'warehouse' = 'oss://bucket/refactor-test/iceberg-fs-oss-warehouse',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```

GCS

```sql
CREATE CATALOG iceberg_fs_on_gcs_catalog PROPERTIES (
    'type'='iceberg',
    'iceberg.catalog.type'='hadoop',
    'warehouse' = 's3://bucket/iceberg_warehouse',
    'gs.endpoint'='storage.googleapis.com',
    'gs.access_key' = '<ak>',
    'gs.secret_key' = '<sk>'
);
```

MinIO

```sql
CREATE CATALOG test_iceberg_fs_on_minio PROPERTIES (
    'type' = 'iceberg',
```

```
    'iceberg.catalog.type'='hadoop',
    'warehouse' = 's3://warehouse/wh',
    's3.region' = 'ap-east-1',
    's3.endpoint' = 'http://minio:9000',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```

AWS S3 Tables

3.1+ Version

S3 Tables Rest Catalog

Please refer to the [S3 Tables Integration](../best-practices/doris-aws-s3tables.md) documentation.

```sql
CREATE CATALOG aws_s3_tables PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'warehouse' = 'arn:aws:s3tables:us-east-1:<account_id>:bucket/<s3_table_bucket_name>',
    'iceberg.rest.uri' = 'https://s3tables.us-east-1.amazonaws.com/iceberg',
    'iceberg.rest.sigv4-enabled' = 'true',
    'iceberg.rest.signing-name' = 's3tables',
    'iceberg.rest.signing-region' = 'us-east-1',
    'iceberg.rest.access-key-id' = '<ak>',
    'iceberg.rest.secret-access-key' = '<sk>'
);
```

AWS Glue Rest Catalog

Please refer to the [S3 Tables Integration](../best-practices/doris-aws-s3tables.md) documentation.

```sql
CREATE CATALOG glue_s3 PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'https://glue.ap-east-1.amazonaws.com/iceberg',
    'warehouse' = '<acount_id>:s3tablescatalog/<s3_table_bucket_name>',
    'iceberg.rest.sigv4-enabled' = 'true',
    'iceberg.rest.signing-name' = 'glue',
    'iceberg.rest.access-key-id' = '<ak>',
    'iceberg.rest.secret-access-key' = '<sk>',
    'iceberg.rest.signing-region' = 'ap-east-1'
);
```

FileSystem Catalog

Please refer to the [S3 Tables Integration](../best-practices/doris-aws-s3tables.md) documentation.

```sql
CREATE CATALOG test_s3tables_write_insert PROPERTIES (
    'type' = 'iceberg',
```

```
            'iceberg.catalog.type' = 's3tables',
            'warehouse' = 'arn:aws:s3tables:ap-east-1:12345:bucket/doris-s3-table-bucket',
            's3.region' = 'ap-east-1',
            's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
            's3.access_key' = '<ak>',
            's3.secret_key' = '<sk>'
        );
        ```
```

#### 3.0.6+ Version

### FileSystem Catalog

```
    Please refer to the [S3 Tables Integration](../best-practices/doris-aws-s3tables.md) documentation.
    ```sql
    CREATE CATALOG test_s3tables_catalog PROPERTIES (
        'type' = 'iceberg',
        'iceberg.catalog.type' = 's3tables',
        'warehouse' = 'arn:aws:s3tables:ap-east-1:9527:bucket/doris-s3-table-bucket',
        's3.region' = 'ap-east-1',
        's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
        's3.access_key' = '<ak>',
        's3.secret_key' = '<sk>'
    );
    ```
```

## Query Operations

### Basic Query

```
-- 1. switch to catalog, use database and query
SWITCH iceberg;
USE iceberg_db;
SELECT * FROM iceberg_tbl LIMIT 10;

-- 2. use iceberg database directly
USE iceberg.iceberg_db;
SELECT * FROM iceberg_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM iceberg.iceberg_db.iceberg_tbl LIMIT 10;
```

Time Travel

You can read a specific snapshot of an Iceberg table.

By default, read requests will access the latest snapshot version.

You can query a specific snapshot of an Iceberg table using the `iceberg_meta()` table function:

```
SELECT * FROM iceberg_meta(
    'table' = 'iceberg_ctl.iceberg_db.iceberg_tbl',
    'query_type' = 'snapshots'
)\G
```
```

```
*************************** 1. row ***************************
 committed_at: 2024-11-28 11:07:29
  snapshot_id: 8903826400153112036
    parent_id: -1
    operation: append
manifest_list: oss://path/to/metadata/snap-8903826400153112036-1-3835e66d-9a18-4cb0-b9b0-9ec80527ad8d.avro
      summary: {"added-data-files":"2","added-records":"3","added-files-size":"2742","changed-partition-count":"2","total-
records":"3","total-files-size":"2742","total-data-files":"2","total-delete-files":"0","total-position-deletes":"0","total-equality-
deletes":"0"}
*************************** 2. row ***************************
 committed_at: 2024-11-28 11:10:11
  snapshot_id: 6099853805930794326
    parent_id: 8903826400153112036
    operation: append
manifest_list: oss://path/to/metadata/snap-6099853805930794326-1-dd46a1bd-219b-4fb0-bb46-ac441d8b3105.avro
      summary: {"added-data-files":"1","added-records":"1","added-files-size":"1367","changed-partition-count":"1","total-
records":"4","total-files-size":"4109","total-data-files":"3","total-delete-files":"0","total-position-deletes":"0","total-equality-
deletes":"0"}
```

You can use the `FOR TIME AS OF` and `FOR VERSION AS OF` clauses to read historical data based on snapshot ID or the time the snapshot was created. Here are some examples:

```
-- Read data as of a specific timestamp
SELECT * FROM iceberg_table FOR TIME AS OF '2023-01-01 00:00:00';

-- Read data as of a specific snapshot ID
SELECT * FROM iceberg_table FOR VERSION AS OF 123456789;
```

Branch and Tag

> Since 3.1.0
>
> For creating, dropping and managing branch and tag, please refer to Managing Branch & Tag

Reading specific branches and tags of Iceberg tables is supported.

Multiple syntax forms are supported to be compatible with systems such as Spark/Trino.

```
-- BRANCH
SELECT * FROM iceberg_tbl@branch(branch1);
SELECT * FROM iceberg_tbl@branch("name" = "branch1");
SELECT * FROM iceberg_tbl FOR VERSION AS OF 'branch1';

-- TAG
SELECT * FROM iceberg_tbl@tag(tag1);
SELECT * FROM iceberg_tbl@tag("name" = "tag1");
SELECT * FROM iceberg_tbl FOR VERSION AS OF 'tag1';
```

For the `FOR VERSION AS OF` syntax, Doris will automatically determine whether the parameter is a timestamp or a Branch/Tag name.

View

> Since 3.1.0

Supports querying Iceberg views. View queries work the same way as regular table queries. Please note the following:

- Only `hms` type Iceberg Catalog is supported.
- The view definition SQL must be compatible with Doris SQL dialect, otherwise parsing errors will occur. (Dialect conversion functionality will be provided in future versions).

System Tables

> Since 3.1.0

Doris supports querying Iceberg system tables to retrieve metadata information about tables. You can use system tables to view snapshot history, manifest files, data files, partitions, and other metadata.

To access metadata of an Iceberg table, append a `$` symbol followed by the system table name to the table name:

```
SELECT * FROM iceberg_table$system_table_name;
```

For example, to view the table's history, you can execute:

```
SELECT * FROM iceberg_table$history;
```

> Currently, the `all_manifests` and `position_deletes` system tables are not yet supported and are planned to be supported in future versions.

entries

Shows all manifest entries for the current snapshot of the table:

`all_entries` and `entries` are similar, with the difference that `all_entries` contains entries from all snapshots, while `entries` only contains entries from the current snapshot.

```
SELECT * FROM iceberg_table$entries;
```

Result:

```
+--------+--------------------+----------------+---------------------+-------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
| status | snapshot_id         | sequence_number | file_sequence_number | data_file
| readable_metrics
|
+--------+---------------------+-----------------+----------------------+------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
|      2 | 4890031351138056789 |               1 |                    1 | {"content":0,
"file_path":"s3://.../iceberg_table/data/id=1/00000-16-79ef2fd7-9997-47eb-a91a-9f7af8201315-0-00001.parquet",
"file_format":"PARQUET", "spec_id":0, "partition":{"id":1}, "record_count":1, "file_size_in_bytes":625, "column_sizes":{1:36, 2:41},
"value_counts":{1:1, 2:1}, "null_value_counts":{1:0, 2:0}, "nan_value_counts":{}, "lower_bounds":{1:"   ", 2:"Alice"},
"upper_bounds":{1:"   ", 2:"Alice"}, "key_metadata":null, "split_offsets":[4], "equality_ids":null, "sort_order_id":0,
"first_row_id":null, "referenced_data_file":null, "content_offset":null, "content_size_in_bytes":null} | {"id":{"column_size":36,
"value_count":1, "null_value_count":0, "nan_value_count":null, "lower_bound":1, "upper_bound":1}, "name":{"column_size":41,
"value_count":1, "null_value_count":0, "nan_value_count":null, "lower_bound":"Alice", "upper_bound":"Alice"}} |
|      0 | 1851184769713369003 |               1 |                    1 | {"content":0,
"file_path":"s3://.../iceberg_table/data/id=2/00000-16-79ef2fd7-9997-47eb-a91a-9f7af8201315-0-00002.parquet",
"file_format":"PARQUET", "spec_id":0, "partition":{"id":2}, "record_count":1, "file_size_in_bytes":611, "column_sizes":{1:36, 2:39},
"value_counts":{1:1, 2:1}, "null_value_counts":{1:0, 2:0}, "nan_value_counts":{}, "lower_bounds":{1:"   ", 2:"Bob"}, "upper_bounds":
{1:"   ", 2:"Bob"}, "key_metadata":null, "split_offsets":[4], "equality_ids":null, "sort_order_id":0, "first_row_id":null,
"referenced_data_file":null, "content_offset":null, "content_size_in_bytes":null}     | {"id":{"column_size":36, "value_count":1,
"null_value_count":0, "nan_value_count":null, "lower_bound":2, "upper_bound":2}, "name":{"column_size":39, "value_count":1,
"null_value_count":0, "nan_value_count":null, "lower_bound":"Bob", "upper_bound":"Bob"}}     |
+--------+---------------------+-----------------+----------------------+------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------+
```

files

Shows file list for the current snapshot of the table:

```
SELECT * FROM iceberg_table$files;
```

Result:

```
+---------+-------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------+-------------+---------+-----------+--------------+--------------------+--------------+--
------------+------------------+------------------+----------------------+----------------------+--------------+--------------+-
-------------+---------------+--------------+----------------------+---------------+----------------------+----------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| content | file_path                                                                                                 | file_format | spec_id
| partition | record_count | file_size_in_bytes | column_sizes | value_counts | null_value_counts | nan_value_counts | lower_bounds
| upper_bounds          | key_metadata | split_offsets | equality_ids | sort_order_id | first_row_id | referenced_data_file |
content_offset | content_size_in_bytes | readable_metrics
```

```
|
+---------+-----------------------
```

The result format of these tables is similar to the `files` system table, but each focuses specifically on data files or delete files. System tables with the `all_` prefix contain files from all snapshots, not just files from the current snapshot.

Note: When specific types of files do not exist in the table (for example, querying `delete_files` when there are no delete files in the table), the query result may be empty.

history

Shows all history of the table:

```
SELECT * FROM iceberg_table$history;
```

Result:

```
+----------------------------+---------------------+---------------------+---------------------+
| made_current_at            | snapshot_id         | parent_id           | is_current_ancestor |
+----------------------------+---------------------+---------------------+---------------------+
| 2025-06-12 22:29:16.357000 | 1851184769713369003 |                NULL |                   1 |
| 2025-06-12 22:29:39.922000 | 4890031351138056789 | 1851184769713369003 |                   1 |
+----------------------------+---------------------+---------------------+---------------------+
```

manifests

Shows manifest file info of the table:

```
SELECT * FROM iceberg_table$manifests;
```

Result:

```
+---------+-----------------------------
```

```
SELECT * FROM iceberg_table$metadata_log_entries;
```

Result:

```
+----------------------------+-------------------------------------------------------------------------------------------------------------------------------------------------------+---------------------+-----------------+------------------------+
| timestamp                  | file                                                                                                                                                  |
latest_snapshot_id  | latest_schema_id | latest_sequence_number |
+----------------------------+-------------------------------------------------------------------------------------------------------------------------------------------------------+---------------------+-----------------+------------------------+
| 2025-06-12 22:29:06.948000 | s3://.../iceberg_table/metadata/00000-e373aa16-15f1-4e69-ae7d-5ed64199cf9a.metadata.json |
NULL |             NULL |                   NULL |
| 2025-06-12 22:29:16.357000 | s3://.../iceberg_table/metadata/00001-bbc8e244-e41c-4958-92f4-63b8c3ee1196.metadata.json |
1851184769713369003 |                0 |                     1 |
| 2025-06-12 22:29:39.922000 | s3://.../iceberg_table/metadata/00002-7dc00d6a-6269-4200-9d28-5f8c1c6b9f99.metadata.json |
4890031351138056789 |                0 |                     2 |
+----------------------------+-------------------------------------------------------------------------------------------------------------------------------------------------------+---------------------+-----------------+------------------------+
```

partitions

Shows partitions of the table:

```
SELECT * FROM iceberg_table$partitions;
```

Result:

```
+-----------+---------+--------------+------------+-----------------------------+-----------------------------+-----------------
----------+-----------------------------+---------------------------+---------------------------+-------------------------+
| partition | spec_id | record_count | file_count | total_data_file_size_in_bytes | position_delete_record_count |
position_delete_file_count | equality_delete_record_count | equality_delete_file_count | last_updated_at            |
last_updated_snapshot_id |
+-----------+---------+--------------+------------+-----------------------------+-----------------------------+-----------------
----------+-----------------------------+---------------------------+---------------------------+-------------------------+
| {"id":8}  |       0 |            1 |          1 |                         625 |                           0 |
0 |                           0 |                           0 | 2025-06-12 22:29:16.357000 |      1851184769713369003 |
| {"id":6}  |       0 |            1 |          1 |                         625 |                           0 |
0 |                           0 |                           0 | 2025-06-12 22:29:16.357000 |      1851184769713369003 |
| {"id":10} |       0 |            1 |          1 |                         618 |                           0 |
0 |                           0 |                           0 | 2025-06-12 22:29:16.357000 |      1851184769713369003 |
| {"id":4}  |       0 |            1 |          1 |                         618 |                           0 |
0 |                           0 |                           0 | 2025-06-12 22:29:16.357000 |      1851184769713369003 |
| {"id":2}  |       0 |            1 |          1 |                         611 |                           0 |
0 |                           0 |                           0 | 2025-06-12 22:29:16.357000 |      1851184769713369003 |
+-----------+---------+--------------+------------+-----------------------------+-----------------------------+-----------------
----------+-----------------------------+---------------------------+---------------------------+-------------------------+
```

Note:

1. For non-partitioned tables, the `partitions` table will not contain the `partition` and `spec_id` fields.
2. The `partitions` table shows partitions that contain data files or delete files in the current snapshot. However, delete files are not applied, so in some cases, a partition may still be displayed even if all data rows in the partition have been marked as deleted by delete files.

refs

Shows all known snapshot references (branches and tags) for the table:

```
SELECT * FROM iceberg_table$refs;
```

Result:

```
+------+--------+---------------------+-------------------------+----------------------+------------------------+
| name | type   | snapshot_id         | max_reference_age_in_ms | min_snapshots_to_keep | max_snapshot_age_in_ms |
+------+--------+---------------------+-------------------------+----------------------+------------------------+
| main | BRANCH | 4890031351138056789 |                    NULL |                 NULL |                   NULL |
+------+--------+---------------------+-------------------------+----------------------+------------------------+
```

snapshots

Shows all snapshots of the table:

```
SELECT * FROM iceberg_table$snapshots;
```

Result:

```
+----------------------------+---------------------+---------------------+-----------+-----------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+---------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------+
| committed_at               | snapshot_id         | parent_id           | operation | manifest_list
| summary
|
+----------------------------+---------------------+---------------------+-----------+-----------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+---------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------+
| 2025-06-12 22:29:16.357000 | 1851184769713369003 |                NULL | append    | s3://.../iceberg_table/metadata/snap-
1851184769713369003-1-82059f57-821a-4983-b083-002cc2cde313.avro | {"spark.app.id":"application_1738810850199_0472", "added-data-
files":"10", "added-records":"10", "added-files-size":"6200", "changed-partition-count":"10", "total-records":"10", "total-files-
size":"6200", "total-data-files":"10", "total-delete-files":"0", "total-position-deletes":"0", "total-equality-deletes":"0"}  |
| 2025-06-12 22:29:39.922000 | 4890031351138056789 | 1851184769713369003 | overwrite | s3://.../iceberg_table/metadata/snap-
4890031351138056789-1-3194eb8b-5ea4-4cbe-95ba-073229458e7b.avro | {"spark.app.id":"application_1738810850199_0472", "deleted-data-
files":"5", "deleted-records":"5", "removed-files-size":"3103", "changed-partition-count":"5", "total-records":"5", "total-files-
size":"3097", "total-data-files":"5", "total-delete-files":"0", "total-position-deletes":"0", "total-equality-deletes":"0"} |
+----------------------------+---------------------+---------------------+-----------+-----------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------+---------
----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------+
```

Write Operations

INSERT INTO

The INSERT operation appends data to the target table.

For example:

```
INSERT INTO iceberg_tbl VALUES (val1, val2, val3, val4);
INSERT INTO iceberg.iceberg_db.iceberg_tbl SELECT col1, col2 FROM internal.db1.tbl1;

INSERT INTO iceberg_tbl(col1, col2) VALUES (val1, val2);
INSERT INTO iceberg_tbl(col1, col2, partition_col1, partition_col2) VALUES (1, 2, 'beijing', '2023-12-12');
```

Since version 3.1.0, support for writing data to specified branches:

```
INSERT INTO iceberg_tbl@branch(b1) values (val1, val2, val3, val4);
INSERT INTO iceberg_tbl@branch(b1) (col3, col4) values (val3, val4);
```

INSERT OVERWRITE

The INSERT OVERWRITE operation completely replaces the existing data in the table with new data.

```
INSERT OVERWRITE TABLE iceberg_tbl VALUES (val1, val2, val3, val4);
INSERT OVERWRITE TABLE iceberg.iceberg_db.iceberg_tbl(col1, col2) SELECT col1, col2 FROM internal.db1.tbl1;
```

Since version 3.1.0, support for writing data to specified branches:

```
INSERT OVERWRITE TABLE iceberg_tbl@branch(b1) values (val1, val2, val3, val4);
INSERT OVERWRITE TABLE iceberg_tbl@branch(b1) (col3, col4) values (val3, val4);
```

CTAS

You can create an Iceberg table and write data using the `CTAS` (Create Table As Select) statement:

```
CREATE TABLE iceberg_ctas AS SELECT * FROM other_table;
```

CTAS supports specifying file formats, partitioning, and other properties:

```
CREATE TABLE iceberg_ctas
PARTITION BY LIST (pt1, pt2) ()
AS SELECT col1, pt1, pt2 FROM part_ctas_src WHERE col1 > 0;

CREATE TABLE iceberg.iceberg_db.iceberg_ctas (col1, col2, pt1)
PARTITION BY LIST (pt1) ()
PROPERTIES (
    'write-format'='parquet',
    'compression-codec'='zstd'
)
AS SELECT col1, pt1 AS col2, pt2 AS pt1 FROM test_ctas.part_ctas_src WHERE col1 > 0;
```

INSERT INTO BRANCH

> Since 3.1.0

```
INSERT INTO iceberg_table@branch(b1) SELECT * FROM other_table;
INSERT OVERWRITE TABLE iceberg_table@branch(b1) SELECT * FROM other_table;
```

Related Parameters

- BE (Backend)

Parameter Name	Default Value	Description
iceberg_sink_max_file_size	1GB	Maximum data file size. When the written data exceeds this size, the current file is closed and a new file is created to continue writing.
table_sink_partition_write_max_partition_nums_per_writer	128	Maximum number of partitions each instance can write to on a BE node.

Parameter Name	Default Value	Description
table_sink_non_partition_write_scaling_data_processed_threshold	25MB	Data threshold for starting scaling-write in non-partitioned tables. A new writer (instance) is used for every additional table_sink_non_partition_write_scaling_data_processed_threshold of data. This mechanism adjusts the number of writers based on data volume to enhance throughput while conserving resources and minimizing file numbers for smaller data volumes.

Parameter Name	Default Value	Description
table_sink_partition_write_min_data_processed_rebalance_threshold	25MB	Minimum data volume threshold to trigger rebalancing for partitioned tables. Rebalancing starts if current accumulated data volume - data volume since last rebalancing >= table_sink_partition_write_min_data_processed_rebalance_threshold. Lowering this threshold can improve balance if file size differences are significant, but may increase rebalancing costs and impact performance.

Parameter Name	Default Value	Description
table_sink_partition_write_min_partition_data_processed_rebalance_threshold		Minimum partition data volume threshold to trigger rebalancing. Rebalancing starts if current partition data volume >= threshold * number of tasks already allocated to the partition. Lowering this threshold can improve balance if file size differences are significant, but may increase rebalancing costs and impact performance.

Database and Table Management

Creating and Dropping Databases

You can switch to the desired catalog using the `SWITCH` statement and execute the `CREATE DATABASE` command:

```
SWITCH iceberg;
CREATE DATABASE [IF NOT EXISTS] iceberg_db;
```

You can also create a database using a fully qualified name or specify a location (currently, only HMS-type catalogs support specifying a location), such as:

```
CREATE DATABASE [IF NOT EXISTS] iceberg.iceberg_db;

CREATE DATABASE [IF NOT EXISTS] iceberg.iceberg_db
PROPERTIES ('location'='hdfs://172.21.16.47:4007/path/to/db/');
```

You can view the database's location information using the `SHOW CREATE DATABASE` command:

```
mysql> SHOW CREATE DATABASE iceberg_db;
+-------------+-----------------------------------------------------------------------------------------------+
| Database    | Create Database                                                                               |
+-------------+-----------------------------------------------------------------------------------------------+
| iceberg_db  | CREATE DATABASE iceberg_db LOCATION 'hdfs://172.21.16.47:4007/usr/hive/warehouse/iceberg_db.db' |
+-------------+-----------------------------------------------------------------------------------------------+
```

To drop a database:

```
DROP DATABASE [IF EXISTS] iceberg.iceberg_db;
```

> **For an Iceberg Database, you must first drop all tables under the database before you can drop the database itself; otherwise, an error will occur.**

Creating and Dropping Tables

- **Creating Tables**

  Doris supports creating both partitioned and non-partitioned tables in Iceberg.

  For example:

```
-- Create unpartitioned iceberg table
CREATE TABLE unpartitioned_table (
  `col1` BOOLEAN COMMENT 'col1',
  `col2` INT COMMENT 'col2',
  `col3` BIGINT COMMENT 'col3',
  `col4` FLOAT COMMENT 'col4',
  `col5` DOUBLE COMMENT 'col5',
  `col6` DECIMAL(9,4) COMMENT 'col6',
  `col7` STRING COMMENT 'col7',
  `col8` DATE COMMENT 'col8',
  `col9` DATETIME COMMENT 'col9'
)
PROPERTIES (
  'write-format'='parquet'
);

-- Create partitioned iceberg table
-- The partition columns must be in table's column definition list
CREATE TABLE partition_table (
  `ts` DATETIME COMMENT 'ts',
  `col1` BOOLEAN COMMENT 'col1',
  `col2` INT COMMENT 'col2',
  `col3` BIGINT COMMENT 'col3',
  `col4` FLOAT COMMENT 'col4',
  `col5` DOUBLE COMMENT 'col5',
```

```
  `col6` DECIMAL(9,4) COMMENT 'col6',
  `col7` STRING COMMENT 'col7',
  `col8` DATE COMMENT 'col8',
  `col9` DATETIME COMMENT 'col9',
  `pt1` STRING COMMENT 'pt1',
  `pt2` STRING COMMENT 'pt2'
)
PARTITION BY LIST (day(ts), pt1, pt2) ()
PROPERTIES (
  'write-format'='orc',
  'compression-codec'='zlib'
);
```

After creation, you can use the `SHOW CREATE TABLE` command to view the Iceberg table creation statement. For details about partition functions, see the Partitioning section.

- **Dropping Tables**

  You can drop an Iceberg table using the `DROP TABLE` statement. Dropping a table will also remove its data, including partition data.

  For example:

  ```
  DROP TABLE [IF EXISTS] iceberg_tbl;
  ```

- **Column Type Mapping**

  Refer to the Column Type Mapping section.

- **Partitioning**

  Partition types in Iceberg correspond to List partitions in Doris. Therefore, when creating an Iceberg partitioned table in Doris, you should use the List partitioning syntax, but you don't need to explicitly enumerate each partition. Doris will automatically create the corresponding Iceberg partitions based on the data values during data insertion.

  - Supports creating single-column or multi-column partitioned tables.
  - Supports partition transformation functions to enable Iceberg implicit partitioning and partition evolution. For specific Iceberg partition transformation functions, see Iceberg partition transforms:
    - `year(ts)` or `years(ts)`
    - `month(ts)` or `months(ts)`
    - `day(ts)` or `days(ts)` or `date(ts)`
    - `hour(ts)` or `hours(ts)` or `date_hour(ts)`
    - `bucket(N, col)`
    - `truncate(L, col)`

- **File Formats**
  - Parquet (default)

Note that for the Iceberg table created by Doris, the Datetime corresponds to the `timestamp_ntz` type.

In versions after 3.1.0, when the Datetime type is written to the Parquet file, the physical type used is INT64 instead of INT96.

And if the Iceberg table is created by other systems, although the `timestamp` and `timestamp_ntz` types are both mapped to the Doris Datetime type. However, when writing, it will determine whether the time zone needs to be processed based on the actual type.

    - ORC

- **Compression Formats**
    - Parquet: snappy, zstd (default), plain (no compression).
    - ORC: snappy, zlib (default), zstd, plain (no compression).
- **Storage Medium**
    - HDFS
    - Object storage

Schema Change

Start from 3.1.0, Doris supports schema changes for Iceberg tables, which can be modified using the `ALTER TABLE` statement.

Supported schema change operations include:

- **Rename Column**

  Use the `RENAME COLUMN` clause to rename columns. Renaming columns within nested types is not supported.

```
ALTER TABLE iceberg_table RENAME COLUMN old_col_name TO new_col_name;
```

- **Add a Column**

  Use `ADD COLUMN` to add a new column. Adding new columns to nested types is not supported.

  When adding a new column, you can specify nullable attributes, default values, comments, and column position.

```
ALTER TABLE iceberg_table ADD COLUMN col_name col_type [NULL|NOT NULL, [DEFAULT default_value, [COMMENT 'comment',
[FIRST|AFTER col_name]]]];
```

  Example:

```
ALTER TABLE iceberg_table ADD COLUMN new_col STRING NOT NULL DEFAULT 'default_value' COMMENT 'This is a new col' AFTER
old_col;
```

- **Add Columns**

  You can also use `ADD COLUMN` to add multiple columns. The new columns will be added to the end of the table. Column positioning is not supported for multiple columns. Adding new columns to nested types is not supported.

  The syntax for each column is the same as adding a single column.

  ```
  ALTER TABLE iceberg_table ADD COLUMN (col_name1 col_type1 [NULL|NOT NULL, [DEFAULT default_value, [COMMENT 'comment']]],
  col_name2 col_type2 [NULL|NOT NULL, [DEFAULT default_value, [COMMENT 'comment']]] ...);
  ```

- **Drop Column**

  Use `DROP COLUMN` to drop columns. Dropping columns within nested types is not supported.

  ```
  ALTER TABLE iceberg_table DROP COLUMN col_name;
  ```

- **Modify Column**

  Use the `MODIFY COLUMN` statement to modify column attributes, including type, nullable, default value, comment, and column position.

  Note: When modifying column attributes, all attributes that are not being modified should also be explicitly specified with their original values.

  ```
  ALTER TABLE iceberg_table MODIFY COLUMN col_name col_type [NULL|NOT NULL, [DEFAULT default_value, [COMMENT 'comment',
  [FIRST|AFTER col_name]]]];
  ```

  Example:

  ```
  CREATE TABLE iceberg_table (
      id INT,
      name STRING
  );
  -- Modify the id column type to BIGINT, set as NOT NULL, default value to 0, and add comment
  ALTER TABLE iceberg_table MODIFY COLUMN id BIGINT NOT NULL DEFAULT 0 COMMENT 'This is a modified id column';
  ```

- **Reorder Columns**

  Use `ORDER BY` to reorder columns by specifying the new column order.

  ```
  ALTER TABLE iceberg_table ORDER BY (col_name1, col_name2, ...);
  ```

Partition Evolution

Starting from version 4.0.2, Doris supports Partition Evolution for Iceberg tables through `ALTER` statements.

Supported partition transforms include:

Transform	Syntax	Example
bucket	`bucket(N, column)`	`bucket(16, id)`
truncate	`truncate(N, column)`	`truncate(10, name)`
year	`year(column)`	`year(ts)`
month	`month(column)`	`month(ts)`
day	`day(column)`	`day(ts)`
hour	`hour(column)`	`hour(ts)`
identity	`column`	`category`

Supported operations include:

- **Add partition key**

```
-- use optional AS keyword to specify a custom name for the partition field
ALTER TABLE table_name ADD PARTITION KEY partition_transform [AS key_name];

-- example
ALTER TABLE prod.db.sample ADD PARTITION KEY bucket(16, id);
ALTER TABLE prod.db.sample ADD PARTITION KEY truncate(4, data);
ALTER TABLE prod.db.sample ADD PARTITION KEY year(ts);
-- use optional AS keyword to specify a custom name for the partition field
ALTER TABLE prod.db.sample ADD PARTITION KEY bucket(16, id) AS shard;
```

- **Drop partition key**

```
ALTER TABLE table_name DROP PARTITION KEY partition_transform | key_name;

-- example
ALTER TABLE prod.db.sample DROP PARTITION KEY catalog;
ALTER TABLE prod.db.sample DROP PARTITION KEY bucket(16, id);
ALTER TABLE prod.db.sample DROP PARTITION KEY truncate(4, data);
ALTER TABLE prod.db.sample DROP PARTITION KEY year(ts);
ALTER TABLE prod.db.sample DROP PARTITION KEY shard;
```

- **Replace partition key**

```
-- use optional AS keyword to specify a custom name for the partition field
ALTER TABLE table_name REPLACE PARTITION KEY key_name WITH partition_transform [AS key_name];

-- example
ALTER TABLE prod.db.sample REPLACE PARTITION KEY ts_day WITH day(ts);
-- use optional AS keyword to specify a custom name for the new partition field
ALTER TABLE prod.db.sample REPLACE PARTITION KEY ts_day WITH day(ts) AS day_of_ts;
```

Managing Branch & Tag

> Since 3.1.0

- **Create Branch**

  Syntax:

```
ALTER TABLE [catalog.][database.]table_name
CREATE [OR REPLACE] BRANCH [IF NOT EXISTS] <branch_name>
[AS OF VERSION <snapshot_id>]
[RETAIN <num> { DAYS | HOURS | MINUTES }]
[WITH SNAPSHOT RETENTION { snapshotKeep | timeKeep }]

snapshotKeep:
  <num> SNAPSHOTS [<num> { DAYS | HOURS | MINUTES }]

timeKeep:
  <num> { DAYS | HOURS | MINUTES }
```

  Examples:

```
-- Create branch "b1".
ALTER TABLE tbl CREATE BRANCH b1;
ALTER TABLE tb1 CREATE BRANCH IF NOT EXISTS b1;
-- Create or replace branch "b1".
ALTER TABLE tb1 CREATE OR REPLACE BRANCH b1;
-- Create or replace branch "b1" based on snapshot "123456".
ALTER TABLE tb1 CREATE OR REPLACE BRANCH b1 AS OF VERSION 123456;
-- Create or replace branch "b1" based on snapshot "123456", branch retained for 1 day.
ALTER TABLE tb1 CREATE OR REPLACE BRANCH b1 AS OF VERSION 123456 RETAIN 1 DAYS;
-- Create branch "b1" based on snapshot "123456", branch retained for 30 days. Keep the latest 3 snapshots in the branch.
ALTER TABLE tb1 CREATE BRANCH b1 AS OF VERSION 123456 RETAIN 30 DAYS WITH SNAPSHOT RETENTION 3 SNAPSHOTS;
-- Create branch "b1" based on snapshot "123456", branch retained for 30 days. Snapshots in the branch are retained for at
most 2 days.
ALTER TABLE tb1 CREATE BRANCH b1 AS OF VERSION 123456 RETAIN 30 DAYS WITH SNAPSHOT RETENTION 2 DAYS;
-- Create branch "b1" based on snapshot "123456", branch retained for 30 days. Keep the latest 3 snapshots in the branch,
and snapshots in the branch are retained for at most 2 days.
ALTER TABLE tb1 CREATE BRANCH b1 AS OF VERSION 123456 RETAIN 30 DAYS WITH SNAPSHOT RETENTION 3 SNAPSHOTS 2 DAYS;
```

- **Drop Branch**

  Syntax:

```
ALTER TABLE [catalog.][database.]table_name
DROP BRANCH [IF EXISTS] <branch_name>;
```

  Example:

```
ALTER TABLE tbl DROP BRANCH b1;
```

- **Create Tag**

  Syntax:

```
ALTER TABLE [catalog.][database.]table_name
CREATE [OR REPLACE] TAG [IF NOT EXISTS] <tag_name>
[AS OF VERSION <snapshot_id>]
[RETAIN <num> { DAYS | HOURS | MINUTES }]
```

Examples:

```
-- Create tag "t1".
ALTER TABLE tbl CREATE TAG t1;
ALTER TABLE tb1 CREATE TAG IF NOT EXISTS t1;
-- Create or replace tag "t1".
ALTER TABLE tb1 CREATE OR REPLACE TAG t1;
-- Create or replace tag "t1" based on snapshot "123456".
ALTER TABLE tb1 CREATE OR REPLACE TAG b1 AS OF VERSION 123456;
-- Create or replace tag "b1" based on snapshot "123456", tag retained for 1 day.
ALTER TABLE tb1 CREATE OR REPLACE TAG b1 AS OF VERSION 123456 RETAIN 1 DAYS;
```

- **Drop Tag**

  Syntax:

  ```
  ALTER TABLE [catalog.][database.]table_name
  DROP TAG [IF EXISTS] <tag_name>;
  ```

  Example:

  ```
  ALTER TABLE tbl DROP TAG t1;
  ```

Iceberg Table Actions

> This feature is supported since 4.0.2
>
> This is an experimental feature.

Doris supports executing specific actions on Iceberg tables through the `ALTER TABLE EXECUTE` syntax. This syntax provides a unified interface for executing various actions such as table optimization and snapshot management.

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE <action_name> ("key1" = "value1", "key2" = "value2", ...)
[WHERE <condition>]
```

- `action_name`: The name of the action to execute
- `("key1" = "value1", ...)`: Action-related parameters provided as key-value pairs. Parameter names must be wrapped in double quotes
- `[WHERE <condition>]`: Optional WHERE condition to specify the partition or data range for the operation (only supported by some operations)

rewrite_data_files

The `rewrite_data_files` operation rewrites data files in an Iceberg table to implement small file compaction, optimizing query performance and storage efficiency. This is a synchronized operation.

Syntax:

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE rewrite_data_files ("key1" = "value1", "key2" = "value2", ...) [WHERE <condition>]
```

Parameters:

Parameter Name	Type	Default Value	Description
target-file-size-bytes	Long	536870912 (512MB)	Target file size in bytes for output files
min-file-size-bytes	Long	target-file-size-bytes * 0.75	Minimum file size in bytes. Files smaller than this will be rewritten. If not specified, defaults to 75% of target file size
max-file-size-bytes	Long	target-file-size-bytes * 1.8	Maximum file size in bytes. Files larger than this will be split. If not specified, defaults to 180% of target file size
min-input-files	Integer	5	Minimum number of input files to trigger rewrite. Rewrite only executes when the number of files to rewrite reaches this threshold
rewrite-all	Boolean	false	Whether to rewrite all files regardless of file size
max-file-group-size-bytes	Long	107374182400 (100GB)	Maximum size in bytes for a file group. Used to limit the amount of data processed by a single rewrite task
delete-file-threshold	Integer	Integer.MAX_VALUE	Minimum number of delete files required to trigger rewrite
delete-ratio-threshold	Double	0.3	Minimum ratio of delete records to total records required to trigger rewrite (delete records / total records). Range: 0.0 - 1.0
output-spec-id	Long	2	Partition specification ID for output files

Return Value:

Executing `rewrite_data_files` returns a result set with the following 4 columns:

Column Name	Type	Description
rewritten_data_files_count	INT	Number of data files that were rewritten
added_data_files_count	INT	Number of new data files that were written
rewritten_bytes_count	INT	Number of bytes that were rewritten
removed_delete_files_count	BIGINT	Number of delete files that were removed

Examples:

```
-- Rewrite data files with default parameters
ALTER TABLE iceberg_db.iceberg_table EXECUTE rewrite_data_files();

-- Specify target file size and minimum input files
ALTER TABLE iceberg_db.iceberg_table
EXECUTE rewrite_data_files (
  "target-file-size-bytes" = "134217728",
  "min-input-files" = "10"
);

-- Use WHERE condition to rewrite specific partition data
ALTER TABLE iceberg_db.iceberg_table
EXECUTE rewrite_data_files (
  "target-file-size-bytes" = "268435456"
)
WHERE date_col = '2024-01-01';

-- Rewrite all files
ALTER TABLE iceberg_db.iceberg_table
EXECUTE rewrite_data_files (
  "rewrite-all" = "true",
  "target-file-size-bytes" = "536870912"
);
```

Notes:

1. The `rewrite_data_files` operation reads and rewrites data files, which will generate additional I/O and computational overhead. Please allocate cluster resources reasonably.
2. Before execution, you can use the SQL in the View Data File Distribution section to evaluate whether a rewrite operation is needed
3. The WHERE condition can be used to limit the partition or data range for rewriting. This condition will filter out files that do not contain data matching the WHERE condition, thereby reducing the number of files and amount of data to rewrite
4. Before execution, you can use the SQL in the Rewrite File Selection Logic section to calculate which files will be rewritten

cherrypick_snapshot

The `cherrypick_snapshot` operation merges changes from a specified snapshot into the current table state, creating a new snapshot without modifying or deleting the original snapshot.

Syntax:

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE cherrypick_snapshot ("snapshot_id" = "<snapshot_id>")
```

Parameters:

Parameter Name	Type	Required	Description
`snapshot_id`	Long	Yes	The snapshot ID to merge

Return Value:

Executing `cherrypick_snapshot` returns a result set with the following 2 columns:

Column Name	Type	Description
source_snapshot_id	BIGINT	The snapshot ID whose changes were merged
current_snapshot_id	BIGINT	The snapshot ID of the new snapshot created and set as current after the merge operation

Example:

```
-- Merge changes from snapshot 123456789 into current table state
ALTER TABLE iceberg_db.iceberg_table
EXECUTE cherrypick_snapshot ("snapshot_id" = "123456789");
```

Notes:

1. This operation does not support WHERE conditions
2. If the specified snapshot does not exist, the operation will fail
3. The merge operation creates a new snapshot and does not delete the original snapshot

fast_forward

The `fast_forward` operation fast-forwards the current snapshot of one branch to the latest snapshot of another branch.

Syntax:

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE fast_forward ("branch" = "<branch_name>", "to" = "<target_branch>")
```

Parameters:

Parameter Name	Type	Required	Description
branch	String	Yes	Name of the branch to fast-forward
to	String	Yes	Target branch name. The `branch` will be fast-forwarded to the latest snapshot of this branch

Return Value:

Executing `fast_forward` returns a result set with the following 3 columns:

Column Name	Type	Description
branch_updated	STRING	Name of the branch that was fast-forwarded
previous_ref	BIGINT	Snapshot ID the branch was pointing to before the fast-forward operation (can be NULL)
updated_ref	BIGINT	Snapshot ID the branch is pointing to after the fast-forward operation

Example:

```
-- Fast-forward feature branch to the latest snapshot of main branch
ALTER TABLE iceberg_db.iceberg_table
EXECUTE fast_forward ("branch" = "feature", "to" = "main");
```

Notes:

1. This operation does not support WHERE conditions
2. If the specified branch does not exist, the operation will fail
3. Only branches can be fast-forwarded, not tags

rollback_to_snapshot

The `rollback_to_snapshot` operation rolls back an Iceberg table to a specified snapshot.

Syntax:

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE rollback_to_snapshot ("snapshot_id" = "<snapshot_id>")
```

Parameters:

Parameter Name	Type	Required	Description
`snapshot_id`	Long	Yes	The snapshot ID to rollback to

Return Value:

Executing `rollback_to_snapshot` returns a result set with the following 2 columns:

Column Name	Type	Description
`previous_snapshot_id`	BIGINT	The snapshot ID that was current before the rollback operation
`current_snapshot_id`	BIGINT	The snapshot ID that is set as current after rolling back to the specified snapshot

Example:

```
-- Rollback to snapshot 987654321
ALTER TABLE iceberg_db.iceberg_table
EXECUTE rollback_to_snapshot ("snapshot_id" = "987654321");
```

Notes:

1. This operation does not support WHERE conditions
2. If the specified snapshot does not exist, the operation will fail
3. If the current snapshot is already the target snapshot, the operation returns directly without creating a new snapshot

rollback_to_timestamp

The `rollback_to_timestamp` operation rolls back an Iceberg table to the snapshot at a specified timestamp.

Syntax:

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE rollback_to_timestamp ("timestamp" = "<timestamp>")
```

Parameters:

Parameter Name	Type	Required	Description
`timestamp`	String	Yes	The timestamp to rollback to. Supports two formats: 1. ISO datetime format: `yyyy-MM-dd HH:mm:ss.SSS` (e.g., `2024-01-01 10:30:00.000`) 2. Milliseconds timestamp (e.g., `1704067200000`)

Return Value:

Executing `rollback_to_timestamp` returns a result set with the following 2 columns:

Column Name	Type	Description
`previous_snapshot_id`	BIGINT	The snapshot ID that was current before the rollback operation
`current_snapshot_id`	BIGINT	The snapshot ID that was current at the specified timestamp, now set as current

Examples:

```
-- Rollback to specified timestamp (ISO format)
ALTER TABLE iceberg_db.iceberg_table
EXECUTE rollback_to_timestamp ("timestamp" = "2024-01-01 10:30:00.000");

-- Rollback to specified timestamp (timestamp format)
ALTER TABLE iceberg_db.iceberg_table
EXECUTE rollback_to_timestamp ("timestamp" = "1704067200000");
```

Notes:

1. This operation does not support WHERE conditions
2. The `timestamp` parameter supports ISO datetime format (`yyyy-MM-dd HH:mm:ss.SSS`) or milliseconds timestamp format
3. If there is no snapshot corresponding to the specified timestamp, the operation will fail

set_current_snapshot

The `set_current_snapshot` operation sets the current snapshot of an Iceberg table to a specified snapshot ID or reference (branch or tag).

Syntax:

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE set_current_snapshot ("snapshot_id" = "<snapshot_id>" | "ref" = "<ref_name>")
```

Parameters:

Parameter Name	Type	Required	Description
snapshot_id	Long	Either one	The snapshot ID to set as current
ref	String	Either one	The reference name (branch or tag) to set as current

Return Value:

Executing set_current_snapshot returns a result set with the following 2 columns:

Column Name	Type	Description
previous_snapshot_id	BIGINT	The snapshot ID that was current before setting the new current snapshot
current_snapshot_id	BIGINT	The snapshot ID that is now set as current (from snapshot_id parameter or resolved from ref parameter)

Examples:

```
-- Set current snapshot by snapshot ID
ALTER TABLE iceberg_db.iceberg_table
EXECUTE set_current_snapshot ("snapshot_id" = "123456789");

-- Set current snapshot by branch name
ALTER TABLE iceberg_db.iceberg_table
EXECUTE set_current_snapshot ("ref" = "feature_branch");

-- Set current snapshot by tag name
ALTER TABLE iceberg_db.iceberg_table
EXECUTE set_current_snapshot ("ref" = "v1.0");
```

Notes:

1. This operation does not support WHERE conditions
2. The snapshot_id and ref parameters are mutually exclusive; only one can be specified
3. If the specified snapshot ID or reference does not exist, the operation will fail
4. If the current snapshot is already the target snapshot, the operation returns directly without creating a new snapshot

Iceberg Table Optimization

View Data File Distribution

You can use the following SQL to analyze the data distribution and delete file count of Iceberg tables, helping determine whether optimization like rewrite_data_files is needed.

- Display data file size distribution. This can help identify if there are too many

```
SELECT
  CASE
    WHEN file_size_in_bytes BETWEEN 0 AND 8 * 1024 * 1024 THEN '0-8M'
    WHEN file_size_in_bytes BETWEEN 8 * 1024 * 1024 + 1 AND 32 * 1024 * 1024 THEN '8-32M'
    WHEN file_size_in_bytes BETWEEN 2 * 1024 * 1024 + 1 AND 128 * 1024 * 1024 THEN '32-128M'
    WHEN file_size_in_bytes BETWEEN 128 * 1024 * 1024 + 1 AND 512 * 1024 * 1024 THEN '128-512M'
    WHEN file_size_in_bytes > 512 * 1024 * 1024 THEN '> 512M'
    ELSE 'Unknown'
  END AS SizeRange,
  COUNT(*) AS FileNum
FROM store_sales$data_files
GROUP BY
  SizeRange;

+-----------+---------+
| SizeRange | FileNum |
+-----------+---------+
| 0-8M      |       8 |
| 8-32M     |       6 |
+-----------+---------+
```

- Display the number of data files and delete files

```
SELECT
  CASE
    WHEN content = 0 THEN 'DataFile'
    WHEN content = 1 THEN 'PositionDeleteFile'
    WHEN content = 2 THEN 'EqualityDeleteFile'
    ELSE 'Unknown'
  END AS ContentType,
  COUNT(*) AS FileNum,
  SUM(file_size_in_bytes) AS SizeInBytes,
  SUM(record_count) AS Records
FROM
  iceberg_table$files
GROUP BY
  ContentType;

+--------------------+---------+-------------+---------+
| ContentType        | FileNum | SizeInBytes | Records |
+--------------------+---------+-------------+---------+
| EqualityDeleteFile |    2787 |     1432518 |   27870 |
| DataFile           |    2787 |     4062416 |   38760 |
| PositionDeleteFile |      11 |       36608 |   10890 |
+--------------------+---------+-------------+---------+
```

Dangling Delete

In some cases, after executing the `rewrite_data_files` action, references to certain Position Deletes may not have been removed from the Snapshot metadata (Dangling Delete). If you directly use the row number information from the metadata in this situation, the result may be incorrect.

Therefore, by default, for `COUNT(*)` queries, if a Position Delete file is found, COUNT pushdown optimization is not enabled; instead, the file is read directly to obtain the actual `COUNT(*)` result. However, this method is time-consuming.

If the user can ensure there is no Dangling Delete issue, this check can be skipped using the Doris session variable `ignore_iceberg_dangling_delete`. This variable defaults to `false`. When set to `true`, the system will directly return the `COUNT(*)` result based on the row count information in the metadata, improving query efficiency.

This feature is supported from versions 3.1.4 and 4.0.3.

Appendix

`rewrite_data_files` File Selection Strategy

You can check this section to understand the logic for selecting files to be rewritten. You can also use the following SQL queries to analyze which files will be rewritten based on system table information. This helps you debug and understand rewrite behavior before executing actual rewrite operations.

The rewrite operation uses a two-stage filtering and grouping strategy:

1. Stage 1: File-level filtering

   First, files are filtered based on file-level conditions. Only files that meet at least one of the following conditions will be considered for rewriting:

   1. **File size out of range**: File is smaller than `min-file-size-bytes` or larger than `max-file-size-bytes`
   2. **Too many delete files**: Number of associated delete files >= `delete-file-threshold`
   3. **High delete ratio**: Delete record ratio >= `delete-ratio-threshold`

   Note about delete file conditions: Conditions related to delete files (`delete-file-threshold` and `delete-ratio-threshold`) affect file rewrite selection, but cannot be accurately calculated through system table SQL queries. This is because current system table information is insufficient to accurately determine which data files are affected by delete files. Delete files may only affect specific data files (especially Position Delete files), and the association between delete files and data files requires more detailed metadata that is not available in current system tables. Actual rewrite operations will evaluate these conditions based on more detailed metadata not exposed in system tables.

2. Stage 2: BinPack grouping and group-level filtering

   After file-level filtering, files are grouped by partition, then within each partition, files are grouped using the **BinPack algorithm** based on file size and `max-file-group-size-bytes`. The BinPack algorithm intelligently packs files into groups to optimize rewrite operations.

   After grouping, file groups are filtered based on group-level conditions. File groups will be rewritten when they meet at least one of the following conditions:

   1. **Sufficient input files**: Number of files in group > 1 and file count >= `min-input-files`
   2. **Sufficient content**: Number of files in group > 1 and total size > `target-file-size-bytes`
   3. **Too much content**: Group total size > `max-file-group-size-bytes`
   4. **Contains files with delete file issues**: At least one file in the group has too many delete files or high delete ratio.

   Note about BinPack grouping: The BinPack algorithm used to group files within each partition is a complex optimization algorithm that cannot be accurately simulated through system table SQL queries.

This algorithm considers file size and `max-file-group-size-bytes` to intelligently pack files into groups, which may result in different grouping results than simple partition-based grouping. Therefore, the SQL queries provided below can only identify files that meet file-level conditions, but cannot accurately calculate the final file groups that will be rewritten.

Here are some example query strategies:

- Query 1: Identify files meeting file-level conditions

  This query identifies files that meet file-level conditions (file size) and will be considered for rewriting:

```
-- Set parameters (adjust according to your rewrite parameters)
SET @min_file_size_bytes = 16 * 1024 * 1024;   -- 16 MB (default: target-file-size-bytes * 0.75)
SET @max_file_size_bytes = 768 * 1024 * 1024;  -- 768 MB (default: target-file-size-bytes * 1.8)

SELECT
    file_path,
    `partition`,
    file_size_in_bytes / 1024.0 / 1024.0 AS file_size_mb,
    record_count,
    CASE
        WHEN file_size_in_bytes < @min_file_size_bytes THEN 'Too small'
        WHEN file_size_in_bytes > @max_file_size_bytes THEN 'Too large'
    END AS size_issue
FROM iceberg_table$data_files
WHERE file_size_in_bytes < @min_file_size_bytes
OR file_size_in_bytes > @max_file_size_bytes
ORDER BY `partition`, file_size_in_bytes DESC;
```

  Sample output:

```
+-------------------------------------------------------------------------------------------+--------------------------+-----
--------+--------------+------------+
| file_path                                                                                 | partition                |
file_size_mb | record_count | size_issue |
+-------------------------------------------------------------------------------------------+--------------------------+-----
--------+--------------+------------+
| s3a://bucket/path/to/table/data/date_col=2024-01-01/00000-0-00000-00000-0.parquet   | {"date_col":"2024-01-01"} |
0.00215  |            5 | Too small  |
| s3a://bucket/path/to/table/data/date_col=2024-01-01/00001-0-00001-00001-0.parquet     | {"date_col":"2024-01-01"} |
0.00198  |            5 | Too small  |
| s3a://bucket/path/to/table/data/date_col=2024-01-02/00002-0-00002-00002-0.parquet     | {"date_col":"2024-01-02"} |
0.00231  |            5 | Too small  |
| s3a://bucket/path/to/table/data/date_col=2024-01-02/00003-0-00003-00003-0.parquet     | {"date_col":"2024-01-02"} |
850.23456  |      1250000 | Too large  |
| s3a://bucket/path/to/table/data/date_col=2024-01-03/00004-0-00004-00004-0.parquet     | {"date_col":"2024-01-03"} |
0.00245  |            5 | Too small  |
+-------------------------------------------------------------------------------------------+--------------------------+-----
--------+--------------+------------+
5 rows in set
```

  Note: This query can only identify files that meet file-level conditions (file size). The actual file groups to be rewritten depend on the BinPack grouping algorithm, which cannot be accurately simulated through SQL queries. Files identified by this query may be grouped differently by the BinPack algorithm, and some groups may be filtered out based on group-level conditions.

- Query 2: Overall statistics summary

This query provides overall statistics about files that meet file-level conditions:

```
-- Set parameters (adjust according to your rewrite parameters)
SET @min_file_size_bytes = 16 * 1024 * 1024;
SET @max_file_size_bytes = 768 * 1024 * 1024;

WITH file_analysis AS (
    SELECT
        `partition`,
        file_path,
        file_size_in_bytes,
        record_count,
        (file_size_in_bytes < @min_file_size_bytes OR file_size_in_bytes > @max_file_size_bytes) AS
meets_file_level_conditions
    FROM iceberg_table$data_files
)
SELECT
    'Total files' AS metric,
    COUNT(*) AS value
FROM file_analysis
UNION ALL
SELECT
    'Files meeting file-level conditions',
    SUM(CASE WHEN meets_file_level_conditions THEN 1 ELSE 0 END)
FROM file_analysis
UNION ALL
SELECT
    'Total size (GB)',
    ROUND(SUM(file_size_in_bytes) / 1024.0 / 1024.0 / 1024.0, 2)
FROM file_analysis
UNION ALL
SELECT
    'Size meeting file-level conditions (GB)',
    ROUND(SUM(CASE WHEN meets_file_level_conditions THEN file_size_in_bytes ELSE 0 END) / 1024.0 / 1024.0 / 1024.0, 2)
FROM file_analysis
UNION ALL
SELECT
    'Percentage meeting file-level conditions (%)',
    ROUND(SUM(CASE WHEN meets_file_level_conditions THEN 1 ELSE 0 END) * 100.0 / COUNT(*), 2)
FROM file_analysis;
```

Sample output:

```
+-----------------------------------------------+--------+
| metric                                        | value  |
+-----------------------------------------------+--------+
| Total files                                   |  15.00 |
| Total size (GB)                               |   2.45 |
| Files meeting file-level conditions           |  12.00 |
| Size meeting file-level conditions (GB)       |   1.85 |
| Percentage meeting file-level conditions (%) |  80.00 |
+-----------------------------------------------+--------+
5 rows in set
```

Notes:

- Replace `iceberg_table` in the example with the actual table name
- Adjust parameter values (`@min_file_size_bytes`, `@max_file_size_bytes`) according to your actual rewrite parameters

    - The default values shown above correspond to default rewrite parameters (min-file-size-bytes = target-file-size-bytes * 0.75, max-file-size-bytes = target-file-size-bytes * 1.8)

- View snapshot and branch relationships

```
SELECT
refs_data.snapshot_id,
snapshots.committed_at,
snapshots.operation,
ARRAY_SORT(refs_data.refs)
FROM (
SELECT
    snapshot_id,
    ARRAY_AGG(CONCAT(type, ':', name)) AS refs
FROM
    iceberg_table$refs
GROUP BY
    snapshot_id
) AS refs_data
JOIN (
SELECT
    snapshot_id,
    committed_at,
    operation
FROM
    iceberg_table$snapshots
) AS snapshots
ON refs_data.snapshot_id = snapshots.snapshot_id
ORDER BY
snapshots.committed_at;

+---------------------+---------------------------+-----------+-----------------------------------+
| snapshot_id         | committed_at              | operation | ARRAY_SORT(refs_data.refs)        |
+---------------------+---------------------------+-----------+-----------------------------------+
| 8272911997874079853 | 2025-07-10 15:27:07.177000 | append    | ["BRANCH:b1", "TAG:t1"]           |
| 1325777059626757917 | 2025-07-10 15:27:07.530000 | append    | ["BRANCH:b2", "TAG:t2"]           |
|   76492482642020578 | 2025-07-10 15:27:07.865000 | append    | ["BRANCH:b3", "TAG:t3"]           |
| 1788715857849070138 | 2025-07-12 04:15:19.626000 | append    | ["BRANCH:main", "TAG:t4", "TAG:t5"] |
+---------------------+---------------------------+-----------+-----------------------------------+
```

Hudi Catalog

Hudi Catalog reuses the Hive Catalog. By connecting to the Hive Metastore, or a metadata service compatible with the Hive Metastore, Doris can automatically obtain Hudi's database and table information and perform data queries.

Quick start with Apache Doris and Apache Hudi.

Applicable Scenarios

Scenario	Description
Query Acceleration	Use Doris's distributed computing engine to directly access Hudi data for query acceleration.
Data Integration	Read Hudi data and write it into Doris internal tables, or perform ZeroETL operations using the Doris computing engine.
Data Write-back	Not supported.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name PROPERTIES (
    'type' = 'hms', -- required
    'hive.metastore.uris' = '<metastore_thrift_url>', -- required
    {MetaStoreProperties},
    {StorageProperties},
    {HudiProperties},
    {CommonProperties}
);
```

- `[MetaStoreProperties]`

  The MetaStoreProperties section is used to fill in the connection and authentication information for the Metastore metadata service. See the section Supported Metadata Services for details.

- `[StorageProperties]`

  The StorageProperties section is used to fill in the connection and authentication information related to the storage system. See the section Supported Storage Systems for details.

- `[CommonProperties]`

  The CommonProperties section is used to fill in common properties. Please refer to the Data Catalog Overview section on [Common Properties].

- `{HudiProperties}`

Parameter Name	Former Name	Description	Default Value
`hudi.use_hive_sync_partition`	`use_hive_sync_partition`	Whether to use the partition information already synchronized by Hive Metastore. If true, partition information will be obtained directly from Hive Metastore. Otherwise, it will be obtained from the metadata file of the file system. Obtaining information from Hive Metastore is more efficient, but users need to ensure that the latest metadata has been synchronized to Hive Metastore.	false

Supported Hudi Versions

The current dependent Hudi version is 0.15. It is recommended to access Hudi data version 0.14 and above.

Supported Query Types

Table Type	Supported Query Types
Copy On Write	Snapshot Query, Time Travel, Incremental Read
Merge On Read	Snapshot Queries, Read Optimized Queries, Time Travel, Incremental Read

Supported Metadata Services

- Hive Metastore

Supported Storage Systems

- HDFS
- AWS S3
- Google Cloud Storage
- Alibaba Cloud OSS
- Tencent Cloud COS
- Huawei Cloud OBS
- MINIO

Supported Data Formats

- Parquet
- ORC

Column Type Mapping

Hudi Type	Doris Type	Comment
boolean	boolean	
int	int	
long	bigint	

Hudi Type	Doris Type	Comment
float	float	
double	double	
decimal(P, S)	decimal(P, S)	
bytes	string	
string	string	
date	date	
timestamp	datetime(N)	Automatically maps to datetime(3) or datetime(6) based on precision
array	array	
map	map	
struct	struct	
other	UNSUPPORTED	

Examples

The creation of a Hudi Catalog is similar to a Hive Catalog. For more examples, please refer to Hive Catalog.

```
CREATE CATALOG hudi_hms PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://172.21.0.1:7004',
    'hadoop.username' = 'hive',
    'dfs.nameservices'='your-nameservice',
    'dfs.ha.namenodes.your-nameservice'='nn1,nn2',
    'dfs.namenode.rpc-address.your-nameservice.nn1'='172.21.0.2:4007',
    'dfs.namenode.rpc-address.your-nameservice.nn2'='172.21.0.3:4007',
    'dfs.client.failover.proxy.provider.your-
nameservice'='org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider'
);
```

Query Operations

Basic Query

Once the Catalog is configured, you can query the tables within the Catalog using the following method:

```
-- 1. switch to catalog, use database and query
SWITCH hudi_ctl;
USE hudi_db;
SELECT * FROM hudi_tbl LIMIT 10;

-- 2. use hudi database directly
USE hudi_ctl.hudi_db;
SELECT * FROM hudi_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM hudi_ctl.hudi_db.hudi_tbl LIMIT 10;
```

Time Travel

Every write operation to a Hudi table creates a new snapshot. Doris supports reading a specified snapshot of a Hudi table. By default, query requests only read the latest snapshot.

You can query the timeline of a specified Hudi table using the `hudi_meta()` table function:

This table function is supported since 3.1.0.

```
SELECT * FROM hudi_meta(
    'table' = 'hudi_ctl.hudi_db.hudi_tbl',
    'query_type' = 'timeline'
);

+-------------------+--------+-------------------------+-----------+----------------------+
| timestamp         | action | file_name               | state     | state_transition_time |
+-------------------+--------+-------------------------+-----------+----------------------+
| 20241202171214902 | commit | 20241202171214902.commit | COMPLETED | 20241202171215756     |
| 20241202171217258 | commit | 20241202171217258.commit | COMPLETED | 20241202171218127     |
| 20241202171219557 | commit | 20241202171219557.commit | COMPLETED | 20241202171220308     |
| 20241202171221769 | commit | 20241202171221769.commit | COMPLETED | 20241202171222541     |
| 20241202171224269 | commit | 20241202171224269.commit | COMPLETED | 20241202171224995     |
| 20241202171226401 | commit | 20241202171226401.commit | COMPLETED | 20241202171227155     |
| 20241202171228827 | commit | 20241202171228827.commit | COMPLETED | 20241202171229570     |
| 20241202171230907 | commit | 20241202171230907.commit | COMPLETED | 20241202171231686     |
| 20241202171233356 | commit | 20241202171233356.commit | COMPLETED | 20241202171234288     |
| 20241202171235940 | commit | 20241202171235940.commit | COMPLETED | 20241202171236757     |
+-------------------+--------+-------------------------+-----------+----------------------+
```

You can use the `FOR TIME AS OF` statement to read historical versions of data based on the snapshot's timestamp. The time format is consistent with the Hudi documentation. Here are some examples:

```
SELECT * FROM hudi_tbl FOR TIME AS OF "2022-10-07 17:20:37";
SELECT * FROM hudi_tbl FOR TIME AS OF "20221007172037";
SELECT * FROM hudi_tbl FOR TIME AS OF "2022-10-07";
```

Note that Hudi tables do not support the `FOR VERSION AS OF` statement. Attempting to use this syntax with a Hudi table will result in an error.

Incremental Query

Incremental Read allows querying data changes within a specified time range, returning the final state of the data at the end of that period.

Doris provides the `@incr` syntax to support Incremental Read:

```
SELECT * from hudi_table@incr('beginTime'='xxx', ['endTime'='xxx'], ['hoodie.read.timeline.holes.resolution.policy'='FAIL'], ...);
```

- `beginTime`

  Required. The time format must be consistent with the Hudi official hudi_table_changes, supporting "earliest".

- `endTime`

Optional, defaults to the latest commitTime.

You can add more options in the `@incr` function, compatible with Spark Read Options.

By using `desc` to view the execution plan, you can see that Doris converts `@incr` into predicates pushed down to `VHUDI_SCAN_NODE`:

```
|   0:VHUDI_SCAN_NODE(113)                                                                                                  |
|      table: lineitem_mor                                                                                                  |
|      predicates: (_hoodie_commit_time[#0] > '20240311151019723'), (_hoodie_commit_time[#0] <= '20240311151606605') |
|      inputSplitNum=1, totalFileSize=13099711, scanRanges=1
```

FAQ

1. Query blocked when using Java SDK to read incremental data through JNI

   Please add `-Djol.skipHotspotSAAttach=true` to `JAVA_OPTS_FOR_JDK_17` or `JAVA_OPTS` in `be.conf`.

Appendix

Change Log

Doris Version	Feature Support
2.1.8/3.0.4	Hudi dependency upgraded to 0.15. Added Hadoop Hudi JNI Scanner.

Paimon Catalog

Doris currently supports accessing Paimon table metadata through various metadata services and querying Paimon data.

At present, only read operations on Paimon tables are supported. Write operations to Paimon tables will be supported in the future.

Quick start with Apache Doris and Apache Paimon.

Applicable Scenarios

Scenario	Description
Query Acceleration	Use Doris's distributed computing engine to directly access Paimon data for query acceleration.
Data Integration	Read Paimon data and write it into Doris internal tables, or perform ZeroETL operations using the Doris computing engine.
Data Write-back	Not supported yet.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = '<paimon_catalog_type>',
    'warehouse' = '<paimon_warehouse>'
    {MetaStoreProperties},
    {StorageProperties},
    {PaimonProperties},
    {CommonProperties}
);
```

- `<paimon_catalog_type>`

  The type of Paimon Catalog, supporting the following:

  - `filesystem`: Default. Directly accesses metadata stored on the file system.
  - `hms`: Uses Hive Metastore as the metadata service.
  - `dlf`: Uses Alibaba Cloud DLF as the metadata service.

- `<paimon_warehouse>`

  The warehouse path for Paimon. This parameter must be specified when `<paimon_catalog_type>` is `filesystem`.

The `warehouse` path must point to the level above the `Database` path. For example, if your table path is: `s3://bucket/path/to/db1/table1`, then `warehouse` should be: `s3://bucket/path/to/`.

- `{MetaStoreProperties}`

  The MetaStoreProperties section is used to fill in connection and authentication information for the Metastore metadata service. Refer to the section on Supported Metadata Services for details.

- `{StorageProperties}`

  The StorageProperties section is used to fill in connection and authentication information related to the storage system. Refer to the section on Supported Storage Systems for details.

- `{PaimonProperties}`

  The PaimonProperties section is used to fill Paimon-related properties.

  When Doris accesses Paimon metadata, it partially uses the Paimon Java SDK, so some custom parameters may need to be passed through to the Paimon Java SDK. You can use the following prefixes:

  ```
  paimon.s3.xxx
  paimon.s3a.xxx
  paimon.fs.xxx
  ```

  These parameters will be automatically converted and passed through. Here are some examples:

Input Parameter	Converted Parameter for Paimon Java SDK
“paimon.s3.list.version” = “1”	“fs.s3a.list.version” = “1”
“paimon.s3.paging.maximum” = “100”	“fs.s3a.paging.maximum” = “100”
“paimon.fs.s3.read.ahead.buffer.size” = “1”	“fs.s3a.read.ahead.buffer.size” = “1”
“paimon.s3a.replication.factor” = “3”	“fs.s3a.replication.factor” = “3”

- `{CommonProperties}`

  The CommonProperties section is used to fill in common properties. Please refer to the Catalog Overview section on [Common Properties].

Supported Paimon Versions

The currently dependent Paimon version is 1.0.0.

Supported Paimon Formats

- Supports reading Paimon Deletion Vector

Supported Metadata Services

- Hive Metastore

- Aliyun DLF
- FileSystem

> Note: The service types and parameters supported by different Doris versions are slightly different. Please refer to the Examples section.

Supported Storage Systems

- HDFS
- AWS S3
- Google Cloud Storage
- Alibaba Cloud OSS
- Tencent Cloud COS
- Huawei Cloud OBS
- MINIO

> Note: The service types and parameters supported by different Doris versions are slightly different. Please refer to the Examples section.

Supported Data Formats

- Parquet
- ORC

Column Type Mapping

Paimon Type	Doris Type	Comment
boolean	boolean	
tinyint	tinyint	
smallint	smallint	
integer	int	
bigint	bigint	
float	float	
double	double	

Paimon Type	Doris Type	Comment
decimal(P, S)	decimal(P, S)	
varchar	string	
char	string	
binary, varbinary	string/varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
date	date	
timestamp_without_time_zone	datetime(N)	Mapped according to precision. If precision is greater than 6, it maps to a maximum of 6 (may cause precision loss).
timestamp_with_local_time_zone	datetime(N)	Mapped according to precision. If precision is greater than 6, it maps to a maximum of 6 (may cause precision loss).
array	array	
map	map	
row	struct	
other	UNSUPPORTED	

> Note:
>
> Doris currently does not support `Timestamp` types with timezone. All `timestamp_without_time_zone` and `timestamp_with_local_time_zone` will be uniformly mapped to `datetime(N)` type. However, during reading, Doris will correctly handle timezones based on the actual source type. For example, after specifying a timezone with `SET time_zone=<tz>`, it will affect the return results of `timestamp_with_local_time_zone` columns.
>
> You can check whether the source type contains timezone information in the Extra column of the `DESCRIBE table_name` statement. If it displays `WITH_TIMEZONE`, it indicates that the source type is a timezone-aware type. (This feature is supported since 3.0.8)

Examples

Hive Metastore

Version 3.1+

HDFS

````
```sql
CREATE CATALOG test_paimon_on_hms_hdfs_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'hdfs://127.0.0.1:8320/user/hive/warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
````
````

```
    'fs.defaultFS' = 'hdfs://127.0.0.1:8320',
    'hadoop.username' = 'doris'
);
```

S3

```sql
CREATE CATALOG test_paimon_on_hms_aws_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 's3://bucket/paimon_warehouse',
    's3.region' = 'ap-east-1',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```

Obtaining S3 access credentials using an IAM Assumed Role (3.1.221+)

```sql
CREATE CATALOG paimon_hms_on_s3_iamrole PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 's3://bucket/warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.region' = 'us-east-1',
    's3.role_arn' = 'arn:aws:iam::543815668950:role/role'
);
```

OSS

```sql
CREATE CATALOG test_paimon_on_hms_ali_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 'oss://bucket/regression/paimon_warehouse',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```

COS

```sql
CREATE CATALOG test_paimon_on_hms_tx_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'cosn://bucket/paimon_warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'cos.region' = 'ap-beijing',
```

```
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```

OBS

```sql
CREATE CATALOG test_paimon_on_hms_hw_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'obs://bucket/regression/paimon_warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```

GCS

```sql
CREATE CATALOG `paimon_on_gcs` PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'gs://bucket/paimon_warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'fs.gcs.support' = 'true',
    'gs.secret_key' = '<sk>',
    'gs.access_key' = '<ak>'
);
```

MinIO

```sql
CREATE CATALOG paimon_hms_on_minio_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 's3://test-bucket/paimon-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'fs.minio.support' = 'true',
    'minio.endpoint' = 'http://127.0.0.1:19000',
    'minio.access_key' = 'minioadmin',
    'minio.secret_key' = 'minioadmin',
    'minio.use_path_style' = 'true'
);
```

Version 2.1 & 3.0

HDFS

Access HMS without Kerberos authentication

```sql
CREATE CATALOG paimon_hms_on_hdfs_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 'hdfs://127.0.0.1:8320/user/paimon/warehouse/paimon-hms-hdfs-warehouse',
    'hadoop.username' = 'doris',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8320'
);
```

Access HMS with Kerberos authentication enabled

```sql
CREATE CATALOG paimon_hms_on_hdfs_kerberos_hdfs_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'hdfs://127.0.0.1:8520/paimon-hms-hdfs-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.kerberos.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.sasl.enabled' = 'true',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8520',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
    RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
    DEFAULT',
    'hadoop.security.authentication' = 'kerberos',
    'hadoop.kerberos.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hadoop.kerberos.keytab' = '/keytabs/hive-presto-master.keytab'
);
```

S3

```sql
CREATE CATALOG test_paimon_on_hms_aws_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 's3://bucket-hk/paimon_warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.region' = 'ap-east-1',
    'paimon.s3.region' = 'ap-east-1',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
    'paimon.s3.endpoint' = 'ap-east-1',
    's3.access_key' = '<ak>',
    'paimon.s3.access-key' = '<ak>',
    's3.secret_key' = '<sk>',
    'paimon.s3.secret-key' = '<sk>'
);
```

OSS

```sql
CREATE CATALOG test_paimon_on_hms_ali_catalog PROPERTIES (
    'type' = 'paimon',
```

```
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'oss://bucket/regression/paimon_warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'oss.region' = 'cn-beijing',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com',
    'oss.access_key' = '<ak>',
    'oss.secret_key' = '<sk>'
);
```

COS

```
```sql
CREATE CATALOG test_paimon_on_hms_tx_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'cosn://bucket/paimon_warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```
```

OBS

```
```sql
CREATE CATALOG test_paimon_on_hms_hw_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 'obs://bucket/regression/paimon_warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```
```

MinIO

```
```sql
CREATE CATALOG test_paimon_on_hms_minio_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'hms',
    'warehouse' = 's3://bucket/paimon-warehouse',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.endpoint' = 'http://127.0.0.1:19000',
    'paimon.s3.endpoint' = 'http://127.0.0.1:19000',
    's3.access_key' = '<ak>',
    'paimon.s3.access-key' = '<ak>',
    's3.secret_key' = '<sk>',
    'paimon.s3.secret-key' = '<sk>',
```
```

```
    's3.path.style.access' = 'true'
);
```

Aliyun DLF Metastore

Version 3.1+

DLF 1.0

```sql
CREATE CATALOG paimon_dlf_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'dlf',
    'warehouse' = 'oss://bucket/p2_regression_case',
    'dlf.uid' = '772905',
    'dlf.catalog_id' = 'p2_regression_case',
    'dlf.region' = 'cn-beijing',
    'dlf.endpoint' = 'dlf.cn-beijing.aliyuncs.com',
    'dlf.access_key' = '<ak>',
    'dlf.secret_key' = '<sk>'
);
```

DLF(2.5+)

``` sql
CREATE CATALOG paimon_dlf_test PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'rest',
    'uri' = 'http://cn-beijing-vpc.dlf.aliyuncs.com',
    'warehouse' = 'new_dfl_paimon_catalog',
    'paimon.rest.token.provider' = 'dlf',
    'paimon.rest.dlf.access-key-id' = '<ak>',
    'paimon.rest.dlf.access-key-secret' = '<sk>'
);
```

Version 2.1 & 3.0

DLF 1.0

```sql
CREATE CATALOG paimon_dlf_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'dlf',
    'warehouse' = 'oss://bucket/paimon-dlf-oss-warehouse',
    'dlf.uid' = '7716890',
    'dlf.catalog.id' = 'p2_regression_case',
    'dlf.region' = 'cn-beijing',
    'dlf.endpoint' = 'dlf.cn-beijing.aliyuncs.com',
    'dlf.access_key' = '<ak>',
    'dlf.secret_key' = '<sk>'
```

```
);
```

FileSystem Metastore

Version 3.1+

HDFS

````
Access HMS and HDFS services without Kerberos authentication
```sql
CREATE CATALOG ctl_test_paimon_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'hdfs://127.0.0.1:8020/user/doris/paimon1',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8020'
);
```
````

S3

````
```sql
CREATE CATALOG test_paimon_on_fs_aws_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 's3://bucket/paimon_warehouse',
    's3.region' = 'ap-east-1',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```
Obtaining S3 access credentials using an IAM Assumed Role (3.1.2+)
```sql
CREATE CATALOG paimon_fs_on_s3_iamrole PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 's3://bucket/warehouse',
    's3.region' = 'us-east-1',
    's3.role_arn' = 'arn:aws:iam::543815668950:role/role'
);
```
````

OSS

````
Use OSS

```sql
CREATE CATALOG paimon_base_filesystem_paimon_oss PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'oss://bucket/regression/paimon1',
    'oss.region'='cn-beijing',
````
````

```
    'oss.endpoint'='oss-cn-beijing.aliyuncs.com',
    'oss.access_key'='<ak>',
    'oss.secret_key'='<sk>'
);
```

Use OSS-HDFS

```sql
CREATE CATALOG paimon_base_filesystem_paimon_oss_hdfs PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'oss://bucket/regression/paimon1',
    'fs.oss-hdfs.support' = 'true',
    'oss.hdfs.access_key' = '<ak>',
    'oss.hdfs.secret_key' = '<sk>',
    'oss.hdfs.endpoint' = 'cn-beijing.oss-dls.aliyuncs.com',
    'oss.hdfs.region' = 'cn-beijing'
);
```

COS

```sql
CREATE CATALOG paimon_base_filesystem_paimon_cosn PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'cosn://bucket/regression/paimon1',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```

OBS

```sql
CREATE CATALOG paimon_base_filesystem_paimon_obs PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'obs://bucket/regression/paimon1',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
```

GCS

```sql
CREATE CATALOG paimon_on_gcs_s3_fs PROPERTIES (
    'type' = 'paimon',
```

```
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 's3://bucket/paimon_warehouse',
    'fs.gcs.support' = 'true',
    'gs.secret_key' = '<sk>',
    'gs.access_key' = '<ak>'
);
```

MinIO

```
```sql
CREATE CATALOG test_paimon_minio_minio PROPERTIES (
    'type' = 'paimon',
    'warehouse' = 's3://warehouse/wh',
    'fs.minio.support' = 'true',
    'minio.endpoint' = 'http://127.0.0.1:19001',
    'minio.access_key' = 'admin',
    'minio.secret_key' = 'password',
    'minio.use_path_style' = 'true'
);
```
```

Version 2.1 & 3.0

HDFS

```
Access HMS and HDFS services without Kerberos authentication
```sql
CREATE CATALOG ctl_test_paimon_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'hdfs://127.0.0.1:8020/user/doris/paimon1',
    'fs.defaultFS' = 'hdfs://127.0.0.1:8020'
);
```
```

S3

```
```sql
CREATE CATALOG test_paimon_on_fs_aws_catalog PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 's3://bucket/paimon_warehouse',
    's3.region' = 'ap-east-1',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```
```

OSS

````
Use OSS

```sql
CREATE CATALOG paimon_base_filesystem_paimon_oss PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'oss://bucket/regression/paimon1',
    'oss.region'='cn-beijing',
    'oss.endpoint'='oss-cn-beijing.aliyuncs.com',
    'oss.access_key'='<ak>',
    'oss.secret_key'='<sk>'
);
```

Use OSS-HDFS

```sql
CREATE CATALOG paimon_base_filesystem_paimon_oss_hdfs PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'oss://bucket/regression/paimon1',
    'oss.hdfs.enabled' = 'true',
    'oss.access_key' = 'your-access-key',
    'oss.secret_key' = 'your-secret-key',
    'oss.endpoint' = 'cn-hangzhou.oss-dls.aliyuncs.com',
    'oss.region' = 'cn-hangzhou'
);
```
````

COS

````
```sql
CREATE CATALOG paimon_base_filesystem_paimon_cosn PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'cosn://bucket/regression/paimon1',
    'cos.region' = 'ap-beijing',
    'cos.endpoint' = 'cos.ap-beijing.myqcloud.com',
    'cos.access_key' = '<ak>',
    'cos.secret_key' = '<sk>'
);
```
````

OBS

````
```sql
CREATE CATALOG paimon_base_filesystem_paimon_obs PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'filesystem',
    'warehouse' = 'obs://bucket/regression/paimon1',
    'obs.region' = 'cn-north-4',
    'obs.endpoint' = 'obs.cn-north-4.myhuaweicloud.com',
    'obs.access_key' = '<ak>',
    'obs.secret_key' = '<sk>'
);
````
````

```
```

MinIO

```sql
CREATE CATALOG paimon_base_filesystem_paimon_minio PROPERTIES (
    'type' = 'paimon',
    'warehouse' = 's3://warehouse/wh',
    'paimon.catalog.type' = 'filesystem',
    's3.endpoint' = 'http://127.0.0.1:19001',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.path.style.access' = 'true'
);
```

Query Operations

Basic Query

Once the Catalog is configured, you can query the table data in the Catalog as follows:

```
-- 1. Switch to catalog, use database, and query
SWITCH paimon_ctl;
USE paimon_db;
SELECT * FROM paimon_tbl LIMIT 10;

-- 2. Use Paimon database directly
USE paimon_ctl.paimon_db;
SELECT * FROM paimon_tbl LIMIT 10;

-- 3. Use fully qualified name to query
SELECT * FROM paimon_ctl.paimon_db.paimon_tbl LIMIT 10;
```

Batch Incremental Query

> Since version 3.1.0

Supports Batch Incremental queries for Paimon, similar to Flink.

Supports querying incremental data within specified snapshot or timestamp intervals. The interval is left-closed and right-open.

```
-- between snapshots [0, 5)
SELECT * FROM paimon_table@incr('startSnapshotId'='0', 'endSnapshotId'='5');

-- between snapshots [0, 5) with specified scan mode
SELECT * FROM paimon_table@incr('startSnapshotId'='0', 'endSnapshotId'='5', 'incrementalBetweenScanMode'='diff');

-- read from start timestamp
```

```
SELECT * FROM paimon_table@incr('startTimestamp'='1750844949000');

-- read between timestamp
SELECT * FROM paimon_table@incr('startTimestamp'='1750844949000', 'endTimestamp'='1750944949000');
```

Parameter:

Parameter	Description	Example
`startSnapshotId`	Starting snapshot ID, must be greater than 0. Must be specified with `endSnapshotId` together.	`'startSnapshotId'='3'`
`endSnapshotId`	Ending snapshot ID, must be greater than `startSnapshotId`. Must be specified with `startSnapshotId` together.	`'endSnapshotId'='10'`
`incrementalBetweenScanMode`	Specifies the incremental read mode, default is `auto`, supports `delta`, `changelog` and `diff`	`'incrementalBetweenScanMode'='delta'`
`startTimestamp`	Starting snapshot timestamp, must be greater than or equal to 0. Unit is millisecond.	`'startTimestamp'='1750844949000'`
`endTimestamp`	Ending snapshot timestamp, must be greater than `startTimestamp`. Optional, if not specified, reads from `startTimestamp` to the latest snapshot. Unit is millisecond.	`'endTimestamp'='1750944949000'`

> Notice:
>
> `startSnapshotId` and `endSnapshotId` will compose the Paimon parameter `'incremental-between'='3,10'`
>
> `startTimestamp` and `endTimestamp` will compose the Paimon parameter `'incremental-between-timestamp'='1750844949000,1750944949000'`
>
> `incrementalBetweenScanMode` corresponds to the Paimon parameter `incremental-between-scan-mode`.

Refer to the Paimon documentation for further details about these parameters.

Time Travel

> Since version 3.1.0

Supports reading specified snapshots of Paimon tables.

By default, read requests only read the latest version of snapshots.

You can query snapshots of a specified Paimon table through the `$snapshots` table function:

```
Doris > SELECT snapshot_id,commit_time FROM paimon_tbl$snapshots;
+-------------+-------------------------+
| snapshot_id | commit_time             |
+-------------+-------------------------+
```

```
|           1 | 2025-08-17 06:05:52.740 |
|           2 | 2025-08-17 06:05:52.979 |
|           3 | 2025-08-17 06:05:53.240 |
|           4 | 2025-08-17 06:05:53.561 |
+-------------+-------------------------+
```

You can use `FOR TIME AS OF` and `FOR VERSION AS OF` statements to read historical version data based on snapshot ID or the time when the snapshot was created. Examples:

```
-- Notice:
-- Before 3.1.1, only support format like YYYY-MM-DD HH:MM:SS.SSS
-- After 3.1.2, support YYYY-MM-DD HH:MM:SS and YYYY-MM-DD
SELECT * FROM paimon_tbl FOR TIME AS OF "2025-08-17 06:05:52.740";
SELECT * FROM paimon_tbl FOR TIME AS OF "2025-08-17 06:05:52";
SELECT * FROM paimon_tbl FOR TIME AS OF "2025-08-17";

-- Notice: The timestamp must be precise down to the millisecond.
SELECT * FROM paimon_tbl FOR TIME AS OF 1755381952740;
-- Use snapshot id
SELECT * FROM paimon_tbl FOR VERSION AS OF 1;
```

Branch and Tag

> Since version 3.1.0

Supports reading branches and tags of specified Paimon tables.

You can use the `$branches` and `$tags` system tables to view branches and tags of Paimon tables:

```
Doris > SELECT * FROM paimon_tbl$branches;
+-------------+-------------------------+
| branch_name | create_time             |
+-------------+-------------------------+
| b_1         | 2025-08-17 06:34:37.294 |
| b_2         | 2025-08-17 06:34:37.297 |
+-------------+-------------------------+

Doris > SELECT * FROM paimon_tbl$tags;
+----------+-------------+-----------+-------------------------+--------------+-------------+---------------+
| tag_name | snapshot_id | schema_id | commit_time             | record_count | create_time | time_retained |
+----------+-------------+-----------+-------------------------+--------------+-------------+---------------+
| t_1      |           1 |         0 | 2025-08-17 06:05:52.740 |            3 | NULL        | NULL          |
| t_2      |           2 |         0 | 2025-08-17 06:05:52.979 |            6 | NULL        | NULL          |
| t_3      |           3 |         0 | 2025-08-17 06:05:53.240 |            9 | NULL        | NULL          |
| t_4      |           4 |         0 | 2025-08-17 06:05:53.561 |           12 | NULL        | NULL          |
+----------+-------------+-----------+-------------------------+--------------+-------------+---------------+
```

Supports various syntax forms to be compatible with systems like Spark/Trino:

```
-- BRANCH
SELECT * FROM paimon_tbl@branch(branch1);
SELECT * FROM paimon_tbl@branch("name" = "branch1");

-- TAG
SELECT * FROM paimon_tbl@tag(tag1);
```

```
SELECT * FROM paimon_tbl@tag("name" = "tag1");
SELECT * FROM paimon_tbl FOR VERSION AS OF 'tag1';
```

For the `FOR VERSION AS OF` syntax, Doris will automatically determine whether the parameter is a timestamp

System Tables

> Since version 3.1.0

Doris supports querying Paimon system tables to retrieve table-related metadata. System tables can be used to view snapshot history, manifest files, data files, partitions, and other information.

To access metadata of a Paimon table, add a `$` symbol after the table name, followed by the system table name:

```
SELECT * FROM my_table$system_table_name;
```

> Note: Doris does not support reading Paimon global system tables, which are only supported in Flink.

schemas

Shows current and historical schema information of the table. When modifying table schema using `ALTER TABLE`, `CREATE TABLE AS`, or `CREATE DATABASE AS` statements, each modification generates a record in the schemas table:

```
SELECT * FROM my_table$schemas;
```

Result:

```
+-----------+-----------------------------------------------------------------------------------------------------------------------------+--
--------------+--------------+---------+---------+------------------------+
| schema_id | fields                                                                                                                                 |
partition_keys | primary_keys | options | comment | update_time             |
+-----------+-----------------------------------------------------------------------------------------------------------------------------+--
--------------+--------------+---------+---------+------------------------+
|         0 | [{"id":0,"name":"k","type":"INT NOT NULL"},{"id":1,"name":"f0","type":"INT"},{"id":2,"name":"f1","type":"STRING"}] |
[]             | ["k"]        | {}      |         | 2025-03-04 22:48:41.666 |
+-----------+-----------------------------------------------------------------------------------------------------------------------------+--
--------------+--------------+---------+---------+------------------------+
```

snapshots

Shows all valid snapshot information of the table, including snapshot creation time, commit user, operation type, etc.:

```
SELECT * FROM my_table$snapshots;
```

Result:

```
+-------------+-----------+--------------------------------------+--------------------+-------------+-------------------------+----
-------------------------------------------------+-------------------------------------------------------------+-------------------------
+--------------------+--------------------+-----------------------+----------------------+
| snapshot_id | schema_id | commit_user                          | commit_identifier   | commit_kind | commit_time             |
base_manifest_list                               | delta_manifest_list                                 |
changelog_manifest_list | total_record_count | delta_record_count | changelog_record_count | watermark            |
+-------------+-----------+--------------------------------------+--------------------+-------------+-------------------------+----
-------------------------------------------------+-------------------------------------------------------------+-------------------------
+--------------------+--------------------+-----------------------+----------------------+
|           1 |         0 | d7ea4996-92c7-469f-b9ff-c76525954f1c | 9223372036854775807 | APPEND      | 2025-03-04 22:48:45.575 |
manifest-list-dc5490ba-420c-445a-b6f7-6962d394935c-0 | manifest-list-dc5490ba-420c-445a-b6f7-6962d394935c-1 | NULL
|                   1 |                   1 |                     0 | -9223372036854775808 |
|           2 |         0 | 34de47f6-31d1-4f06-b378-c85ef4fbca41 | 9223372036854775807 | APPEND      | 2025-07-01 23:11:35.406 |
manifest-list-dca6aa5b-6fc6-4b4f-ac22-acfa15bbf171-0 | manifest-list-dca6aa5b-6fc6-4b4f-ac22-acfa15bbf171-1 | NULL
|                   2 |                   1 |                     0 | -9223372036854775808 |
|           3 |         0 | 89f67183-a1f8-4ee9-b73c-3f7e992b79a7 | 9223372036854775807 | APPEND      | 2025-07-01 23:11:45.114 |
manifest-list-6d624d1b-c774-4d95-905e-8258a7b89ecb-0 | manifest-list-6d624d1b-c774-4d95-905e-8258a7b89ecb-1 | NULL
|                   3 |                   1 |                     0 | -9223372036854775808 |
|           4 |         0 | 31924a7c-1389-490c-adf1-3bb805b33cd7 | 9223372036854775807 | APPEND      | 2025-07-01 23:12:42.042 |
manifest-list-09097a51-afde-485e-929b-d2cc39eb6eb2-0 | manifest-list-09097a51-afde-485e-929b-d2cc39eb6eb2-1 | NULL
|                   5 |                   2 |                     0 | -9223372036854775808 |
|           5 |         0 | 1e90a80b-41cb-4242-b97c-889728f76810 | 9223372036854775807 | APPEND      | 2025-07-01 23:14:26.445 |
manifest-list-b8471969-9c4d-41cd-b790-64f6efb2d142-0 | manifest-list-b8471969-9c4d-41cd-b790-64f6efb2d142-1 | NULL
|                   6 |                   1 |                     0 | -9223372036854775808 |
|           6 |         0 | 1e90a80b-41cb-4242-b97c-889728f76810 | 9223372036854775807 | COMPACT     | 2025-07-01 23:14:29.317 |
manifest-list-b8471969-9c4d-41cd-b790-64f6efb2d142-2 | manifest-list-b8471969-9c4d-41cd-b790-64f6efb2d142-3 | NULL
|                   5 |                  -1 |                     0 | -9223372036854775808 |
+-------------+-----------+--------------------------------------+--------------------+-------------+-------------------------+----
-------------------------------------------------+-------------------------------------------------------------+-------------------------
+--------------------+--------------------+-----------------------+----------------------+
```

options

Shows current configuration options of the table. If a table option is not included in the table, that option is set to its default value:

```
SELECT * FROM my_table$options;
```

Result:

```
+------------------------+--------------------+
|         key            |        value       |
+------------------------+--------------------+
| snapshot.time-retained |         5 h        |
+------------------------+--------------------+
```

files

Shows information about all data files pointed to by the current snapshot, including file format, record count, file size, etc.:

```
SELECT * FROM my_table$files;
```

Result:

```
mysql> SELECT * FROM my_table$files;
+-----------+--------+----------------------------------------------------------------------------------------------------------------------------------------------------
----------+-------------+-----------+-------+--------------+--------------------+---------+---------+-------------------+-----------
----------+---------------------+---------------------+--------------------+-------------------------+-------------+
| partition | bucket | file_path
| file_format | schema_id | level | record_count | file_size_in_bytes | min_key | max_key | null_value_counts | min_value_stats
| max_value_stats     | min_sequence_number | max_sequence_number | creation_time           | file_source |
+-----------+--------+----------------------------------------------------------------------------------------------------------------------------------------------------
----------+-------------+-----------+-------+--------------+--------------------+---------+---------+-------------------+-----------
----------+---------------------+---------------------+--------------------+-------------------------+-------------+
| {}        |      0 | s3://paimon-warehouse-dev/test-flink/cookbook.db/my_table/bucket-0/data-b4a49c57-6ef6-4c04-8813-07a4960d987c-
0.parquet | parquet     |         0 |     5 |            5 |               1321 | [1]     | [6]     | {f0=0, f1=0, k=0} | {f0=4,
f1=111, k=1} | {f0=11, f1=k7, k=6} |                   0 |                   5 | 2025-07-01 23:14:23.967 | COMPACT     |
+-----------+--------+----------------------------------------------------------------------------------------------------------------------------------------------------
----------+-------------+-----------+-------+--------------+--------------------+---------+---------+-------------------+-----------
----------+---------------------+---------------------+--------------------+-------------------------+-------------+
```

tags

Shows all tag information of the table, including tag names and associated snapshots:

```
SELECT * FROM my_table$tags;
```

Result:

```
+----------+-------------+-----------+-------------------------+--------------+--------------+
| tag_name | snapshot_id | schema_id |             commit_time | record_count |   branches   |
+----------+-------------+-----------+-------------------------+--------------+--------------+
|     tag1 |           1 |         0 | 2025-03-04 14:55:29.344 |            3 |      []      |
|     tag3 |           3 |         0 | 2025-03-04 14:58:24.691 |            7 |  [branch-1]  |
+----------+-------------+-----------+-------------------------+--------------+--------------+
```

branches

Shows all known branch information of the table:

```
SELECT * FROM my_table$branches;
```

Result:

```
+----------------------+-------------------------+
|          branch_name |             create_time |
+----------------------+-------------------------+
|              branch1 | 2025-03-04 20:31:39.084 |
|              branch2 | 2025-03-04 21:11:14.373 |
+----------------------+-------------------------+
```

consumers

Shows consumer information of the table, used to track data consumption:

```
SELECT * FROM my_table$consumers;
```

Result:

```
+-------------+------------------+
| consumer_id | next_snapshot_id |
+-------------+------------------+
|         id1 |                1 |
|         id2 |                3 |
+-------------+------------------+
```

manifests

Shows manifest file information of the table's current snapshot:

```
SELECT * FROM my_table$manifests;
```

Result:

```
+-------------------------------------------------+-----------+-----------------+-------------------+-----------+--------------------
--+---------------------+
| file_name                                       | file_size | num_added_files | num_deleted_files | schema_id |
min_partition_stats | max_partition_stats |
+-------------------------------------------------+-----------+-----------------+-------------------+-----------+--------------------
--+---------------------+
| manifest-3df9bb64-5c11-4aef-994e-d8717fedfc70-0 |      1949 |               1 |                 0 |         0 | {}
| {}                  |
| manifest-d7eb4ec4-7238-478a-9ae6-91a4ccebd561-0 |      1946 |               1 |                 0 |         0 | {}
| {}                  |
| manifest-3b6f4079-c893-4413-aedc-1e8fbcea6db1-0 |      1948 |               1 |                 0 |         0 | {}
| {}                  |
| manifest-abe5177f-82da-4e86-9864-40efffb391bd-0 |      1964 |               1 |                 0 |         0 | {}
| {}                  |
| manifest-ee89dff3-a523-4655-a4b8-d7c9e471a1d6-0 |      1949 |               1 |                 0 |         0 | {}
| {}                  |
| manifest-ee89dff3-a523-4655-a4b8-d7c9e471a1d6-1 |      2232 |               1 |                 5 |         0 | {}
| {}                  |
+-------------------------------------------------+-----------+-----------------+-------------------+-----------+--------------------
--+---------------------+
```

aggregation_fields

Shows aggregation field information of the table, used for field configuration in aggregate tables:

```
SELECT * FROM my_table$aggregation_fields;
```

Result:

```
+------------+--------------+----------+------------------+---------+
| field_name | field_type   | function | function_options | comment |
+------------+--------------+----------+------------------+---------+
| k          | INT NOT NULL | []       | []               | NULL    |
| f0         | INT          | []       | []               | NULL    |
```

```
| f1         | STRING       | []       | []               | NULL    |
+------------+--------------+----------+------------------+---------+
```

partitions

Shows partition information of the table, including total record count and total file size for each partition:

```
SELECT * FROM my_table$partitions;
```

Result:

```
+-----------+--------------+--------------------+------------+-------------------------+
| partition | record_count | file_size_in_bytes | file_count | last_update_time        |
+-----------+--------------+--------------------+------------+-------------------------+
| {}        |            5 |               1321 |          1 | 2025-07-01 23:14:23.967 |
+-----------+--------------+--------------------+------------+-------------------------+
```

buckets

Shows bucket information of the table, including statistics for each bucket:

```
SELECT * FROM my_table$buckets;
```

Result:

```
+-----------+--------+--------------+--------------------+------------+-------------------------+
| partition | bucket | record_count | file_size_in_bytes | file_count | last_update_time        |
+-----------+--------+--------------+--------------------+------------+-------------------------+
| {}        |      0 |            5 |               1321 |          1 | 2025-07-01 23:14:23.967 |
+-----------+--------+--------------+--------------------+------------+-------------------------+
```

statistics

Shows statistical information of the table, including row count, data size, and other statistics:

```
SELECT * FROM my_table$statistics;
```

Result:

```
+--------------+------------+-----------------------+------------------+----------+
|  snapshot_id |  schema_id |     mergedRecordCount | mergedRecordSize |  colstat |
+--------------+------------+-----------------------+------------------+----------+
|            2 |          0 |              2        |         2        |    {}    |
+--------------+------------+-----------------------+------------------+----------+
```

table_indexes

Shows index information of the table:

```
SELECT * FROM my_table$table_indexes;
```

Result:

```
+--------------------------------+-------------+--------------------------------+--------------------------------+-----------------
----+---------------------+--------------------------------+
|                      partition |      bucket |                     index_type |                      file_name |
file_size |           row_count |                     dv_ranges |
+--------------------------------+-------------+--------------------------------+--------------------------------+-----------------
----+---------------------+--------------------------------+
|                   {2025-03-01} |           0 |                           HASH | index-70abfebf-149e-4796-9f... |
12 |                   3 |                        <NULL> |
|                   {2025-04-01} |           0 |             DELETION_VECTORS | index-633857e7-cdce-47d2-87... |
33 |                   1 | [(data-346cb9c8-4032-4d66-a... |
+--------------------------------+-------------+--------------------------------+--------------------------------+-----------------
----+---------------------+--------------------------------+
```

System Table Use Cases

Through system tables, you can easily accomplish the following operations and monitoring tasks.

View the latest snapshot information of a table to understand its current state

```
SELECT snapshot_id, commit_time, commit_kind, total_record_count FROM catalog_sales$snapshots ORDER BY snapshot_id DESC;
```

Result:

```
+-------------+-------------------------+-------------+--------------------+
| snapshot_id | commit_time             | commit_kind | total_record_count |
+-------------+-------------------------+-------------+--------------------+
|           1 | 2025-07-01 21:21:54.179 | APPEND      |           14329288 |
+-------------+-------------------------+-------------+--------------------+
```

View table information for snapshots

```
SELECT s.snapshot_id, t.schema_id, t.fields FROM store_sales$snapshots s JOIN store_sales$schemas t ON s.schema_id=t.schema_id;
```

Result:

```
+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| snapshot_id | schema_id | fields
|
+-------------+-----------+-----------------------------------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------+
|          1 |         0 | [{"id":0,"name":"ss_sold_date_sk","type":"INT"},{"id":1,"name":"ss_item_sk","type":"INT NOT NULL"},
{"id":2,"name":"ss_ticket_number","type":"INT NOT NULL"},{"id":3,"name":"ss_sold_time_sk","type":"INT"},
{"id":4,"name":"ss_customer_sk","type":"INT"},{"id":5,"name":"ss_cdemo_sk","type":"INT"},{"id":6,"name":"ss_hdemo_sk","type":"INT"},
{"id":7,"name":"ss_addr_sk","type":"INT"},{"id":8,"name":"ss_store_sk","type":"INT"},{"id":9,"name":"ss_promo_sk","type":"INT"},
{"id":10,"name":"ss_quantity","type":"INT"},{"id":11,"name":"ss_wholesale_cost","type":"DECIMAL(7, 2)"},
{"id":12,"name":"ss_list_price","type":"DECIMAL(7, 2)"},{"id":13,"name":"ss_sales_price","type":"DECIMAL(7, 2)"},
{"id":14,"name":"ss_ext_discount_amt","type":"DECIMAL(7, 2)"},{"id":15,"name":"ss_ext_sales_price","type":"DECIMAL(7, 2)"},
{"id":16,"name":"ss_ext_wholesale_cost","type":"DECIMAL(7, 2)"},{"id":17,"name":"ss_ext_list_price","type":"DECIMAL(7, 2)"},
{"id":18,"name":"ss_ext_tax","type":"DECIMAL(7, 2)"},{"id":19,"name":"ss_coupon_amt","type":"DECIMAL(7, 2)"},
{"id":20,"name":"ss_net_paid","type":"DECIMAL(7, 2)"},{"id":21,"name":"ss_net_paid_inc_tax","type":"DECIMAL(7, 2)"},
{"id":22,"name":"ss_net_profit","type":"DECIMAL(7, 2)"}] |
|          2 |         0 | [{"id":0,"name":"ss_sold_date_sk","type":"INT"},{"id":1,"name":"ss_item_sk","type":"INT NOT NULL"},
{"id":2,"name":"ss_ticket_number","type":"INT NOT NULL"},{"id":3,"name":"ss_sold_time_sk","type":"INT"},
{"id":4,"name":"ss_customer_sk","type":"INT"},{"id":5,"name":"ss_cdemo_sk","type":"INT"},{"id":6,"name":"ss_hdemo_sk","type":"INT"},
{"id":7,"name":"ss_addr_sk","type":"INT"},{"id":8,"name":"ss_store_sk","type":"INT"},{"id":9,"name":"ss_promo_sk","type":"INT"},
{"id":10,"name":"ss_quantity","type":"INT"},{"id":11,"name":"ss_wholesale_cost","type":"DECIMAL(7, 2)"},
{"id":12,"name":"ss_list_price","type":"DECIMAL(7, 2)"},{"id":13,"name":"ss_sales_price","type":"DECIMAL(7, 2)"},
{"id":14,"name":"ss_ext_discount_amt","type":"DECIMAL(7, 2)"},{"id":15,"name":"ss_ext_sales_price","type":"DECIMAL(7, 2)"},
{"id":16,"name":"ss_ext_wholesale_cost","type":"DECIMAL(7, 2)"},{"id":17,"name":"ss_ext_list_price","type":"DECIMAL(7, 2)"},
{"id":18,"name":"ss_ext_tax","type":"DECIMAL(7, 2)"},{"id":19,"name":"ss_coupon_amt","type":"DECIMAL(7, 2)"},
{"id":20,"name":"ss_net_paid","type":"DECIMAL(7, 2)"},{"id":21,"name":"ss_net_paid_inc_tax","type":"DECIMAL(7, 2)"},
{"id":22,"name":"ss_net_profit","type":"DECIMAL(7, 2)"}] |
+-------------+-----------+-------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------+
```

View data distribution of buckets

```
SELECT `bucket` , COUNT(*) as file_count, SUM(file_size_in_bytes)/1024/1024 as total_size_mb from
paimon_s3.tpcds.catalog_sales$files GROUP BY `bucket`  ORDER BY total_size_mb;
```

> Note: Many fields in Paimon system tables are keywords in Doris, so they need to be enclosed in backticks.

Result:

```
+--------+------------+--------------------+
| bucket | file_count | total_size_mb      |
+--------+------------+--------------------+
|     35 |          1 | 12.144722938537598 |
|     81 |          1 | 12.143454551696777 |
```

```
|     37 |          1 |  12.14071273803711 |
|     36 |          1 | 12.139023780822754 |
|     63 |          1 | 12.137332916259766 |
|      7 |          1 | 12.122495651245117 |
|     15 |          1 | 12.117934226989746 |
|     11 |          1 | 12.116133689880371 |
|     12 |          1 |  12.11155891418457 |
|     46 |          1 | 12.111005783081055 |
+--------+------------+--------------------+
```

Appendix

FAQ

1. `Could not find a file io implementation for scheme 's3a' in the classpath`

   In versions prior to 3.1, when using HMS as metadata storage with ‘s3a’ or ‘s3’ protocol on the storage side, the following error occurs:

   ```
   java.io.UncheckedIOException: org.apache.paimon.fs.UnsupportedSchemeException: Could not find a file io implementation for
   scheme 's3a' in the classpath. Hadoop FileSystem also cannot access this path 's3a://`.
   ```

   Versions prior to 3.1 Versions prior to 3.1 do not support the `s3a` protocol. If you must use it, you can temporarily add the `s3a`-related parameters as a workaround.

   Version 3.1 and above Starting from version 3.1, explicitly adding `s3a` parameters is no longer necessary. If you need to override Paimon storage parameters, you can add parameters with the `paimon.fs.` prefix. Since Paimon internally uses the HDFS API, all `fs.s3a.*` parameters are supported.

   For example:

   ```
   CREATE CATALOG `paimon_hms_on_s3a` PROPERTIES (
       "type" = "paimon",
       "paimon.catalog.type" = "hms",
       "hive.metastore.uris" = "thrift://172.20.48.119:9383"
       "warehouse" = "s3a://bucket/paimon_warehouse",
       "s3.access_key" = "<ak>",
       "s3.secret_key" = "<sk>",
       "s3.region" = "ap-east-1",
       "s3.endpoint" = "https://s3.ap-east-1.amazonaws.com",
       -- explicitly add s3a parameters
       "fs.s3a.access.key"="<ak>",
       "fs.s3a.secret.key"="<sk>",
       "fs.s3a.endpoint.region"="ap-east-1",
       "fs.s3a.endpoint"="https://s3.ap-east-1.amazonaws.com",
       "fs.s3a.impl"= "org.apache.hadoop.fs.s3a.S3AFileSystem",
   );
   ```

   When using the `s3` protocol, you need to explicitly add Paimon S3-related parameters, i.e., add the `paimon.` prefix. For example:

   ```
   CREATE CATALOG test_paimon_on_hms_aws_catalog PROPERTIES (
       'type' = 'paimon',
       'paimon.catalog.type' = 'hms',
       'warehouse' = 's3://bucket-hk/paimon_warehouse',
   ```

```
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.region' = 'ap-east-1',
    's3.endpoint' = 'https://s3.ap-east-1.amazonaws.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    -- explicitly add s3a parameters
    'paimon.s3.region' = 'ap-east-1',
    'paimon.s3.endpoint' = 'ap-east-1',
    'paimon.s3.access-key' = '<ak>',
    'paimon.s3.secret-key' = '<sk>'
);
```

MaxCompute Catalog

MaxCompute is an enterprise-level SaaS (Software as a Service) cloud data warehouse on Alibaba Cloud. Through the open storage SDK provided by MaxCompute, Doris can access MaxCompute table information and perform queries.

Applicable Scenarios

Scenario	Description
Data Integration	Read MaxCompute data and write it to Doris internal tables.
Data Write-back	Not supported.

Notes

1. Starting from version 2.1.7, the MaxCompute Catalog is developed based on the open storage SDK. Before this, it was developed based on the Tunnel API.
2. There are certain restrictions on the use of the open storage SDK. Please refer to the `Usage Restrictions` section in this document.
3. Before Doris version 3.1.3, a `Project` in MaxCompute is equivalent to a `Database` in Doris. After 3.1.3, You can use the `mc.enable.namespace.schema` parameter to introduce the MaxCompute schema level.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name PROPERTIES (
    'type' = 'max_compute',
    {McRequiredProperties},
    {McOptionalProperties},
    {CommonProperties}
);
```

- `{McRequiredProperties}`

Property Name	Description	Supported Doris Version
`mc.default.project`	The name of the MaxCompute project you want to access. You can create and manage it in the MaxCompute Project List.	

Property Name	Description	Supported Doris Version
mc.access_key	AccessKey. You can create and manage it in the Alibaba Cloud Console.	
mc.secret_key	SecretKey. You can create and manage it in the Alibaba Cloud Console.	
mc.region	The region where MaxCompute is enabled. You can find the corresponding region from the Endpoint.	Before 2.1.7
mc.endpoint	The region where MaxCompute is enabled. Please refer to the section below on how to obtain Endpoint and Quota for configuration.	2.1.7 and later

- {McOptionalProperties}

Property Name	Default Value	Description	Supported Doris Version
mc.tunnel_endpoint	None	Refer to the appendix on Custom Service Address.	Before 2.1.7
mc.odps_endpoint	None	Refer to the appendix on Custom Service Address.	Before 2.1.7
mc.quota	pay-as-you-go	Quota name. Please refer to the section on how to obtain Endpoint and Quota for configuration.	2.1.7 and later
mc.split_strategy	byte_size	Sets the split strategy. Can be set to byte_size (split by byte size) or row_count (split by number of rows).	2.1.7 and later
mc.split_byte_size	268435456	The file size (in bytes) read by each split. Default is 256 MB. Effective only when "mc.split_strategy" = "byte_size".	2.1.7 and later
mc.split_row_count	1048576	The number of rows read by each split. Effective only when "mc.split_strategy" = "row_count".	2.1.7 and later
mc.split_cross_partition	false	Whether the generated split crosses partitions.	2.1.8 and later
mc.connect_timeout	10s	Timeout for connecting to MaxCompute.	2.1.8 and later
mc.read_timeout	120s	Timeout for reading from MaxCompute.	2.1.8 and later
mc.retry_count	4	Number of retries after a timeout.	2.1.8 and later
mc.datetime_predicate_push_down	true	Whether to allow pushdown of predicate conditions of timestamp/timestamp_ntz types. Doris will lose precision (9 -> 6) when synchronizing these two types. Therefore, if the original data has a precision higher than 6 digits, condition pushdown may lead to inaccurate results.	2.1.9/3.0.5 and later
mc.account_format	name	The account systems of Alibaba Cloud International and China are different. For users of the International site, if an error such as user 'RAM$xxxxxx:xxxxx' is not a valid aliyun account occurs, you can specify this parameter as id.	3.0.9/3.1.1 later
mc.enable.namespace.schema	false	Whether MaxCompute's schema level is supported. For details, see: https://help.aliyun.com/zh/maxcompute/user-guide/schema-related-operations	3.1.3 and later

- [CommonProperties]

The CommonProperties section is used to fill in common properties. Please refer to the Catalog Overview section on Common Properties.

Supported MaxCompute Versions

Only the public cloud version of MaxCompute is supported. For support with the private cloud version, please contact the Doris community.

Supported MaxCompute Formats

- Supports reading partitioned tables, clustered tables, and materialized views.
- Does not support reading MaxCompute external tables, logical views, or Delta Tables.

Hierarchical Mapping

- `mc.enable.namespace.schema` is false

Doris	MaxCompute
Catalog	N/A
Database	Project
Table	Table

- `mc.enable.namespace.schema` is true

Doris	MaxCompute
Catalog	Project
Database	Schema
Table	Table

Column Type Mapping

MaxCompute Type	Doris Type	Comment
bolean	boolean	
tiny	tinyint	
tinyint	tinyint	
smallint	smallint	
int	int	
bigint	bigint	
float	float	
double	double	

MaxCompute Type	Doris Type	Comment
decimal(P, S)	decimal(P, S)	1 <= P <= 38 ,0 <= scale <= 18
char(N)	char(N)	
varchar(N)	varchar(N)	
string	string	
date	date	
datetime	datetime(3)	Fixed mapping to precision 3. You can specify the time zone using `SET [GLOBAL] time_zone = 'Asia/Shanghai'`.
timestamp_ntz	datetime(6)	The precision of MaxCompute's `timestamp_ntz` is 9, but Doris' DATETIME supports a maximum precision of 6. Therefore, the extra part will be directly truncated when reading data.
timestamp	datetime(6)	Since 2.1.9 & 3.0.5. The precision of MaxCompute's `timestamp` is 9, but Doris' DATETIME supports a maximum precision of 6. Therefore, the extra part will be directly truncated when reading data.
array	array	
map	map	
struct	struct	
other	UNSUPPORTED	

Examples

```
CREATE CATALOG mc_catalog PROPERTIES (
    'type' = 'max_compute',
    'mc.default.project' = 'project',
    'mc.access_key' = 'sk',
    'mc.secret_key' = 'ak',
    'mc.endpoint' = 'http://service.cn-beijing-vpc.MaxCompute.aliyun-inc.com/api'
);
```

If you are using a version earlier than 2.1.7 (exclusive), please use the following statements. (It is recommended to upgrade to 2.1.8 or later for usage)

```
CREATE CATALOG mc_catalog PROPERTIES (
    'type' = 'max_compute',
    'mc.region' = 'cn-beijing',
    'mc.default.project' = 'project',
    'mc.access_key' = 'ak',
    'mc.secret_key' = 'sk'
    'mc.odps_endpoint' = 'http://service.cn-beijing.maxcompute.aliyun-inc.com/api',
    'mc.tunnel_endpoint' = 'http://dt.cn-beijing.maxcompute.aliyun-inc.com'
);
```

Support Schema:

```
CREATE CATALOG mc_catalog PROPERTIES (
    'type' = 'max_compute',
    'mc.region' = 'cn-beijing',
```

```
    'mc.default.project' = 'project',
    'mc.access_key' = 'ak',
    'mc.secret_key' = 'sk'
    'mc.odps_endpoint' = 'http://service.cn-beijing.maxcompute.aliyun-inc.com/api',
    'mc.tunnel_endpoint' = 'http://dt.cn-beijing.maxcompute.aliyun-inc.com',
    'mc.enable.namespace.schema' = 'true'
);
```

Query Operations

Basic Query

```
-- 1. Switch to catalog, use database, and query
SWITCH mc_ctl;
USE mc_ctl;
SELECT * FROM mc_tbl LIMIT 10;

-- 2. Use mc database directly
USE mc_ctl.mc_db;
SELECT * FROM mc_tbl LIMIT 10;

-- 3. Use fully qualified name to query
SELECT * FROM mc_ctl.mc_db.mc_tbl LIMIT 10;
```

Appendix

How to Obtain Endpoint and Quota (For Doris 2.1.7 and Later)

1. If using a dedicated resource group for Data Transmission Service (DTS)

   Refer to the documentation under the section **“Use Dedicated Data Service Resource Groups”**, specifically **“2. Authorization”**, to enable the required permissions. Then, navigate to the **“Quota Management”** list to view and copy the corresponding `QuotaName`, and specify it using `"mc.quota" = "QuotaName"`. At this point, you can choose to access MaxCompute via VPC or public network. However, VPC provides guaranteed bandwidth, while public network bandwidth is limited.

2. If using `pay-as-you-go`

   Refer to the documentation under the section **“Using Open Storage (Pay-As-You-Go)”** to enable the Open Storage (Storage API) switch and grant permissions to the user corresponding to the Ak and SK. In this case, `mc.quota` defaults to `pay-as-you-go`, and no additional value needs to be specified. When using the pay-as-you-go model, MaxCompute can only be accessed via VPC, and public network access is not available. Only prepaid users can access MaxCompute via the public network.

3. Configure `mc.endpoint` based on the Alibaba Cloud Endpoints Documentation

   For users accessing via VPC, refer to the **“VPC Network Endpoint”** column in the **“Regional Endpoint Table (Alibaba Cloud VPC Network Connection Method)”** to configure `mc.endpoint`.

   For users accessing via the public network, you can choose from the **“Classic Network Endpoint”** column in the **“Regional Endpoint Table (Alibaba Cloud Classic Network Connection Method)”**, or

the **“External Network Endpoint”** column in the **“Regional Endpoint Table (External Network Connection Method)”** to configure `mc.endpoint`.

Custom Service Address (For Doris Versions before 2.1.7)

In Doris versions before 2.1.7, the **Tunnel SDK** is used to interact with MaxCompute. Therefore, the following two endpoint properties need to be configured:

- `mc.odps_endpoint`: MaxCompute Endpoint, used to retrieve MaxCompute metadata (e.g., database and table information).
- `mc.tunnel_endpoint`: Tunnel Endpoint, used to read MaxCompute data.

By default, the MaxCompute Catalog generates endpoints based on the values of `mc.region` and `mc.public_access`.

The generated endpoint formats are as follows:

mc.public_access	mc.odps_endpoint	mc.tunnel_endpoint
false	http://service.{mc.region}.maxcompute.aliyun-inc.com/api	http://dt.{mc.region}.maxcompute.aliyun-inc.com
true	http://service.{mc.region}.maxcompute.aliyun.com/api	http://dt.{mc.region}.maxcompute.aliyun.com

Users can also manually specify `mc.odps_endpoint` and `mc.tunnel_endpoint` to customize the service addresses. This is particularly useful for private deployments of MaxCompute environments.

For details on configuring MaxCompute Endpoint and Tunnel Endpoint, refer to the documentation on Endpoints for Different Regions and Network Connection Methods.

Resource Usage Control

Users can adjust the `parallel_pipeline_task_num` and `num_scanner_threads` session variables to control the table-level request concurrency and manage resource consumption in the data transmission service. The corresponding concurrency equals `max(parallel_pipeline_task_num * be num * num_scanner_threads)`.

Note:

1. This method can only control the concurrent request count for a single table in a single query, and cannot control resource usage across multiple SQL statements.
2. Reducing the concurrency will increase the query time.

Delta Lake Catalog

Delta Lake Catalog uses the Trino Connector compatibility framework to access Delta Lake tables through the Delta Lake Connector.

> **This feature is experimental and has been supported since version 3.0.1.**

Application Scenarios

Scenario	Description
Data Integration	Read Delta Lake data and write it into Doris internal tables.
Data Writeback	Not supported.

Environment Preparation

Compile the Delta Lake Connector Plugin

> JDK 17 is required.

```
$ git clone https://github.com/apache/doris-thirdparty.git
$ cd doris-thirdparty
$ git checkout trino-435
$ cd plugin/trino-delta-lake
$ mvn clean install -DskipTests
$ cd ../../lib/trino-hdfs
$ mvn clean install -DskipTests
```

After compiling, you will find the `trino-delta-lake-435` directory under `trino/plugin/trino-delta-lake/target/` and the `hdfs` directory under `trino/lib/trino-hdfs/target/`.

You can also directly download the precompiled trino-delta-lake-435-20240724.tar.gz and hdfs.tar.gz, then extract them.

Deploy the Delta Lake Connector

Place the `trino-delta-lake-435/` directory in the `connectors/` directory of all FE and BE deployment paths(If it does not exist, you can create it manually) and extract `hdfs.tar.gz` into the `trino-delta-lake-435/` directory.

```
├── bin
├── conf
├── connectors
│   ├── trino-delta-lake-435
```

```
|   |   ├── hdfs
...
```

After deployment, it is recommended to restart the FE and BE nodes to ensure the Connector is loaded correctly.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name
PROPERTIES (
    'type' = 'trino-connector', -- required
    'trino.connector.name' = 'delta_lake', -- required
    {TrinoProperties},
    {CommonProperties}
);
```

- `{TrinoProperties}`

  The TrinoProperties section is used to specify properties that will be passed to the Trino Connector. These properties use the `trino.` prefix. In theory, all properties supported by Trino are also supported here. For more information about Delta Lake, refer to the Trino documentation.

- `[CommonProperties]`

  The CommonProperties section is used to specify general properties. Please refer to the Catalog Overview under the “Common Properties” section.

Supported Delta Lake Versions

For more information about Delta Lake, refer to the Trino documentation.

Supported Metadata Services

For more information about Delta Lake, refer to the Trino documentation.

Supported Storage Systems

For more information about Delta Lake, refer to the Trino documentation.

Column Type Mapping

Delta Lake Type	Trino Type	Doris Type	Comment
boolean	boolean	boolean	
int	int	int	
byte	tinyint	tinyint	
short	smallint	smallint	

Delta Lake Type	Trino Type	Doris Type	Comment
long	bigint	bigint	
float	real	float	
double	double	double	
decimal(P, S)	decimal(P, S)	decimal(P, S)	
string	varchar	string	
bianry	varbinary	string	
date	date	date	
timestamp_ntz	timestamp(N)	datetime(N)	
timestamp	timestamp with time zone(N)	datetime(N)	
array	array	array	
map	map	map	
struct	row	struct	

Examples

```
CREATE CATALOG delta_lake_hms properties (
    'type' = 'trino-connector',
    'trino.connector.name' = 'delta_lake',
    'trino.hive.metastore' = 'thrift',
    'trino.hive.metastore.uri'= 'thrift://ip:port',
    'trino.hive.config.resources'='/path/to/core-site.xml,/path/to/hdfs-site.xml'
);
```

Query Operations

After configuring the Catalog, you can query the table data in the Catalog using the following methods:

```
-- 1. Switch to the catalog, use the database, and query
SWITCH delta_lake_ctl;
USE delta_lake_db;
SELECT * FROM delta_lake_tbl LIMIT 10;

-- 2. Use the Delta Lake database directly
USE delta_lake_ctl.delta_lake_db;
SELECT * FROM delta_lake_tbl LIMIT 10;

-- 3. Use the fully qualified name to query
SELECT * FROM delta_lake_ctl.delta_lake_db.delta_lake_tbl LIMIT 10;
```

BigQuery Catalog

BigQuery Catalog uses the Trino Connector compatibility framework to access BigQuery tables through the BigQuery Connector.

> **This feature is experimental and has been supported since version 3.0.1.**

Application Scenarios

Scenario	Description
Data Integration	Read BigQuery data and write it into Doris internal tables.
Data Writeback	Not supported.

Environment Preparation

Compile the BigQuery Connector Plugin

> JDK 17 is required.

```
$ git clone https://github.com/apache/doris-thirdparty.git
$ cd doris-thirdparty
$ git checkout trino-435
$ cd plugin/trino-bigquery
$ mvn clean install -DskipTests
```

After compilation, the `trino/plugin/trino-bigquery/target/` directory will contain the `trino-bigquery-435` folder.

You can also directly download the precompiled trino-bigquery-435-20240724.tar.gz and extract it.

Deploy the BigQuery Connector

Place the `trino-bigquery-435/` directory into the `connectors/` directory of the deployment paths for all FE and BE nodes. (If the directory does not exist, you can create it manually.)

```
├── bin
├── conf
├── connectors
│   └── trino-bigquery-435
...
```

After deployment, it is recommended to restart the FE and BE nodes to ensure the Connector is loaded correctly.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name
PROPERTIES (
    'type' = 'trino-connector', -- required
    'trino.connector.name' = 'bigquery', -- required
    {TrinoProperties},
    {CommonProperties}
);
```

- `{TrinoProperties}`

  The TrinoProperties section is used to specify properties that will be passed to the Trino Connector. These properties use the `trino.` prefix. In theory, all properties supported by Trino are also supported here. For more information about BigQuery, refer to the Trino documentation.

- `[CommonProperties]`

  The CommonProperties section is used to specify general properties. Please refer to the Catalog Overview under the “Common Properties” section.

Supported BigQuery Versions

For more information about BigQuery properties, refer to the Trino documentation.

Column Type Mapping

BigQuery Type	Trino Type	Doris Type
boolean	boolean	boolean
int64	bigint	bigint
float64	double	double
numeric	decimal(P, S)	decimal(P, S)
bignumric	decimal(P, S)	decimal(P, S)
string	varchar	string
bytes	varbinary	string
date	date	date
datetime	timestamp(6)	datetime(6)
time	time(6)	string
timestamp	timestamp with time zone(6)	datetime(6)
geography	varchar	string
array	array	array
map	map	map

BigQuery Type	Trino Type	Doris Type
struct	row	struct

Examples

```
CREATE CATALOG bigquery_catalog PROPERTIES (
    'type' = 'trino-connector',
    'trino.connector.name' = 'bigquery',
    'trino.bigquery.project-id' = 'your-bigquery-project-id',
    'trino.bigquery.credentials-file' = '/path/to/application_default_credentials.json',
);
```

Query Operations

After configuring the Catalog, you can query the table data in the Catalog using the following methods:

```
-- 1. switch to catalog, use database and query
SWITCH bigquery_ctl;
USE bigquery_db;
SELECT * FROM bigquery_tbl LIMIT 10;

-- 2. use bigquery database directly
USE bigquery_ctl.bigquery_db;
SELECT * FROM bigquery_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM bigquery_ctl.bigquery_db.bigquery_tbl LIMIT 10;
```

Kudu Catalog

Kudu Catalog uses the Trino Connector compatibility framework to access Kudu tables through the Kudu Connector.

> **This feature is experimental and has been supported since version 3.0.1.**

Application Scenarios

Scenario	Description
Data Integration	Read Kudu data and write it into Doris internal tables.
Data Writeback	Not supported.

Environment Preparation

Compile the Kudu Connector Plugin

> JDK 17 is required.

```
$ git clone https://github.com/apache/doris-thirdparty.git
$ cd doris-thirdparty
$ git checkout trino-435
$ cd plugin/trino-kudu
$ mvn clean package -Dmaven.test.skip=true
```

After compilation, the `trino/plugin/trino-kudu/target/` directory will contain the `trino-kudu-435` folder.

You can also directly download the precompiled trino-kudu-435-20240724.tar.gz and extract it.

Deploy the Kudu Connector

Place the `trino-kudu-435/` directory into the `connectors/` directory of the deployment paths for all FE and BE nodes. (If the directory does not exist, you can create it manually.)

```
├─── bin
├─── conf
├─── connectors
│    ├─── trino-kudu-435
...
```

After deployment, it is recommended to restart the FE and BE nodes to ensure the Connector is loaded correctly.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name PROPERTIES (
    'type' = 'trino-connector', -- required
    'trino.connector.name' = 'kudu', -- required
    {TrinoProperties},
    {CommonProperties}
);
```

- `{TrinoProperties}`

  The `TrinoProperties` section is used to specify properties that will be passed to the Trino Connector. These properties are prefixed with `trino.`. In theory, all properties supported by Trino are also supported here. For more information about Kudu properties, refer to the Trino documentation.

- `[CommonProperties]`

  The CommonProperties section is used to specify general properties. Please refer to the Catalog Overview under the “Common Properties” section.

Supported Kudu Versions

For more information about Kudu, refer to the Trino documentation.

Supported Metadata Services

For more information about Kudu, refer to the Trino documentation.

Supported Storage Systems

For more information about Kudu, refer to the Trino documentation.

Column Type Mapping

Kudu Type	Trino Type	Doris Type	Comment
boolean	boolean	boolean	
int8	tinyint	tinyint	
int16	smallint	smallint	
int32	integer	int	
int64	bigint	bigint	
float	real	float	
double	double	double	
decimal(P, S)	decimal(P, S)	decimal(P, S)	

Kudu Type	Trino Type	Doris Type	Comment
binary	varbinary	string	Requires using `HEX(col)` in queries to return results displayed the same as in Trino.
string	varchar	string	
date	date	date	
unixtime_micros	timestamp(3)	datetime(3)	
other		UNSUPPORTED	

Examples

```
CREATE CATALOG kudu_catalog PROPERTIES (
    'type' = 'trino-connector',
    'trino.connector.name' = 'kudu',
    'trino.kudu.client.master-addresses' = 'ip1:port1,ip2:port2,ip3,port3',
    'trino.kudu.authentication.type' = 'NONE'
);
```

Query Operations

After configuring the Catalog, you can query the table data in the Catalog using the following methods:

```
-- 1. switch to catalog, use database and query
SWITCH kudu_ctl;
USE kudu_db;
SELECT * FROM kudu_tbl LIMIT 10;

-- 2. use kudu database directly
USE kudu_ctl.kudu_db;
SELECT * FROM kudu_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM kudu_ctl.kudu_db.kudu_tbl LIMIT 10;
```

Elasticsearch Catalog

Elasticsearch Catalog not only supports automatic mapping of ES metadata, but also combines Doris's distributed query planning capabilities with ES (Elasticsearch)'s full-text search capabilities to provide a more comprehensive OLAP analysis solution:

1. Multi-index distributed Join queries in ES.
2. Joint queries between tables in Doris and ES, with more complex full-text search filtering.

Prerequisites

Supports Elasticsearch 5.x and above.

Connecting to Elasticsearch

```
CREATE CATALOG es_catalog PROPERTIES (
    'type' = 'es',
    'hosts' = 'http://127.0.0.1:9200'
    {ElasticsearchProperties}
);
```

- {ElasticsearchProperties}

Parameter	Required	Default	Description
hosts	Yes		ES address, can be one or multiple, or ES load balancer address
user	No	Empty	ES username
password	No	Empty	Password for the corresponding user
doc_value_scan	No	true	Whether to enable getting query field values through ES/Lucene columnar storage
keyword_sniff	No	true	Whether to detect text.fields for string tokenization types in ES and query through keyword. Setting to false will match based on tokenized content
nodes_discovery	No	true	Whether to enable ES node discovery, default is true. Set to false in network isolated environments to connect only to specified nodes
ssl	No	false	Whether ES enables https access mode, currently implemented in fe/be as trust all
mapping_es_id	No	false	Whether to map the _id field in ES index
like_push_down	No	true	Whether to convert like to wildcard and push down to ES, will increase ES CPU consumption
include_hidden_index	No	false	Whether to include hidden indexes, default is false

Notes:

1. Authentication currently only supports HTTP Basic authentication, and you need to ensure that the user has access to paths like `/_cluster/state/`, `_nodes/http` and Index read permissions; username and password are not required when cluster security authentication is not enabled.
2. In 5.x and 6.x versions, the first type is taken by default for multiple types in one Index.

Hierarchy Mapping

Since Elasticsearch doesn't have the concept of Database, after connecting to ES, a unique Database will be automatically generated: `default_db`.

After switching to ES Catalog using the SWITCH command, it will automatically switch to `default_db`, no need to execute `USE default_db` command.

Column Type Mapping

ES Type	Doris Type	Comment
null	null	
boolean	boolean	
byte	tinyint	
short	smallint	
integer	int	
long	bigint	
unsigned_long	largeint	
float	float	
half_float	float	
double	double	
scaled_float	double	
date	date	Only supports `default/yyyy-MM-dd HH:mm:ss/yyyy-MM-dd/epoch_millis` formats
keyword	string	
text	string	
ip	string	
constant_keyword	string	
wildcard	string	
nested	json	
object	json	
flattened	json	Supported since version 3.1.4, 4.0.3

ES Type	Doris Type	Comment
other	UNSUPPORTED	

Array Types

Elasticsearch doesn't have an explicit array type, but its fields can contain 0 or more values.

To indicate that a field is an array type, you can add a specific `doris` structure annotation in the _meta section of the index mapping.

For Elasticsearch 6.x and earlier versions, please refer to _meta.

For example, suppose there's an index `doc` containing the following data structure:

```
{
  "array_int_field": [1, 2, 3, 4],
  "array_string_field": ["doris", "is", "the", "best"],
  "id_field": "id-xxx-xxx",
  "timestamp_field": "2022-11-12T12:08:56Z",
  "array_object_field": [
    {
      "name": "xxx",
      "age": 18
    }
  ]
}
```

The array fields of this structure can be defined by adding field property definitions to the `_meta.doris` property of the target index mapping using the following commands.

```
# ES 7.x and above
curl -X PUT "localhost:9200/doc/_mapping?pretty" -H 'Content-Type:application/json' -d '
{
    "_meta": {
        "doris":{
            "array_fields":[
                "array_int_field",
                "array_string_field",
                "array_object_field"
            ]
        }
    }
}'

# ES 6.x and before
curl -X PUT "localhost:9200/doc/_mapping/_doc?pretty" -H 'Content-Type: application/json' -d '
{
    "_meta": {
        "doris":{
            "array_fields":[
                "array_int_field",
                "array_string_field",
                "array_object_field"
            ]
        }
```

```
    }
}
```

`array_fields`: Used to indicate fields that are array types.

flattened Type

For `flattened` type, when the `enable_docvalue_scan` property is `false`, the JSON data format read out is flattened. If the `enable_docvalue_scan` property is `true`, then the original JSON format is read out. Example as follows:

Index definition:

```
"mappings": {
   "properties": {
      "column1": {
      "type": "keyword"
      },
      "extra": {
      "type": "flattened"
      }
   }
}
```

Data:

```
{
  "column1": 1,
  "extra": {
    "subcol1": "abc",
    "sub_array": [
      {"k1": "element1"},
      {"k2": "element2"},
      {"k3": "element3"}
    ]
  }
}
```

- When `enable_docvalue_scan` property is `false`

  The query result for `extra` column is:

  ```
  {
     "subcol1": "abc",
     "sub_array": [
        {"k1": "element1"},
        {"k2": "element2"},
        {"k3": "element3"}
     ]
  }
  ```

- When `enable_docvalue_scan` property is `true`

  The query result for `extra` column is:

  ```
  ["abc","element1","element2","element3"]
  ```

Query Operations

After configuring the Catalog, you can query table data in the Catalog through the following methods:

```
-- 1. switch to catalog, use database and query
SWITCH es_ctl;
SELECT * FROM es_tbl LIMIT 10;

-- 2. use default_db directly
USE es_ctl.default_db;
SELECT * FROM es_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM es_ctl.default_db.es_tbl LIMIT 10;
```

For more advanced query features, please refer to the Best Practices section.

Best Practices

Filter Predicate Pushdown

ES Catalog supports filter predicate pushdown: filter conditions are pushed down to ES, so only data that truly meets the conditions will be returned, which can significantly improve query performance and reduce CPU, Memory, IO usage of Doris and Elasticsearch.

The following operators will be optimized into the corresponding ES Query:

SQL syntax	ES 5.x+ syntax
`=`	term query
`in`	terms query
`> , < , >= , <=`	range query
`and`	bool.filter
`or`	bool.should
`not`	bool.must_not
`not in`	bool.must_not + terms query
`is_not_null`	exists query
`is_null`	bool.must_not + exists query
`esquery`	ES native json form QueryDSL

Enable Columnar Scan to Optimize Query Speed (enable_docvalue_scan=true)

Set `"enable_docvalue_scan" = "true"`

After enabling, Doris will follow these two principles when getting data from ES:

- **Best Effort**: Automatically detect whether the fields to be read have columnar storage enabled (doc_value: true). If all fields have columnar storage, Doris will get all field values from columnar

storage.

- **Automatic Degradation**: If any field to be retrieved doesn't have columnar storage, all field values will be parsed and retrieved from row storage `_source`.

Advantages:

By default, Doris On ES will get all required columns from row storage, i.e., `_source`. `_source` storage uses row-based + JSON format, which has inferior batch reading performance compared to columnar storage, especially when only a few columns are needed. In cases where only a few columns are needed, docvalue performance is about ten times better than _source performance.

Notes:

1. `text` type fields don't have columnar storage in ES, so if any field value to be retrieved is of `text` type, it will automatically degrade to getting from `_source`.
2. When the number of fields to retrieve is too many (`>= 25`), the performance of getting field values from `docvalue` will be basically the same as getting field values from `_source`.
3. `keyword` type fields may appear empty due to the `ignore_above` parameter limitation for long text fields that exceed this limit. In this case, you need to disable `enable_docvalue_scan` and get results from `_source`.

Detect Keyword Type Fields

Set `"enable_keyword_sniff" = "true"`

In ES, you can import data directly without establishing an index. In this case, ES will automatically create a new index. For string type fields, ES will create fields that have both `text` type and `keyword` type, which is ES's multi fields feature, with mapping as follows:

```
"k4": {
   "type": "text",
   "fields": {
      "keyword": {
         "type": "keyword",
         "ignore_above": 256
      }
   }
}
```

When filtering on k4, such as =, Doris On ES will convert the query to ES's TermQuery.

SQL filter condition:

```
k4 = "Doris On ES"
```

Converted to ES query DSL:

```
"term" : {
    "k4": "Doris On ES"
```

```
}
```

Because k4's first field type is `text`, during data import it will be tokenized according to k4's configured tokenizer (if not set, it's the standard tokenizer) to get three Terms: doris, on, es, as analyzed by ES analyze API:

```
POST /_analyze
{
  "analyzer": "standard",
  "text": "Doris On ES"
}
```

The tokenization result is:

```
{
   "tokens": [
      {
         "token": "doris",
         "start_offset": 0,
         "end_offset": 5,
         "type": "<ALPHANUM>",
         "position": 0
      },
      {
         "token": "on",
         "start_offset": 6,
         "end_offset": 8,
         "type": "<ALPHANUM>",
         "position": 1
      },
      {
         "token": "es",
         "start_offset": 9,
         "end_offset": 11,
         "type": "<ALPHANUM>",
         "position": 2
      }
   ]
}
```

The query uses:

```
"term" : {
    "k4": "Doris On ES"
}
```

The term `Doris On ES` cannot match any term in the dictionary and won't return any results. Enabling `enable_keyword_sniff: true` will automatically convert `k4 = "Doris On ES"` to `k4.keyword = "Doris On ES"` to completely match SQL semantics. The converted ES query DSL is:

```
"term" : {
    "k4.keyword": "Doris On ES"
}
```

`k4.keyword` type is `keyword`, and data is written to ES as a complete term, so it can match.

Enable Node Auto Discovery (nodes_discovery=true)

Set `"nodes_discovery" = "true"`

When configured as true, Doris will find all available related data nodes (allocated shards above) from ES. If ES data node addresses are not accessible by Doris BE, set to false. ES cluster is deployed in an intranet isolated from the public Internet, and users access through proxy.

Whether ES Cluster Enables HTTPS Access Mode

Set `"ssl" = "true"`

Currently FE/BE implementation is trust all, this is a temporary solution, real user configured certificates will be used later.

Extended esquery() Function

Use `esquery(field, QueryDSL)` function to push down some queries that cannot be expressed in SQL, such as `match_phrase`, `geoshape`, etc., to ES for filtering. The first column name parameter of `esquery` is used to associate `index`, and the second parameter is the JSON representation of ES's basic `Query DSL`, enclosed in curly braces `{}`. The JSON `root key` must have one and only one, such as `match_phrase`, `geo_shape`, `bool`, etc.

`match_phrase` query:

```
select * from es_table where esquery(k4, '{
        "match_phrase": {
           "k4": "doris on es"
        }
    }');
```

`geo` related queries:

```
select * from es_table where esquery(k4, '{
      "geo_shape": {
         "location": {
            "shape": {
               "type": "envelope",
               "coordinates": [
                  [
                     13,
                     53
                  ],
                  [
                     14,
                     52
                  ]
               ]
            },
            "relation": "within"
         }
      }
   }');
```

`bool` query:

```
select * from es_table where esquery(k4, ' {
         "bool": {
            "must": [
               {
                  "terms": {
                     "k1": [
                        11,
                        12
                     ]
                  }
               },
               {
                  "terms": {
                     "k2": [
                        100
                     ]
                  }
               }
            ]
         }
      }');
```

Time Type Field Usage Recommendations

Only applicable to ES external tables. In ES Catalog, date types are automatically mapped to Date or Datetime.

In ES, time type fields are very flexible to use, but in ES external tables, if time type fields are not properly configured, filter conditions cannot be pushed down.

When creating an index, set time type format for maximum format compatibility:

```
 "dt": {
     "type": "date",
     "format": "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis"
 }
```

When establishing this field in Doris, it's recommended to set it as `date` or `datetime`, or it can be set as `varchar` type. Using the following SQL statements can directly push filter conditions down to ES:

```
select * from doe where k2 > '2020-06-21';

select * from doe where k2 < '2020-06-21 12:00:00';

select * from doe where k2 < 1593497011;

select * from doe where k2 < now();

select * from doe where k2 < date_format(now(), '%Y-%m-%d');
```

Note:

- If you don't set `format` for time type fields in ES, the default time type field format is:

  ```
  strict_date_optional_time||epoch_millis
  ```

- If date fields imported to ES are timestamps, they need to be converted to `ms`. ES processes timestamps internally in `ms`, otherwise ES external tables will have display errors.

Getting ES Metadata Field ID

When importing documents without specifying `_id`, ES will assign a globally unique `_id` (primary key) to each document. Users can also specify a `_id` with special business meaning when importing.

If you need to get this field value in ES external tables, you can add a `_id` field of `varchar` type when creating the table:

```
CREATE EXTERNAL TABLE `doe` (
  `_id` varchar COMMENT "",
  `city`  varchar COMMENT ""
) ENGINE=ELASTICSEARCH
PROPERTIES (
"hosts" = "http://127.0.0.1:8200",
"user" = "root",
"password" = "root",
"index" = "doe"
}
```

If you need to get this field value in ES Catalog, please set `"mapping_es_id" = "true"`.

Note:

1. Filter conditions for `_id` field only support `=` and `in`.
2. `_id` field must be of `varchar` type.

Getting Globally Ordered Query Results

In scenarios like relevance ranking and priority display of important content, it's very useful for ES query results to be sorted by score. Doris queries ES to fully utilize the advantages of MPP architecture by pulling data according to the shard distribution of ES indexes. To get globally ordered sorting results, ES needs to be queried from a single point. This can be controlled by the session variable `enable_es_parallel_scroll` (default is true). When `enable_es_parallel_scroll=false` is set, Doris will send `scroll` queries without `shard_preference` and `sort` information to the ES cluster, thus getting globally ordered results. Note: Use with caution when the query result set is large.

Modifying scroll Request batch Size

The default `batch` size for `scroll` requests is 4064. It can be modified through the session variable `batch_size`.

Frequently Asked Questions

1. **Does it support ES clusters with X-Pack authentication?**

   Supports all ES clusters using HTTP Basic authentication.

2. **Some queries are much slower than requesting ES directly**

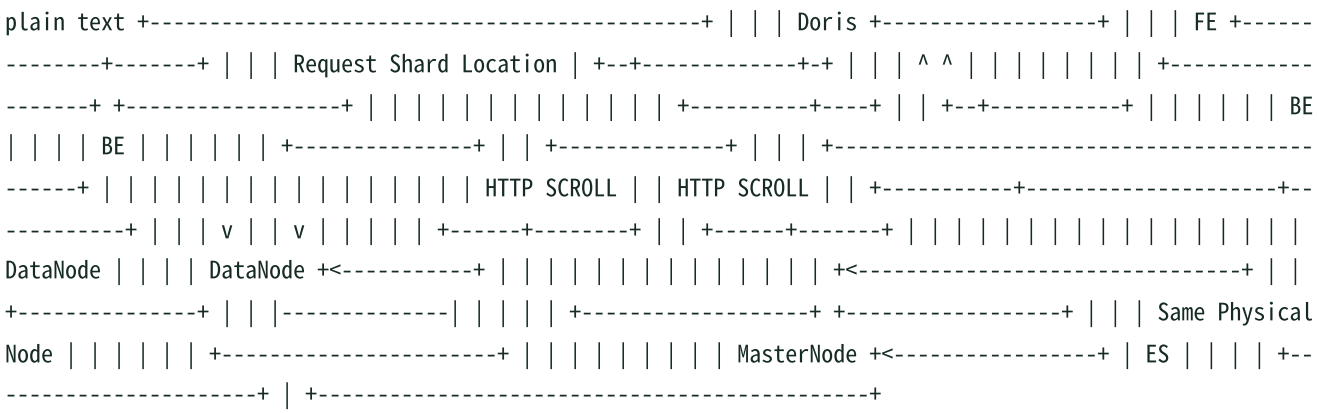
Yes, such as _count related queries, etc. ES will directly read metadata related to the number of documents that meet the conditions internally, without filtering actual data.

3. **Can aggregation operations be pushed down?**

   Currently Doris On ES doesn't support pushing down aggregation operations like sum, avg, min/max, etc. The calculation method is to batch stream all documents meeting the conditions from ES, then calculate in Doris.

Appendix

Principle of Doris Querying ES



1. FE will request the hosts specified when creating the table to get HTTP port information of all nodes and shard distribution information of indexes, etc. If the request fails, it will traverse the host list sequentially until success or complete failure.

2. When querying, a query plan will be generated and sent to corresponding BE nodes based on some node information and index metadata information obtained by FE.

3. BE nodes will request ES nodes deployed locally first according to the `proximity principle`. BE gets data concurrently from each shard of ES index through `HTTP Scroll` method streaming from `_source` or `docvalue`.

4. After Doris completes the calculation, it returns the results to the user.

LakeSoul Catalog

> **Note!**
>
> This feature is deprecated since 3.1.0. For usage inquiries, please contact the developers.

Doris supports accessing and reading LakeSoul table data using metadata stored in PostgreSQL.

Quickly Experience Apache Doris & LakeSoul with Docker

Applicable Scenarios

Scenario	Description
Data Integration	Read LakeSoul data and write it to Doris internal tables, or perform ZeroETL operations using the Doris computing engine.
Data Write-back	Not supported.

Configuring Catalog

Syntax

```
CREATE CATALOG lakesoul_catalog PROPERTIES (
    'type' = 'lakesoul',
    {LakeSoulProperties},
    {CommonProperties}
);
```

- `{LakeSoulProperties}`

Property	Description	Example
`lakesoul.pg.username`	Username for the PG source database	
`lakesoul.pg.password`	Password for the PG source database	
`lakesoul.pg.url`	JDBC URL for the PG metadata database	`jdbc:postgresql://127.0.0.1:5432/lakesoul_test?stringtype=unspecified`

- `[CommonProperties]`

> The CommonProperties section is for filling in general properties. Please refer to the Data Catalog Overview under the “Common Properties” section.

If LakeSoul data is stored on HDFS, place `core-site.xml`, `hdfs-site.xml`, and `hive-site.xml` in the `conf/` directory of both FE and BE. The Hadoop configuration files in the `conf/` directory are read first, followed by the configuration files specified by the `HADOOP_CONF_DIR` environment variable.

Supported LakeSoul Versions

The currently supported LakeSoul version is 2.6.2.

Supported LakeSoul Formats

- Supports LakeSoul primary key and non-primary key tables.
- Supports reading LakeSoul MOR (Merge-On-Read) tables.

Column Type Mapping

LakeSoul Type	Doris Type	Comment
boolean	boolean	
int8	tinyint	
int16	smallint	
int32	int	
int64	bigint	
float	float	
double	double	
decimal(P, S)	decimal(P, S)	
string	string	
date	date	
timestamp(S)	datetime(S)	
list	array	
map	map	
row	struct	
other	UNSUPPORTED	

Examples

```
CREATE CATALOG lakesoul PROPERTIES (
    'type' = 'lakesoul',
    'lakesoul.pg.username' = 'lakesoul_test',
    'lakesoul.pg.password' = 'lakesoul_test',
```

```
    'lakesoul.pg.url' = 'jdbc:postgresql://127.0.0.1:5432/lakesoul_test?stringtype=unspecified'
);
```

Query Operations

Basic Query

Once the Catalog is configured, you can query the table data in the Catalog as follows:

```
-- 1. switch to catalog, use database and query
SWITCH ls_ctl;
USE ls_db;
SELECT * FROM ls_tbl LIMIT 10;

-- 2. use lakesoul database directly
USE ls_ctl.ls_db;
SELECT * FROM ls_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM ls_ctl.ls_db.ls_tbl LIMIT 10;
```

JDBC Catalog

JDBC Catalog supports connecting to databases that are compatible with the JDBC protocol through a standard JDBC interface.

This document introduces the general configuration and usage of the JDBC Catalog. For different JDBC sources, please refer to the respective documentation.

> **Note**
>
> The JDBC Catalog feature in Doris relies on the Java layer to read and process data, and its overall performance can be affected by the JDK version. Some internal libraries in older versions of the JDK (such as JDK 8) are less efficient and may lead to higher resource consumption. If higher performance is required, it is recommended to use Doris 3.0, which is compiled with JDK 17 by default and offers better overall performance.

Applicable Scenarios

The JDBC Catalog is only suitable for data integration, such as importing small amounts of data from a data source into Doris or performing join queries on small tables in a JDBC data source. The JDBC Catalog cannot accelerate queries on the data source or access large amounts of data at once.

Supported Databases

Doris JDBC Catalog supports connections to the following databases:

Supported Data Sources
MySQL
PostgreSQL
Oracle
SQL Server
IBM DB2
ClickHouse
SAP HANA
Oceanbase

You can refer to the Developer Guide to develop support for new, unsupported JDBC data sources.

Configuring Catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] catalog_name PROPERTIES (
    'type' = 'jdbc', -- required
    {JdbcProperties},
    {CommonProperties}
);
```

- {JdbcProperties}
    - Required Properties

Parameter Name	Description	Example
user	Data source username	
password	Data source password	
jdbc_url	Data source connection URL	jdbc:mysql://host:3306
driver_url	Path to the JDBC driver file. For driver package security, see the appendix.	Supports three methods, see below.
driver_class	Class name of the JDBC driver	

driver_url supports the following three specifications:

1. Filename. For example, mysql-connector-j-8.3.0.jar. The Jar file must be pre-placed in the jdbc_drivers/ directory under the FE and BE deployment directories. The system will automatically search in this directory. The location can also be modified by the jdbc_drivers_dir configuration in fe.conf and be.conf.
2. Local absolute path. For example, file:///path/to/mysql-connector-j-8.3.0.jar. The Jar file must be pre-placed in the specified path on all FE/BE nodes.
3. HTTP URL. For example: https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.3.0/mysql-connector-j-8.3.0.jar. The system will download the driver file from this HTTP address. Only supports HTTP services without authentication.

    - Optional Properties

Parameter Name	Default Value	Description
lower_case_meta_names	false	Whether to synchronize the database, table, and column names from the external data source in lowercase
meta_names_mapping		When the external data source has names that differ only in case, such as MY_TABLE and my_table, Doris will report an error due to ambiguity when querying the Catalog. You need to configure the meta_names_mapping parameter to resolve conflicts.
only_specified_database	false	Whether to synchronize only the database specified in the jdbc_url (this Database maps to the Database level in Doris)

Parameter Name	Default Value	Description
`connection_pool_min_size`	1	Defines the minimum number of connections in the connection pool, used to initialize the pool and ensure at least this number of active connections when keep-alive is enabled.
`connection_pool_max_size`	30	Defines the maximum number of connections in the connection pool. Each FE or BE node corresponding to each Catalog can hold up to this number of connections.
`connection_pool_max_wait_time`	5000	Defines the maximum wait time in milliseconds for a client to wait for a connection if none are available in the pool.
`connection_pool_max_life_time`	1800000	Sets the maximum active duration (in milliseconds) for a connection in the pool. Connections exceeding this time will be recycled. Additionally, half of this value is used as the minimum eviction idle time for the pool, making connections reaching this time eligible for eviction.
`connection_pool_keep_alive`	false	Effective only on BE nodes, it determines whether to keep connections active that have reached the minimum eviction idle time but not the maximum lifetime. Disabled by default to reduce unnecessary resource usage.

- `[CommonProperties]`

  The CommonProperties section is used to configure common properties. Please refer to the Catalog Overview section on **Common Properties**.

Query Operations

Basic Query

```
-- 1. switch to catalog, use database and query
SWITCH mysql_ctl;
USE mysql_db;
SELECT * FROM mysql_tbl LIMIT 10;

-- 2. use mysql database directly
USE mysql_ctl.mysql_db;
SELECT * FROM mysql_tbl LIMIT 10;

-- 3. use full qualified name to query
SELECT * FROM mysql_ctl.mysql_db.mysql_tbl LIMIT 10;
```

Query Optimization

Predicate Pushdown

When the JDBC Catalog accesses a data source, it essentially selects a BE node, uses the JDBC Client to send a generated SQL query to the source, and retrieves the data. Therefore, the performance depends solely on how efficiently the generated SQL executes on the source side. Doris will attempt to push down predicate conditions and incorporate them into the generated SQL. You can use the `EXPLAIN SQL` statement to view the generated SQL.

```
EXPLAIN SELECT smallint_u, sum(int_u)
FROM all_types WHERE smallint_u > 10 GROUP BY smallint_u;
```

```
...
|   0:VJdbcScanNode(206)                                                                                  |
|      TABLE: `doris_test`.`all_types`                                                                    |
|      QUERY: SELECT `smallint_u`, `int_u` FROM `doris_test`.`all_types` WHERE ((`smallint_u` > 10)) |
|      PREDICATES: (smallint_u[#1] > 10)                                                                  |
|      final projections: smallint_u[#1], int_u[#3]                                                       |
|      final project output tuple id: 1
...                                                  |
```

Function Pushdown

For predicate conditions, the semantics or behavior in Doris and external data sources may be inconsistent. Therefore, Doris restricts and controls predicate pushdown in JDBC external table queries through the following parameter variables:

> Note: Currently, Doris only supports predicate pushdown for MySQL, Clickhouse, and Oracle data sources. More data sources will be supported in the future.

- `enable_jdbc_oracle_null_predicate_push_down`

  Session variable. The default is `false`. That is, when the predicate condition contains a `NULL` value, the predicate will not be pushed down to the Oracle data source. This is because, before Oracle version 21, Oracle does not support `NULL` as an operator.

  This parameter is supported since 2.1.7 and 3.0.3.

- `enable_jdbc_cast_predicate_push_down`

  Session variable. The default is `false`. That is, when there is an explicit or implicit CAST in the predicate condition, the predicate will not be pushed down to the JDBC data source. Since the behavior of CAST is inconsistent across different databases, to ensure correctness, CAST is not pushed down by default. However, users can manually verify whether the behavior of CAST is consistent. If so, this parameter can be set to `true` to allow more predicates to be pushed down for better performance.

  This parameter is supported since 2.1.7 and 3.0.3.

- Function pushdown blacklist and whitelist

  Functions with the same signature may have inconsistent semantics in Doris and external data sources. Doris has predefined some blacklists and whitelists for function pushdown:

Data Source	Blacklist	Whitelist	Description
MySQL	- `DATE_TRUNC` - `MONEY_FORMAT` - `NEGTIVE`		MySQL can also set additional blacklist items through the FE configuration item `jdbc_mysql_unsupported_pushdown_functions`, e.g.: `jdbc_mysql_unsupported_pushdown_functions==func1,func2`

Data Source	Blacklist	Whitelist	Description
Clickhouse		- FROM_UNIXTIME - UNIX_TIMESTAMP	
Oracle		- NVL - IFNULL	

- Function rewrite rules

  There are some functions in Doris and external data sources that have consistent behavior but different names. Doris supports rewriting these functions during function pushdown. The following rewrite rules are currently built-in:

Data Source	Doris Function	Target Function
MySQL	nvl	ifnull
MySQL	to_date	date
Clickhouse	from_unixtime	FROM_UNIXTIME
Clickhouse	unix_timestamp	toUnixTimestamp
Oracle	ifnull	nvl

- Custom function pushdown and rewrite rules

  In subsequent versions after 3.0.7, Doris supports more flexible function pushdown and rewrite rules. Users can set function pushdown and rewrite rules for a specific Catalog in the Catalog properties:

```
create catalog jdbc properties (
...
'function_rules' = '{"pushdown" : {"supported": ["to_date"], "unsupported" : ["abs"]}, "rewrite" : {"to_date" : "date2"}}'
)
```

  Through `function_rules`, the following rules can be specified:

  - `pushdown`

    Specifies function pushdown rules. The `supported` and `unsupported` arrays specify the function names that can and cannot be pushed down, respectively. If a function exists in both arrays, `supported` takes precedence.

    Doris will first apply the system's predefined blacklists and whitelists, and then apply the user-specified blacklists and whitelists.

  - `rewrite`

    Defines function rewrite rules. As in the example above, the function name `to_date` will be rewritten as `date2` and pushed down.

Note: Only functions that are allowed to be pushed down will be rewritten.

Row Count Limitation

If a query includes the `LIMIT` keyword, Doris will push the `LIMIT` clause down to the data source to reduce data transfer volume. You can use the `EXPLAIN` statement to confirm whether the generated SQL includes the `LIMIT` clause.

Write Operations

Doris supports writing data back to the corresponding data source via the JDBC protocol.

```
INSERT INTO mysql_table SELECT * FROM internal.doris_db.doris_tbl;
```

Statement Passthrough

Applicable Scenarios

Doris supports directly executing corresponding DDL, DML, and query statements in the JDBC data source through statement passthrough. This feature is applicable in the following scenarios:

- Improving Complex Query Performance

  By default, the Doris query optimizer parses the original SQL and generates the SQL to be sent to the data source based on certain rules. This generated SQL is typically a simple single-table query and cannot include operators like aggregation or join queries. For example, consider the following query:

  ```
  SELECT smallint_u, sum(int_u)
  FROM all_types WHERE smallint_u > 10 GROUP BY smallint_u;
  ```

  The final generated SQL would be:

  ```
  SELECT smallint_u, int_u
  FROM all_types
  WHERE smallint_u > 10;
  ```

  In this case, the aggregation operation is performed within Doris. Therefore, in some scenarios, a large amount of data may need to be read from the source over the network, resulting in low query efficiency. With statement passthrough, the original SQL can be directly passed to the data source, leveraging the data source's own computation capabilities to execute the SQL, thereby improving query performance.

- Unified Management

  In addition to query SQL, the statement passthrough feature can also pass DDL and DML statements. This allows users to perform database and table operations on the source data directly through Doris, such as creating, deleting tables, or modifying table structures.

Passthrough SQL

```
SELECT * FROM
QUERY(
  'catalog' = 'mysql_catalog',
  'query' = 'SELECT smallint_u, sum(int_u) FROM db.all_types WHERE smallint_u > 10 GROUP BY smallint_u;'
);
```

The `QUERY` table function takes two parameters:

- `catalog`: The name of the catalog, which should match the name of the catalog.
- `query`: The query statement to execute, written directly using the syntax of the corresponding data source.

Passthrough DDL and DML

```
CALL EXECUTE_STMT("jdbc_catalog", "insert into db1.tbl1 values(1,2), (3, 4)");

CALL EXECUTE_STMT("jdbc_catalog", "delete from db1.tbl1 where k1 = 2");

CALL EXECUTE_STMT("jdbc_catalog", "create table dbl1.tbl2 (k1 int)");
```

The `EXECUTE_STMT()` function takes two parameters:

- First parameter: The name of the catalog. Currently, only JDBC-type catalogs are supported.
- Second parameter: The SQL statement to execute. Currently, only DDL and DML statements are supported, and they must be written using the syntax of the corresponding data source.

Usage Restrictions

When using the `CALL EXECUTE_STMT()` command, Doris directly sends the user-written SQL statement to the JDBC data source associated with the catalog for execution. As a result, this operation has the following restrictions:

- The SQL statement must follow the syntax of the corresponding data source. Doris does not perform syntax or semantic checks.
- Table names referenced in the SQL statement should preferably use fully qualified names, such as `db.tbl`. If the database is not specified, the database name in the JDBC URL of the JDBC catalog will be used.
- The SQL statement cannot reference databases or tables outside the JDBC data source, nor can it reference Doris's databases or tables. However, it can reference tables within the JDBC data source that are not synchronized with the Doris JDBC catalog.
- When executing DML statements, the number of rows inserted, updated, or deleted cannot be retrieved; only the success or failure of the command can be determined.
- Only users with `LOAD` permission on the catalog can execute the `CALL EXECUTE_STMT()` command.
- Only users with `SELECT` permission on the catalog can execute the `query()` table function.

- The JDBC user specified when creating the catalog must have the necessary permissions on the source to execute the corresponding statements.
- The data types of the results read by the `query()` table function are consistent with the data types supported by the queried catalog type.

Appendix

Case Sensitivity Settings

By default, database and table names in Doris are case-sensitive, while column names are case-insensitive. This behavior can be modified through configuration parameters. Additionally, the case sensitivity rules for database, table, and column names in some JDBC data sources may differ from those in Doris. This discrepancy can cause naming conflicts during name mapping via the JDBC Catalog. The following section explains how to resolve such issues.

Display Name and Query Name

In Doris, an object name (here we use table names as an example) can be divided into a **Display Name** and a **Query Name**. For instance, for a table name, the **Display Name** refers to the name shown in the result of `SHOW TABLES`, while the **Query Name** refers to the name that can be used in a `SELECT` statement.

For example, if the actual name of a table is `MyTable`, the **Display Name** and **Query Name** of this table may differ depending on the configuration of the Frontend (FE) parameter `lower_case_table_names`:

Configuration	Description	Actual Name	Display Name	Query Name
`lower_case_table_names=0`	Default configuration. The original name is stored and displayed, and queries are case-sensitive.	`MyTable`	`MyTable`	Case-sensitive in queries, must use: `MyTable`
`lower_case_table_names=1`	The name is stored and displayed in lowercase, and queries are case-insensitive.	`MyTable`	`mytable`	Case-insensitive in queries, e.g., you can use `MyTable` or `mytable`.
`lower_case_table_names=2`	The original name is stored and displayed, but queries are case-insensitive.	`MyTable`	`MyTable`	Case-insensitive in queries, e.g., you can use `MyTable` or `mytable`.

JDBC Catalog Name Case Sensitivity Rules

Doris itself only allows configuration of the case sensitivity rules for **table names**. However, JDBC Catalog requires additional handling for **database names** and **column names**. Therefore, we use an additional Catalog property, `lower_case_meta_names`, to work in conjunction with `lower_case_table_names`.

Configuration	Description
`lower_case_meta_names`	Specified via `properties` when creating a Catalog, and it applies only to that Catalog. The default value is `false`. When set to `true`, Doris converts all database, table, and column names to lowercase for storage and display. Queries must use the lowercase names in Doris.
`lower_case_table_names`	A Frontend (FE) configuration item, configured in `fe.conf`, and it applies to the entire cluster. The default value is `0`.

> Note: If `lower_case_meta_names = true`, the `lower_case_table_names` configuration will be ignored, and all database, table, and column names will be converted to lowercase.

Based on the combinations of `lower_case_meta_names` (true/false) and `lower_case_table_names` (0/1/2), the behavior of database, table, and column names during **storage** and **queries** is shown in the following table ("Original" means retaining the case from the external data source, "Lowercase" means automatically converting to lowercase, and "Any Case" means queries can use any case):

`lower_case_table_names` & `lower_case_meta_names`	Database Display Name	Table Display Name	Column Display Name	Database Query Name	Table Query Name	Column Query Name
`0 & false`	Original	Original	Original	Original	Original	Any Case
`0 & true`	Lowercase	Lowercase	Lowercase	Lowercase	Lowercase	Any Case
`1 & false`	Original	Lowercase	Original	Original	Any Case	Any Case
`1 & true`	Lowercase	Lowercase	Lowercase	Lowercase	Any Case	Any Case
`2 & false`	Original	Original	Original	Original	Any Case	Any Case
`2 & true`	Lowercase	Lowercase	Lowercase	Lowercase	Any Case	Any Case

Case Conflict Check

When performing name mapping through a JDBC Catalog, naming conflicts may occur. For example, if the source column names are case-sensitive and there are two columns, `ID` and `id`. If `lower_case_meta_names = true` is set, these two columns will conflict after being converted to lowercase. Doris performs conflict checks according to the following rules:

- In any scenario, Doris will check for **column name** case conflicts (e.g., whether `id` and `ID` exist simultaneously).
- When `lower_case_meta_names = true`, Doris will check for case conflicts in database names, table names, and column names (e.g., whether `DORIS` and `doris` exist simultaneously).
- When `lower_case_meta_names = false` and `lower_case_table_names` is set to `1` or `2`, Doris will check for **table name** conflicts (e.g., whether `orders` and `ORDERS` exist simultaneously).
- When `lower_case_table_names = 0`, database and table names are case-sensitive, and no additional conversion is required.

Solutions to Case Conflicts

When conflicts occur, Doris will throw an error, and the conflicts must be resolved using the following approach.

For cases where databases, tables, or columns with only case differences (e.g., `DORIS` and `doris`) cause Doris to be unable to distinguish them properly, you can resolve the conflict by setting `meta_names_mapping` for the Catalog to specify manual mappings.

Examples

```
{
  "databases": [
    {
      "remoteDatabase": "DORIS",
      "mapping": "doris_1"
    },
    {
      "remoteDatabase": "doris",
      "mapping": "doris_2"
    }
  ],
  "tables": [
    {
      "remoteDatabase": "DORIS",
      "remoteTable": "DORIS",
      "mapping": "doris_1"
    },
    {
      "remoteDatabase": "DORIS",
      "remoteTable": "doris",
      "mapping": "doris_2"
    }
  ],
  "columns": [
    {
      "remoteDatabase": "DORIS",
      "remoteTable": "DORIS",
      "remoteColumn": "DORIS",
      "mapping": "doris_1"
    },
    {
      "remoteDatabase": "DORIS",
      "remoteTable": "DORIS",
      "remoteColumn": "doris",
      "mapping": "doris_2"
    }
  ]
}
```

Driver Package Security

Driver packages are uploaded by users to the Doris cluster, which poses certain security risks. Users can enhance security through the following measures:

1. Doris considers all driver packages in the `jdbc_drivers_dir` directory to be secure and does not perform path checks on them. Administrators must manage the files in this directory themselves to ensure their security.

2. If a driver package is specified using a local path or an HTTP path, Doris performs the following checks:

    - The allowed driver package paths are controlled via the FE configuration item `jdbc_driver_secure_path`. This configuration can include multiple paths, separated by semicolons. When this configuration is set, Doris checks whether the `driver_url` path prefix is included in `jdbc_driver_secure_path`. If it is not included, the creation will be denied.

      Example:

      If `jdbc_driver_secure_path = "file:///path/to/jdbc_drivers;http://path/to/jdbc_drivers"` is configured, only driver package paths starting with `file:///path/to/jdbc_drivers` or `http://path/to/jdbc_drivers` are allowed.

    - This parameter defaults to *. If it is empty or set to *, all Jar package paths are allowed.

3. When creating a data directory, you can specify the checksum of the driver package using the `checksum` parameter. After loading the driver package, Doris will validate the checksum, and if the validation fails, the creation will be denied.

Connection Pool Cleanup

In Doris, each FE and BE node maintains a connection pool to avoid the frequent opening and closing of individual data source connections. Each connection in the pool can be used to establish a connection with the data source and execute queries. After the task is completed, these connections are returned to the pool for reuse, which not only improves performance but also reduces the system overhead of establishing connections and helps prevent reaching the connection limit of the data source.

The size of the connection pool can be adjusted according to actual needs to better accommodate different workloads. Typically, the minimum number of connections in the pool should be set to 1 to ensure that at least one connection remains active when the keep-alive mechanism is enabled. The maximum number of connections should be set to a reasonable value to avoid excessive resource consumption.

To prevent the accumulation of unused connection pool caches on the BE, you can set the `jdbc_connection_pool_cache_clear_time_sec` parameter on the BE to specify the cache cleanup interval. The default value is 28,800 seconds (8 hours), after which the BE will forcibly clear all connection pool caches that have not been used within this time.

Credential Update

When using the JDBC Catalog to connect to external data sources, it is important to carefully update database credentials.

Doris maintains active connections through the connection pool to respond quickly to queries. However, after changing credentials, the connection pool might continue using the old credentials to attempt new connections and fail. Since the system tries to maintain a certain number of active connections, these erroneous attempts will be repeated, and in some database systems, frequent failures might lead to account lockout.

It is recommended to update the Doris JDBC Catalog configuration in sync when credentials must be changed and restart the Doris cluster to ensure all nodes use the latest credentials, preventing connection failures and potential account lockouts.

Possible account lockouts include:

```
MySQL: account is locked

Oracle: ORA-28000: the account is locked

SQL Server: Login is locked out
```

Connection Pool Troubleshooting

1. HikariPool Connection Timeout Error: `Connection is not available, request timed out after 5000ms`
    - Possible Causes
        - Cause 1: Network issues (e.g., server unreachable)
        - Cause 2: Authentication issues, such as invalid username or password
        - Cause 3: High network latency causing connection creation to exceed the 5-second timeout
        - Cause 4: Too many concurrent queries exceeding the maximum connections configured in the pool
    - Solutions
        - If you only encounter the error `Connection is not available, request timed out after 5000ms`, check Causes 3 and 4:
            - Check for high network latency or resource exhaustion.
            - Increase the maximum number of connections in the pool:

              ```
              ALTER CATALOG catalog_name SET PROPERTIES ('connection_pool_max_size' = '100');
              ```

            - Increase the connection timeout:

              ```
              ALTER CATALOG catalog_name SET PROPERTIES ('connection_pool_max_wait_time' = '10000');
              ```

        - If there are additional error messages besides `Connection is not available, request timed out after 5000ms`, check these additional errors:
            - Network issues (e.g., server unreachable) may cause connection failures. Ensure the network connection is stable.
            - Authentication issues (e.g., invalid username or password) may also cause connection failures. Verify the database credentials in the configuration to ensure the username and password are correct.
            - Investigate issues related to network, database, or authentication based on the specific error messages to identify the root cause.

MySQL JDBC Catalog

Doris JDBC Catalog supports connecting to MySQL databases via the standard JDBC interface. This document describes how to configure MySQL database connections.

For an overview of JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to a MySQL database, you need:

- MySQL 5.7, 8.0, or higher.
- The JDBC driver for MySQL, which you can download from the Maven Repository. It is recommended to use MySQL Connector/J version 8.0.31 or above.
- Network connection between each Doris FE and BE node and the MySQL server, with the default port being 3306.

Connecting to MySQL

```
CREATE CATALOG mysql_catalog PROPERTIES (
    'type' = 'jdbc',
    'user' = 'username',
    'password' = 'pwd',
    'jdbc_url' = 'jdbc:mysql://host:3306',
    'driver_url' = 'mysql-connector-j-8.3.0.jar',
    'driver_class' = 'com.mysql.cj.jdbc.Driver'
);
```

The `jdbc_url` defines the connection information and parameters to be passed to the MySQL JDBC driver. Supported URL parameters can be found in the MySQL Developer Guide.

Connection Security

If the user has configured TLS with a globally trusted certificate installed on the data source, TLS can be enabled between the cluster and the data source by appending parameters to the JDBC connection string set in the `jdbc_url` property.

For example, for `MySQL Connector/J 8.0`, use the `sslMode` parameter to secure the connection via TLS. By default, this parameter is set to PREFERRED, which can secure the connection if the server is enabled. This parameter can also be set to REQUIRED, which will cause the connection to fail if TLS is not established.

It can be configured by adding the sslMode parameter in the `jdbc_url`:

```
'jdbc_url' = 'jdbc:mysql://host:3306/?sslMode=REQUIRED'
```

For more information on TLS configuration options, please refer to the MySQL JDBC Security Documentation.

Hierarchical Mapping

When mapping MySQL, a Database in Doris corresponds to a Database in MySQL. A Table under a Database in Doris corresponds to Tables under that Database in MySQL. The mapping relationship is as follows:

Doris	MySQL
Catalog	MySQL Server
Database	Database
Table	Table

Column Type Mapping

MySQL Type	Doris Type	Comment
boolean	tinyint	
tinyint	tinyint	
smallint	smallint	
mediumint	int	
int	int	
bigint	bigint	
unsigned tinyint	smallint	Doris does not support unsigned data types, so unsigned data types are mapped to the corresponding larger data type in Doris.
unsigned mediumint	int	Same as above.
unsigned int	bigint	Same as above.
unsigned bigint	largeint	Same as above.
float	float	
double	double	
decimal(P, S)	decimal(P, S)	
unsigned decimal(P, S)	decimal(P + 1, S) / string	If it exceeds the maximum precision supported by Doris, it will be handled by String. Note that when this type is mapped to String, it only supports queries and cannot write to MySQL.
date	date	
timestamp(S)	datetime(S)	
datetime(S)	datetime(S)	
year	smallint	Doris does not support the year type, so the year type is mapped to smallint.
time	string	Doris does not support the time type, so the time type is mapped to string.
char	char	
varchar	varchar	

MySQL Type	Doris Type	Comment
json	string	For better balance between reading and computing performance, Doris maps the json type to the string type.
set	string	
enum	string	
bit	boolean / string	Doris does not support the bit type, so the bit type is mapped to boolean when bit(1), and to string in other cases.
tinytext, text, mediumtext, longtext	string	
blob, mediumblob, longblob, tinyblob, binary, varbinary	string/varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
other	UNSUPPORTED	

Appendix

Time Zone Issues

When accessing data through JDBC Catalog, the JNI part of BE uses the JVM time zone. The JVM time zone defaults to the time zone of the BE deployment machine, which affects time zone conversion when JDBC reads data. To ensure time zone consistency, it is recommended to set the JVM time zone in `be.conf`'s `JAVA_OPTS` to be consistent with the Doris session variable `time_zone`.

When reading MySQL's timestamp type, add the parameters `connectionTimeZone=LOCAL` and `forceConnectionTimeZoneToSession=true` in the JDBC URL. These parameters are applicable to MySQL Connector/J version 8 and above, ensuring that the read time is the Doris BE JVM time zone, not the MySQL server's time zone.

Common Issues

Connection Exception Troubleshooting

- Communications link failure The last packet successfully received from the server was 7 milliseconds ago.
    - Cause:
        - Network issues:
            - Unstable network or connection interruption.
            - High network latency between the client and server.
        - MySQL server settings
            - The MySQL server may have configured connection timeout parameters, such as `wait_timeout` or `interactive_timeout`, causing the connection to be closed due to

timeout.

        - Firewall settings
            - Firewall rules may block communication between the client and server.
        - Connection pool settings
            - The configuration `connection_pool_max_life_time` in the connection pool may cause the connection to be closed or recycled, or not kept alive in time.
        - Server resource issues
            - The MySQL server may lack resources to handle new connection requests.
        - Client configuration
            - Incorrect client JDBC driver configuration, such as the `autoReconnect` parameter not set or set improperly.
    - Solution
        - Check network connection:
            - Ensure stable network connection between the client and server, avoiding high network latency.
        - Check MySQL server configuration:
            - Review and adjust the MySQL server's `wait_timeout` and `interactive_timeout` parameters to ensure they are set reasonably.
        - Check firewall configuration:
            - Ensure firewall rules allow communication between the client and server.
        - Adjust connection pool settings:
            - Review and adjust the connection pool's configuration parameters `connection_pool_max_life_time`, ensuring it is less than MySQL's `wait_timeout` and `interactive_timeout` parameters and greater than the longest execution time of SQL.
        - Monitor server resources:
            - Monitor the MySQL server's resource usage to ensure there are enough resources to handle connection requests.
        - Optimize client configuration:
            - Ensure the JDBC driver's configuration parameters are correct, such as `autoReconnect=true`, to ensure the connection can automatically reconnect after interruption.

- java.io.EOFException MESSAGE: Can not read response from server. Expected to read 819 bytes, read 686 bytes before connection was unexpectedly lost.

- Cause: Connection was killed by MySQL or MySQL crashed
- Solution: Check if MySQL has a mechanism to actively kill connections, or if a large query crashed MySQL

Other Issues

1. Garbled characters when reading and writing MySQL emoji

   When Doris queries MySQL, the default utf8 encoding in MySQL is utf8mb3, which cannot represent emoji that require 4-byte encoding. Here, you need to change MySQL's encoding to utf8mb4 to support 4-byte encoding.

   You can globally modify the configuration item

   ```
   Modify the my.ini file in the mysql directory (for Linux systems, it is the my.cnf file in the etc directory)
   [client]
   default-character-set=utf8mb4

   [mysql]
   Set mysql default character set
   default-character-set=utf8mb4

   [mysqld]
   Set mysql character set server
   character-set-server=utf8mb4
   collation-server=utf8mb4_unicode_ci
   init_connect='SET NAMES utf8mb4

   Modify the type of the corresponding table and column
   ALTER TABLE table_name MODIFY colum_name VARCHAR(100) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
   ALTER TABLE table_name CHARSET=utf8mb4;
   SET NAMES utf8mb4
   ```

2. Exception when reading MySQL DATE/DATETIME type

   ```
   ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.6)[INTERNAL_ERROR]UdfRuntimeException: get next block failed:
   CAUSED BY: SQLException: Zero date value prohibited
   CAUSED BY: DataReadException: Zero date value prohibited
   ```

   In JDBC, illegal DATE/DATETIME is handled by default by throwing an exception. The behavior can be controlled by the URL parameter `zeroDateTimeBehavior`. The optional parameters are: `exception`, `convertToNull`, `round`, which are: exception error; convert to `NULL` value; convert to `"0001-01-01 00:00:00"`

   You need to add `zeroDateTimeBehavior=convertToNull` to the JDBC connection string at the end of the `jdbc_url` when creating the Catalog, such as `"jdbc_url" = "jdbc:mysql://127.0.0.1:3306/test?zeroDateTimeBehavior=convertToNull"`. In this case, JDBC will convert 0000-00-00 or 0000-00-00 00:00:00 to null, and then Doris will handle all Date/DateTime type columns of the current Catalog as nullable types, so they can be read normally.

3. When reading MySQL Catalog or other JDBC Catalog, class loading failure occurs, such as `failed to load driver class com.mysql.cj.jdbc.driver in either of hikariconfig class loader`

This is because the `driver_class` filled in when creating the Catalog is incorrect and needs to be filled in correctly, such as the example above is a case issue, it should be filled in as `'driver_class' = 'com.mysql.cj.jdbc.Driver'`

4. Communication link exception when reading MySQL

```
ERROR 1105 (HY000): errCode = 2, detailMessage = PoolInitializationException: Failed to initialize pool: Communications link failure

The last packet successfully received from the server was 7 milliseconds ago.  The last packet sent successfully to the server was 4 milliseconds ago.
CAUSED BY: CommunicationsException: Communications link failure

The last packet successfully received from the server was 7 milliseconds ago.  The last packet sent successfully to the server was 4 milliseconds ago.
CAUSED BY: SSLHandshakeException
```

You can check the be.out log of be, if it contains the following information:

```
WARN: Establishing SSL connection without server's identity verification is not recommended.
According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't set.
For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'.
You need either to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
```

You can add `useSSL=false` in the `jdbc_url`, such as `'jdbc_url' = 'jdbc:mysql://127.0.0.1:3306/test?useSSL=false'`.

- When querying large amounts of data from MySQL, if the query occasionally succeeds and occasionally reports the following error, and when this error occurs, all connections to MySQL are disconnected and cannot connect to the MySQL Server, but after a while, MySQL returns to normal, but the previous connections are gone:

  ```
  ERROR 1105 (HY000): errCode = 2, detailMessage = [INTERNAL_ERROR]UdfRuntimeException: JDBC executor sql has error:
  CAUSED BY: CommunicationsException: Communications link failure
  The last packet successfully received from the server was 4,446 milliseconds ago. The last packet sent successfully to the server was 4,446 milliseconds ago.
  ```

  When the above phenomenon occurs, it may be that the MySQL Server's own memory or CPU resources are exhausted, causing the MySQL service to be unavailable. You can try to increase the memory or CPU configuration of the MySQL Server.

- During the process of querying MySQL, if you find that the query results are inconsistent with those in the MySQL database

  First, check whether there are case-sensitive situations in the query fields. For example, if there are two pieces of data "aaa" and "AAA" in a field `c_1` in the Table, if the MySQL database is not specified to distinguish case when initialized, MySQL does not distinguish case by default, but Doris strictly distinguishes case, so the following situation will occur:

  ```
  MySQL behavior:
  select count(c_1) from table where c_1 = "aaa"; Does not distinguish case, so the result is: 2
  ```

```
Doris behavior:
select count(c_1) from table where c_1 = "aaa"; Strictly distinguishes case, so the result is: 1
```

If the above phenomenon occurs, you need to adjust according to the requirements, as follows:

- Add the “BINARY” keyword in MySQL queries to force case sensitivity: `select count(c_1) from table where BINARY c_1 = "aaa";`
- Or specify when creating a table in MySQL: `CREATE TABLE table (c_1 VARCHAR(255) CHARACTER SET binary);`
- Or specify collation rules to distinguish case when initializing the MySQL database:

  ```text
  [mysqld]
  character-set-server=utf8
  collation-server=utf8_bin
  [client]
  default-character-set=utf8
  [mysql]
  default-character-set=utf8
  ```

- When querying MySQL, if it gets stuck for a long time without returning results, or gets stuck for a long time and a large number of write lock logs appear in fe.warn.log.

  You can try adding socketTimeout to the URL, for example: `jdbc:mysql://host:port/database?socketTimeout=30000`, to prevent the JDBC client from waiting indefinitely after being closed by MySQL.

- During the use of MySQL Catalog, if you find that the JVM memory or Threads number of FE continues to grow without decreasing, and may simultaneously report Forward to master connection timed out error

  Print the FE thread stack `jstack fe_pid > fe.js`, if a large number of `mysql-cj-abandoned-connection-cleanup` threads appear, it indicates a problem with the MySQL JDBC driver.

  Handle it as follows:

  - Upgrade the MySQL JDBC driver to version 8.0.31 or above
  - Add the parameter `-Dcom.mysql.cj.disableAbandonedConnectionCleanup=true` in the `JAVA_OPTS` of the FE and BE conf files to disable the connection cleanup function of the MySQL JDBC driver, and restart the cluster

    Note: If the version of Doris is 2.0.13 or above, or 2.1.5 or above, there is no need to add this parameter, because Doris has already disabled the connection cleanup function of the MySQL JDBC driver by default. Just replace the MySQL JDBC driver version. However, you need to restart the Doris cluster to clean up the previously leaked threads.

PostgreSQL JDBC Catalog

Doris JDBC Catalog supports connecting to PostgreSQL databases via the standard JDBC interface. This document describes how to configure a PostgreSQL database connection.

For an overview of JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to a PostgreSQL database, you need

- PostgreSQL 11.x or higher
- JDBC driver for PostgreSQL database, which you can download the latest or specified version from Maven Repository. It is recommended to use PostgreSQL JDBC Driver version 42.5.x or above.
- Network connection between each FE and BE node of Doris and the PostgreSQL server, with the default port being 5432.

Connecting to PostgreSQL

```
CREATE CATALOG postgresql_catalog PROPERTIES (
    'type' = 'jdbc',
    'user' = 'username',
    'password' = 'pwd',
    'jdbc_url' = 'jdbc:postgresql://host:5432/postgres',
    'driver_url' = 'postgresql-42.5.6.jar',
    'driver_class' = 'org.postgresql.Driver'
);
```

`jdbc_url` defines the connection information and parameters to be passed to the PostgreSQL JDBC driver. Supported URL parameters can be found in the PostgreSQL JDBC Driver Documentation.

Connection Security

If you have configured TLS with a globally trusted certificate installed on the data source, you can enable TLS between the cluster and the data source by appending parameters to the JDBC connection string set in the jdbc_url property.

For example, for version 42 of the PostgreSQL JDBC driver, enable TLS by adding the ssl=true parameter to the jdbc_url configuration property:

```
"jdbc_url"="jdbc:postgresql://example.net:5432/database?ssl=true"
```

For more information on TLS configuration options, please refer to the PostgreSQL JDBC Driver Documentation.

Hierarchical Mapping

When mapping PostgreSQL, a Database in Doris corresponds to a Schema under a specified database in PostgreSQL (as in the example `jdbc_url` parameter under `postgres`). A Table under a Database in Doris corresponds to Tables under that Schema in PostgreSQL. The mapping relationship is as follows:

Doris	PostgreSQL
Catalog	Database
Database	Schema
Table	Table

Column Type Mapping

PostgreSQL Type	Doris Type	
boolean	boolean	
smallint/int2	smallint	
integer/int4	int	
bigint/int8	bigint	
decimal/numeric	decimal(P, S) / string	Numeric without precision will be mapped to string type, and needs to be converted to decimal type for numerical calculations, and does not support write-back.
real/float4	float	
double	double	
smallserial	smallint	
serial	int	
bigserial	bigint	
char(N)	char(N)	
varchar/text	string	
timestamp(S)/timestampz(S)	datetime(S)	
date	date	
json/jsonb	string	For better reading and computing performance balance, Doris maps JSON type to STRING type.
time	string	Doris does not support time type, time type will be mapped to string.
interval	string	
point/line/lseg/box/path/polygon/circle	string	
cidr/inet/macaddr	string	
uuid	string	
bit	boolean / string	Doris does not support bit type, bit type will be mapped to boolean when bit(1), otherwise mapped to string.

PostgreSQL Type	Doris Type	
bytea	varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
array	array	Please refer to the following explanation regarding the mapping methods for array types.
other	UNSUPPORTED	

- Array Types

  In PostgreSQL, array types can be defined as follows:

  ```
  col1 text[]
  col2 in4[][]
  ```

  However, the array dimensions cannot be directly obtained from the PostgreSQL metadata. For example, `text[]` could be a one-dimensional array or a two-dimensional array. The array dimensions can only be determined after data is written.

  Doris requires the array dimensions to be explicitly declared. Therefore, Doris can only map correctly if the corresponding array column in PostgreSQL contains data; otherwise, the array column will be mapped as `UNSUPPORTED`.

Appendix

Time Zone Issues

Since Doris does not support timestamp types with time zones, when reading the timestampz type from PostgreSQL, Doris will map it to DATETIME type and convert it to local time zone time when reading.

And because when reading data from JDBC type Catalog, the Java part of BE uses the JVM time zone. The JVM time zone defaults to the time zone of the BE deployment machine, which affects the time zone conversion when JDBC reads data.

To ensure time zone consistency, it is recommended to set the JVM time zone in `JAVA_OPTS` of `be.conf` to be consistent with the `time_zone` of Doris session.

Oracle JDBC Catalog

Apache Doris JDBC Catalog supports connecting to Oracle databases via the standard JDBC interface. This document describes how to configure Oracle database connections.

For an overview of JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to an Oracle database, you need

- Oracle 19c, 18c, 12c, 11g, or 10g.
- The JDBC driver for Oracle databases, which you can download from the Maven Repository for Ojdbc8 and above versions of the Oracle JDBC driver.
- Network connection between each FE and BE node of Apache Doris and the Oracle server, with the default port being 1521; if Oracle RAC is ONS-enabled, port 6200 is also required.

Connecting to Oracle

```
CREATE CATALOG oracle_catalog PROPERTIES (
    'type' = 'jdbc',
    'user' = 'username',
    'password'='pwd',
    'jdbc_url' = 'jdbc:oracle:thin:@example.net:1521:orcl',
    'driver_url' = 'ojdbc8.jar',
    'driver_class' = 'oracle.jdbc.driver.OracleDriver'
)
```

`jdbc_url` defines the connection information and parameters to be passed to the JDBC driver. When using the Oracle JDBC Thin driver, the syntax of the URL may vary depending on your Oracle configuration. For example, if you are connecting to an Oracle SID or Oracle service name, the connection URL will differ. For more information, please refer to the Oracle Database JDBC Driver Documentation. The above example URL connects to an Oracle SID named `orcl`.

Hierarchical Mapping

When mapping Oracle, a Database in Apache Doris corresponds to a User in Oracle. And a Table under a Database in Apache Doris corresponds to a Table accessible by that User in Oracle. The mapping relationship is as follows:

Doris	Oracle
Catalog	Database
Database	User
Table	Table

Column Type Mapping

Oracle Type	Doris Type	Comment
number(P) / number(P, 0)	tinyint/smallint/int/bigint/largeint	Doris will choose the corresponding type based on the size of P: P < 3: TINYINT; P < 5: SMALLINT; P < 10: INT; P < 19: BIGINT; P > 19: LARGEINT
number(P, S), if (S > 0 && P > S)	decimal(P, S)	
number(P, S), if (S > 0 && P < S)	decimal(S, S)	
number(P, S), if (S < 0)	tinyint/smallint/int/bigint/largeint	In the case of S < 0, Doris will set P to `P + |S|`, and perform the same mapping as `number(P) / number(P, 0)`
number		Doris currently does not support the number type without specified P and S
decimal(P, S)	decimal(P, S)	
float/real	double	
date	date	
timestamp	datetime(S)	
char/nchar	string	
varchar2/nvarchar2	string	
long/raw/long raw/internal	string	
BLOB	varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
other	UNSUPPORTED	

Common Issues

1. `ONS configuration failed` occurs when creating or querying Oracle Catalog

   Add -Doracle.jdbc.fanEnabled=false to JAVA_OPTS in be.conf and upgrade the driver to https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc8/19.23.0.0/ojdbc8-19.23.0.0.jar

2. `Non supported character set (add orai18n.jar in your classpath): ZHS16GBK` exception occurs when creating or querying Oracle Catalog

   Download orai18n.jar and place it in the `custom_lib/` directory under each FE and BE (create manually if it does not exist) and restart each FE and BE.

SQL Server JDBC Catalog

Doris JDBC Catalog supports connecting to SQL Server databases through the standard JDBC interface. This document describes how to configure the SQL Server database connection.

For an overview of JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to a SQL Server database, you need

- SQL Server 2012 or later, or Azure SQL Database.
- JDBC driver for SQL Server database, which you can download the latest or specified version from Maven Repository. It is recommended to use SQL Server JDBC Driver 11.2.x and above.
- Network connection between each FE and BE node of Doris and the SQL Server server, with the default port being 1433.

Connecting to SQL Server

```
CREATE CATALOG sqlserver_catalog PROPERTIES (
    'type' = 'jdbc',
    'user' = 'username',
    'password' = 'pwddd',
    'jdbc_url' = 'jdbc:sqlserver://<host>:<port>;databaseName=<databaseName>;encrypt=false',
    'driver_url' = 'mssql-jdbc-11.2.3.jre8.jar',
    'driver_class' = 'com.microsoft.sqlserver.jdbc.SQLServerDriver'
)
```

`jdbc_url` defines the connection information and parameters to be passed to the SQL Server JDBC driver. The parameters supported by the URL are provided in the SQL Server JDBC Driver Documentation.

Connection Security

The JDBC driver and connector automatically use Transport Layer Security (TLS) encryption and certificate verification. This requires configuring appropriate TLS certificates on the SQL Server database host.

If you have not established the necessary configuration, you can disable encryption in the connection string using the encrypt property:

```
'jdbc_url' = 'jdbc:sqlserver://<host>:<port>;databaseName=<databaseName>;encrypt=false'
```

The TLS section of the SQL Server JDBC Driver Documentation details other parameters such as `trustServerCertificate`, `hostNameInCertificate`, `trustStore`, and `trustStorePassword`.

Hierarchical Mapping

When mapping SQLServer, a Database in Doris corresponds to a Schema under the specified Database in SQL Server (`<databaseName>` parameter in `jdbc_url`). A Table under a Database in Doris corresponds to Tables under a Schema in SQLServer. The mapping relationship is as follows:

Doris	SQL Server
Catalog	Database
Database	Schema
Table	Table

Column Type Mapping

SQL Server Type	Doris Type	Comment
bit	boolean	
tinyint	smallint	SQLServer's tinyint is unsigned, so it is mapped to Doris's smallint
smallint	smallint	
int	int	
bigint	bigint	
real	float	
float	double	
money	decimal(19,4)	
smallmoney	decimal(10,4)	
decimal(P, S)/numeric(P, S)	decimal(P, S)	
date	date	
datetime/datetime2/smalldatetime	datetime(S)	
char/varchar/text/nchar/nvarchar/ntext	string	
time/datetimeoffset	string	
timestamp	string	Displays hexadecimal representation of binary data, no actual meaning
image/binary/varbinary	varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
other	UNSUPPORTED	

Common Issues

1. Certificate authentication exception when connecting to SQL Server

```
SQLServerException: The driver could not establish a secure connection to SQL Server by using Secure Sockets Layer (SSL)
encryption.
Error: "sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target". ClientConnectionId:a92f3817-e8e6-4311-bc21-7c66
```

You can add `encrypt=false` to the end of the JDBC connection string when creating the Catalog, such as `"jdbc_url" = "jdbc:sqlserver://127.0.0.1:1433;DataBaseName=doris_test;encrypt=false"`

2. TLS exception when connecting to SQL Server

```
The server selected protocol version TLS10 is not accepted by client preferences [TLS13, TLS12]
```

This is because the TLS protocol version between SQL Server and the JDBC client does not match. The connected SQL Server only supports TLS 1.0, while the JAVA environment where the JDBC client is located has TLS 1.0 disabled by default.

The solution is as follows:

1. Enable TLS 1.2 on SQL Server. Refer to: SQL Server TLS 1.2 Support

2. Enable TLS 1.0 for JDK.

```
vim ${JAVA_HOME}/lib/security/java.security
# Find this section
jdk.tls.disabledAlgorithms=SSLv3, TLSv1, TLSv1.1, RC4, DES, MD5withRSA, \
DH keySize < 1024, EC keySize < 224, 3DES_EDE_CBC, anon, NULL, \
include jdk.disabled.namedCurves

# Remove TLSv1, TLSv1.1, change it to the following
jdk.tls.disabledAlgorithms=SSLv3, RC4, DES, MD5withRSA, \
DH keySize < 1024, EC keySize < 224, anon, NULL, \
include jdk.disabled.namedCurves
```

IBM Db2 JDBC Catalog

The Doris JDBC Catalog supports connecting to IBM Db2 databases through the standard JDBC interface. This document describes how to configure the IBM Db2 database connection.

For an overview of JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to an IBM Db2 database, you need:

- IBM Db2 11.5.x or higher
- The JDBC driver for the IBM Db2 database, which you can download from the Maven Repository for the latest or specified version of the IBM Db2 driver. It is recommended to use IBM db2 jcc version 11.5.8.0.
- Network connection between each Doris FE and BE node and the IBM Db2 server, with the default port being 51000.

Connecting to IBM Db2

```
CREATE CATALOG db2_catalog PROPERTIES (
    'type' = 'jdbc',
    'user' = 'USERNAME',
    'password' = 'PASSWORD',
    'jdbc_url' = 'jdbc:db2://host:port/database',
    'driver_url' = 'jcc-11.5.8.0.jar',
    'driver_class' = 'com.ibm.db2.jcc.DB2Driver'
)
```

The `jdbc_url` defines the connection information and parameters to be passed to the IBM Db2 driver. Supported URL parameters can be found in the Db2 JDBC Driver Documentation.

Hierarchical Mapping

When mapping IBM Db2, Doris's Database corresponds to a Schema under the specified DataBase in DB2 (the "database" parameter in `jdbc_url`). And the Table under Doris's Database corresponds to the Tables under the Schema in DB2. The mapping relationship is as follows:

Doris	IBM Db2
Catalog	DataBase
Database	Schema
Table	Table

Column Type Mapping

IBM Db2 Type	Doris Type	Comment
smallint	smallint	
int	int	
bigint	bigint	
double	double	
double precision	double	
float	double	
real	float	
decimal(P, S)	decimal(P, S)	
decfloat(P, S)	decimal(P, S)	
date	date	
timestamp(S)	datetime(S)	
char(N)	char(N)	
varchar(N)	varchar(N)	
long varchar(N)	varchar(N)	
vargraphic	string	
long vargraphic	string	
time	string	
clob	string	
xml	string	
BLOB	varbinary	Controlled by the `enable.mapping.varbinary` property of Catalog (supported since 4.0.2). The default is `false`, which maps to `string`; when `true`, it maps to `varbinary` type.
other	UNSUPPORTED	

Common Issues

1. When reading IBM Db2 data through JDBC Catalog, an `Invalid operation: result set is closed. ERRORCODE=-4470` exception occurs.

   Add connection parameters to the jdbc_url connection string when creating the IBM Db2 Catalog: `allowNextOnExhaustedResultSet=1;resultSetHoldability=1;`. For example: `jdbc:db2://host:port/database:allowNextOnExhaustedResultSet=1;resultSetHoldability=1;`.

2. Caught java.io.CharConversionException

   This issue may be caused by a character set problem. You can try adding the configuration `-Ddb2.jcc.charsetDecoderEncoder=3` to the `JAVA_OPTS` in `be.conf`, then restart the BE to see if it resolves the issue. You may also experiment with values such as `1` or `2`. For more details, please refer to: https://www.ibm.com/docs/en/content-collector/4.0.1?topic=manager-jdbc-throws-javaiocharconversionexception

Clickhouse JDBC Catalog

The Doris JDBC Catalog supports connecting to the ClickHouse database via the standard JDBC interface. This document describes how to configure the ClickHouse database connection.

For an overview of the JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to the ClickHouse database, you need

- ClickHouse version 23.x or higher (versions below this have not been fully tested).
- The JDBC driver for the ClickHouse database, which you can download from the Maven Repository for the latest or specified version of the ClickHouse JDBC driver. It is recommended to use version 0.4.6 of the ClickHouse JDBC Driver.
- Network connection between each FE and BE node of Doris and the ClickHouse server, with the default port being 8123.

Connecting to ClickHouse

```
CREATE CATALOG clickhouse PROPERTIES (
    'type' = 'jdbc',
    'user' = 'username',
    'password' = 'pwd',
    'jdbc_url' = 'jdbc:clickhouse://example.net:8123/',
    'driver_url' = 'clickhouse-jdbc-0.4.6-all.jar',
    'driver_class' = 'com.clickhouse.jdbc.ClickHouseDriver'
)
```

The `jdbc_url` defines the connection information and parameters to be passed to the ClickHouse JDBC driver. Supported URL parameters can be found in the ClickHouse JDBC Driver Configuration.

Connection Security

If you have configured TLS with a globally trusted certificate installed on the data source, you can enable TLS between the cluster and the data source by appending parameters to the JDBC connection string set in the jdbc_url property.

For example, enable TLS by adding the ssl=true parameter to the jdbc_url configuration property:

```
'jdbc_url' = 'jdbc:clickhouse://example.net:8123/db?ssl=true'
```

For more information on TLS configuration options, please refer to the Clickhouse JDBC Driver Documentation SSL Configuration Section

Hierarchical Mapping

When mapping ClickHouse, a Database in Doris corresponds to a Database in ClickHouse. And a Table under a Database in Doris corresponds to Tables under that Database in ClickHouse. The mapping relationship is as follows:

Doris	ClickHouse
Catalog	ClickHouse Server
Database	Database
Table	Table

Column Type Mapping

ClickHouse Type	Doris Type	Comment
bool	boolean	
string	string	
date/date32	date	
datetime(S)/datetime64(S)	datetime(S)	
float32	float	
float64	double	
int8	tinyint	
int16/uint8	smallint	Doris does not have UNSIGNED data types, so it is scaled up by one magnitude
int32/uInt16	int	Same as above
int64/uint32	bigint	Same as above
int128/uint64	largeint	Same as above
int256/uint128/uint256	string	Doris does not have data types of this magnitude, so it is handled with STRING
decimal(P, S)	decimal(P, S) or string	If it exceeds the maximum precision supported by Doris, use string to handle it
enum/ipv4/ipv6/uuid	string	
array	array	
other	UNSUPPORTED	

Related Parameters

- `jdbc_clickhouse_query_final`

  Session variable, default is false. When set to true, `SETTINGS final = 1` will be appended to the SQL statements sent to Clickhouse.

Common Issues

1. Encountering `NoClassDefFoundError: net/jpountz/lz4/LZ4Factory` error message when reading Clickhouse data

   You can first download the lz4-1.3.0.jar package and place it in the `custom_lib/` directory under each FE and BE directory (if it does not exist, create it manually).

SAP HANA JDBC Catalog

The Doris JDBC Catalog supports connecting to the SAP HANA database through the standard JDBC interface. This document describes how to configure the SAP HANA database connection.

For an overview of the JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to the SAP HANA database, you need

- SAP HANA 2.0 or higher.
- The JDBC driver for the SAP HANA database, which you can download from the Maven Repository. It is recommended to use version ngdbc 2.4.51 or above.
- Network connection between each Doris FE and BE node and the SAP HANA server, with the default port being 30015.

Connecting to SAP HANA

```
CREATE CATALOG saphana_catalog PROPERTIES (
    'type' = 'jdbc',
    'user' = 'username',
    'password' = 'pwd',
    'jdbc_url' = 'jdbc:sap://Hostname:Port/?optionalparameters',
    'driver_url' = 'ngdbc-2.4.51.jar',
    'driver_class' = 'com.sap.db.jdbc.Driver'
)
```

For more information on the JDBC URL format and parameters supported by the SAP HANA JDBC driver, please refer to SAP HANA.

Hierarchical Mapping

When mapping SAP HANA, the Database in Doris corresponds to a Schema under the specified DataBase in SAP HANA (the “DATABASE” in the `jdbc_url` parameter). The Table under the Database in Doris corresponds to the Tables under the Schema in SAP HANA. The mapping relationship is as follows:

Doris	SAP HANA
Catalog	Database
Database	Schema
Table	Table

Column Type Mapping

SAP HANA Type	Doris Type	Comment
boolean	boolean	
tinyint	tinyint	
smallint	smallint	
integer	int	
bigint	bigint	
smalldecimal(P, S)	decimal(P, S) or double or string	If precision is not specified, the double type is used. If the precision exceeds the maximum precision supported by Doris, the string type is used.
decimal(P, S)	decimal(P, S) or double or string	Same as above.
real	float	
double	double	
date	date	
time	string	
timestamp(S)	datetime(S)	
seconddate	datetime(S)	
varchar	string	
nvarchar	string	
alphanum	string	
shorttext	string	
char(N)	char(N)	
nchar(N)	char(N)	
other	UNSUPPORTED	

Oceanbase JDBC Catalog

Doris JDBC Catalog supports connecting to OceanBase database through the standard JDBC interface. This document describes how to configure the OceanBase database connection.

For an overview of JDBC Catalog, please refer to: JDBC Catalog Overview

Usage Notes

To connect to the OceanBase database, you need

- OceanBase 3.1.0 or higher
- JDBC driver for OceanBase database, which you can download the latest or specified version of OceanBase JDBC driver from Maven Repository. It is recommended to use OceanBase Connector/J 2.4.8 or above.
- Network connection between each FE and BE node of Doris and the OceanBase server.

Connect to OceanBase

```
CREATE CATALOG oceanbase_catalog PROPERTIES (
    'type' = 'jdbc',
    'user' = 'username',
    'password' = 'pwd',
    'jdbc_url' = 'jdbc:oceanbase://host:port/db',
    'driver_url' = 'oceanbase-client-2.4.8.jar',
    'driver_class' = 'com.oceanbase.jdbc.Driver'
)
```

`jdbc_url` defines the connection information and parameters to be passed to the OceanBase JDBC driver. Supported URL parameters can be found in OceanBase JDBC Driver Configuration.

Schema Compatibility

When creating an OceanBase Catalog, Doris will automatically recognize whether OceanBase is in MySQL or Oracle mode to correctly parse metadata.

The hierarchy mapping, type mapping, and query optimization in different modes are handled in the same way as the Catalog of MySQL or Oracle databases, and you can refer to the documentation

- MySQL Catalog
- Oracle Catalog

Analyze Files on S3/HDFS

Through the Table Value Function feature, Doris can directly query and analyze files on object storage or HDFS as a Table. It also supports automatic column type inference.

For more usage methods, refer to the Table Value Function documentation:

- S3: Supports file analysis on S3-compatible object storage.
- HDFS: Supports file analysis on HDFS.

Basic Usage

Here we illustrate how to analyze files on object storage using the S3 Table Value Function as an example.

Query

```
SELECT * FROM S3 (
    'uri' = 's3://bucket/path/to/tvf_test/test.parquet',
    'format' = 'parquet',
    's3.endpoint' = 'https://s3.us-east-1.amazonaws.com',
    's3.region' = 'us-east-1',
    's3.access_key' = 'ak',
    's3.secret_key'='sk'
)
```

The `S3(...)` is a TVF (Table Value Function). A Table Value Function is essentially a table, so it can appear in any SQL statement where a "table" can appear.

The attributes of a TVF include the file path to be analyzed, file format, connection information of the object storage, etc. The file path (URI) can use wildcards to match multiple files. The following file paths are valid:

- Match a specific file

  `s3://bucket/path/to/tvf_test/test.parquet`

- Match all files starting with `test_`

  `s3://bucket/path/to/tvf_test/test_*`

- Match all files with the `.parquet` suffix

  `s3://bucket/path/to/tvf_test/*.parquet`

- Match all files in the `tvf_test` directory

  `s3://bucket/path/to/tvf_test/*`

- Match files with `test` in the filename

  `s3://bucket/path/to/tvf_test/*test*`

Automatic Inference of File Column Types

You can view the Schema of a TVF using the `DESC FUNCTION` syntax:

```
DESC FUNCTION s3 (
    "URI" = "s3://bucket/path/to/tvf_test/test.parquet",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "format" = "parquet",
    "use_path_style"="true"
);
+---------------+--------------+------+-------+---------+-------+
| Field         | Type         | Null | Key   | Default | Extra |
+---------------+--------------+------+-------+---------+-------+
| p_partkey     | INT          | Yes  | false | NULL    | NONE  |
| p_name        | TEXT         | Yes  | false | NULL    | NONE  |
| p_mfgr        | TEXT         | Yes  | false | NULL    | NONE  |
| p_brand       | TEXT         | Yes  | false | NULL    | NONE  |
| p_type        | TEXT         | Yes  | false | NULL    | NONE  |
| p_size        | INT          | Yes  | false | NULL    | NONE  |
| p_container   | TEXT         | Yes  | false | NULL    | NONE  |
| p_retailprice | DECIMAL(9,0) | Yes  | false | NULL    | NONE  |
| p_comment     | TEXT         | Yes  | false | NULL    | NONE  |
+---------------+--------------+------+-------+---------+-------+
```

Doris infers the Schema based on the following rules:

- For Parquet and ORC formats, Doris obtains the Schema from the file metadata.
- In the case of matching multiple files, the Schema of the first file is used as the TVF's Schema.
- For CSV and JSON formats, Doris parses the **first line of data** to obtain the Schema based on fields, delimiters, etc.

  By default, all column types are `string`. You can specify column names and types individually using the `csv_schema` attribute. Doris will use the specified column types for file reading. The format is: `name1:type1;name2:type2;...`. For example:

  ```
  S3 (
      'uri' = 's3://bucket/path/to/tvf_test/test.parquet',
      's3.endpoint' = 'https://s3.us-east-1.amazonaws.com',
      's3.region' = 'us-east-1',
      's3.access_key' = 'ak'
      's3.secret_key'='sk',
      'format' = 'csv',
      'column_separator' = '|',
      'csv_schema' = 'k1:int;k2:int;k3:int;k4:decimal(38,10)'
  )
  ```

  The currently supported column type names are as follows:

Column Type Name
tinyint
smallint
int

Column Type Name
bigint
largeint
float
double
decimal(p,s)
date
datetime
char
varchar
string
boolean

- For columns with mismatched formats (e.g., the file contains a string, but the user defines it as `int`; or other files have a different Schema than the first file), or missing columns (e.g., the file has 4 columns, but the user defines 5 columns), these columns will return `null`.

Applicable Scenarios

Query Analysis

TVF is very suitable for directly analyzing independent files on storage systems without having to import the data into Doris in advance.

You can use any SQL statement for file analysis, such as:

```
SELECT * FROM s3(
    'uri' = 's3://bucket/path/to/tvf_test/test.parquet',
    'format' = 'parquet',
    's3.endpoint' = 'https://s3.us-east-1.amazonaws.com',
    's3.region' = 'us-east-1',
    's3.access_key' = 'ak',
    's3.secret_key'='sk'
)
ORDER BY p_partkey LIMIT 5;
+-----------+-----------------------------------------+---------------+----------+-------------------------+--------+-------------
+---------------+--------------------+
| p_partkey | p_name                                  | p_mfgr        | p_brand  | p_type                  | p_size | p_container
| p_retailprice | p_comment           |
+-----------+-----------------------------------------+---------------+----------+-------------------------+--------+-------------
+---------------+--------------------+
|         1 | goldenrod lavender spring chocolate lace | Manufacturer#1 | Brand#13 | PROMO BURNISHED COPPER  |      7 | JUMBO PKG
|           901 | ly. slyly ironi     |
|         2 | blush thistle blue yellow saddle         | Manufacturer#1 | Brand#13 | LARGE BRUSHED BRASS     |      1 | LG CASE
|           902 | lar accounts amo    |
|         3 | spring green yellow purple cornsilk      | Manufacturer#4 | Brand#42 | STANDARD POLISHED BRASS |     21 | WRAP CASE
|           903 | egular deposits hag |
|         4 | cornflower chocolate smoke green pink    | Manufacturer#3 | Brand#34 | SMALL PLATED BRASS      |     14 | MED DRUM
|           904 | p furiously r       |
|         5 | forest brown coral puff cream            | Manufacturer#3 | Brand#32 | STANDARD POLISHED TIN   |     15 | SM PKG
```

```
|          905 |  wake carefully     |
+-----------+-----------------------------------------+---------------+----------+-----------------------+--------+------------
+---------------+--------------------+
```

TVF can appear in any position in SQL where a Table can appear, such as in the `WITH` clause of a `CTE`, in the `FROM` clause, etc. This way, you can treat the file as a regular table for any analysis.

You can also create a logical view for a TVF using the `CREATE VIEW` statement. After that, you can access this TVF like other views, manage permissions, etc., and allow other users to access this View without having to repeatedly write connection information and other attributes.

```
-- Create a view based on a TVF
CREATE VIEW tvf_view AS
SELECT * FROM s3(
    'uri' = 's3://bucket/path/to/tvf_test/test.parquet',
    'format' = 'parquet',
    's3.endpoint' = 'https://s3.us-east-1.amazonaws.com',
    's3.region' = 'us-east-1',
    's3.access_key' = 'ak',
    's3.secret_key'='sk'
);

-- Describe the view as usual
DESC tvf_view;

-- Query the view as usual
SELECT * FROM tvf_view;

-- Grant SELECT priv to other user on this view
GRANT SELECT_PRIV ON db.tvf_view TO other_user;
```

Data Import

TVF can be used as a method for data import into Doris. With the `INSERT INTO SELECT` syntax, we can easily import files into Doris.

```
-- Create a Doris table
CREATE TABLE IF NOT EXISTS test_table
(
    id int,
    name varchar(50),
    age int
)
DISTRIBUTED BY HASH(id) BUCKETS 4
PROPERTIES("replication_num" = "1");

-- 2. Load data into table from TVF
INSERT INTO test_table (id,name,age)
SELECT cast(id as INT) as id, name, cast (age as INT) as age
FROM s3(
    'uri' = 's3://bucket/path/to/tvf_test/test.parquet',
    'format' = 'parquet',
    's3.endpoint' = 'https://s3.us-east-1.amazonaws.com',
    's3.region' = 'us-east-1',
    's3.access_key' = 'ak',
    's3.secret_key'='sk'
);
```

Notes

1. If the specified `uri` does not match any files, or all matched files are empty, the TVF will return an empty result set. In this case, using `DESC FUNCTION` to view the Schema of this TVF will yield a virtual column `__dummy_col`, which is meaningless and only serves as a placeholder.

2. If the specified file format is `csv`, and the file read is not empty but the first line of the file is empty, an error `The first line is empty, can not parse column numbers` will be prompted, as the Schema cannot be parsed from the first line of the file.

Hive Metastore

This document describes all supported parameters when connecting to and accessing Hive MetaStore services through the `CREATE CATALOG` statement.

Supported Catalog Types

Catalog Type	Type Identifier (type)	Description
Hive	hms	Catalog for connecting to Hive Metastore
Iceberg	iceberg	Catalog for Iceberg table format
Paimon	paimon	Catalog for Apache Paimon table format

Common Parameters Overview

The following parameters are common to different Catalog types.

Parameter Name	Former Name	Required	Default	Description
hive.metastore.uris		Yes	None	URI address of Hive Metastore, supports multiple URIs separated by commas. Example: 'hive.metastore.uris' = 'thrift://127.0.0.1:9083','hive.metastore.uris' = 'thrift://127.0.0.1:9083,thrift://127.0.0.1:9084'
hive.metastore.authentication.type	hadoop.security.authentication	No	simple	Metastore authentication method: supports simple (default) or kerberos. In versions 3.0 and earlier, authentication method was determined by hadoop.security.authentication property. Starting from version 3.1, Hive Metastore authentication method can be specified separately. Example: 'hive.metastore.authentication.type' = 'kerberos'
hive.metastore.service.principal	hive.metastore.kerberos.principal	No	Empty	Hive server principal, supports _HOST placeholder. Example: 'hive.metastore.service.principal' = 'hive/_HOST@EXAMPLE.COM'
hive.metastore.client.principal	hadoop.kerberos.principal	No	Empty	Kerberos principal used by Doris to connect to Hive MetaStore service.

Parameter Name	Former Name	Required	Default	Description
hive.metastore.client.keytab	hadoop.kerberos.keytab	No	Empty	Kerberos keytab file path
hive.metastore.username	hadoop.username	No	hadoop	Hive Metastore username, used in non-Kerberos mode
hive.conf.resources		No	Empty	hive-site.xml configuration file path, using relative path

Parameter Name	Former Name	Required	Default	Description
hive.metastore.client.socket.timeout		No	Default value is hive_metastore_client_timeout_seconds	This parameter is supported since version 4.0.3. Timeout for accessing metadata through Hive Metastore Client connection. If the metadata is large (e.g., when there are many partitions), this value can be increased.

Parameter Name	Former Name	Required	Default	Description
			nFEconfigurationparameters,defaultsto10seconds.	

Note:

> For versions before 3.1.0, please use the former names.

Required Parameters

- `hive.metastore.uris`: Must specify the URI address of Hive Metastore

Optional Parameters

- `hive.metastore.authentication.type`: Authentication method, default is `simple`, optional `kerberos`
- `hive.metastore.service.principal`: Kerberos principal of Hive MetaStore service, must be specified when using Kerberos authentication.
- `hive.metastore.client.principal`: Kerberos principal used by Doris to connect to Hive MetaStore service, must be specified when using Kerberos authentication.
- `hive.metastore.client.keytab`: Kerberos keytab file path, must be specified when using Kerberos authentication.
- `hive.metastore.username`: Username for connecting to Hive MetaStore service, used in non-Kerberos mode, default is `hadoop`.
- `hive.conf.resources`: hive-site.xml configuration file path, used when configuration for connecting to Hive Metastore service needs to be read from configuration files.

Authentication Methods

Simple Authentication

- `simple`: Non-Kerberos mode, directly connects to Hive Metastore service.

Kerberos Authentication

To use Kerberos authentication to connect to Hive Metastore service, configure the following parameters:

- `hive.metastore.authentication.type`: Set to `kerberos`
- `hive.metastore.service.principal`: Kerberos principal of Hive MetaStore service
- `hive.metastore.client.principal`: Kerberos principal used by Doris to connect to Hive MetaStore service
- `hive.metastore.client.keytab`: Kerberos keytab file path

```
'hive.metastore.authentication.type' = 'kerberos',
'hive.metastore.service.principal' = 'hive/_HOST@EXAMPLE.COM',
'hive.metastore.client.principal' = 'hive/doris.cluster@EXAMPLE.COM',
'hive.metastore.client.keytab' = '/etc/security/keytabs/hive.keytab'
```

When using Hive MetaStore service with Kerberos authentication enabled, ensure that the same keytab file exists on all FE nodes, the user running the Doris process has read permission to the keytab file, and the krb5

configuration file is properly configured.

For detailed Kerberos configuration, refer to Kerberos Authentication.

Configuration File Parameters

`hive.conf.resources`

If you need to read configuration for connecting to Hive Metastore service through configuration files, you can configure the `hive.conf.resources` parameter to set the conf file path.

> Note: The `hive.conf.resources` parameter only supports relative paths, do not use absolute paths. The default path is under the `${DORIS_HOME}/plugins/hadoop_conf/` directory. You can specify other directories by modifying hadoop_config_dir in fe.conf.

Example: `'hive.conf.resources' = 'hms-1/hive-site.xml'`

Catalog Type-Specific Data

The following parameters are specific to each Catalog type, in addition to the common parameters.

Hive Catalog

Parameter Name	Former Name	Required	Default	Description
type		Yes	None	Catalog type, fixed as hms for Hive Catalog
hive.metastore.type		No	'hms'	Metadata Catalog type, fixed as hms for Hive Metastore, must be hms when using HiveMetaStore

Examples

1. Create a Hive Catalog using unauthenticated Hive Metastore as metadata service, with S3 storage service.

```
CREATE CATALOG hive_hms_s3_test_catalog PROPERTIES (
    'type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    's3.access_key' = 'S3_ACCESS_KEY',
    's3.secret_key' = 'S3_SECRET_KEY',
    's3.region' = 's3.ap-east-1.amazonaws.com'
);
```

2. Create a Hive Catalog using Hive Metastore with Kerberos authentication enabled as metadata service, with S3 storage service.

```
 CREATE CATALOG hive_hms_on_oss_kerberos_new_catalog PROPERTIES (
    'type' = 'hms',
```

```
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'hive.metastore.client.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/mnt/keytabs/hive-presto-master.keytab',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.authentication.type'='kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
                        RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
                        RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
                        DEFAULT',
    'oss.access_key' = 'OSS_ACCESS_KEY',
    'oss.secret_key' = 'OSS_SECRET_KEY',
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com'
);
```

Iceberg Catalog

Parameter Name	Former Name	Required	Default	Description
type		Yes	None	Catalog type, fixed as iceberg for Iceberg
iceberg.catalog.type		No	None	Metadata Catalog type, fixed as hms for Hive Metastore, must be hms when using HiveMetaStore
warehouse		No	None	Iceberg warehouse path

Examples

1. Create an Iceberg Catalog using Hive Metastore as metadata service, with S3 storage service.

```
 CREATE CATALOG iceberg_hms_s3_test_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
    'warehouse' = 's3://doris/iceberg_warehouse/',
    's3.access_key' = 'S3_ACCESS_KEY',
    's3.secret_key' = 'S3_SECRET_KEY',
    's3.region' = 's3.ap-east-1.amazonaws.com'
);
```

2. Create an Iceberg Catalog using Hive Metastore with Kerberos authentication enabled as metadata service in a multi-Kerberos environment, with S3 storage service.

```
CREATE CATALOG IF NOT EXISTS iceberg_hms_on_oss_kerberos_new_catalog PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
    'warehouse' = 'oss://doris/iceberg_warehouse/',
    'hive.metastore.client.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
    'hive.metastore.client.keytab' = '/mnt/keytabs/hive-presto-master.keytab',
    'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
    'hive.metastore.authentication.type'='kerberos',
    'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
                        RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
                        RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
                        DEFAULT',
    'oss.access_key' = 'OSS_ACCESS_KEY',
    'oss.secret_key' = 'OSS_SECRET_KEY',
```

```
    'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com'
);
```

Paimon Catalog

Parameter Name	Former Name	Required	Default	Description
type		Yes	None	Catalog type, fixed as paimon for Paimon
paimon.catalog.type		No	filesystem	Must be hms when using HiveMetaStore, default is filesystem for storing metadata in filesystem
warehouse		Yes	None	Paimon warehouse path

Examples

1. Create a Paimon Catalog using Hive Metastore as metadata service, with S3 storage service.

```
CREATE CATALOG IF NOT EXISTS paimon_hms_s3_test_catalog PROPERTIES (
   'type' = 'paimon',
   'paimon.catalog.type' = 'hms',
   'hive.metastore.uris' = 'thrift://127.0.0.1:9383',
   'warehouse' = 's3://doris/paimon_warehouse/',
   's3.access_key' = 'S3_ACCESS_KEY',
   's3.secret_key' = 'S3_SECRET_KEY',
   's3.region' = 's3.ap-east-1.amazonaws.com'
);
```

2. Create a Paimon Catalog using Hive Metastore with Kerberos authentication enabled as metadata service in a multi-Kerberos environment, with S3 storage service.

```
CREATE CATALOG IF NOT EXISTS paimon_hms_on_oss_kerberos_new_catalog PROPERTIES (
   'type' = 'paimon',
   'paimon.catalog.type' = 'hms',
   'hive.metastore.uris' = 'thrift://127.0.0.1:9583',
   'warehouse' = 's3://doris/iceberg_warehouse/',
   'hive.metastore.client.principal'='hive/presto-master.docker.cluster@LABS.TERADATA.COM',
   'hive.metastore.client.keytab' = '/mnt/keytabs/hive-presto-master.keytab',
   'hive.metastore.service.principal' = 'hive/hadoop-master@LABS.TERADATA.COM',
   'hive.metastore.authentication.type'='kerberos',
   'hadoop.security.auth_to_local' = 'RULE:[2:\$1@\$0](.*@LABS.TERADATA.COM)s/@.*//
                      RULE:[2:\$1@\$0](.*@OTHERLABS.TERADATA.COM)s/@.*//
                      RULE:[2:\$1@\$0](.*@OTHERREALM.COM)s/@.*//
                      DEFAULT',
   'oss.access_key' = 'OSS_ACCESS_KEY',
   'oss.secret_key' = 'OSS_SECRET_KEY',
   'oss.endpoint' = 'oss-cn-beijing.aliyuncs.com'
);
```

HMS Access Port Requirements

Doris requires at least the following ports to be open to access HMS:

Service	Port Purpose	Default Port	Protocol
Hive Metastore	Thrift (metadata access)	9083	TCP

Notes: - Ports may be customized in `hive-site.xml`. Always follow your actual configuration. - When Kerberos authentication is enabled, ensure network connectivity from Doris to the Kerberos KDC. The KDC listens on TCP port 88 by default, unless customized in your KDC configuration.

Frequently Asked Questions (FAQ)

- Q1: Is hive-site.xml mandatory?

  No, it's only used when configuration needs to be read from it.

- Q2: Must the keytab file exist on every node?

  Yes, all FE nodes must be able to access the specified path.

- Q3: What should be noted when using write-back functionality, i.e., creating Hive/Iceberg databases/tables in Doris?

  Since creating tables involves metadata operations on the storage side, i.e., accessing the storage system, the Hive MetaStore service server side needs to configure corresponding storage parameters, such as access parameters for S3, OSS and other storage services. When using object storage as the underlying storage system, ensure that the bucket being written to matches the configured Region.

AWS Glue

This document describes the parameter configuration when using **AWS Glue Catalog** to access **Iceberg tables** or **Hive tables** through `CREATE CATALOG`.

Supported Glue Catalog Types

AWS Glue Catalog currently supports three types of Catalogs:

Catalog Type	Type Identifier (`type`)	Description
Hive	glue	Catalog for connecting to Hive Metastore
Iceberg	glue	Catalog for connecting to Iceberg table format
Iceberg	rest	Catalog for connecting to Iceberg table format via Glue Rest Catalog

This documentation provides detailed parameter descriptions for each type to facilitate user configuration.

Common Parameters Overview

Parameter Name	Description	Required	Default Value
`glue.region`	AWS Glue region, e.g., `us-east-1`	Yes	None
`glue.endpoint`	AWS Glue endpoint, e.g., `https://glue.us-east-1.amazonaws.com`	Yes	None
`glue.access_key`	AWS Access Key ID	Yes	Empty
`glue.secret_key`	AWS Secret Access Key	Yes	Empty
`glue.catalog_id`	Glue Catalog ID (not supported yet)	No	Empty
`glue.role_arn`	IAM Role ARN for accessing Glue (supported since 3.1.2+)	No	Empty
`glue.external_id`	IAM External ID for accessing Glue (supported since 3.1.2+)	No	Empty

Authentication Parameters

Accessing Glue requires authentication information, supporting the following two methods:

1. Access Key Authentication

   Authenticate access to Glue through Access Key provided by `glue.access_key` and `glue.secret_key`.

2. IAM Role Authentication (supported since 3.1.2+)

   Authenticate access to Glue through IAM Role provided by `glue.role_arn`.

   This method requires Doris to be deployed on AWS EC2, and the EC2 instance needs to be bound to an IAM Role that has permission to access Glue.

If access through External ID is required, you need to configure `glue.external_id` as well.

Notes:

- At least one of the two methods must be configured. If both methods are configured, Access Key authentication takes priority.

Example:

```sql
CREATE CATALOG hive_glue_catalog PROPERTIES (
  'type' = 'hms',
  'hive.metastore.type' = 'glue',
  'glue.region' = 'us-east-1',
  'glue.endpoint' = 'https://glue.us-east-1.amazonaws.com',
  -- Using Access Key authentication
  'glue.access_key' = '<YOUR_ACCESS_KEY>',
  'glue.secret_key' = '<YOUR_SECRET_KEY>'
  -- Or using IAM Role authentication
  -- 'glue.role_arn' = '<YOUR_ROLE_ARN>',
  -- 'glue.external_id' = '<YOUR_EXTERNAL_ID>'
);
```

Hive Glue Catalog

Hive Glue Catalog is used to access Hive tables through AWS Glue's Hive Metastore compatible interface. Configuration as follows:

Parameter Name	Description	Required	Default Value
`type`	Fixed as `hms`	Yes	None
`hive.metastore.type`	Fixed as `glue`	Yes	None
`glue.region`	AWS Glue region, e.g., `us-east-1`	Yes	None
`glue.endpoint`	AWS Glue endpoint, e.g., `https://glue.us-east-1.amazonaws.com`	Yes	None
`glue.access_key`	AWS Access Key ID	No	Empty
`glue.secret_key`	AWS Secret Access Key	No	Empty
`glue.catalog_id`	Glue Catalog ID (not supported yet)	No	Empty
`glue.role_arn`	IAM Role ARN for accessing Glue	No	Empty
`glue.external_id`	IAM External ID for accessing Glue	No	Empty

Example

```
CREATE CATALOG hive_glue_catalog PROPERTIES (
  'type' = 'hms',
  'hive.metastore.type' = 'glue',
  'glue.region' = 'us-east-1',
  'glue.endpoint' = 'https://glue.us-east-1.amazonaws.com',
  'glue.access_key' = 'YOUR_ACCESS_KEY',
```

```
  'glue.secret_key' = 'YOUR_SECRET_KEY'
);
```

Iceberg Glue Catalog

Iceberg Glue Catalog accesses Glue through Glue Client. Configuration as follows:

Parameter Name	Description	Required	Default Value
`type`	Fixed as `iceberg`	Yes	None
`iceberg.catalog.type`	Fixed as `glue`	Yes	None
`warehouse`	Iceberg data warehouse path, e.g., `s3://my-bucket/iceberg-warehouse/`	Yes	s3://doris
`glue.region`	AWS Glue region, e.g., `us-east-1`	Yes	None
`glue.endpoint`	AWS Glue endpoint, e.g., `https://glue.us-east-1.amazonaws.com`	Yes	None
`glue.access_key`	AWS Access Key ID	No	Empty
`glue.secret_key`	AWS Secret Access Key	No	Empty
`glue.catalog_id`	Glue Catalog ID (not supported yet)	No	Empty
`glue.role_arn`	IAM Role ARN for accessing Glue (not supported yet)	No	Empty
`glue.external_id`	IAM External ID for accessing Glue (not supported yet)	No	Empty

Example

```
CREATE CATALOG iceberg_glue_catalog PROPERTIES (
  'type' = 'iceberg',
  'iceberg.catalog.type' = 'glue',
  'glue.region' = 'us-east-1',
  'glue.endpoint' = 'https://glue.us-east-1.amazonaws.com',
  'glue.access_key' = '<YOUR_ACCESS_KEY>',
  'glue.secret_key' = '<YOUR_SECRET_KEY>'
);
```

Iceberg Glue Rest Catalog

Iceberg Glue Rest Catalog accesses Glue through Glue Rest Catalog interface. Currently only supports Iceberg tables stored in AWS S3 Table Bucket. Configuration as follows:

Parameter Name	Description	Required	Default Value
`type`	Fixed as `iceberg`	Yes	None
`iceberg.catalog.type`	Fixed as `rest`	Yes	None
`iceberg.rest.uri`	Glue Rest service endpoint, e.g., `https://glue.ap-east-1.amazonaws.com/iceberg`	Yes	None
`warehouse`	Iceberg data warehouse path, e.g., `<account_id>:s3tablescatalog/<bucket_name>`	Yes	None

Parameter Name	Description	Required	Default Value
iceberg.rest.sigv4-enabled	Enable V4 signature format, fixed as true	Yes	None
iceberg.rest.signing-name	Signature type, fixed as glue	Yes	Empty
iceberg.rest.access-key-id	Access Key for accessing Glue (also used for accessing S3 Bucket)	Yes	Empty
iceberg.rest.secret-access-key	Secret Key for accessing Glue (also used for accessing S3 Bucket)	Yes	Empty
iceberg.rest.signing-region	AWS Glue region, e.g., us-east-1	Yes	Empty

Example

```
CREATE CATALOG glue_s3 PROPERTIES (
  'type' = 'iceberg',
  'iceberg.catalog.type' = 'rest',
  'iceberg.rest.uri' = 'https://glue.<region>.amazonaws.com/iceberg',
  'warehouse' = '<acount_id>:s3tablescatalog/<s3_table_bucket_name>',
  'iceberg.rest.sigv4-enabled' = 'true',
  'iceberg.rest.signing-name' = 'glue',
  'iceberg.rest.access-key-id' = '<ak>',
  'iceberg.rest.secret-access-key' = '<sk>',
  'iceberg.rest.signing-region' = '<region>'
);
```

Permission Policies

Depending on usage scenarios, they can be divided into **read-only** and **read-write** policies.

1. Read-Only Permissions

Only allows reading database and table information from Glue Catalog.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GlueCatalogReadOnly",
      "Effect": "Allow",
      "Action": [
        "glue:GetCatalog",
        "glue:GetDatabase",
        "glue:GetDatabases",
        "glue:GetTable",
        "glue:GetTables",
        "glue:GetPartitions"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<account-id>:catalog",
        "arn:aws:glue:<region>:<account-id>:database/*",
        "arn:aws:glue:<region>:<account-id>:table/*/*"
      ]
    }
  ]
}
```

2. Read-Write Permissions

Based on read-only permissions, allows creating/modifying/deleting databases and tables.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GlueCatalogReadWrite",
      "Effect": "Allow",
      "Action": [
        "glue:GetCatalog",
        "glue:GetDatabase",
        "glue:GetDatabases",
        "glue:GetTable",
        "glue:GetTables",
        "glue:GetPartitions",
        "glue:CreateDatabase",
        "glue:UpdateDatabase",
        "glue:DeleteDatabase",
        "glue:CreateTable",
        "glue:UpdateTable",
        "glue:DeleteTable"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<account-id>:catalog",
        "arn:aws:glue:<region>:<account-id>:database/*",
        "arn:aws:glue:<region>:<account-id>:table/*/*"
      ]
    }
  ]
}
```

Notes

1. Placeholder Replacement
    - `<region>` → Your AWS region (e.g., `us-east-1`).
    - `<account-id>` → Your AWS account ID (12-digit number).
2. Principle of Least Privilege
    - If only querying, do not grant write permissions.
    - Can replace * with specific database/table ARNs to further restrict permissions.
3. S3 Permissions
    - The above policies only involve Glue Catalog.
    - If you need to read data files, additional S3 permissions are required (such as `s3:GetObject`, `s3:ListBucket`, etc.).

Google Dataproc

This document describes the supported parameters when connecting to and accessing Google Dataproc Metastore through the `CREATE CATALOG` statement.

Property Name	Description	Default Value	Required
`hive.metastore.uri`	The URI address of the Metastore. Can be obtained from the Dataproc Metastore Services page	None	Yes

Aliyun DLF

This document describes how to use the `CREATE CATALOG` statement to connect and access Alibaba Cloud Data Lake Formation (DLF) metadata service.

DLF Version Notes

- For DLF 1.0 version, Doris accesses DLF through DLF's Hive Metastore compatible interface. Supports Paimon Catalog and Hive Catalog.
- For DLF versions 2.5 and later, Doris accesses DLF through DLF's Rest interface. Only supports Paimon Catalog.

DLF 1.0

Parameter Name	Legacy Name	Description	Default Value	Required
`dlf.endpoint`	-	DLF endpoint, see: Alibaba Cloud Documentation	None	Yes
`dlf.region`	-	DLF region, see: Alibaba Cloud Documentation	None	Yes
`dlf.uid`	-	Alibaba Cloud account ID. Can be viewed in personal information at the top right corner of the console.	None	Yes
`dlf.access_key`	-	Alibaba Cloud AccessKey for accessing DLF service.	None	Yes
`dlf.secret_key`	-	Alibaba Cloud SecretKey for accessing DLF service.	None	Yes
`dlf.catalog_id`	`dlf.catalog.id`	Catalog ID. Used to specify metadata catalog. If not set, the default catalog is used.	None	No
`warehouse`	-	Storage path of the Warehouse, only required for Paimon Catalog. Note that the object storage path must end with `/`.	None	No

> Note:

> For versions before 3.1.0, please use the former names.

DLF 2.5+ (Rest Catalog)

> Supported since version 3.1.0

Parameter Name	Legacy Name	Description	Default Value	Required
`uri`	-	DLF REST URI. Example: http://cn-beijing-vpc.dlf.aliyuncs.com	None	Yes
`warehouse`	-	Warehouse name. Note: directly fill in the name of the Catalog to connect to, not the Paimon table storage path	None	Yes
`paimon.rest.token.provider`	-	Token provider, fixed value `dlf`	None	Yes
`paimon.rest.dlf.access-key-id`	-	Alibaba Cloud AccessKey for accessing DLF service.	None	Yes
`paimon.rest.dlf.access-key-secret`	-	Alibaba Cloud SecretKey for accessing DLF service.	None	Yes

DLF Rest Catalog does not require providing storage service (OSS) Endpoint and Region information. Doris will use DLF Rest Catalog's Vended Credential to obtain temporary credential information for accessing OSS.

Examples

DLF 1.0

Create a Hive Catalog with DLF as the metadata service:

```
CREATE CATALOG hive_dlf_catalog WITH (
  'type' = 'hms',
  'hive.metastore.type' = 'dlf',
  'dlf.endpoint' = '<DLF_ENDPOINT>',
  'dlf.region' = '<DLF_REGION>',
  'dlf.uid' = '<YOUR_ALICLOUD_UID>',
  'dlf.access_key' = '<YOUR_ACCESS_KEY>',
  'dlf.secret_key' = '<YOUR_SECRET_KEY>'
);
```

Create a Paimon Catalog with DLF as the metadata service:

```
CREATE CATALOG paimon_dlf PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'dlf',
    'warehouse' = 'oss://xx/yy/',
    'dlf.proxy.mode' = 'DLF_ONLY',
    'dlf.endpoint' = '<DLF_ENDPOINT>',
    'dlf.region' = '<DLF_REGION>',
    'dlf.uid' = '<YOUR_ALICLOUD_UID>',
    'dlf.access_key' = '<YOUR_ACCESS_KEY>',
    'dlf.secret_key' = '<YOUR_SECRET_KEY>'
);
```

DLF 2.5+ (Rest Catalog)

```
CREATE CATALOG paimon_dlf_test PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'rest',
    'uri' = 'http://cn-beijing-vpc.dlf.aliyuncs.com',
    'warehouse' = 'my_catalog_name',
    'paimon.rest.token.provider' = 'dlf',
    'paimon.rest.dlf.access-key-id' = '<YOUR_ACCESS_KEY>',
    'paimon.rest.dlf.access-key-secret' = '<YOUR_SECRET_KEY>'
);
```

Iceberg Rest Catalog API

This document describes the supported parameters when connecting to and accessing metadata services that support the Iceberg Rest Catalog interface through the `CREATE CATALOG` statement.

Parameter Overview

Property Name	Legacy Name	Description	Default Value	Required
iceberg.rest.uri	uri	Specifies the Rest service address	-	Yes
warehouse	warehouse	Specifies the iceberg warehouse	-	Yes
iceberg.rest.security.type		Specifies the Rest service authentication method, supports `oauth2`, defaults to `none` (no authentication)	`none`	No
iceberg.rest.oauth2.token		Bearer token when using `oauth2` authentication	-	No
iceberg.rest.oauth2.scope		Specifies the resource scope and permission level that users can access after authorization when using `oauth2` authentication	-	No
iceberg.rest.oauth2.credential		`oauth2` credentials used to access `server-uri` to obtain token	-	No
iceberg.rest.oauth2.server-uri		URI address for obtaining `oauth2` token, used in conjunction with `iceberg.rest.oauth2.credential`	-	No
iceberg.rest.vended-credentials-enabled		Whether to enable `vended-credentials` functionality. When enabled, it will obtain storage system access credentials such as `access-key` and `secret-key` from the rest server, eliminating the need for manual specification. Requires rest server support for this capability.	`false`	No

Property Name	Legacy Name	Description	Default Value	Required
iceberg.rest.nested-namespace-enabled		(Supported since version 3.1.2+) Whether to enable support for Nested Namespace. Default is `false`. If `true`, Nested Namespace will be flattened and displayed as Database names, such as `parent_ns.child_ns`. Some Rest Catalog services do not support Nested Namespace, such as AWS Glue, so this parameter should be set to `false`	No	

> Note:
>
> 1. OAuth2 authentication and `vended-credentials` functionality are supported starting from version 3.1.0.
> 2. For versions prior to 3.1.0, please use the legacy names.
> 3. For AWS Glue Rest Catalog, please refer to the AWS Glue documentation

Nested Namespace

Since 3.1.2, to fully access Nested Namespace, in addition to setting `iceberg.rest.nested-namespace-enabled` to `true` in the Catalog properties, you also need to enable the following global parameter:

```
SET GLOBAL enable_nested_namespace=true;
```

Assuming the Catalog is “ice”, Namespace is “ns1.ns2”, and Table is “tbl1”, you can access Nested Namespace in the following ways:

```
mysql> USE ice.ns1.ns2;
mysql> SELECT k1 FROM ice.`ns1.ns2`.tbl1;
mysql> SELECT tbl1.k1 FROM `ns1.ns2`.tbl1;
mysql> SELECT `ns1.ns2`.tbl1.k1 FROM ice.`ns1.ns2`.tbl1;
mysql> SELECT ice.`ns1.ns2`.tbl1.k1 FROM tbl1;
mysql> REFRESH CATALOG ice;
mysql> REFRESH DATABASE ice.`ns1.ns2`;
mysql> REFRESH TABLE ice.`ns1.ns2`.tbl1;
```

Example Configurations

- Rest Catalog service without authentication

```
CREATE CATALOG minio_iceberg PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'uri' = 'http://172.21.0.1:8181',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.endpoint' = 'http://10.0.0.1:9000',
    's3.region' = 'us-east-1'
);
```

- Connecting to AWS Glue Rest Catalog

```
CREATE CATALOG glue_iceberg PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'https://glue.<region>.amazonaws.com/iceberg',
    'warehouse' = '<acount_id>:s3tablescatalog/<s3_table_bucket_name>',
    'iceberg.rest.sigv4-enabled' = 'true',
    'iceberg.rest.signing-name' = 'glue',
    'iceberg.rest.access-key-id' = '<ak>',
    'iceberg.rest.secret-access-key' = '<sk>',
    'iceberg.rest.signing-region' = '<region>'
);
```

- Connecting to Databricks Unity Iceberg Rest Catalog

```
CREATE CATALOG unity_iceberg properties(
    "uri" = "https://dbc-59918a85-6c3a.cloud.databricks.com/api/2.1/unity-catalog/iceberg-rest/",
    "type" = "iceberg",
    "warehouse" = "<catalog_name>",
    "iceberg.catalog.type" = "rest",
    "iceberg.rest.security.type" = "oauth2",
    "iceberg.rest.oauth2.token" = "<token>",
    "iceberg.rest.vended-credentials-enabled" = "true",
    's3.endpoint' = 'https://s3.us-east-2.amazonaws.com',
    's3.region' = 'us-east-2'
);
```

- Connecting to Apache Polaris Rest Catalog

```
-- Enable vended-credentials
CREATE CATALOG polaris_iceberg PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://YOUR_POLARIS_HOST:8181/api/catalog',
    'warehouse' = '<catalog_name>',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = 'client_id:client_secret',
    'iceberg.rest.oauth2.server-uri' = 'http://YOUR_POLARIS_HOST:8181/api/catalog/v1/oauth/tokens',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:doris_pr_role',
    'iceberg.rest.vended-credentials-enabled' = 'true',
    's3.endpoint' = 'https://s3.us-west-2.amazonaws.com',
    's3.region' = 'us-west-2'
);

-- Disable vended-credentials
CREATE CATALOG polaris_iceberg PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
```

```
    'iceberg.rest.uri' = 'http://YOUR_POLARIS_HOST:8181/api/catalog',
    'warehouse' = '<catalog_name>',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = '6e155b128dc06c13:ce9fbb4cc91c43ff2955f2c6545239d7',
    'iceberg.rest.oauth2.server-uri' = 'http://YOUR_POLARIS_HOST:8181/api/catalog/v1/oauth/tokens',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:doris_pr_role',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.endpoint' = 'https://s3.us-west-2.amazonaws.com',
    's3.region' = 'us-west-2'
);
```

- Connecting to Snowflake Open Catalog (Since 3.1.2)

```
-- Enable vended-credentials
CREATE CATALOG snowflake_open_catalog PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = '<catalog_name>',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'https://<open_catalog_account>.snowflakecomputing.com/polaris/api/catalog',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = '<client_id>:<client_secret>',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:<principal_role>',
    'iceberg.rest.vended-credentials-enabled' = 'true',
    's3.endpoint' = 'https://s3.us-west-2.amazonaws.com',
    's3.region' = 'us-west-2',
    'iceberg.rest.nested-namespace-enabled' = 'true'
);
```

```
-- Disable vended-credentials
CREATE CATALOG snowflake_open_catalog PROPERTIES (
    'type' = 'iceberg',
    'warehouse' = '<catalog_name>',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'https://<open_catalog_account>.snowflakecomputing.com/polaris/api/catalog',
    'iceberg.rest.security.type' = 'oauth2',
    'iceberg.rest.oauth2.credential' = '<client_id>:<client_secret>',
    'iceberg.rest.oauth2.scope' = 'PRINCIPAL_ROLE:<principal_role>',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.endpoint' = 'https://s3.us-west-2.amazonaws.com',
    's3.region' = 'us-west-2',
    'iceberg.rest.nested-namespace-enabled' = 'true'
);
```

- Connecting to Apache Gravitino Rest Catalog

```
-- Enable vended-credentials
CREATE CATALOG gravitino_iceberg PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:9001/iceberg/',
    'warehouse' = 's3://gravitino-iceberg-demo/warehouse',
    'iceberg.rest.vended-credentials-enabled' = 'true',
    's3.endpoint' = 'https://s3.us-west-2.amazonaws.com',
    's3.region' = 'us-west-2'
);

-- Disable vended-credentials
CREATE CATALOG gravitino_iceberg PROPERTIES (
```

```
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'http://127.0.0.1:9001/iceberg/',
    'warehouse' = 's3://gravitino-iceberg-demo/warehouse',
    'iceberg.rest.vended-credentials-enabled' = 'false',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>',
    's3.endpoint' = 'https://s3.us-west-2.amazonaws.com',
    's3.region' = 'us-west-2'
);
```

File System

Iceberg Catalog and Paimon Catalog support accessing metadata directly from the file system.

For related connection information, please refer to the documentation of each respective Catalog and the parameters related to the corresponding

HDFS | Storages

HDFS

This document describes the parameters required for accessing HDFS. These parameters apply to:

- Catalog properties
- Table Valued Function properties
- Broker Load properties
- Export properties
- Outfile properties
- Backup and restore

Parameter Overview

Property Name	Legacy Name	Description	Default Value	Required
hdfs.authentication.type	hadoop.security.authentication	Specifies the authentication type. Optional values are kerberos or simple. If kerberos is selected, the system will use Kerberos authentication to interact with HDFS; if simple is used, it means no authentication is used, suitable for open HDFS clusters. Selecting kerberos requires configuring the corresponding principal and keytab.	simple	No
hdfs.authentication.kerberos.principal	hadoop.kerberos.principal	When the authentication type is kerberos, specify the Kerberos principal. A Kerberos principal is a unique identity string that typically includes service name, hostname, and domain name.	-	No

Property Name	Legacy Name	Description	Default Value	Required
hdfs.authentication.kerberos.keytab	hadoop.kerberos.keytab	This parameter specifies the keytab file path for Kerberos authentication. The keytab file stores encrypted credentials, allowing the system to authenticate automatically without requiring users to manually enter passwords.	-	No
hdfs.impersonation.enabled	-	If true, enables HDFS impersonation functionality. Uses the proxy user configured in core-site.xml to proxy Doris login users for HDFS operations	Not supported yet	-
hadoop.username	-	When the authentication type is simple, this user will be used to access HDFS. By default, the Linux system user running the Doris process will be used for access	-	-
hadoop.config.resources	-	Specifies the HDFS configuration file directory (must include hdfs-site.xml and core-site.xml), using relative paths. The default directory is /plugins/hadoop/conf/ under the (FE/BE) deployment directory (can be changed by modifying hadoop_config_dir in fe.conf/be.conf to change the default path). All FE and BE nodes need to configure the same relative path. Example: hadoop/conf/core-site.xml,hadoop/conf/hdfs-site.xml	-	-

Property Name	Legacy Name	Description	Default Value	Required
dfs.nameservices	-	Manually configure HDFS high availability cluster parameters. If using hadoop.config.resources configuration, parameters will be automatically read from hdfs-site.xml. Need to be used with the following parameters: dfs.ha.namenodes.your-nameservice, dfs.namenode.rpc-address.your-nameservice.nn1, dfs.client.failover.proxy.provider, etc.	-	-

> For versions before 3.1, please use the legacy names.

Authentication Configuration

HDFS supports two authentication methods:

- Simple
- Kerberos

Simple Authentication

Simple authentication is suitable for HDFS clusters that have not enabled Kerberos.

Using Simple authentication, you can set the following parameters or use the default values directly:

```
"hdfs.authentication.type" = "simple"
```

In Simple authentication mode, you can use the `hadoop.username` parameter to specify the username. If not specified, it defaults to the username of the current process.

Examples:

Using `lakers` username to access HDFS

```
"hdfs.authentication.type" = "simple",
"hadoop.username" = "lakers"
```

Using default system user to access HDFS

```
"hdfs.authentication.type" = "simple"
```

Kerberos Authentication

Kerberos authentication is suitable for HDFS clusters with Kerberos enabled.

Using Kerberos authentication, you need to set the following parameters:

```
"hdfs.authentication.type" = "kerberos",
"hdfs.authentication.kerberos.principal" = "<your_principal>",
"hdfs.authentication.kerberos.keytab" = "<your_keytab>"
```

In Kerberos authentication mode, you need to set the Kerberos principal and keytab file path.

Doris will access HDFS with the identity specified by the `hdfs.authentication.kerberos.principal` property, using the keytab specified by keytab to authenticate the Principal.

> Note:
>
> The keytab file must exist on every FE and BE node with the same path, and the user running the Doris process must have read permissions for the keytab file.

Example:

```
"hdfs.authentication.type" = "kerberos",
"hdfs.authentication.kerberos.principal" = "hdfs/hadoop@HADOOP.COM",
"hdfs.authentication.kerberos.keytab" = "/etc/security/keytabs/hdfs.keytab",
```

HDFS HA Configuration

If HDFS HA mode is enabled, need to configure `dfs.nameservices` related parameters:

```
'dfs.nameservices' = '<your-nameservice>',
'dfs.ha.namenodes.<your-nameservice>' = '<nn1>,<nn2>',
'dfs.namenode.rpc-address.<your-nameservice>.<nn1>' = '<nn1_host:port>',
'dfs.namenode.rpc-address.<your-nameservice>.<nn2>' = '<nn2_host:port>',
'dfs.client.failover.proxy.provider.<your-nameservice>' =
'org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider',
```

Example:

```
'dfs.nameservices' = 'nameservice1',
'dfs.ha.namenodes.nameservice1' = 'nn1,nn2',
'dfs.namenode.rpc-address.nameservice1.nn1' = '172.21.0.2:8088',
'dfs.namenode.rpc-address.nameservice1.nn2' = '172.21.0.3:8088',
'dfs.client.failover.proxy.provider.nameservice1' = 'org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider',
```

Configuration Files

> This feature is supported since version 3.1.0

Doris supports specifying the HDFS configuration file directory through the `hadoop.config.resources` parameter.

The configuration file directory must contain `hdfs-site.xml` and `core-site.xml` files. The default directory is `/plugins/hadoop_conf/` under the (FE/BE) deployment directory. All FE and BE nodes need to configure the same relative path.

If the configuration files contain the above parameters mentioned in this document, user-explicitly configured parameters take priority. Configuration files can specify multiple files, separated by commas, such as `hadoop/conf/core-site.xml,hadoop/conf/hdfs-site.xml`.

Examples:

```
-- Multiple configuration files
'hadoop.config.resources'='hdfs-cluster-1/core-site.xml,hdfs-cluster-1/hdfs-site.xml'
-- Single configuration file
'hadoop.config.resources'='hdfs-cluster-2/hdfs-site.xml'
```

HDFS IO Optimization

In some cases, high HDFS load may cause reading data replicas on HDFS to take a long time, thereby slowing down overall query efficiency. The following introduces some related optimization configurations.

Hedged Read

HDFS Client provides Hedged Read functionality. This feature can start another read thread to read the same data when a read request exceeds a certain threshold without returning, using whichever returns first.

Note: This feature may increase the load on the HDFS cluster, please use it judiciously.

You can enable this feature in the following way:

```
"dfs.client.hedged.read.threadpool.size" = "128",
"dfs.client.hedged.read.threshold.millis" = "500"
```

- `dfs.client.hedged.read.threadpool.size`

  Represents the number of threads used for Hedged Read, which are shared by one HDFS Client. Usually, for one HDFS cluster, BE nodes share one HDFS Client.

- `dfs.client.hedged.read.threshold.millis`

  Read threshold in milliseconds. When a read request exceeds this threshold without returning, it will trigger Hedged Read.

After enabling, you can see related parameters in Query Profile:

- `TotalHedgedRead`

  Number of times Hedged Read was initiated.

- `HedgedReadWins`

  Number of successful Hedged Read attempts (initiated and returned faster than the original request).

Note that the values here are cumulative values for a single HDFS Client, not values for a single query. The same HDFS Client is reused by multiple queries.

dfs.client.socket-timeout

`dfs.client.socket-timeout` is a client configuration parameter in Hadoop HDFS used to set the socket timeout when the client establishes connections or reads data with DataNode or NameNode, in milliseconds. The default value of this parameter is usually 60,000 milliseconds.

Reducing this parameter value allows the client to timeout faster and retry or switch to other nodes when encountering network delays, slow DataNode responses, or connection exceptions. This helps reduce wait times and improve system response speed. For example, in some tests, setting `dfs.client.socket-timeout` to a smaller value (such as 5000 milliseconds) can quickly detect DataNode delays or failures, avoiding long waits.

Note:

- Setting the timeout too small may cause frequent timeout errors during network fluctuations or high node load, affecting task stability.
- It is recommended to reasonably adjust this parameter value based on actual network environment and system load conditions to balance response speed and system stability.
- This parameter should be set in the client configuration file (such as `hdfs-site.xml`) to ensure the client uses the correct timeout when communicating with HDFS.

In summary, properly configuring the `dfs.client.socket-timeout` parameter can improve I/O response speed while ensuring system stability and reliability.

HDFS Access Port Requirements (NameNode & DataNode only)

Doris requires the following ports to be open to access HDFS:

Service	Port Purpose	Default Port	Protocol
NameNode	RPC (client/metadata access)	8020	TCP
DataNode	Data transfer (block I/O)	9866	TCP

Notes: - Ports may be customized in `core-site.xml` & `hdfs-site.xml`. Use actual configs. - When Kerberos authentication is enabled, Doris must also be able to reach the Kerberos KDC service. The KDC listens on TCP port 88 by default, but the actual port should follow your KDC configuration.

Debugging HDFS

Hadoop environment configuration is complex, and in some cases, connectivity issues and poor access performance may occur. Here are some third-party tools to help users quickly troubleshoot connectivity issues and basic performance problems.

HDFS Client

- Java: https://github.com/morningman/hdfs-client-java
- CPP: https://github.com/morningman/hdfs-client-cpp

These two tools can be used to quickly verify HDFS connectivity and read performance. Most of their Hadoop dependencies are the same as Doris's own Hadoop dependencies, so they can simulate Doris's HDFS access scenarios to the greatest extent.

The Java version uses Java to access HDFS and can simulate the logic of Doris FE side accessing HDFS.

The CPP version accesses HDFS through C++ calling libhdfs and can simulate the logic of Doris BE side accessing HDFS.

For specific usage, please refer to the README in each code repository.

S3 | Storages

S3

This document describes the parameters required for accessing AWS S3. These parameters apply to:

- Catalog properties.
- Table Valued Function properties.
- Broker Load properties.
- Export properties.
- Outfile properties.

Configure BE CA Certificate for HTTPS

Starting from Doris 2.1, you can explicitly configure `ca_cert_file_paths` in `be.conf` when Doris BE accesses S3 over HTTPS.

By default, if `ca_cert_file_paths` is not configured, Doris uses the operating system's default CA certificates. In most environments, you do not need to set this parameter manually. Configure it in the following cases:

- The BE node is missing system CA certificates, or the installed CA bundle is too old.
- The BE node runs in a minimal container or image that does not include the `ca-certificates` package.
- The default CA file path on the BE node is invalid, or the Doris process does not have read permission on the CA file.
- Your environment uses a self-signed certificate, a private CA, or a corporate proxy or gateway that re-signs TLS traffic.
- You encounter errors such as `Problem with the SSL CA cert` or `curl 77: Problem with the SSL CA cert (path? access rights?)` when accessing S3.

Example:

```
# be.conf
ca_cert_file_paths = /etc/ssl/certs/ca-certificates.crt
```

Common CA bundle paths:

- Debian / Ubuntu: `/etc/ssl/certs/ca-certificates.crt`
- CentOS / RockyLinux: `/etc/pki/ca-trust/extracted/openssl/ca-bundle.trust.crt`

Configure this item on every BE node that may access S3, and ensure that the certificate file exists and is readable by the Doris process. After updating `be.conf`, restart the affected BE nodes to apply the change.

Parameter Overview

Property Name	Legacy Name	Description	Default	Required
s3.endpoint		S3 service access endpoint, e.g., `s3.us-east-1.amazonaws.com`	None	No
s3.access_key		AWS Access Key for authentication	None	No
s3.secret_key		AWS Secret Key for authentication	None	No
s3.region		S3 region, e.g., `us-east-1`. Strongly recommended	None	Yes
s3.use_path_style		Whether to use path-style access	FALSE	No
s3.connection.maximum		Maximum number of connections for high concurrency scenarios	50	No
s3.connection.request.timeout		Request timeout (milliseconds), controls connection acquisition timeout	3000	No
s3.connection.timeout		Connection establishment timeout (milliseconds)	1000	No
s3.role_arn		Role ARN specified when using Assume Role mode	None	No
s3.external_id		External ID used with `s3.role_arn`	None	No
s3.credentials_provider_type		Credentials provider type for AWS authentication (used without AK/SK; used as STS source credentials in IAM Role mode)	DEFAULT	No

> Version note: `s3.credentials_provider_type` is supported since **3.1.4** and **4.0.3**.

Authentication Configuration

Doris supports the following three methods to access S3:

1. Direct Access Key and Secret Key (AK/SK)

```
"s3.access_key"="your-access-key",
"s3.secret_key"="your-secret-key",
"s3.endpoint"="https://s3.us-east-1.amazonaws.com",
"s3.region"="us-east-1"
```

2. IAM Role (Assume Role) mode

Suitable for cross-account and temporary authorization access. Doris automatically obtains temporary credentials through role authorization.

```
"s3.role_arn"="arn:aws:iam::123456789012:role/demo-role",
"s3.external_id"="external-identifier",
"s3.endpoint"="https://s3.us-east-1.amazonaws.com",
"s3.region"="us-east-1"
```

Configure `s3.credentials_provider_type` in IAM Role mode

When `s3.role_arn` is configured, `s3.credentials_provider_type` controls which source credentials provider is used for STS `AssumeRole`:

1. Get source credentials from `s3.credentials_provider_type`.
2. Call STS `AssumeRole` with source credentials.
3. Access S3 with the returned temporary credentials.

IAM Role + `s3.credentials_provider_type` examples

Example 1: EC2 Instance Profile as STS source credentials

```
"s3.role_arn"="arn:aws:iam::123456789012:role/demo-role",
"s3.external_id"="external-identifier",
"s3.credentials_provider_type"="INSTANCE_PROFILE",
"s3.endpoint"="https://s3.us-east-1.amazonaws.com",
"s3.region"="us-east-1"
```

Example 2: Web Identity (for example IRSA) as STS source credentials

```
"s3.role_arn"="arn:aws:iam::123456789012:role/demo-role",
"s3.credentials_provider_type"="WEB_IDENTITY",
"s3.endpoint"="https://s3.us-east-1.amazonaws.com",
"s3.region"="us-east-1"
```

Example 3: Container metadata as STS source credentials

```
"s3.role_arn"="arn:aws:iam::123456789012:role/demo-role",
"s3.credentials_provider_type"="CONTAINER",
"s3.endpoint"="https://s3.us-east-1.amazonaws.com",
"s3.region"="us-east-1"
```

Example 4: Default provider chain as STS source credentials

```
"s3.role_arn"="arn:aws:iam::123456789012:role/demo-role",
"s3.credentials_provider_type"="DEFAULT",
"s3.endpoint"="https://s3.us-east-1.amazonaws.com",
"s3.region"="us-east-1"
```

3. Specify credential source with `s3.credentials_provider_type`

This is suitable for scenarios without explicit AK/SK, such as EC2 Instance Profile, container metadata, or Web Identity.

```
"s3.credentials_provider_type"="INSTANCE_PROFILE",
"s3.endpoint"="https://s3.us-east-1.amazonaws.com",
"s3.region"="us-east-1"
```

Supported values for `s3.credentials_provider_type`

Value	Description
DEFAULT	Use default provider chain
ENV	Read credentials from environment variables
SYSTEM_PROPERTIES	Read credentials from system properties
WEB_IDENTITY	Use Web Identity Token credentials
CONTAINER	Use container metadata credentials
INSTANCE_PROFILE	Use EC2 Instance Profile credentials
ANONYMOUS	Anonymous access (for public buckets)

Effective rules when configured together

1. If `s3.access_key` and `s3.secret_key` are both configured, AK/SK is used first.
2. If AK/SK is not configured and `s3.role_arn` is configured, IAM Role is used. In this case, `s3.credentials_provider_type` is used to select STS source credentials.
3. If neither AK/SK nor `s3.role_arn` is configured, `s3.credentials_provider_type` directly determines the credentials provider used by the S3 client.

> Note: `s3.access_key` and `s3.secret_key` must be configured together.

Accessing S3 Directory Bucket

> This feature is supported since version 3.1.0.

Amazon S3 Express One Zone (also known as Directory Bucket) provides higher performance, but has a different endpoint format.

- Regular bucket: s3.us-east-1.amazonaws.com
- Directory Bucket: s3express-usw2-az1.us-west-2.amazonaws.com

For more available regions, refer to: AWS Official Documentation

Example:

```
"s3.access_key"="ak",
"s3.secret_key"="sk",
"s3.endpoint"="https://s3express-usw2-az1.us-west-2.amazonaws.com",
"s3.region"="us-west-2"
```

Permission Policies

Depending on the use case, permissions can be categorized into **read-only** and **read-write** policies.

1. Read-only Permissions

Only allows reading objects from S3. Suitable for LOAD, TVF, querying EXTERNAL CATALOG, and other scenarios.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "s3:GetObject",
              "s3:GetObjectVersion",
            ],
            "Resource": "arn:aws:s3:::<your-bucket>/your-prefix/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::<your-bucket>"
        }
    ]
}
```

2. Read-write Permissions

Based on read-only permissions, additionally allows deleting, creating, and modifying objects. Suitable for EXPORT, OUTFILE, and EXTERNAL CATALOG write-back scenarios.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
              "s3:PutObject",
              "s3:GetObject",
              "s3:GetObjectVersion",
              "s3:DeleteObject",
              "s3:DeleteObjectVersion",
              "s3:AbortMultipartUpload",
              "s3:ListMultipartUploadParts"
```

```
            ],
            "Resource": "arn:aws:s3:::<your-bucket>/<your-prefix>/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketVersioning",
                "s3:GetLifecycleConfiguration"
            ],
            "Resource": "arn:aws:s3:::<your-bucket>"
        }
    ]
}
```

Notes

1. Placeholder Replacement
    - `<bucket>` → Your S3 Bucket name.
    - `<account-id>` → Your AWS account ID (12-digit number).
2. Principle of Least Privilege
    - If only querying, do not grant write permissions.

Azure Blob

Starting from version 3.1.3, Doris supports accessing Azure Blob storage.

This document describes the parameters required to access Microsoft Azure Blob storage, which apply to the following scenarios:

- Catalog properties
- Table Valued Function properties
- Broker Load properties
- Export properties
- Outfile properties
- Backup / Restore properties

Azure Blob Storage currently does not support ARM architectures.

Configure BE CA Certificate for HTTPS

Starting from Doris 3.1.5 and 4.0.5, you can explicitly configure `ca_cert_file_paths` in `be.conf` when Doris BE accesses Azure Blob Storage over HTTPS.

By default, if `ca_cert_file_paths` is not configured, Doris uses the operating system's default CA certificates. In most environments, you do not need to set this parameter manually. Configure it in the following cases:

- The BE node is missing system CA certificates, or the installed CA bundle is too old.
- The BE node runs in a minimal container or image that does not include the `ca-certificates` package.
- The default CA file path on the BE node is invalid, or the Doris process does not have read permission on the CA file.
- Your environment uses a self-signed certificate, a private CA, or a corporate proxy or gateway that re-signs TLS traffic.
- You encounter errors such as `Problem with the SSL CA cert` or `curl 77: Problem with the SSL CA cert (path? access rights?)` when accessing Azure Blob Storage.

Example:

```
# be.conf
ca_cert_file_paths = /etc/ssl/certs/ca-certificates.crt
```

Common CA bundle paths:

- Debian / Ubuntu: `/etc/ssl/certs/ca-certificates.crt`
- CentOS / RockyLinux: `/etc/pki/ca-trust/extracted/openssl/ca-bundle.trust.crt`

Configure this item on every BE node that may access Azure Blob Storage, and ensure that the certificate file exists and is readable by the Doris process. After updating `be.conf`, restart the affected BE nodes to apply the change.

Parameter Overview

Property Name	Former Name	Description	Default Value	Required
azure.account_name		Azure storage account name (Account Name), which is the storage account name created in the Azure portal.		Yes
azure.account_key		Account Key for Azure Blob storage		Yes
azure.endpoint		Access endpoint for Azure Blob storage, typically formatted as https://.blob.core.windows.net		Yes
fs.azure.support		Whether to enable Azure Blob storage	true	Yes

- Enable Azure Blob Storage

  You must explicitly configure `"provider" = "AZURE"` or `"fs.azure.support" = "true"` to indicate that Azure Blob storage is enabled.

- Get `azure.account_name`

  1. Log in to Azure Portal
  2. Open **Storage Accounts** and select the target account
  3. You can see the **Storage Account Name** on the **Overview** page

  ```
  "azure.account_name" = "myblobstorage"
  ```

- Get `azure.account_key`

  1. Log in to Azure Portal
  2. Open **Storage Accounts** and select the target account
  3. Select **Access keys** in the left navigation bar
  4. Click “Show key” in **key1** or **key2**, and copy the **Key value**

  ```
  "azure.account_key" = "EXAMPLE_I_A...=="
  ```

Google Cloud Storage | Storages

Google Cloud Storage

The document is under development.

Aliyun OSS

This document describes the parameters required to access Alibaba Cloud OSS, which are applicable to the following scenarios:

- Catalog properties
- Table Valued Function properties
- Broker Load properties
- Export properties
- Outfile properties

OSS

Doris uses S3 Client to access Alibaba Cloud OSS through S3-compatible protocol.

Parameter Overview

Property Name	Legacy Name	Description	Default Value
oss.endpoint	s3.endpoint	OSS endpoint, specifies the access endpoint for Alibaba Cloud OSS. Note that OSS and OSS HDFS have different endpoints.	None
oss.access_key	s3.access_key	OSS Access Key for authentication	None
oss.secret_key	s3.secret_key	OSS Secret Key, used together with Access Key	None
oss.region	s3.region	OSS region, specifies the region of Alibaba Cloud OSS	None
oss.use_path_style	s3.use_path_style	Whether to use path-style access. Recommended to set to true for compatibility with MinIO and other non-AWS S3 services	FALSE
oss.connection.maximum	s3.connection.maximum	Maximum number of connections, specifies the maximum number of connections established with OSS service	50
oss.connection.request.timeout	s3.connection.request.timeout	Request timeout (milliseconds), specifies the request timeout when connecting to OSS service	3000
oss.connection.timeout	s3.connection.timeout	Connection timeout (milliseconds), specifies the timeout when establishing connection with OSS service	1000

> Before version 3.1, use legacy name

Example Configuration

```
"oss.access_key" = "your-access-key",
"oss.secret_key" = "your-secret-key",
```

```
"oss.endpoint" = "oss-cn-beijing.aliyuncs.com",
"oss.region" = "cn-beijing"
```

For versions before 3.1:

```
"s3.access_key" = "your-access-key",
"s3.secret_key" = "your-secret-key",
"s3.endpoint" = "oss-cn-beijing.aliyuncs.com",
"s3.region" = "cn-beijing"
```

Usage Recommendations

- It is recommended to use the `oss.` prefix for configuration parameters to ensure consistency and clarity with Alibaba Cloud OSS.
- For versions before 3.1, please use the legacy name `s3.` as the prefix.
- Configuring `oss.region` can improve access accuracy and performance, recommended to set.
- Connection pool parameters can be adjusted according to concurrency requirements to avoid connection blocking.

OSS-HDFS

OSS-HDFS service (JindoFS service) is an Alibaba Cloud native data lake storage functionality. Based on unified metadata management capabilities, it is compatible with HDFS file system interfaces and meets data lake computing scenarios in big data and AI fields.

Accessing data stored on OSS-HDFS is slightly different from directly accessing OSS services. Please refer to this documentation for details.

Parameter Overview

Property Name	Legacy Name	Description	Default Value	Required
oss.hdfs.endpoint	oss.endpoint	Alibaba Cloud OSS-HDFS service endpoint, e.g., `cn-hangzhou.oss-dls.aliyuncs.com`.	None	Yes
oss.hdfs.access_key	oss.access_key	OSS Access Key for authentication	None	Yes
oss.hdfs.secret_key	oss.secret_key	OSS Secret Key, used together with Access Key	None	Yes
oss.hdfs.region	oss.region	Region ID where the OSS bucket is located, e.g., `cn-beijing`.	None	Yes
oss.hdfs.fs.defaultFS		Supported in version 3.1. Specifies the file system access path for OSS, e.g., `oss://my-bucket/`.	None	No
oss.hdfs.hadoop.config.resources		Supported in version 3.1. Specifies the path containing OSS file system configuration. Requires relative path. Default directory is `/plugins/hadoop_conf/` under the (FE/BE) deployment directory (can be changed by modifying hadoop_config_dir in fe.conf/be.conf). All FE and BE nodes need to configure the same relative path. Example: `hadoop/conf/core-site.xml,hadoop/conf/hdfs-site.xml`.	None	No

Property Name	Legacy Name	Description	Default Value	Required
fs.oss-hdfs.support	oss.hdfs.enabled	Supported in version 3.1. Explicitly declares the enabling of OSS-HDFS functionality. Needs to be set to true	None	No

> For versions before 3.1, please use legacy names.

Endpoint Configuration

`oss.hdfs.endpoint`: Used to specify the OSS-HDFS service endpoint.

Endpoint is the entry address for accessing Alibaba Cloud OSS, formatted as `<region>.oss-dls.aliyuncs.com`, e.g., `cn-hangzhou.oss-dls.aliyuncs.com`.

We perform strict format validation to ensure the endpoint conforms to the Alibaba Cloud OSS endpoint format.

For backward compatibility, the endpoint configuration allows the inclusion of https:// or http:// prefixes. The system will automatically parse and ignore the protocol part during format validation.

When using legacy names, the system determines whether it's an OSS-HDFS service based on whether the `endpoint` contains `oss-dls`.

Configuration Files

> Supported in version 3.1

OSS-HDFS supports specifying HDFS-related configuration file directories through the `oss.hdfs.hadoop.config.resources` parameter.

The configuration file directory must contain `hdfs-site.xml` and `core-site.xml` files. The default directory is `/plugins/hadoop_conf/` under the (FE/BE) deployment directory. All FE and BE nodes need to configure the same relative path.

If the configuration files contain the parameters mentioned above in this document, the explicitly configured parameters by the user take precedence. Configuration files can specify multiple files, separated by commas, such as `hadoop/conf/core-site.xml,hadoop/conf/hdfs-site.xml`.

Example Configuration

```
"fs.oss-hdfs.support" = "true",
"oss.hdfs.access_key" = "your-access-key",
"oss.hdfs.secret_key" = "your-secret-key",
"oss.hdfs.endpoint" = "cn-hangzhou.oss-dls.aliyuncs.com",
"oss.hdfs.region" = "cn-hangzhou"
```

For versions before 3.1:

```
"oss.hdfs.enabled" = "true",
"oss.access_key" = "your-access-key",
"oss.secret_key" = "your-secret-key",
"oss.endpoint" = "cn-hangzhou.oss-dls.aliyuncs.com",
"oss.region" = "cn-hangzhou"
```

Tencent COS

This document describes the parameters required to access Tencent Cloud COS, which apply to the following scenarios:

- Catalog properties
- Table Valued Function properties
- Broker Load properties
- Export properties
- Outfile properties

Doris uses S3 Client to access Tencent Cloud COS through S3-compatible protocol.

Parameter Overview

Property Name	Legacy Name	Description	Default Value	Required
cos.endpoint	s3.endpoint	COS endpoint, specifies the access endpoint for Tencent Cloud COS		Yes
cos.access_key	s3.access_key	COS access key, used for authentication to access COS		Yes
cos.secret_key	s3.secret_key	COS secret key, used together with access key for authentication		Yes
cos.region	s3.region	COS region, specifies the region of Tencent Cloud COS		No
cos.connection.maximum	s3.connection.maximum	S3 maximum connections, specifies the maximum number of connections to COS service	50	No
cos.connection.request.timeout	s3.connection.request.timeout	S3 request timeout in milliseconds, specifies the request timeout when connecting to COS service	3000	No
cos.connection.timeout	s3.connection.timeout	S3 connection timeout in milliseconds, specifies the timeout when establishing connection to COS service	1000	No

> Before version 3.1, use legacy name

Configuration Examples

```
"cos.access_key" = "your-access-key",
"cos.secret_key" = "your-secret-key",
"cos.endpoint" = "cos.ap-beijing.myqcloud.com",
"cos.region" = "ap-beijing"
```

For versions before 3.1:

```
"s3.access_key" = "ak",
"s3.secret_key" = "sk",
"s3.endpoint" = "cos.ap-beijing.myqcloud.com",
"s3.region" = "ap-beijing"
```

Usage Recommendations

- It is recommended to use the `cos.` prefix for configuration parameters to ensure consistency and clarity with Tencent Cloud COS.
- For versions before 3.1, please use the legacy `s3.` prefix.
- Configuring `cos.region` can improve access accuracy and performance, and it is recommended to set it.
- Connection pool parameters can be adjusted according to concurrency requirements to avoid connection blocking.

Huawei OBS

This document describes the parameters required to access Huawei Cloud OBS, which are applicable to the following scenarios:

- Catalog properties
- Table Valued Function properties
- Broker Load properties
- Export properties
- Outfile properties

Doris uses S3 Client to access Huawei Cloud OBS through S3-compatible protocol.

Parameter Overview

Property Name	Former Name	Description	Default Value	Required
obs.endpoint	s3.endpoint	OBS endpoint, specifies the access endpoint for Huawei Cloud OBS		Yes
obs.access_key	s3.access_key	OBS access key, used for authentication		Yes
obs.secret_key	s3.secret_key	OBS secret key, used together with access key for authentication		Yes
obs.region	s3.region	OBS region, specifies the region of Huawei Cloud OBS		No
obs.use_path_style	s3.use_path_style	Whether to use path-style access. Recommended to set to true for compatibility with non-AWS S3 services like MinIO/Ceph	FALSE	No
obs.connection.maximum	s3.connection.maximum	Maximum number of connections to OBS service	50	No
obs.connection.request.timeout	s3.connection.request.timeout	Request timeout in milliseconds for connecting to OBS service	3000	No
obs.connection.timeout	s3.connection.timeout	Connection timeout in milliseconds for establishing connection to OBS service	1000	No

> Before version 3.1, use legacy name

Example Configuration

```
"obs.access_key" = "your-access-key",
"obs.secret_key" = "your-secret-key",
"obs.endpoint" = "obs.cn-north-4.myhuaweicloud.com",
"obs.region" = "cn-north-4"
```

For versions before 3.1:

```
"s3.access_key" = "your-access-key",
"s3.secret_key" = "your-secret-key",
"s3.endpoint" = "obs.cn-north-4.myhuaweicloud.com",
"s3.region" = "cn-north-4",
```

Usage Recommendations

- It is recommended to use the `obs.` prefix for configuration parameters to ensure consistency and clarity with Huawei Cloud OBS.
- For versions before 3.1, please use the former name with `s3.` prefix.
- Configuring `obs.region` can improve access accuracy and performance, it is recommended to set it.
- Connection pool parameters can be adjusted according to concurrency requirements to avoid connection blocking.

Baidu BOS

Document is under development.

MinIO | Storages

MinIO

This document describes the parameters required to access MinIO, which apply to the following scenarios:

- Catalog properties
- Table Valued Function properties
- Broker Load properties
- Export properties
- Outfile properties

Doris uses S3 Client to access MinIO through S3-compatible protocol.

Parameter Overview

Property Name	Legacy Name	Description	Default Value	Required
minio.endpoint	s3.endpoint	MinIO endpoint, the access endpoint for MinIO		Yes
minio.access_key	s3.access_key	MinIO access key, the MinIO access key used for authentication		Yes
minio.secret_key	s3.secret_key	MinIO secret key, the secret key used together with access key		Yes
minio.connection.maximum	s3.connection.maximum	S3 maximum connections, specifies the maximum number of connections established with MinIO service	50	No
minio.connection.request.timeout	s3.connection.request.timeout	S3 request timeout, in milliseconds, specifies the request timeout when connecting to MinIO service	3000	No
minio.connection.timeout	s3.connection.timeout	S3 connection timeout, in milliseconds, specifies the timeout when establishing connection with MinIO service	1000	No
minio.use_path_style	s3.use_path_style	Whether to use path-style access. Recommended to set to true for compatibility with MinIO and other non-AWS S3 services	FALSE	No

Using Path-style Access

MinIO uses Host-style access by default, but also supports Path-style access. You can switch by setting the `minio.use_path_style` parameter.

- Host-style access (default): `https://bucket.minio.example.com`
- Path-style access (when enabled): `https://minio.example.com/bucket`

Example Configuration

```
"minio.access_key" = "your-access-key",
"minio.secret_key" = "your-secret-key",
"minio.endpoint" = "http://minio.example.com:9000"
```

For versions before 3.1:

```
"s3.access_key" = "your-access-key",
"s3.secret_key" = "your-secret-key",
"s3.endpoint" = "http://minio.example.com:9000"
```

Usage Recommendations

- It is recommended to use the `minio.` prefix for configuration parameters to ensure consistency and clarity with MinIO.
- For versions before 3.1, please use the legacy name `s3.` as the prefix.
- Connection pool parameters can be adjusted according to concurrency requirements to avoid connection

Parquet | File Formats

Parquet

This document introduces the support for reading and writing Parquet file formats in Doris. It applies to the following features:

- Reading and writing data in the Catalog.
- Reading data using Table Valued Functions.
- Reading data with Broker Load.
- Writing data during Export.
- Writing data with Outfile.

Supported Compression Formats

- uncompressed
- snappy
- lz4
- zstd
- gzip
- lzo
- brotli

Parameters

Session Variables

- `enable_parquet_lazy_mat` (2.1+, 3.0+)

  Controls whether the Parquet Reader enables lazy materialization. Default is true.

- `hive_parquet_use_column_names` (2.1.6+, 3.0.3+)

  When reading Parquet data types from Hive tables, Doris will, by default, read data from columns in the Parquet file that have the same name as the columns in the Hive table. When this variable is set to `false`, Doris will read data from the Parquet file based on the column order in the Hive table, regardless of column names. This is similar to the `parquet.column.index.access` variable in Hive. This parameter only applies to top-level column names and is ineffective for columns inside Structs.

BE Configuration

- `enable_parquet_page_index` (2.1.5+, 3.0+)

  Determines whether the Parquet Reader uses the Page Index to filter data. This is only for debugging purposes, in case the page index sometimes filters incorrect data. Default value is false.

- `parquet_header_max_size_mb` (2.1+, 3.0+)

  The maximum buffer size allocated when reading the Parquet Page header. Default is 1M.

- `parquet_rowgroup_max_buffer_mb` (2.1+, 3.0+)

  The maximum buffer size allocated when reading a Parquet Row Group. Default is 128M.

- `parquet_column_max_buffer_mb` (2.1+, 3.0+)

  The maximum buffer size allocated when reading a Column within a Parquet Row Group. Default is 8M.

ORC | File Formats

ORC

This document introduces the support for reading and writing ORC file formats in Doris. It applies to the following functionalities:

- Reading and writing data in the Catalog.
- Reading data using Table Valued Functions.
- Reading data with Broker Load.
- Writing data during Export.
- Writing data with Outfile.

Supported Compression Formats

- uncompressed
- snappy
- lz4
- zstd
- lzo
- zlib

Parameters

Session Variables

- `enable_orc_lazy_mat` (2.1+, 3.0+)

  Controls whether the ORC Reader enables lazy materialization. Default is true.

- `hive_orc_use_column_names` (2.1.6+, 3.0.3+)

  When reading ORC data types from Hive tables, Doris will, by default, read data from columns in the ORC file that have the same name as the columns in the Hive table. When this variable is set to `false`, Doris will read data from the ORC file based on the column order in the Hive table, regardless of column names. This is similar to the `orc.force.positional.evolution` variable in Hive. This parameter only applies to top-level column names and is ineffective for columns inside Structs.

- `orc_tiny_stripe_threshold_bytes` (2.1.8+, 3.0.3+)

  In ORC files, if the byte size of a Stripe is less than `orc_tiny_stripe_threshold`, it is considered a Tiny Stripe. For multiple consecutive Tiny Stripes, read optimization will be performed, i.e., multiple Tiny Stripes will be read at once to reduce the number of IO operations. If you do not want to use this optimization, you can set this value to 0. Default is 8M.

- `orc_once_max_read_bytes` (2.1.8+, 3.0.3+)

When using Tiny Stripe read optimization, multiple Tiny Stripes will be merged into a single IO operation. This parameter controls the maximum number of bytes for each IO request. You should not set this value smaller than `orc_tiny_stripe_threshold`. Default is 8M.

- `orc_max_merge_distance_bytes` (2.1.8+, 3.0.3+)

  When using Tiny Stripe read optimization, since two Tiny Stripes to be read may not be consecutive, if the distance between two Tiny Stripes is greater than this parameter, they will not be merged into a single IO operation. Default is 1M.

- `orc_tiny_stripe_amplification_factor` (3.1.0+)

  In Tiny Stripe optimization, if there are many columns in the ORC file but only a few are used in the query, Tiny Stripe optimization may cause severe read amplification. When the proportion of actually read bytes to the entire Stripe exceeds this parameter, Tiny Stripe read optimization will be used. The default value is 0.4, and the minimum value is 0.

- `check_orc_init_sargs_success` (3.1.0+)

  Checks whether ORC predicate pushdown is successful, used for debugging. Default is false.

BE Configuration

- `orc_natural_read_size_mb` (2.1+, 3.0+)

  The maximum number of bytes that the ORC Reader reads at one time. Default is 8 MB.

Text/CSV/JSON

This document introduces the support for reading and writing text file formats in Doris.

Text/CSV

- Catalog

  Supports reading Hive tables in the `org.apache.hadoop.mapred.TextInputFormat` format.

  Support following SerDes:

  - `org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe`
  - `org.apache.hadoop.hive.serde2.OpenCSVSerde` (Since 2.1.7)
  - `org.apache.hadoop.hive.serde2.MultiDelimitSerDe` (Since 3.1.0)

- Table Valued Function

- Import

  Import functionality supports Text/CSV formats. See the import documentation for details.

- Export

  Export functionality supports Text/CSV formats. See the export documentation for details.

Supported Compression Formats

- uncompressed
- gzip
- deflate
- bzip2
- zstd
- lz4
- snappy
- lzo

JSON

Catalog

- `org.apache.hadoop.hive.serde2.JsonSerDe` (Since 3.0.4)
- `org.apache.hive.hcatalog.data.JsonSerDe` (Since 3.0.4)

  1. Supports both primitive and complex types.
  2. Does not support the `timestamp.formats` SERDEPROPERTIES.

- Hive table in `org.openx.data.jsonserde.JsonSerDe` (Since 3.0.6)

    1. Supports both primitive and complex types.
    2. SERDEPROPERTIES: Only `ignore.malformed.json` is supported and behaves the same as in this JsonSerDe. Other SERDEPROPERTIES are not effective.
    3. Does not support `Using Arrays` (similar to Text/CSV format, where all column data is placed into a single array).
    4. Does not support `Promoting a Scalar to an Array` (promoting a scalar to a single-element array).
    5. By default, Doris can correctly recognize the table schema. However, due to the lack of support for certain parameters, automatic schema recognition might fail. In this case, you can set `read_hive_json_in_one_column = true` to place the entire JSON row into the first column to ensure the original data is fully read. Users can then process it manually. This feature requires the first column's data type to be `String`.

Import

Import functionality supports JSON formats. See the import documentation for details.

Character Set

Currently, Doris only supports the UTF-8 character set encoding. However, some data, such as the data in Hive Text-formatted tables, may contain content encoded in non-UTF-8 encoding, which will cause reading failures and result in the following error:

```
Only support csv data in utf8 codec
```

In this case, you can set the session variable as follows:

```
SET enable_text_validate_utf8 = false
```

This will ignore the UTF-8 encoding check, allowing you to read this content. Note that this parameter is only used to skip the check, and non-UTF-8 encoded content will still be displayed as garbled text.

This parameter has been supported since version 3.0.4.

Data Cache

Data Cache accelerates subsequent queries of the same data by caching recently accessed data files from remote storage systems (HDFS or object storage) to local disks. In scenarios where the same data is frequently accessed, Data Cache can avoid the overhead of repeated remote data access, improving the performance and stability of query analysis on hot data.

Applicable Scenarios

The data cache function only works for queries on Hive, Iceberg, Hudi, and Paimon tables. It has no effect on internal table queries or non-file external table queries (such as JDBC, Elasticsearch).

Whether data caching can improve query efficiency depends on multiple factors. Below are the applicable scenarios for data caching:

- High-speed local disk

  It is recommended to use high-speed local disks, such as SSD or NVME media local disks, as the data cache directory. It is not recommended to use mechanical hard drives as the data cache directory. Essentially, the local disk's IO bandwidth and IOPS must be significantly higher than the network bandwidth and the source storage system's IO bandwidth and IOPS to bring noticeable performance improvements.

- Sufficient cache space size

  Data caching uses the LRU strategy as the cache eviction policy. If the queried data does not have a clear distinction between hot and cold, the cached data may be frequently updated and replaced, which may reduce query performance. It is recommended to enable data caching in scenarios where the query pattern has a clear distinction between hot and cold (e.g., most queries only access today's data and rarely access historical data), and the cache space is sufficient to store hot data.

- Unstable IO latency of remote storage

  This situation usually occurs on HDFS storage. In most enterprises, different business departments share the same HDFS, which may lead to very unstable IO latency during peak periods. In this case, if you need to ensure stable IO latency, it is recommended to enable data caching. However, the first two conditions should still be considered.

Enabling Data Cache

The data cache function is disabled by default and needs to be enabled by setting relevant parameters in FE and BE.

BE Configuration

First, configure the cache path information in `be.conf` and restart the BE node to make the configuration effective.

Parameter	Required	Description
enable_file_cache	Yes	Whether to enable Data Cache, default is false
file_cache_path	Yes	Configuration related to the cache directory, in JSON format.
clear_file_cache	No	Default is false. If true, the cache directory will be cleared when the BE node restarts.

Example configuration of `file_cache_path`:

```
file_cache_path=[{"path": "/path/to/file_cache1", "total_size":53687091200},{"path": "/path/to/file_cache2",
"total_size":53687091200},{"path": "/path/to/file_cache3", "total_size":53687091200}]
```

`path` is the path where the cache is stored, and one or more paths can be configured. It is recommended to configure only one path per disk.

`total_size` is the upper limit of the cache space size, in bytes. When the cache space is exceeded, the LRU strategy will be used to evict cached data.

FE Configuration

Enable Data Cache in a single session:

```
SET enable_file_cache = true;
```

Enable Data Cache globally:

```
SET GLOBAL enable_file_cache = true;
```

Note that if `enable_file_cache` is not enabled, the cache will not be used even if the BE is configured with a cache directory. Similarly, if the BE is not configured with a cache directory, the cache will not be used even if `enable_file_cache` is enabled.

Cache Observability

View Cache Hit Rate

Execute `set enable_profile=true` to open the session variable, and you can view the job's Profile on the `Queries` tab of the FE web page. The data cache-related metrics are as follows:

```
-  FileCache:  0ns
    -  BytesScannedFromCache:  2.02  GB
    -  BytesScannedFromRemote:  0.00
    -  BytesWriteIntoCache:  0.00
    -  LocalIOUseTimer:  2s723ms
    -  NumLocalIOTotal:  444
    -  NumRemoteIOTotal:  0
    -  NumSkipCacheIOTotal:  0
```

```
    -  RemoteIOUseTimer:  0ns
    -  WriteCacheIOUseTimer:  0ns
```

- `BytesScannedFromCache`: The amount of data read from the local cache.
- `BytesScannedFromRemote`: The amount of data read from the remote.
- `BytesWriteIntoCache`: The amount of data written into the cache.
- `LocalIOUseTimer`: The IO time of the local cache.
- `RemoteIOUseTimer`: The IO time of remote reading.
- `NumLocalIOTotal`: The number of IO operations on the local cache.
- `NumRemoteIOTotal`: The number of remote IO operations.
- `WriteCacheIOUseTimer`: The IO time of writing into the cache.

If `BytesScannedFromRemote` is 0, it means the cache is fully hit.

Monitoring Metrics

Users can view cache statistics for each Backend node through the system table `file cache statistics`.

Appendix

Principle

Data caching caches accessed remote data to the local BE node. The original data file is split into Blocks based on the accessed IO size, and Blocks are stored in the local file `cache_path/hash(filepath).substr(0, 3)/hash(filepath)/offset`, with Block metadata saved in the BE node. When accessing the same remote file, doris checks whether the cache data of the file exists in the local cache and determines which data to read from the local Block and which data to pull from the remote based on the Block's offset and size, caching the newly pulled remote data. When the BE node restarts, it scans the `cache_path` directory to restore Block metadata. When the cache size reaches the upper limit, it cleans up long-unused Blocks according to the LRU principle.

Metadata Cache

To improve the performance of accessing external data sources, Apache Doris caches the **metadata** of external data sources.

Metadata includes information such as databases, tables, columns, partitions, snapshots, file lists, etc.

This article details the types, strategies, and related parameter configurations of cached metadata.

For **data cache**, refer to the data cache documentation.

> **This document applies to versions after 2.1.6.**

Cache Strategies

Most caches have the following three strategy indicators:

- Maximum cache count

  The maximum number of objects the cache can hold. For example, up to 1000 tables can be cached. When the cache count exceeds the threshold, the LRU (Least-Recent-Used) strategy is used to remove some caches.

- Eviction time

  - Before version 3.0.6 (inclusive):

    After a cache object is written to the cache for a period of time, the object will be automatically removed from the cache. The next access will re-fetch the latest information from the data source and update the cache.

    For example, a user first accesses table A at 08:00 and writes it to the cache. If the eviction time is 4 hours, then, unless replaced due to capacity issues, the user will directly access table A in the cache between 08:00-14:00. After 14:00, the cache is evicted. If the user accesses table A again, the latest information will be fetched from the data source and the cache will be updated.

  - After version 3.0.7 (inclusive):

    Starting from version 3.0.7, this strategy is changed to **the cache object will be automatically removed after being accessed** for a period of time, instead of **after being written** for a period of time. Each time the cache object is accessed, the timer is reset to ensure that frequently accessed objects always remain in the cache.

    For example, a user first accesses table A at 08:00 and writes it to the cache. If the eviction time is 4 hours, then, unless replaced due to capacity issues, the user will directly access table A in the cache between 08:00-14:00. Suppose the user accesses this object again at 09:00, then the cache eviction time will be recalculated from 09:00, i.e., it becomes 15:00.

- Minimum refresh time

  After a cache object is written to the cache for a period of time, it will automatically trigger a refresh.

  For example, a user first accesses table A at 08:00 and writes it to the cache. If the minimum refresh time is 10 minutes, then, unless replaced due to capacity issues, the user will directly access table A in the cache between 08:00-8:10. At 08:10, the cache object will be marked as [ready to refresh]. When the user accesses this cache object again, the current object will still be returned, but the cache refresh operation will be triggered at the same time. Suppose the cache update takes 1 minute, then after 1 minute, accessing the cache again will get the updated cache object.

  Note that the time to trigger the cache refresh is [the first access to the cache object after exceeding the minimum refresh time], and it is an asynchronous refresh. So, for example, if the minimum refresh time is 10 minutes, it does not mean that the latest object will be obtained after 10 minutes.

  This strategy is different from [eviction time], mainly used to adjust the timeliness of the cache, and avoids blocking the current operation by asynchronously refreshing the cache.

Cache Types

Database and Table Name Lists

The database name list refers to the list of all database names under a Catalog.

The table name list refers to the list of all table names under a database.

The name list is only used for operations that need to enumerate names, such as the `SHOW TABLES` or `SHOW DATABASES` statements.

Each Catalog has a database name list cache. Each database has a table name list cache.

- Maximum cache count

  Each cache has only one entry. So the maximum cache count is 1.

- Eviction time

  Fixed at 86400 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest name list in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Database and Table Objects

Caches individual database and table objects. Any access operation to a database or table, such as query, write, etc., will obtain the corresponding object from this cache.

Note that the list of objects in this cache may be inconsistent with the **database and table name list** cache.

For example, through the `SHOW TABLES` command, you get tables `A`, `B`, and `C` from the name list cache. Suppose table `D` is added to the external data source at this time, then `SELECT * FROM D` can access table `D`, and the [table object] cache will add the table `D` object, but the [table name list] cache may still be `A`, `B`, `C`. Only when the [table name list] cache is refreshed will it become `A`, `B`, `C`, `D`.

Each Catalog has a database name list cache. Each database has a table name list cache.

- Maximum cache count

  Controlled by the FE configuration item `max_meta_object_cache_num`, default is 1000. You can adjust this parameter appropriately according to the number of databases under a single Catalog or the number of tables under a single database.

- Eviction time

  Fixed at 86400 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest database or table in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Table Schema

Caches the schema information of tables, such as column names. This cache is mainly used to load the schema of accessed tables on demand, to prevent synchronizing a large number of unnecessary table schemas and occupying FE memory.

This cache is shared by all Catalogs and is globally unique.

- Maximum cache count

  Controlled by the FE configuration item `max_external_schema_cache_num`, default is 10000.

  You can adjust this parameter appropriately according to the total number of tables under a Catalog.

- Eviction time

  Fixed at 86400 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest schema in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Hive Metastore Table Partition List

Used to cache the partition list of tables synchronized from Hive Metastore. The partition list is used for partition pruning during queries.

This cache, each Hive Catalog has one.

- Maximum cache count

  Controlled by the FE configuration item `max_hive_partition_table_cache_num`, default is 1000.

  You can adjust this parameter appropriately according to the total number of tables under a Catalog.

- Eviction time

  Fixed at 28800 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest partition list in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Hive Metastore Table Partition Properties

Used to cache the properties of each partition of a Hive table, such as file format, partition root path, etc. For each query, after partition pruning to get the list of partitions to be accessed, the detailed properties of each partition are obtained through this cache.

This cache, each Hive Catalog has one.

- Maximum cache count

  Controlled by the FE configuration item `max_hive_partition_cache_num`, default is 10000.

  You can adjust this parameter appropriately according to the total number of partitions to be accessed under a Catalog.

- Eviction time

Fixed at 28800 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest partition properties in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Hive Metastore Table Partition File List

Used to cache the file list information under a single partition of a Hive table. This cache is used to reduce the overhead of file system List operations.

- Maximum cache count

  Controlled by the FE configuration item `max_external_file_cache_num`, default is 100000.

  You can adjust this parameter appropriately according to the number of files to be accessed.

- Eviction time

  Default is 28800 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

  If the `file.meta.cache.ttl-second` property is set in the Catalog properties, the set time is used.

  In some cases, the data files of Hive tables change frequently, resulting in the cache not meeting timeliness requirements. You can set this parameter to 0 to disable this cache. In this case, the file list will be obtained in real time for each query, which may reduce performance but improve file timeliness.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest partition properties in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Hudi Table Partitions

Used to cache partition information of Hudi tables.

This cache, each Hudi Catalog has one.

- Maximum cache count

  Controlled by the FE configuration item `max_external_table_cache_num`, default is 1000.

You can adjust this parameter appropriately according to the number of Hudi tables.

- Eviction time

  Fixed at 28800 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest Hudi partition properties in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Iceberg Table Information

Used to cache Iceberg table objects. The object is loaded and constructed through the Iceberg API.

This cache, each Iceberg Catalog has one.

- Maximum cache count

  Controlled by the FE configuration item `max_external_table_cache_num`, default is 1000.

  You can adjust this parameter appropriately according to the number of Iceberg tables.

- Eviction time

  Fixed at 28800 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest Iceberg table properties in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Iceberg Table Snapshot

Used to cache the snapshot list of Iceberg tables. The object is loaded and constructed through the Iceberg API. This cache, each Iceberg Catalog has one.

- Maximum cache count

  Controlled by the FE configuration item `max_external_table_cache_num`, default is 1000.

  You can adjust this parameter appropriately according to the number of Iceberg tables.

- Eviction time

  Fixed at 28800 seconds. After version 3.0.7, configured by the FE parameter `external_cache_expire_time_seconds_after_access`, default is 86400 seconds.

- Minimum refresh time

  Controlled by the FE configuration item `external_cache_expire_time_minutes_after_access`, in minutes. Default is 10 minutes. Reducing this time allows you to see the latest Iceberg table properties in Doris more in real time, but increases the frequency of accessing external data sources.

  After version 3.0.7, the configuration item name is changed to `external_cache_refresh_time_minutes`. The default value remains unchanged.

Cache Refresh

In addition to the refresh and eviction strategies of each cache above, users can also directly refresh the metadata cache manually or on a schedule.

Manual Refresh

Users can manually refresh metadata using the `REFRESH` command.

1. REFRESH CATALOG

   Refresh the specified Catalog.

   ```
   REFRESH CATALOG ctl1 PROPERTIES("invalid_cache" = "true");
   ```

   - This command refreshes the database list, table column names, and all cache information of the specified Catalog.
   - `invalid_cache` indicates whether to refresh caches such as partitions and file lists. The default is true. If false, only the database and table lists of the Catalog will be refreshed, but not caches such as partitions and file lists. This parameter is suitable for cases where the user only wants to synchronize newly added or deleted databases and tables, and can be set to false.

2. REFRESH DATABASE

   Refresh the specified Database.

   ```
   REFRESH DATABASE [ctl.]db1 PROPERTIES("invalid_cache" = "true");
   ```

   - This command refreshes the table column names and all cache information under the specified Database.
   - The meaning of the `invalid_cache` property is the same as above. The default is true. If false, only the table list of the Database will be refreshed, but not the cache information. This parameter is suitable for cases where the user only wants to synchronize newly added or deleted tables.

3. REFRESH TABLE

   Refresh the specified Table.

```
REFRESH TABLE [ctl.][db.]tbl1;
```

- This command refreshes all cache information under the specified Table.

Scheduled Refresh

Users can set up scheduled refreshes for a Catalog when creating it.

```
CREATE CATALOG hive PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://172.0.0.1:9083',
    'metadata_refresh_interval_sec' = '3600'
);
```

In the above example, `metadata_refresh_interval_sec` means the Catalog is refreshed every 3600 seconds. This is equivalent to automatically executing once every 3600 seconds:

```
REFRESH CATALOG ctl1 PROPERTIES("invalid_cache" = "true");
```

Best Practices

Caching can significantly improve metadata access performance and avoid frequent remote access to metadata, which can cause performance jitter or put pressure on the metadata service. However, caching also reduces data timeliness. For example, if the cache refresh time is 10 minutes, only cached metadata can be read within ten minutes. Therefore, it is necessary to set the cache reasonably according to the situation.

Default Behavior

This section mainly introduces the cache behavior that users may be concerned about under the default parameter configuration.

- After a new database or table is added to the external data source, it can be queried in real time in Doris through SELECT. But SHOW DATABASES and SHOW TABLES may not see it; you need to manually refresh the cache or wait up to 10 minutes.
- When a new partition is added to the external data source, you need to manually refresh the cache or wait up to 10 minutes to query the new partition data.
- When partition data files change, you need to manually refresh the cache or wait up to 10 minutes to query the new partition data.

Disable Schema Cache

For all types of External Catalogs, if you want to see the latest Table Schema in real time, you can disable the Schema Cache:

- Disable globally

  ```
  -- fe.conf
  max_external_schema_cache_num=0 // Disable Schema cache.
  ```

- Disable at Catalog level

```
-- Catalog property
"schema.cache.ttl-second" = "0" // For a specific Catalog, disable Schema cache (supported in 2.1.11, 3.0.6)
```

After setting, Doris will see the latest Table Schema in real time. However, this setting may increase the pressure on the metadata service.

Disable Hive Catalog Metadata Cache

For Hive Catalog, if you want to disable the cache to query real-time updated data, you can configure the following parameters:

- Disable globally

```
-- fe.conf
max_external_file_cache_num=0    // Disable file list cache
max_hive_partition_table_cache_num=0  // Disable partition list cache
```

- Disable at Catalog level

```
-- Catalog property
"file.meta.cache.ttl-second" = "0" // For a specific Catalog, disable file list cache
"partition.cache.ttl-second" = "0" // For a specific Catalog, disable partition list cache (supported in 2.1.11, 3.0.6)
```

After setting the above parameters:

- New partitions in the external data source can be queried in real time.
- Changes in partition data files can be queried in real time.

But this will increase the access pressure on external data sources (such as Hive Metastore and HDFS), which may cause unstable metadata access latency and other phenomena.

Elastic Compute Node

Elastic compute nodes, as a special type of BE node, do not have data storage capabilities and are only responsible for data computation. Therefore, compute nodes can be regarded as stateless BE nodes, which can be easily added and removed.

In lakehouse data analysis scenarios, elastic compute nodes can be used to query external data sources such as Hive, Iceberg, Hudi, Paimon, JDBC, etc. Doris does not handle the storage of external data source data, so elastic compute nodes can be used to easily expand the computing power for external data sources. Additionally, compute nodes can also be configured with cache directories to cache hot data from external data sources, further accelerating data reading.

Elastic compute nodes **are suitable for elastic resource control in the integrated storage and computing mode of Doris**. In the storage-compute separation architecture of Doris 3.0, BE nodes are stateless, so separate elastic compute nodes are no longer needed.

Usage of Compute Nodes

BE Node Types

In the integrated storage and computing mode, BE nodes are divided into two types:

- Mix

  Mixed nodes. This is the default type of BE node. These nodes participate in both computation and the storage of Doris internal table data.

- Computation

  Elastic compute nodes. They do not handle data storage, only data computation.

Adding Compute Nodes

Add the following configuration in the BE `be.conf` configuration file:

```
be_node_role=computation
```

Afterwards, start the BE node, and it will run as a Computation type.

Then you can connect to Doris and execute:

```
ALTER SYSTEM ADD BACKEND
```

to add this BE node. Once added successfully, you can see the node type as `computation` in the `NodeRole` column of `SHOW BACKENDS`.

Using Compute Nodes

You need to configure the following parameters in the FE configuration file `fe.conf` to enable compute nodes and control their behavior:

Parameter Name	Description
`prefer_compute_node_for_external_table`	Default is `false`. If set to `true`, queries for external tables will be preferentially assigned to compute nodes. If `false`, queries for external tables will be assigned to any BE node. If there are no compute nodes in the cluster, this parameter has no effect.
`min_backend_num_for_external_table`	Only effective when `prefer_compute_node_for_external_table` is `true`. If the number of compute nodes in the cluster is less than this value, queries for external tables will attempt to acquire some mixed nodes to allocate, so that the total number of nodes reaches this value. If the number of compute nodes in the cluster is greater than this value, queries for external tables will only be assigned to compute nodes. Before version 2.0 (inclusive), the default value of this parameter was 3. After version 2.1, the default value is `-1`, which means only the current number of compute nodes will be used.

Further explanation of `min_backend_num_for_external_table`:

Assume there are 3 compute nodes and 5 mixed nodes in the cluster.

If `min_backend_num_for_external_table` is set to less than or equal to 3, then external table queries will only use 3 compute nodes. If set to greater than 3, say 6, then external table queries will use 3 compute nodes plus an additional 3 mixed nodes for computation.

In summary, this parameter is mainly used for the minimum number of BE nodes that can participate in external table computation, and will preferentially select compute nodes. Increasing this parameter will allow more BE nodes (not limited to compute nodes) to participate in external table query processing; decreasing this parameter can limit the number of BE nodes participating in external table query processing.

> Note:
>
> 1. After version 2.1, `min_backend_num_for_external_table` can be set to `-1`. In previous versions, this parameter must be a positive number. This parameter is only effective when `prefer_compute_node_for_external_table = true`.
> 2. If the value of `min_backend_num_for_external_table` is greater than the total number of BE nodes, at most all BEs will be selected.
> 3. The above parameters can be dynamically modified using the `ADMIN SET FRONTEND CONFIG` command without restarting the FE node. All FE nodes need to be configured. Alternatively, add the configuration in `fe.conf` and restart the FE node.

Best Practices

Resource Isolation and Elastic Scaling of Federated Queries

In federated query scenarios, users can deploy a dedicated set of compute nodes for querying external table data. This allows the isolation of external table query loads (such as large-scale analysis on Hive) from internal

table query loads (such as low-latency fast data analysis).

At the same time, as stateless BE nodes, compute nodes can be easily scaled up and down. For example, a set of elastic compute node clusters can be deployed using k8s, utilizing more compute nodes for data lake analysis during peak business periods, and quickly scaling down during off-peak periods to reduce costs.

Common Issues

1. Can mixed nodes and compute nodes be converted to each other?

Compute nodes can be converted to mixed nodes. However, mixed nodes cannot be converted to compute nodes.

2. Do compute nodes need to configure data storage directories?

Yes. The data storage directory of compute nodes will not store user data, only some information files of the BE node itself, such as `cluster_id`, and some temporary files during operation.

The storage directory of compute nodes requires only a small amount of disk space (MB level) and can be destroyed at any time along with the node without affecting user data.

3. Can compute nodes and mixed nodes configure file cache directories?

File cache accelerates subsequent queries of the same data by caching data files from recently accessed remote storage systems (HDFS or object storage).

Both compute nodes and mixed nodes can set up file cache directories. The file cache directory needs to be created in advance.

4. Do compute nodes need to be decommissioned through the DECOMMISION operation?

No. Compute nodes can be directly deleted using the `DROP BACKEND` operation.

Statistics | Lakehouse

Statistics

Doris supports automatic or manual statistics collection for tables from external data sources like Hive, Iceberg and Paimon. The accuracy of statistics directly determines the accuracy of cost estimation, which is crucial for selecting the optimal query plan. This can significantly improve query execution efficiency, especially in complex query scenarios.

For details, please refer to the Statistics document in the “External Table Collection” section.

Data Lake Query Optimization

This document mainly introduces optimization methods and strategies for querying lake data (Hive, Iceberg, Paimon, etc.).

Partition Pruning

By specifying partition column conditions in queries, unnecessary partitions can be pruned, reducing the amount of data that needs to be read.

You can use `EXPLAIN <SQL>` to view the `partition` section of `XXX_SCAN_NODE` to check whether partition pruning is effective and how many partitions need to be scanned in this query.

For example:

```
0:VPAIMON_SCAN_NODE(88)
    table: paimon_ctl.db.table
    predicates: (user_id[#4] = 431304818)
    inputSplitNum=15775, totalFileSize=951754154566, scanRanges=15775
    partition=203/0
```

Local Data Cache

Data Cache accelerates subsequent queries accessing the same data by caching recently accessed data files from remote storage systems (HDFS or object storage) to local disk.

The cache feature is disabled by default. Please refer to the Data Cache documentation to configure and enable it.

Since version 4.0.2, cache warmup functionality is supported, which can further actively utilize data cache to improve query performance.

HDFS Read Optimization

Please refer to the **HDFS IO Optimization** section in the HDFS Documentation.

Merge IO Optimization

For remote storage systems like HDFS and object storage, Doris optimizes IO access through Merge IO technology. Merge IO technology essentially merges multiple adjacent small IO requests into one large IO request, which can reduce IOPS and increase IO throughput.

For example, if the original request needs to read parts [0, 10] and [20, 50] of file `file1`:

```
Request Range: [0, 10], [20, 50]
```

Through Merge IO, it will be merged into one request:

```
Request Range: [0, 50]
```

In this example, two IO requests are merged into one, but it also reads some additional data (data between 10-20). Therefore, while Merge IO reduces the number of IO operations, it may bring potential read amplification issues.

You can view specific Merge IO information through Query Profile:

```
- MergedSmallIO:
    - MergedBytes: 3.00 GB
    - MergedIO: 424
    - RequestBytes: 2.50 GB
    - RequestIO: 65.555K (65555)
```

Where `RequestBytes` and `RequestIO` indicate the data volume and number of requests in the original request. `MergedBytes` and `MergedIO` indicate the data volume and number of requests after merging.

If you find that `MergedBytes` is much larger than `RequestBytes`, it indicates serious read amplification. You can adjust it through the following parameters:

- `merge_io_read_slice_size_bytes`

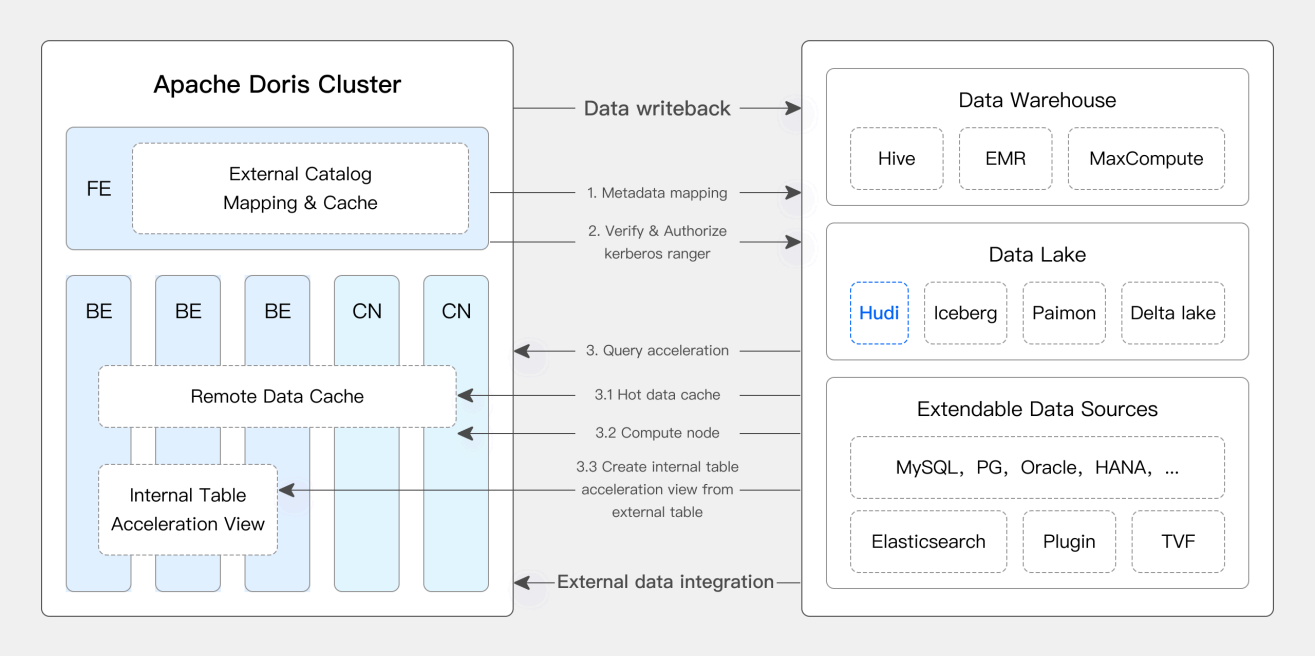
  Session variable, supported since version 3.1.3. Default is 8MB. If you find serious read amplification, you can reduce this parameter, such as to 64KB, and observe whether the modified IO requests and query latency improve.

Using Doris and Hudi

As a new open data management architecture, the Data Lakehouse integrates the high performance and real-time capabilities of data warehouses with the low cost and flexibility of data lakes, helping users more conveniently meet various data processing and analysis needs. It has been increasingly applied in enterprise big data systems.

In recent versions, Apache Doris has deepened its integration with data lakes and has evolved a mature Data Lakehouse solution.

- Since version 0.15, Apache Doris has introduced Hive and Iceberg external tables, exploring the capabilities of combining with Apache Iceberg for data lakes.
- Starting from version 1.2, Apache Doris officially introduced the Multi-Catalog feature, enabling automatic metadata mapping and data access for various data sources, along with numerous performance optimizations for external data reading and query execution. It now fully possesses the ability to build a high-speed and user-friendly Lakehouse architecture.
- In version 2.1, Apache Doris' Data Lakehouse architecture was significantly enhanced, improving the reading and writing capabilities of mainstream data lake formats (Hudi, Iceberg, Paimon, etc.), introducing compatibility with multiple SQL dialects, and seamless migration from existing systems to Apache Doris. For data science and large-scale data reading scenarios, Doris integrated the Arrow Flight high-speed reading interface, achieving a 100-fold increase in data transfer efficiency.



Building lakehouse using doris and huid

Apache Doris & Hudi

Apache Hudi is currently one of the most popular open data lake formats and a transactional data lake management platform, supporting various mainstream query engines including Apache Doris.

Apache Doris has also enhanced its ability to read Apache Hudi data tables:

- Supports Copy on Write Table: Snapshot Query
- Supports Merge on Read Table: Snapshot Queries, Read Optimized Queries
- Supports Time Travel
- Supports Incremental Read

With Apache Doris' high-performance query execution and Apache Hudi's real-time data management capabilities, efficient, flexible, and cost-effective data querying and analysis can be achieved. It also provides robust data lineage, auditing, and incremental processing functionalities. The combination of Apache Doris and Apache Hudi has been validated and promoted in real business scenarios by multiple community users:

- Real-time data analysis and processing: Common scenarios such as real-time data updates and query analysis in industries like finance, advertising, and e-commerce require real-time data processing. Hudi enables real-time data updates and management while ensuring data consistency and reliability. Doris efficiently handles large-scale data query requests in real-time, meeting the demands of real-time data analysis and processing effectively when combined.
- Data lineage and auditing: For industries with high requirements for data security and accuracy like finance and healthcare, data lineage and auditing are crucial functionalities. Hudi offers Time Travel functionality for viewing historical data states, combined with Apache Doris' efficient querying capabilities, enabling quick analysis of data at any point in time for precise lineage and auditing.
- Incremental data reading and analysis: Large-scale data analysis often faces challenges of large data volumes and frequent updates. Hudi supports incremental data reading, allowing users to process only the changed data without full data updates. Additionally, Apache Doris' Incremental Read feature enhances this process, significantly improving data processing and analysis efficiency.
- Cross-data source federated queries: Many enterprises have complex data sources stored in different databases. Doris' Multi-Catalog feature supports automatic mapping and synchronization of various data sources, enabling federated queries across data sources. This greatly shortens the data flow path and enhances work efficiency for enterprises needing to retrieve and integrate data from multiple sources for analysis.

This article will introduce readers to how to quickly set up a test and demonstration environment for Apache Doris + Apache Hudi in a Docker environment, and demonstrate various operations to help readers get started quickly.

For more information, please refer to Hudi Catalog

User Guide

All scripts and code mentioned in this article can be obtained from this address:
https://github.com/apache/doris/tree/master/samples/datalake/hudi

01 Environment Preparation

This article uses Docker Compose for deployment, with the following components and versions:

Component	Version
Apache Doris	Default 2.1.4, can be modified
Apache Hudi	0.14

Component	Version
Apache Spark	3.4.2
Apache Hive	2.1.3
MinIO	2022-05-26T05-48-41Z

02 Environment Deployment

1. Create a Docker network

   ```
   sudo docker network create -d bridge hudi-net
   ```

2. Start all components

   ```
   sudo ./start-hudi-compose.sh
   ```

   > Note: Before starting, you can modify the `DORIS_PACKAGE` and `DORIS_DOWNLOAD_URL` in `start-hudi-compose.sh` to the desired Doris version. It is recommended to use version 2.1.4 or higher.

3. After starting, you can use the following script to log in to Spark command line or Doris command line:

   ```
   -- Spark
   sudo ./login-spark.sh

   -- Doris
   sudo ./login-doris.sh
   ```

03 Data Preparation

Next, generate Hudi data through Spark. As shown in the code below, there is already a Hive table named `customer` in the cluster. You can create a Hudi table using this Hive table:

```
-- ./login-spark.sh
spark-sql> use default;

-- create a COW table
spark-sql> CREATE TABLE customer_cow
USING hudi
TBLPROPERTIES (
  type = 'cow',
  primaryKey = 'c_custkey',
  preCombineField = 'c_name'
)
PARTITIONED BY (c_nationkey)
AS SELECT * FROM customer;

-- create a MOR table
spark-sql> CREATE TABLE customer_mor
USING hudi
TBLPROPERTIES (
  type = 'mor',
  primaryKey = 'c_custkey',
  preCombineField = 'c_name'
```

```
)
PARTITIONED BY (c_nationkey)
AS SELECT * FROM customer;
```

04 Data Query

As shown below, a Catalog named `hudi` has been created in the Doris cluster (can be viewed using `SHOW CATALOGS`). The following is the creation statement for this Catalog:

```
-- Already created, no need to execute again
CREATE CATALOG `hudi` PROPERTIES (
    "type"="hms",
    'hive.metastore.uris' = 'thrift://hive-metastore:9083',
    "s3.access_key" = "minio",
    "s3.secret_key" = "minio123",
    "s3.endpoint" = "http://minio:9000",
    "s3.region" = "us-east-1",
    "use_path_style" = "true"
);
```

1. Manually refresh this Catalog to synchronize the created Hudi table:

   ```
   -- ./login-doris.sh
   doris> REFRESH CATALOG hudi;
   ```

2. Operations on data in Hudi using Spark are immediately visible in Doris without the need to refresh the Catalog. We insert a row of data into both COW and MOR tables using Spark:

   ```
   spark-sql> insert into customer_cow values (100, "Customer#000000100", "jD2xZzi", "25-430-914-2194", 3471.59, "BUILDING",
   "cial ideas. final, furious requests", 25);
   spark-sql> insert into customer_mor values (100, "Customer#000000100", "jD2xZzi", "25-430-914-2194", 3471.59, "BUILDING",
   "cial ideas. final, furious requests", 25);
   ```

3. Through Doris, you can directly query the latest inserted data:

   ```
   doris> use hudi.default;
   doris> select * from customer_cow where c_custkey = 100;
   doris> select * from customer_mor where c_custkey = 100;
   ```

4. Insert data with c_custkey=32 that already exists using Spark, thus overwriting the existing data:

   ```
   spark-sql> insert into customer_cow values (32, "Customer#000000032_update", "jD2xZzi", "25-430-914-2194", 3471.59,
   "BUILDING", "cial ideas. final, furious requests", 15);
   spark-sql> insert into customer_mor values (32, "Customer#000000032_update", "jD2xZzi", "25-430-914-2194", 3471.59,
   "BUILDING", "cial ideas. final, furious requests", 15);
   ```

5. With Doris, you can query the updated data:

   ```
   doris> select * from customer_cow where c_custkey = 32;
   +-----------+--------------------------+-----------+-----------------+-----------+--------------+-------------------------
   ------------+-------------+
   | c_custkey | c_name                   | c_address | c_phone         | c_acctbal | c_mktsegment | c_comment
   | c_nationkey |
   ```

```
+-----------+---------------------------+-----------+-----------------+-----------+--------------+-------------------------
------------+-------------+
|        32 | Customer#000000032_update | jD2xZzi   | 25-430-914-2194 |   3471.59 | BUILDING     | cial ideas. final,
furious requests |          15 |
+-----------+---------------------------+-----------+-----------------+-----------+--------------+-------------------------
------------+-------------+
doris> select * from customer_mor where c_custkey = 32;
+-----------+---------------------------+-----------+-----------------+-----------+--------------+-------------------------
------------+-------------+
| c_custkey | c_name                    | c_address | c_phone         | c_acctbal | c_mktsegment | c_comment
| c_nationkey |
+-----------+---------------------------+-----------+-----------------+-----------+--------------+-------------------------
------------+-------------+
|        32 | Customer#000000032_update | jD2xZzi   | 25-430-914-2194 |   3471.59 | BUILDING     | cial ideas. final,
furious requests |          15 |
+-----------+---------------------------+-----------+-----------------+-----------+--------------+-------------------------
------------+-------------+
```

05 Incremental Read

Incremental Read is one of the features provided by Hudi. With Incremental Read, users can obtain incremental data within a specified time range, enabling incremental processing of data. In this regard, Doris can query the changed data after inserting `c_custkey=100`. As shown below, we inserted a data with `c_custkey=32`:

```
doris> select * from customer_cow@incr('beginTime'='20240603015018572');
+-----------+---------------------------+-----------+-----------------+-----------+--------------+---------------------------------
---+-------------+
| c_custkey | c_name                    | c_address | c_phone         | c_acctbal | c_mktsegment | c_comment
| c_nationkey |
+-----------+---------------------------+-----------+-----------------+-----------+--------------+---------------------------------
---+-------------+
|        32 | Customer#000000032_update | jD2xZzi   | 25-430-914-2194 |   3471.59 | BUILDING     | cial ideas. final, furious
requests |          15 |
+-----------+---------------------------+-----------+-----------------+-----------+--------------+---------------------------------
---+-------------+
spark-sql> select * from hudi_table_changes('customer_cow', 'latest_state', '20240603015018572');

doris> select * from customer_mor@incr('beginTime'='20240603015058442');
+-----------+---------------------------+-----------+-----------------+-----------+--------------+---------------------------------
---+-------------+
| c_custkey | c_name                    | c_address | c_phone         | c_acctbal | c_mktsegment | c_comment
| c_nationkey |
+-----------+---------------------------+-----------+-----------------+-----------+--------------+---------------------------------
---+-------------+
|        32 | Customer#000000032_update | jD2xZzi   | 25-430-914-2194 |   3471.59 | BUILDING     | cial ideas. final, furious
requests |          15 |
+-----------+---------------------------+-----------+-----------------+-----------+--------------+---------------------------------
---+-------------+
spark-sql> select * from hudi_table_changes('customer_mor', 'latest_state', '20240603015058442');
```

06 TimeTravel

Doris supports querying specific snapshot versions of Hudi data, thereby enabling Time Travel functionality for data. First, you can query the commit history of two Hudi tables using Spark:

```
spark-sql> call show_commits(table => 'customer_cow', limit => 10);
20240603033556094        20240603033558249        commit        448833        0        1        1        183        0        0
20240603015444737        20240603015446588        commit        450238        0        1        1        202        1        0
```

```
20240603015018572        20240603015020503        commit        436692        1        0        1        1        0        0
20240603013858098        20240603013907467        commit        44902033        100        0        25        18751        0
0

spark-sql> call show_commits(table => 'customer_mor', limit => 10);
20240603033745977        20240603033748021        deltacommit        1240        0        1        1        0        0        0
20240603015451860        20240603015453539        deltacommit        1434        0        1        1        1        1        0
20240603015058442        20240603015100120        deltacommit        436691        1        0        1        1        0        0
20240603013918515        20240603013922961        deltacommit        44904040        100        0        25        18751        0
0
```

Next, using Doris, you can execute `c_custkey=32` to query the data snapshot before the data insertion. As shown below, the data with `c_custkey=32` has not been updated yet:

> Note: Time Travel syntax is currently not supported by the new optimizer. You need to first execute `set enable_nereids_planner=false;` to disable the new optimizer. This issue will be fixed in future versions.

```
doris> select * from customer_cow for time as of '20240603015018572' where c_custkey = 32 or c_custkey = 100;
+-----------+--------------------+---------------------------------------+-----------------+-----------+--------------+-------------
--------------------------------------+-------------+
| c_custkey | c_name             | c_address                             | c_phone         | c_acctbal | c_mktsegment | c_comment
| c_nationkey |
+-----------+--------------------+---------------------------------------+-----------------+-----------+--------------+-------------
--------------------------------------+-------------+
|        32 | Customer#000000032 | jD2xZzi UmId,DCtNBLXKj9q0Tlp2iQ6ZcO3J | 25-430-914-2194 |   3471.53 | BUILDING     | cial ideas.
final, furious requests across the e |          15 |
|       100 | Customer#000000100 | jD2xZzi                               | 25-430-914-2194 |   3471.59 | BUILDING     | cial ideas.
final, furious requests              |          25 |
+-----------+--------------------+---------------------------------------+-----------------+-----------+--------------+-------------
--------------------------------------+-------------+
-- compare with spark-sql
spark-sql> select * from customer_mor timestamp as of '20240603015018572' where c_custkey = 32 or c_custkey = 100;

doris> select * from customer_mor for time as of '20240603015058442' where c_custkey = 32 or c_custkey = 100;
+-----------+--------------------+---------------------------------------+-----------------+-----------+--------------+-------------
--------------------------------------+-------------+
| c_custkey | c_name             | c_address                             | c_phone         | c_acctbal | c_mktsegment | c_comment
| c_nationkey |
+-----------+--------------------+---------------------------------------+-----------------+-----------+--------------+-------------
--------------------------------------+-------------+
|       100 | Customer#000000100 | jD2xZzi                               | 25-430-914-2194 |   3471.59 | BUILDING     | cial ideas.
final, furious requests              |          25 |
|        32 | Customer#000000032 | jD2xZzi UmId,DCtNBLXKj9q0Tlp2iQ6ZcO3J | 25-430-914-2194 |   3471.53 | BUILDING     | cial ideas.
final, furious requests across the e |          15 |
+-----------+--------------------+---------------------------------------+-----------------+-----------+--------------+-------------
--------------------------------------+-------------+
spark-sql> select * from customer_mor timestamp as of '20240603015058442' where c_custkey = 32 or c_custkey = 100;
```

Query Optimization

Data in Apache Hudi can be roughly divided into two categories - baseline data and incremental data. Baseline data is typically merged Parquet files, while incremental data refers to data increments generated by INSERT, UPDATE, or DELETE operations. Baseline data can be read directly, while incremental data needs to be read through Merge on Read.

For querying Hudi COW tables or Read Optimized queries on MOR tables, the data belongs to baseline data and can be directly read using Doris' native Parquet Reader, providing fast query responses. For incremental data, Doris needs to access Hudi's Java SDK through JNI calls. To achieve optimal query performance, Apache Doris divides the data in a query into baseline and incremental data parts and reads them using the aforementioned methods.

To verify this optimization approach, we can use the EXPLAIN statement to see how many baseline and incremental data are present in a query example below. For a COW table, all 101 data shards are baseline data (`hudiNativeReadSplits=101/101`), so the COW table can be entirely read directly using Doris' Parquet Reader, resulting in the best query performance. For a ROW table, most data shards are baseline data (`hudiNativeReadSplits=100/101`), with one shard being incremental data, which also provides good query performance.

```
-- COW table is read natively
doris> explain select * from customer_cow where c_custkey = 32;
|   0:VHUDI_SCAN_NODE(68)                                      |
|      table: customer_cow                                     |
|      predicates: (c_custkey[#5] = 32)                        |
|      inputSplitNum=101, totalFileSize=45338886, scanRanges=101 |
|      partition=26/26                                         |
|      cardinality=1, numNodes=1                               |
|      pushdown agg=NONE                                       |
|      hudiNativeReadSplits=101/101                            |

-- MOR table: because only the base file contains `c_custkey = 32` that is updated, 100 splits are read natively, while the split
with log file is read by JNI.
doris> explain select * from customer_mor where c_custkey = 32;
|   0:VHUDI_SCAN_NODE(68)                                      |
|      table: customer_mor                                     |
|      predicates: (c_custkey[#5] = 32)                        |
|      inputSplitNum=101, totalFileSize=45340731, scanRanges=101 |
|      partition=26/26                                         |
|      cardinality=1, numNodes=1                               |
|      pushdown agg=NONE                                       |
|      hudiNativeReadSplits=100/101                            |
```

You can further observe the changes in Hudi baseline data and incremental data by performing some deletion operations using Spark:

```
-- Use delete statement to see more differences
spark-sql> delete from customer_cow where c_custkey = 64;
doris> explain select * from customer_cow where c_custkey = 64;

spark-sql> delete from customer_mor where c_custkey = 64;
doris> explain select * from customer_mor where c_custkey = 64;
```

Additionally, you can reduce the data volume further by using partition conditions for partition pruning to improve query speed. In the example below, partition pruning is done using the partition condition `c_nationkey=15`, allowing the query request to access data from only one partition (`partition=1/26`).

```
-- customer_xxx is partitioned by c_nationkey, we can use the partition column to prune data
doris> explain select * from customer_mor where c_custkey = 64 and c_nationkey = 15;
|   0:VHUDI_SCAN_NODE(68)                                      |
|      table: customer_mor                                     |
|      predicates: (c_custkey[#5] = 64), (c_nationkey[#12] = 15) |
|      inputSplitNum=4, totalFileSize=1798186, scanRanges=4    |
```

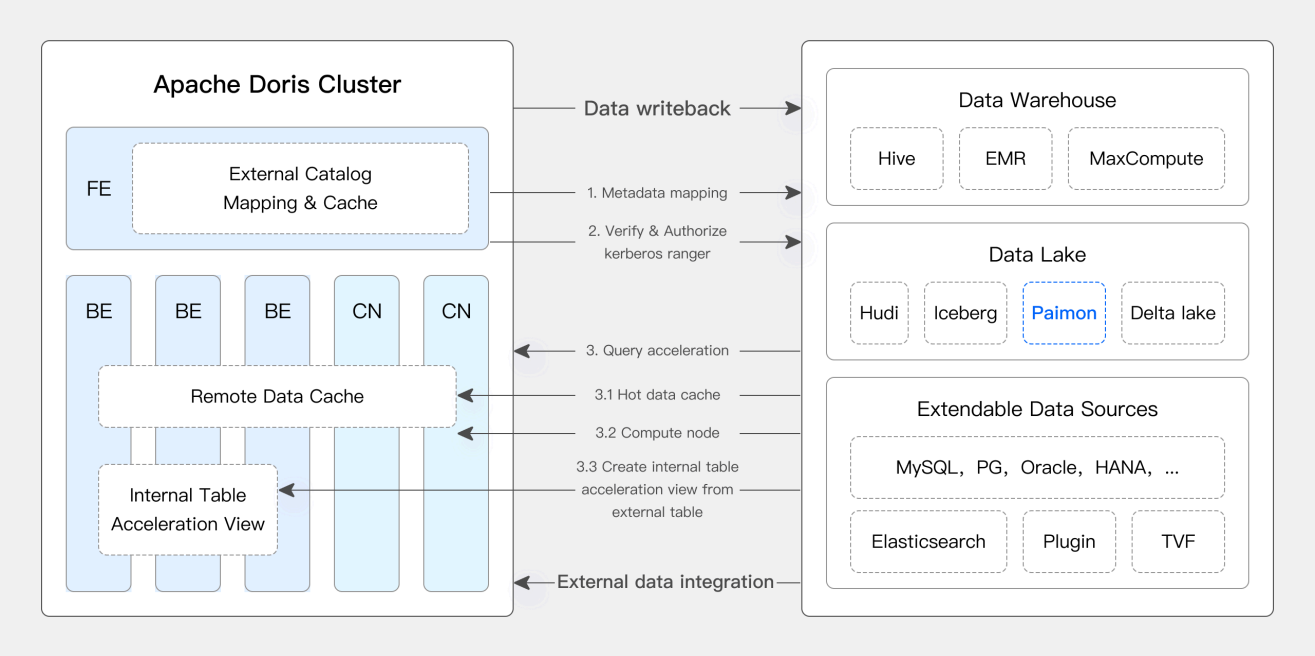
```
|      partition=1/26                                   |
|      cardinality=1, numNodes=1                        |
|      pushdown agg=NONE                                |
|      hudiNativeReadSplits=3/4                         |
```

Using Doris and Paimon

As a new open data management architecture, the Data Lakehouse integrates the high performance and real-time capabilities of data warehouses with the low cost and flexibility of data lakes, helping users more conveniently meet various data processing and analysis needs. It has been increasingly applied in enterprise big data systems.

In recent versions, Apache Doris has deepened its integration with data lakes and has evolved a mature Data Lakehouse solution.

- Since version 0.15, Apache Doris has introduced Hive and Iceberg external tables, exploring the capabilities of combining with Apache Iceberg for data lakes.
- Starting from version 1.2, Apache Doris officially introduced the Multi-Catalog feature, achieving automatic metadata mapping and data access for various data sources, along with many performance optimizations for external data reading and query execution. It now fully possesses the ability to build a high-speed and user-friendly Lakehouse architecture.
- In version 2.1, Apache Doris' Data Lakehouse architecture was significantly enhanced, strengthening the reading and writing capabilities of mainstream data lake formats (Hudi, Iceberg, Paimon, etc.), introducing compatibility with multiple SQL dialects, and seamless migration from existing systems to Apache Doris. For data science and large-scale data reading scenarios, Doris integrated the Arrow Flight high-speed reading interface, achieving a 100-fold improvement in data transfer efficiency.



Building lakehouse using Doris and Paimon

Apache Doris & Paimon

Apache Paimon is a data lake format that innovatively combines the advantages of data lake formats and LSM structures, successfully introducing efficient real-time streaming update capabilities into data lake architecture.

This enables Paimon to efficiently manage data and perform real-time analysis, providing strong support for building real-time Data Lakehouse architecture.

To fully leverage Paimon's capabilities and improve query efficiency for Paimon data, Apache Doris provides native support for several of Paimon's latest features:

- Supports various types of Paimon Catalogs such as Hive Metastore and FileSystem.
- Native support for Paimon 0.6's Primary Key Table Read Optimized feature.
- Native support for Paimon 0.8's Primary Key Table Deletion Vector feature.

With Apache Doris' high-performance query engine and Apache Paimon's efficient real-time streaming update capabilities, users can achieve:

- Real-time data ingestion into the lake: Leveraging Paimon's LSM-Tree model, data ingestion into the lake can be reduced to a minute-level timeliness. Additionally, Paimon supports various data update capabilities including aggregation, deduplication, and partial column updates, making data flow more flexible and efficient.
- High-performance data processing and analysis: Paimon's technologies such as Append Only Table, Read Optimized, and Deletion Vector can be seamlessly integrated with Doris' powerful query engine, enabling fast querying and analysis responses for lake data.

In the future, Apache Doris will gradually support more advanced features of Apache Paimon, including Time Travel and incremental data reading, to jointly build a unified, high-performance, real-time lakehouse platform.

This article will explain how to quickly set up an Apache Doris + Apache Paimon testing & demonstration environment in a Docker environment and demonstrate the usage of various features.

For more information, please refer to Paimon Catalog

User Guide

All scripts and code mentioned in this article can be obtained from the following address:
https://github.com/apache/doris/tree/master/samples/datalake/iceberg_and_paimon

01 Environment Preparation

This article uses Docker Compose for deployment, with the following components and versions:

Component	Version
Apache Doris	Default 2.1.5, can be modified
Apache Paimon	0.8
Apache Flink	1.18
MinIO	RELEASE.2024-04-29T09-56-05Z

02 Environment Deployment

1. Start all components

```
bash ./start_all.sh
```

2. After starting, you can use the following scripts to log in to the Flink command line or Doris command line:

```
-- login flink
bash ./start_flink_client.sh

-- login doris
bash ./start_doris_client.sh
```

03 Data Preparation

After logging into the Flink command line, you can see a pre-built table. The table already contains some data that can be viewed using Flink SQL.

```
Flink SQL> use paimon.db_paimon;
[INFO] Execute statement succeed.

Flink SQL> show tables;
+------------+
| table name |
+------------+
|   customer |
+------------+
1 row in set

Flink SQL> show create table customer;
+------------------------------------------------------------------------+
|                                                                 result |
+------------------------------------------------------------------------+
| CREATE TABLE `paimon`.`db_paimon`.`customer` (
  `c_custkey` INT NOT NULL,
  `c_name` VARCHAR(25),
  `c_address` VARCHAR(40),
  `c_nationkey` INT NOT NULL,
  `c_phone` CHAR(15),
  `c_acctbal` DECIMAL(12, 2),
  `c_mktsegment` CHAR(10),
  `c_comment` VARCHAR(117),
  CONSTRAINT `PK_c_custkey_c_nationkey` PRIMARY KEY (`c_custkey`, `c_nationkey`) NOT ENFORCED
) PARTITIONED BY (`c_nationkey`)
WITH (
  'bucket' = '1',
  'path' = 's3://warehouse/wh/db_paimon.db/customer',
  'deletion-vectors.enabled' = 'true'
)
 |
+------------------------------------------------------------------------+
1 row in set

Flink SQL> desc customer;
+-------------+---------------+-------+----------------------------+--------+-----------+
|         name |           type |  null |                        key | extras | watermark |
+-------------+---------------+-------+----------------------------+--------+-----------+
|    c_custkey |            INT | FALSE | PRI(c_custkey, c_nationkey) |        |           |
|       c_name |    VARCHAR(25) |  TRUE |                            |        |           |
|    c_address |    VARCHAR(40) |  TRUE |                            |        |           |
|  c_nationkey |            INT | FALSE | PRI(c_custkey, c_nationkey) |        |           |
|      c_phone |       CHAR(15) |  TRUE |                            |        |           |
```

```
|    c_acctbal | DECIMAL(12, 2) |  TRUE |                            |        |           |
| c_mktsegment |       CHAR(10) |  TRUE |                            |        |           |
|    c_comment |   VARCHAR(117) |  TRUE |                            |        |           |
+--------------+----------------+-------+----------------------------+--------+-----------+
8 rows in set

Flink SQL> select * from customer order by c_custkey limit 4;
+-----------+--------------------+-------------------------------+-------------+-----------------+-----------+--------------+------
-------------------------+
| c_custkey |             c_name |                     c_address | c_nationkey |         c_phone | c_acctbal | c_mktsegment |
c_comment |
+-----------+--------------------+-------------------------------+-------------+-----------------+-----------+--------------+------
-------------------------+
|         1 | Customer#000000001 |              IVhzIApeRb ot,c,E |          15 | 25-989-741-2988 |    711.56 |     BUILDING | to
the even, regular platel... |
|         2 | Customer#000000002 | XSTf4,NCwDVaWNe6tEgvwfmRchLXak |          13 | 23-768-687-3665 |    121.65 |   AUTOMOBILE | l
accounts. blithely ironic... |
|         3 | Customer#000000003 |                   MG9kdTD2WBHm |           1 | 11-719-748-3364 |   7498.12 |   AUTOMOBILE |
deposits eat slyly ironic,... |
|        32 | Customer#000000032 | jD2xZzi UmId,DCtNBLXKj9q0Tl... |          15 | 25-430-914-2194 |   3471.53 |     BUILDING | cial
ideas. final, furious ... |
+-----------+--------------------+-------------------------------+-------------+-----------------+-----------+--------------+------
-------------------------+
4 rows in set
```

04 Data Query

As shown below, a Catalog named `paimon` has been created in the Doris cluster (can be viewed using SHOW CATALOGS). The following is the statement for creating this Catalog:

```
-- Already created, no need to run
CREATE CATALOG `paimon` PROPERTIES (
    "type" = "paimon",
    "warehouse" = "s3://warehouse/wh/",
    "s3.endpoint"="http://minio:9000",
    "s3.access_key"="admin",
    "s3.secret_key"="password",
    "s3.region"="us-east-1"
);
```

You can query Paimon's data in Doris:

```
mysql> use paimon.db_paimon;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+---------------------+
| Tables_in_db_paimon |
+---------------------+
| customer            |
+---------------------+
1 row in set (0.00 sec)

mysql> select * from customer order by c_custkey limit 4;
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+--------------
+-----------------------------------------------------------------------------------------------------+
| c_custkey | c_name             | c_address                             | c_nationkey | c_phone         | c_acctbal | c_mktsegment
| c_comment                                                                                           |
```

```
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------
+-----------------------------------------------------------------------------------------------------+
|         1 | Customer#000000001 | IVhzIApeRb ot,c,E                     |          15 | 25-989-741-2988 |    711.56 | BUILDING
| to the even, regular platelets. regular, ironic epitaphs nag e                                      |
|         2 | Customer#000000002 | XSTf4,NCwDVaWNe6tEgvwfmRchLXak        |          13 | 23-768-687-3665 |    121.65 | AUTOMOBILE
| l accounts. blithely ironic theodolites integrate boldly: caref                                     |
|         3 | Customer#000000003 | MG9kdTD2WBHm                          |           1 | 11-719-748-3364 |   7498.12 | AUTOMOBILE
|  deposits eat slyly ironic, even instructions. express foxes detect slyly. blithely even accounts abov |
|        32 | Customer#000000032 | jD2xZzi UmId,DCtNBLXKj9q0Tlp2iQ6ZcO3J |          15 | 25-430-914-2194 |   3471.53 | BUILDING
| cial ideas. final, furious requests across the e                                                    |
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------
+-----------------------------------------------------------------------------------------------------+
4 rows in set (1.89 sec)
```

05 Read Incremental Data

You can update the data in the Paimon table using Flink SQL:

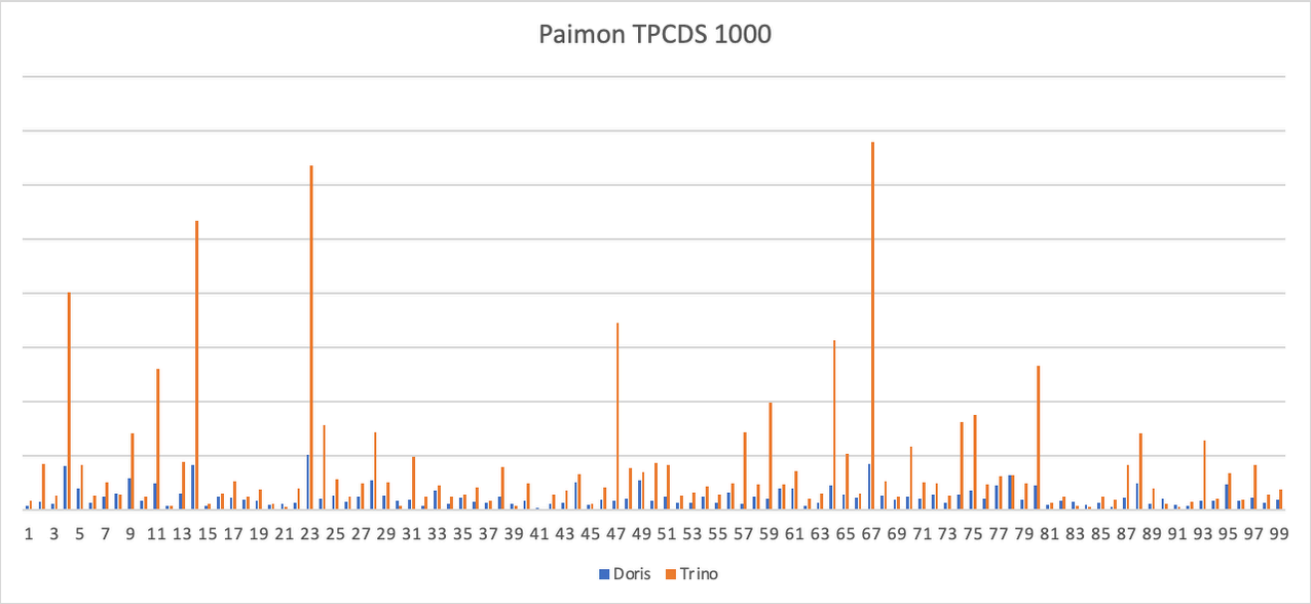
```
Flink SQL> update customer set c_address='c_address_update' where c_nationkey = 1;
[INFO] Submitting SQL update statement to the cluster...
[INFO] SQL update statement has been successfully submitted to the cluster:
Job ID: ff838b7b778a94396b332b0d93c8f7ac
```

After the Flink SQL execution is complete, you can directly view the latest data in Doris:

```
mysql> select * from customer where c_nationkey=1 limit 2;
+-----------+--------------------+-----------------+-------------+-----------------+-----------+--------------+---------------------
---------------------------------------------------------------------------------+
| c_custkey | c_name             | c_address       | c_nationkey | c_phone         | c_acctbal | c_mktsegment | c_comment
|
+-----------+--------------------+-----------------+-------------+-----------------+-----------+--------------+---------------------
---------------------------------------------------------------------------------+
|         3 | Customer#000000003 | c_address_update |           1 | 11-719-748-3364 |   7498.12 | AUTOMOBILE   |  deposits eat slyly
ironic, even instructions. express foxes detect slyly. blithely even accounts abov |
|       513 | Customer#000000513 | c_address_update |           1 | 11-861-303-6887 |    955.37 | HOUSEHOLD    | press along the
quickly regular instructions. regular requests against the carefully ironic s          |
+-----------+--------------------+-----------------+-------------+-----------------+-----------+--------------+---------------------
---------------------------------------------------------------------------------+
2 rows in set (0.19 sec)
```

Benchmark

We conducted a simple test on the TPCDS 1000 dataset in Paimon (0.8) version, using Apache Doris 2.1.5 version and Trino 422 version, both with the Primary Key Table Read Optimized feature enabled.



From the test results, it can be seen that Doris' average query performance on the standard static test set is 3-5 times that of Trino. In the future, we will optimize the Deletion Vector to further improve query efficiency in real business scenarios.

Query Optimization

For baseline data, after introducing the Primary Key Table Read Optimized feature in Apache Paimon version 0.6, the query engine can directly access the underlying Parquet/ORC files, significantly improving the reading efficiency of baseline data. For unmerged incremental data (data increments generated by INSERT, UPDATE, or DELETE), they can be read through Merge-on-Read. In addition, Paimon introduced the Deletion Vector feature in version 0.8, which further enhances the query engine's efficiency in reading incremental data. Apache Doris supports reading Deletion Vector through native Reader and performing Merge on Read. We demonstrate the query methods for baseline data and incremental data in a query using Doris' EXPLAIN statement.

```
mysql> explain verbose select * from customer where c_nationkey < 3;
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------+
| Explain String(Nereids Planner)
|
+------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------+
| ..............
|
|
|
|   0:VPAIMON_SCAN_NODE(68)
|
|      table: customer
|
|      predicates: (c_nationkey[#3] < 3)
|
|      inputSplitNum=4, totalFileSize=238324, scanRanges=4
|
|      partition=3/0
|
|      backends:
```

```
|
|        10002
|
|          s3://warehouse/wh/db_paimon.db/customer/c_nationkey=1/bucket-0/data-15cee5b7-1bd7-42ca-9314-56d92c62c03b-0.orc start: 0
length: 66600 |
|          s3://warehouse/wh/db_paimon.db/customer/c_nationkey=1/bucket-0/data-5d50255a-2215-4010-b976-d5dc656f3444-0.orc start: 0
length: 44501 |
|          s3://warehouse/wh/db_paimon.db/customer/c_nationkey=2/bucket-0/data-e98fb7ef-ec2b-4ad5-a496-713cb9481d56-0.orc start: 0
length: 64059 |
|          s3://warehouse/wh/db_paimon.db/customer/c_nationkey=0/bucket-0/data-431be05d-50fa-401f-9680-d646757d0f95-0.orc start: 0
length: 63164 |
|      cardinality=18751, numNodes=1
|
|      pushdown agg=NONE
|
|      paimonNativeReadSplits=4/4
|
|      PaimonSplitStats:
|
|        SplitStat [type=NATIVE, rowCount=1542, rawFileConvertable=true, hasDeletionVector=true]
|
|        SplitStat [type=NATIVE, rowCount=750, rawFileConvertable=true, hasDeletionVector=false]
|
|        SplitStat [type=NATIVE, rowCount=750, rawFileConvertable=true, hasDeletionVector=false]
|
|      tuple ids: 0
| ..............                                                                                                    |
|
+-------------------------------------------------------------------------------------------------------------------------------------------
-------------+
67 rows in set (0.23 sec)
```

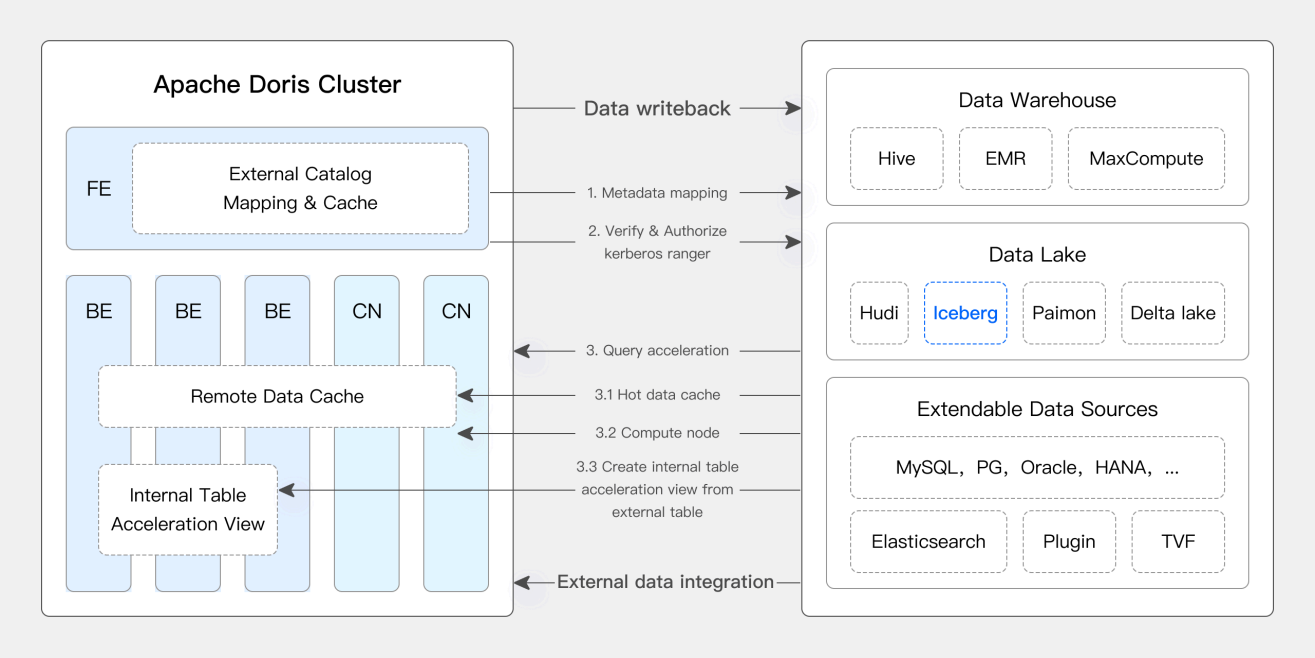
It can be seen that the table just updated by Flink SQL contains 4 shards, and all shards can be accessed through Native Reader (paimonNativeReadSplits=4/4). In addition, the hasDeletionVector property of the first shard is true, indicating that the shard has a corresponding Deletion Vector, and data will be filtered according to the Deletion Vector when reading.

Using Doris and Iceberg

As a new open data management architecture, the Data Lakehouse integrates the high performance and real-time capabilities of a data warehouse with the low cost and flexibility of a data lake, helping users more conveniently meet various data processing and analysis needs. It has been increasingly applied in enterprise big data systems.

In recent versions, Apache Doris has deepened its integration with data lakes and now offers a mature Data Lakehouse solution.

- Since version 0.15, Apache Doris has introduced Hive and Iceberg external tables, exploring the capabilities of combining with Apache Iceberg for data lakes.
- Starting from version 1.2, Apache Doris officially introduced the Multi-Catalog feature, enabling automatic metadata mapping and data access for various data sources, along with numerous performance optimizations for external data reading and query execution. It now fully supports building a high-speed and user-friendly Lakehouse architecture.
- In version 2.1, Apache Doris further strengthened its Data Lakehouse architecture, enhancing the reading and writing capabilities of mainstream data lake formats (Hudi, Iceberg, Paimon, etc.), introducing compatibility with multiple SQL dialects, and seamless migration from existing systems to Apache Doris. For data science and large-scale data reading scenarios, Doris integrated the Arrow Flight high-speed reading interface, achieving a 100x improvement in data transfer efficiency.



Building Lakehouse using Doris and Iceberg

Apache Doris & Iceberg

Apache Iceberg is an open-source, high-performance, and highly reliable data lake table format that enables the analysis and management of massive-scale data. It supports various mainstream query engines, including Apache Doris, is compatible with HDFS and various object cloud storage, and features ACID compliance,

schema evolution, advanced filtering, hidden partitioning, and partition layout evolution to ensure high-performance queries, data reliability, consistency, and flexibility with features like time travel and version rollback.

Apache Doris provides native support for several core features of Iceberg:

- Supports multiple Iceberg Catalog types such as Hive Metastore, Hadoop, REST, Glue, Google Dataproc Metastore, DLF, etc.
- Native support for Iceberg V1/V2 table formats and reading of Position Delete, Equality Delete files.
- Supports querying Iceberg table snapshot history through table functions.
- Supports Time Travel functionality.
- Native support for the Iceberg table engine. It allows Apache Doris to directly create, manage, and write data to Iceberg tables. It supports comprehensive partition Transform functions, providing capabilities like hidden partitioning and partition layout evolution.

Users can quickly build an efficient Data Lakehouse solution based on Apache Doris + Apache Iceberg to flexibly address various real-time data analysis and processing needs.

- Use the high-performance query engine of Doris to perform data analysis by associating Iceberg table data and other data sources, building a **unified federated data analysis platform**.
- Manage and build Iceberg tables directly through Doris, complete data cleaning, processing, and writing to Iceberg tables in Doris, building a **unified data processing platform for data lakes**.
- Share Doris data with other upstream and downstream systems for further processing through the Iceberg table engine, building a **unified open data storage platform**.

In the future, Apache Iceberg will serve as one of the native table engines for Apache Doris, providing more comprehensive analysis and management functions for lake-formatted data. Apache Doris will also gradually support more advanced features of Apache Iceberg, including Update/Delete/Merge, sorting during write-back, incremental data reading, metadata management, etc., to jointly build a unified, high-performance, real-time data lake platform.

For more information, please refer to Iceberg Catalog

User Guide

This document mainly explains how to quickly set up an Apache Doris + Apache Iceberg testing & demonstration environment in a Docker environment and demonstrate the usage of various functions.

All scripts and code mentioned in this document can be obtained from this address: https://github.com/apache/doris/tree/master/samples/datalake/iceberg_and_paimon

01 Environment Preparation

This document uses Docker Compose for deployment, with the following components and versions:

Component	Version
Apache Doris	Default 2.1.5, can be modified
Apache Iceberg	1.4.3

Component	Version
MinIO	RELEASE.2024-04-29T09-56-05Z

02 Environment Deployment

1. Start all components

   ```
   bash ./start_all.sh
   ```

2. After starting, you can use the following script to log in to the Doris command line:

   ```
   -- login doris
   bash ./start_doris_client.sh
   ```

03 Create Iceberg Table

After logging into the Doris command line, an Iceberg Catalog named Iceberg has already been created in the Doris cluster (can be viewed by `SHOW CATALOGS`/`SHOW CREATE CATALOG iceberg`). The following is the creation statement for this Catalog:

```
-- Already created
CREATE CATALOG `iceberg` PROPERTIES (
    "type" = "iceberg",
    "iceberg.catalog.type" = "rest",
    "warehouse" = "s3://warehouse/",
    "uri" = "http://rest:8181",
    "s3.access_key" = "admin",
    "s3.secret_key" = "password",
    "s3.endpoint" = "http://minio:9000"
);
```

Create a database and an Iceberg table in the Iceberg Catalog:

```
mysql> SWITCH iceberg;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE DATABASE nyc;
Query OK, 0 rows affected (0.12 sec)

mysql> CREATE TABLE iceberg.nyc.taxis
       (
           vendor_id BIGINT,
           trip_id BIGINT,
           trip_distance FLOAT,
           fare_amount DOUBLE,
           store_and_fwd_flag STRING,
           ts DATETIME
       )
       PARTITION BY LIST (vendor_id, DAY(ts)) ()
       PROPERTIES (
           "compression-codec" = "zstd",
           "write-format" = "parquet"
       );
Query OK, 0 rows affected (0.15 sec)
```

04 Data Insertion

Insert data into the Iceberg table:

```
mysql> INSERT INTO iceberg.nyc.taxis
       VALUES
        (1, 1000371, 1.8, 15.32, 'N', '2024-01-01 9:15:23'),
        (2, 1000372, 2.5, 22.15, 'N', '2024-01-02 12:10:11'),
        (2, 1000373, 0.9, 9.01, 'N', '2024-01-01 3:25:15'),
        (1, 1000374, 8.4, 42.13, 'Y', '2024-01-03 7:12:33');
Query OK, 4 rows affected (1.61 sec)
{'status':'COMMITTED', 'txnId':'10085'}
```

Create an Iceberg table using `CREATE TABLE AS SELECT`:

```
mysql> CREATE TABLE iceberg.nyc.taxis2 AS SELECT * FROM iceberg.nyc.taxis;
Query OK, 4 rows affected (0.25 sec)
{'status':'COMMITTED', 'txnId':'10088'}
```

05 Data Query

- Simple query

```
mysql> SELECT * FROM iceberg.nyc.taxis;
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         1 | 1000374 |           8.4 |       42.13 | Y                  | 2024-01-03 07:12:33.000000 |
|         1 | 1000371 |           1.8 |       15.32 | N                  | 2024-01-01 09:15:23.000000 |
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
|         2 | 1000372 |           2.5 |       22.15 | N                  | 2024-01-02 12:10:11.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
4 rows in set (0.37 sec)

mysql> SELECT * FROM iceberg.nyc.taxis2;
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         1 | 1000374 |           8.4 |       42.13 | Y                  | 2024-01-03 07:12:33.000000 |
|         1 | 1000371 |           1.8 |       15.32 | N                  | 2024-01-01 09:15:23.000000 |
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
|         2 | 1000372 |           2.5 |       22.15 | N                  | 2024-01-02 12:10:11.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
4 rows in set (0.35 sec)
```

- Partition pruning

```
mysql> SELECT * FROM iceberg.nyc.taxis where vendor_id = 2 and ts >= '2024-01-01' and ts < '2024-01-02';
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
1 row in set (0.06 sec)

mysql> EXPLAIN VERBOSE SELECT * FROM iceberg.nyc.taxis where vendor_id = 2 and ts >= '2024-01-01' and ts < '2024-01-02';
```

```
....
|   0:VICEBERG_SCAN_NODE(71)
|      table: taxis
|      predicates: (ts[#5] < '2024-01-02 00:00:00'), (vendor_id[#0] = 2), (ts[#5] >= '2024-01-01 00:00:00')
|      inputSplitNum=1, totalFileSize=3539, scanRanges=1
|      partition=1/0
|      backends:
|        10002
|          s3://warehouse/wh/nyc/taxis/data/vendor_id=2/ts_day=2024-01-01/40e6ca404efa4a44-b888f23546d3a69c_5708e229-2f3d-
4b68-a66b-44298a9d9815-0.zstd.parquet start: 0 length: 3539
|      cardinality=6, numNodes=1
|      pushdown agg=NONE
|      icebergPredicatePushdown=
|           ref(name="ts") < 1704153600000000
|           ref(name="vendor_id") == 2
|           ref(name="ts") >= 1704067200000000
....
```

By examining the result of the `EXPLAIN VERBOSE` statement, it can be seen that the predicate condition `vendor_id = 2 and ts >= '2024-01-01' and ts < '2024-01-02'` ultimately only hits one partition (`partition=1/0`).

It can also be observed that because a partition Transform function `DAY(ts)` was specified when creating the table, the original value in the data `2024-01-01 03:25:15.000000` will be transformed into the partition information in the file directory `ts_day=2024-01-01`.

06 Time Travel

Let's insert a few more rows of data:

```
INSERT INTO iceberg.nyc.taxis VALUES (1, 1000375, 8.8, 55.55, 'Y', '2024-01-01 8:10:22'), (3, 1000376, 7.4, 32.35, 'N', '2024-01-02
1:14:45');
Query OK, 2 rows affected (0.17 sec)
{'status':'COMMITTED', 'txnId':'10086'}

mysql> SELECT * FROM iceberg.nyc.taxis;
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         3 | 1000376 |           7.4 |       32.35 | N                  | 2024-01-02 01:14:45.000000 |
|         2 | 1000372 |           2.5 |       22.15 | N                  | 2024-01-02 12:10:11.000000 |
|         1 | 1000374 |           8.4 |       42.13 | Y                  | 2024-01-03 07:12:33.000000 |
|         1 | 1000371 |           1.8 |       15.32 | N                  | 2024-01-01 09:15:23.000000 |
|         1 | 1000375 |           8.8 |       55.55 | Y                  | 2024-01-01 08:10:22.000000 |
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
6 rows in set (0.11 sec)
```

Use the `iceberg_meta` table function to query the snapshot information of the table:

```
mysql> select * from iceberg_meta("table" = "iceberg.nyc.taxis", "query_type" = "snapshots");
+---------------------+---------------------+---------------------+-----------+-----------------------------------------------------
----------------------------------------------------+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| committed_at        | snapshot_id         | parent_id           | operation | manifest_list
| summary
|
```

```
+---------------------+---------------------+---------------------+-----------+-----------------------------------------------------
-----------------------------------------------------+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
| 2024-07-29 03:38:22 | 8483933166442433486 |                  -1 | append    | s3://warehouse/wh/nyc/taxis/metadata/snap-
8483933166442433486-1-5f7b7736-8022-4ba1-9db2-51ae7553be4d.avro | {"added-data-files":"4","added-records":"4","added-files-
size":"14156","changed-partition-count":"4","total-records":"4","total-files-size":"14156","total-data-files":"4","total-delete-
files":"0","total-position-deletes":"0","total-equality-deletes":"0"} |
| 2024-07-29 03:40:23 | 4726331391239920914 | 8483933166442433486 | append    | s3://warehouse/wh/nyc/taxis/metadata/snap-
4726331391239920914-1-6aa3d142-6c9c-4553-9c04-08ad4d49a4ea.avro | {"added-data-files":"2","added-records":"2","added-files-
size":"7078","changed-partition-count":"2","total-records":"6","total-files-size":"21234","total-data-files":"6","total-delete-
files":"0","total-position-deletes":"0","total-equality-deletes":"0"}  |
+---------------------+---------------------+---------------------+-----------+-----------------------------------------------------
-----------------------------------------------------+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------+
2 rows in set (0.07 sec)
```

Query a specified snapshot using the `FOR VERSION AS OF` statement:

```
mysql> SELECT * FROM iceberg.nyc.taxis FOR VERSION AS OF 8483933166442433486;
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         1 | 1000371 |           1.8 |       15.32 | N                  | 2024-01-01 09:15:23.000000 |
|         1 | 1000374 |           8.4 |       42.13 | Y                  | 2024-01-03 07:12:33.000000 |
|         2 | 1000372 |           2.5 |       22.15 | N                  | 2024-01-02 12:10:11.000000 |
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
4 rows in set (0.05 sec)

mysql> SELECT * FROM iceberg.nyc.taxis FOR VERSION AS OF 4726331391239920914;
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         1 | 1000374 |           8.4 |       42.13 | Y                  | 2024-01-03 07:12:33.000000 |
|         1 | 1000375 |           8.8 |       55.55 | Y                  | 2024-01-01 08:10:22.000000 |
|         3 | 1000376 |           7.4 |       32.35 | N                  | 2024-01-02 01:14:45.000000 |
|         2 | 1000372 |           2.5 |       22.15 | N                  | 2024-01-02 12:10:11.000000 |
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
|         1 | 1000371 |           1.8 |       15.32 | N                  | 2024-01-01 09:15:23.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
6 rows in set (0.04 sec)
```

Query a specified snapshot using the `FOR TIME AS OF` statement:

```
mysql> SELECT * FROM iceberg.nyc.taxis FOR TIME AS OF "2024-07-29 03:38:23";
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         1 | 1000374 |           8.4 |       42.13 | Y                  | 2024-01-03 07:12:33.000000 |
|         1 | 1000371 |           1.8 |       15.32 | N                  | 2024-01-01 09:15:23.000000 |
|         2 | 1000372 |           2.5 |       22.15 | N                  | 2024-01-02 12:10:11.000000 |
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
4 rows in set (0.04 sec)

mysql> SELECT * FROM iceberg.nyc.taxis FOR TIME AS OF "2024-07-29 03:40:22";
+-----------+---------+---------------+-------------+--------------------+----------------------------+
| vendor_id | trip_id | trip_distance | fare_amount | store_and_fwd_flag | ts                         |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
|         2 | 1000373 |           0.9 |        9.01 | N                  | 2024-01-01 03:25:15.000000 |
```

```
|         1 | 1000374 |           8.4 |       42.13 | Y                  | 2024-01-03 07:12:33.000000 |
|         2 | 1000372 |           2.5 |       22.15 | N                  | 2024-01-02 12:10:11.000000 |
|         1 | 1000371 |           1.8 |       15.32 | N                  | 2024-01-01 09:15:23.000000 |
+-----------+---------+---------------+-------------+--------------------+----------------------------+
4 rows in set (0.05 sec)
```

07 Interacting with PyIceberg

> Please use Doris 2.1.8/3.0.4 or above.

Load an iceberg table:

```
from pyiceberg.catalog import load_catalog

catalog = load_catalog(
    "iceberg",
    **{
        "warehouse" = "warehouse",
        "uri" = "http://rest:8181",
        "s3.access-key-id" = "admin",
        "s3.secret-access-key" = "password",
        "s3.endpoint" = "http://minio:9000"
    },
)
table = catalog.load_table("nyc.taxis")
```

Read table as `Arrow Table`:

```
print(table.scan().to_arrow())

pyarrow.Table
vendor_id: int64
trip_id: int64
trip_distance: float
fare_amount: double
store_and_fwd_flag: large_string
ts: timestamp[us]
----
vendor_id: [[1],[1],[2],[2]]
trip_id: [[1000371],[1000374],[1000373],[1000372]]
trip_distance: [[1.8],[8.4],[0.9],[2.5]]
fare_amount: [[15.32],[42.13],[9.01],[22.15]]
store_and_fwd_flag: [["N"],["Y"],["N"],["N"]]
ts: [[2024-01-01 09:15:23.000000],[2024-01-03 07:12:33.000000],[2024-01-01 03:25:15.000000],[2024-01-02 12:10:11.000000]]
```

Read table as `Pandas DataFrame`:

```
print(table.scan().to_pandas())

vendor_id  trip_id  trip_distance  fare_amount store_and_fwd_flag                  ts
0          1  1000371            1.8        15.32                 N   2024-01-01 09:15:23
1          1  1000374            8.4        42.13                 Y   2024-01-03 07:12:33
2          2  1000373            0.9         9.01                 N   2024-01-01 03:25:15
3          2  1000372            2.5        22.15                 N   2024-01-02 12:10:11
```

Read table as `Polars DataFrame`:

```
import polars as pl

print(pl.scan_iceberg(table).collect())

shape: (4, 6)
┌────────────┬─────────┬───────────────┬─────────────┬────────────────────┬─────────────────────┐
│ vendor_id  ┆ trip_id ┆ trip_distance ┆ fare_amount ┆ store_and_fwd_flag ┆ ts                  │
│ ---        ┆ ---     ┆ ---           ┆ ---         ┆ ---                ┆ ---                 │
│ i64        ┆ i64     ┆ f32           ┆ f64         ┆ str                ┆ datetime[μs]        │
╞════════════╪═════════╪═══════════════╪═════════════╪════════════════════╪═════════════════════╡
│ 1          ┆ 1000371 ┆ 1.8           ┆ 15.32       ┆ N                  ┆ 2024-01-01 09:15:23 │
│ 1          ┆ 1000374 ┆ 8.4           ┆ 42.13       ┆ Y                  ┆ 2024-01-03 07:12:33 │
│ 2          ┆ 1000373 ┆ 0.9           ┆ 9.01        ┆ N                  ┆ 2024-01-01 03:25:15 │
│ 2          ┆ 1000372 ┆ 2.5           ┆ 22.15       ┆ N                  ┆ 2024-01-02 12:10:11 │
└────────────┴─────────┴───────────────┴─────────────┴────────────────────┴─────────────────────┘
```

> Write iceberg table by PyIceberg, please see step

08 Appendix

Write iceberg table by PyIceberg

Load an iceberg table:

```
from pyiceberg.catalog import load_catalog

catalog = load_catalog(
    "iceberg",
    **{
        "warehouse" = "warehouse",
        "uri" = "http://rest:8181",
        "s3.access-key-id" = "admin",
        "s3.secret-access-key" = "password",
        "s3.endpoint" = "http://minio:9000"
    },
)
table = catalog.load_table("nyc.taxis")
```

Write table with `Arrow Table` :

```
import pyarrow as pa

df = pa.Table.from_pydict(
    {
        "vendor_id": pa.array([1, 2, 2, 1], pa.int64()),
        "trip_id": pa.array([1000371, 1000372, 1000373, 1000374], pa.int64()),
        "trip_distance": pa.array([1.8, 2.5, 0.9, 8.4], pa.float32()),
        "fare_amount": pa.array([15.32, 22.15, 9.01, 42.13], pa.float64()),
        "store_and_fwd_flag": pa.array(["N", "N", "N", "Y"], pa.string()),
        "ts": pa.compute.strptime(
```

```
            ["2024-01-01 9:15:23", "2024-01-02 12:10:11", "2024-01-01 3:25:15", "2024-01-03 7:12:33"],
            "%Y-%m-%d %H:%M:%S",
            "us",
        ),
    }
)
table.append(df)
```

Write table with `Pandas DataFrame` :

```
import pyarrow as pa
import pandas as pd

df = pd.DataFrame(
    {
        "vendor_id": pd.Series([1, 2, 2, 1]).astype("int64[pyarrow]"),
        "trip_id": pd.Series([1000371, 1000372, 1000373, 1000374]).astype("int64[pyarrow]"),
        "trip_distance": pd.Series([1.8, 2.5, 0.9, 8.4]).astype("float32[pyarrow]"),
        "fare_amount": pd.Series([15.32, 22.15, 9.01, 42.13]).astype("float64[pyarrow]"),
        "store_and_fwd_flag": pd.Series(["N", "N", "N", "Y"]).astype("string[pyarrow]"),
        "ts": pd.Series(["2024-01-01 9:15:23", "2024-01-02 12:10:11", "2024-01-01 3:25:15", "2024-01-03
7:12:33"]).astype("timestamp[us][pyarrow]"),
    }
)
table.append(pa.Table.from_pandas(df))
```

Write table with `Polars DataFrame` :

```
import polars as pl

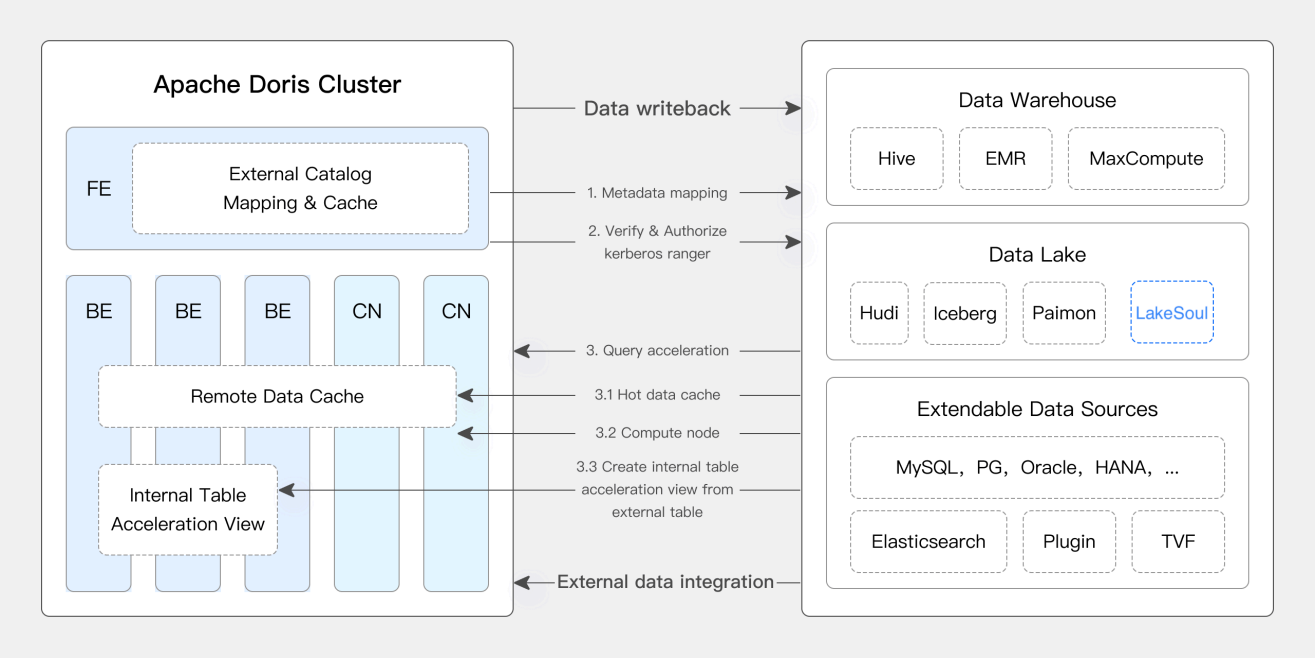
df = pl.DataFrame(
    {
        "vendor_id": [1, 2, 2, 1],
        "trip_id": [1000371, 1000372, 1000373, 1000374],
        "trip_distance": [1.8, 2.5, 0.9, 8.4],
        "fare_amount": [15.32, 22.15, 9.01, 42.13],
        "store_and_fwd_flag": ["N", "N", "N", "Y"],
        "ts": ["2024-01-01 9:15:23", "2024-01-02 12:10:11", "2024-01-01 3:25:15", "2024-01-03 7:12:33"],
    },
    {
        "vendor_id": pl.Int64,
        "trip_id": pl.Int64,
        "trip_distance": pl.Float32,
        "fare_amount": pl.Float64,
        "store_and_fwd_flag": pl.String,
        "ts": pl.String,
    },
).with_columns(pl.col("ts").str.strptime(pl.Datetime, "%Y-%m-%d %H:%M:%S"))
table.append(df.to_arrow())
```

Using Doris and LakeSoul

As a new open data management architecture, the Data Lakehouse integrates the high performance and real-time capabilities of data warehouses with the low cost and flexibility of data lakes, helping users more conveniently meet various data processing and analysis needs. It has been increasingly applied in enterprise big data systems.

In recent versions, Apache Doris has deepened its integration with data lakes and has evolved a mature Data Lakehouse solution.

- Since version 0.15, Apache Doris has introduced Hive and Iceberg external tables, exploring the capabilities of combining with Apache Iceberg for data lakes.
- Starting from version 1.2, Apache Doris officially introduced the Multi-Catalog feature, enabling automatic metadata mapping and data access for various data sources, along with numerous performance optimizations for external data reading and query execution. It now fully possesses the ability to build a high-speed and user-friendly Lakehouse architecture.
- In version 2.1, Apache Doris' Data Lakehouse architecture was significantly enhanced, improving the reading and writing capabilities of mainstream data lake formats (Hudi, Iceberg, Paimon, etc.), introducing compatibility with multiple SQL dialects, and seamless migration from existing systems to Apache Doris. For data science and large-scale data reading scenarios, Doris integrated the Arrow Flight high-speed reading interface, achieving a 100-fold increase in data transfer efficiency.



Building lakehouse using doris and lakesoul

Apache Doris & LakeSoul

LakeSoul is a cloud-native lakehouse framework developed by DMetaSoul and donated to the Linux Foundation AI & Data Foundation in May 2023. It features high scalability in metadata management, ACID

transactions, efficient and flexible upsert operations, schema evolution, and batch-stream integrated processing .

With Apache Doris' high-performance query engine and LakeSoul's efficient data management, users can achieve:

- Real-time data ingestion into the lake: Leveraging LakeSoul's architecture, data can be ingested into the lake with high efficiency and low latency, supporting various data update capabilities including aggregation, deduplication, and partial column updates.
- High-performance data processing and analysis: LakeSoul's capabilities such as batch-stream integrated processing and schema evolution can be seamlessly integrated with Doris' powerful query engine, enabling fast querying and analysis responses for lake data.

In the future, Apache Doris will gradually support more advanced features of LakeSoul, such as CDC stream synchronization and automatic schema evolution, to jointly build a unified, high-performance, real-time lakehouse platform.

This article will explain how to quickly set up an Apache Doris + LakeSoul testing & demonstration environment and demonstrate the usage of various features, showcasing the integration and benefits of using both systems in a lakehouse architecture .

For more information, please refer to LakeSoul Catalog

User Guide

All scripts and code mentioned in this article can be obtained from this address:
https://github.com/apache/doris/tree/master/samples/datalake/lakesoul

01 Environment Preparation

This article uses Docker Compose for deployment, with the following components and versions:

Component	Version
Apache Doris	Default 3.0.2
LakeSoul	2.6.1
Postgres	14.5
Apache Spark	3.3.1
Apache Flink	1.17
MinIO	RELEASE.2024-04-29T09-56-05Z

02 Environment Deployment

1. Start all components

```
bash ./start_all.sh
```

2. After starting, you can use the following script to log in to the Doris command line:

```
-- login doris
bash ./start_doris_client.sh
```

03 Data Query

As shown below, a Catalog named `lakesoul` has been created in the Doris cluster (can be viewed using `SHOW CATALOGS`). The following is the creation statement for this Catalog:

```
-- Already created
CREATE CATALOG `lakesoul` PROPERTIES (
  'type'='lakesoul',
  'lakesoul.pg.username'='lakesoul_test',
  'lakesoul.pg.password'='lakesoul_test',
  'lakesoul.pg.url'='jdbc:postgresql://lakesoul-meta-pg:5432/lakesoul_test?stringtype=unspecified',
  'minio.endpoint'='http://minio:9000',
  'minio.access_key'='admin',
  'minio.secret_key'='password'
);
```

The LakeSoul Table `lakesoul.tpch.customer` has already been loaded into Doris. Query the data in Doris. - Simple query ``sql Doris> uselakesoul.tpch`; Database changed

Doris> show tables; +————————————+ | Tables_in_tpch | +————————————+ | customer_from_spark | +————————————+ 1 row in set (0.00 sec)

Doris> select * from customer_from_spark where c_nationkey = 1 order by c_custkey limit 4; +———————+————————————+———————————————————+———————+———————————+———————+———————+——+ | c_custkey | c_name | c_address | c_nationkey | c_phone | c_acctbal | c_mktsegment | c_comment | +———————+————————————+———————————————————+———————+———————————+———————+———————+——+ | 3 | Customer#000000003 | MG9kdTD2WBHm | 1 | 11-719-748-3364 | 7498.12 | AUTOMOBILE | deposits eat slyly ironic, even instructions. express foxes detect slyly. blithely even accounts abov | | 14 | Customer#000000014 | KXkletMlL2JQEA | 1 | 11-845-129-3851 | 5266.30 | FURNITURE | , ironic packages across the unus | | 30 | Customer#000000030 | nJDsELGAavU63Jl0c5NKsKfL8rIJQQkQnYL2QJY | 1 | 11-764-165-5076 | 9321.01 | BUILDING | lithely final requests. furiously unusual account | | 59 | Customer#000000059 | zLOCP0wh92OtBihgspOGl4 | 1 | 11-355-584-3112 | 3458.60 | MACHINERY | ously final packages haggle blithely after the express deposits. furiou | +———————+————————————+———————————————————+———————+———————————+———————+———————+——+ 4 rows in set (3.14 sec)

Doris> select * from customer_from_spark where c_nationkey = 1 order by c_custkey desc limit 4; +———————+————————————+———————————————————+———————+———————————+———————+———————+——-+ | c_custkey | c_name | c_address | c_nationkey | c_phone | c_acctbal | c_mktsegment | c_comment | +———————+————————————+———————————————————+———————+———————————+———————+———————+——-+ | 14983 | Customer#000014983 | ERN3vq5Fvt4DL | 1 | 11-424-279-1846 | 841.22 | AUTOMOBILE | furiously slyly special foxes. express theodolites cajole carefully. special dinos haggle pinto | | 14968 | Customer#000014968 | ,sykKTZBzVFl7ito1750v2TRYwmkRl2nvqGHwmx | 1 | 11-669-222-9657 | 6106.77 | HOUSEHOLD | ts above the furiously even deposits haggle across | | 14961 | Customer#000014961 | JEIORcsBp6RpLYH 9gNdDyWJ |

1 | 11-490-251-5554 | 4006.35 | HOUSEHOLD | quests detect carefully final platelets! quickly final frays haggle slyly blithely final acc | | 14940 | Customer#000014940 | bNoyCxPuqSwPLjbqjEUNGN d0mSP | 1 | 11-242-677-1085 | 8829.48 | HOUSEHOLD | ver the quickly express braids. regular dependencies haggle fluffily quickly i | +———+———————+———————————+———-+———————+———+———————+ ————————————————————————————————————-+ 4 rows in set (0.10 sec) ```

- Partition pruning Doris can perform partition pruning on LakeSoul and speed up the query process through native reading. We can check this through `explain verbose`.

```
Doris> explain verbose select * from customer_from_spark where c_nationkey < 3;
+------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
| Explain String(Old Planner)
|
+------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
| PLAN FRAGMENT 0
|
|   OUTPUT EXPRS:
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_custkey`
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_name`
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_address`
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_nationkey`
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_phone`
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_acctbal`
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_mktsegment`
|
|     `lakesoul`.`tpch`.`customer_from_spark`.`c_comment`
|
|   PARTITION: UNPARTITIONED
|
|
|
|   HAS_COLO_PLAN_NODE: false
|
|
|
|   VRESULT SINK
|
|      MYSQL_PROTOCAL
|
|
|
|   1:VEXCHANGE
|
|      offset: 0
|
|      tuple ids: 0
|
|
|
| PLAN FRAGMENT 1
|
|
|
```

```
|   PARTITION: RANDOM
|
|
|
|   HAS_COLO_PLAN_NODE: false
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 01
|
|     UNPARTITIONED
|
|
|
|   0:VplanNodeName
|
|      table: customer_from_spark
|
|      predicates: (`c_nationkey` < 3)
|
|      inputSplitNum=12, totalFileSize=0, scanRanges=12
|
|      partition=0/0
|
|      backends:
|
|        10002
|
|          s3://lakesoul-test-bucket/data/tpch/customer_from_spark/c_nationkey=1/part-00000-0568c817-d6bc-4fa1-bb9e-
b311069b131c_00000.c000.parquet start: 0 length: 0 |
|          s3://lakesoul-test-bucket/data/tpch/customer_from_spark/c_nationkey=1/part-00001-d99a8fe6-61ab-4285-94da-
2f84f8746a8a_00001.c000.parquet start: 0 length: 0 |
|          s3://lakesoul-test-bucket/data/tpch/customer_from_spark/c_nationkey=1/part-00002-8a8e396f-685f-4b0f-87fa-
e2a3fe5be87e_00002.c000.parquet start: 0 length: 0 |
|          ... other 8 files ...
|
|          s3://lakesoul-test-bucket/data/tpch/customer_from_spark/c_nationkey=0/part-00003-d5b598cd-5bed-412c-a26f-
bb4bc9c937bc_00003.c000.parquet start: 0 length: 0 |
|      numNodes=1
|
|      pushdown agg=NONE
|
|      tuple ids: 0
|
|
|
| Tuples:
|
| TupleDescriptor{id=0, tbl=customer_from_spark}
|
|   SlotDescriptor{id=0, col=c_custkey, colUniqueId=0, type=int, nullable=false, isAutoIncrement=false, subColPath=null}
|
|   SlotDescriptor{id=1, col=c_name, colUniqueId=1, type=text, nullable=true, isAutoIncrement=false, subColPath=null}
|
|   SlotDescriptor{id=2, col=c_address, colUniqueId=2, type=text, nullable=true, isAutoIncrement=false, subColPath=null}
|
|   SlotDescriptor{id=3, col=c_nationkey, colUniqueId=3, type=int, nullable=false, isAutoIncrement=false, subColPath=null}
|
|   SlotDescriptor{id=4, col=c_phone, colUniqueId=4, type=text, nullable=true, isAutoIncrement=false, subColPath=null}
|
|   SlotDescriptor{id=5, col=c_acctbal, colUniqueId=5, type=decimalv3(15,2), nullable=true, isAutoIncrement=false,
subColPath=null}                                    |
|   SlotDescriptor{id=6, col=c_mktsegment, colUniqueId=6, type=text, nullable=true, isAutoIncrement=false, subColPath=null}
```

```
|
|   SlotDescriptor{id=7, col=c_comment, colUniqueId=7, type=text, nullable=true, isAutoIncrement=false, subColPath=null}
|
+----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------+
57 rows in set (0.03 sec)
```

By examining the result of the `EXPLAIN VERBOSE` statement, it can be seen that the predicate condition `c_nationkey < 3` ultimately only hits one partition (`partition=0/0`).

04 CDC Table Support

Launch Flink CDC Job to sync mysql table. The mysql table is loaded when launching docker compose.

```
bash start_flink_cdc_job.sh
```

```
Start flink-cdc job...
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/flink/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-reload4j-
1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is
automatically registered via the SPI and manual loading of the driver class is generally unnecessary.
Job has been submitted with JobID d1b3641dcd1ad85c6b373d49b1867e68
```

Flink CDC Job will be launched. We can check the process of launching at `doris client` by recreate the lakesoul catalog. After the Flink CDC Job has been launched, we can see the syncing LakeSoul CDC table at `doris client`.

```
Doris> show tables;
+---------------------+
| Tables_in_tpch      |
+---------------------+
| customer_from_spark |
+---------------------+
2 rows in set (0.00 sec)


Doris> drop catalog if exists lakesoul;
Query OK, 0 rows affected (0.00 sec)

Doris> create catalog `lakesoul`  properties ('type'='lakesoul', 'lakesoul.pg.username'='lakesoul_test',
'lakesoul.pg.password'='lakesoul_test', 'lakesoul.pg.url'='jdbc:postgresql://lakesoul-meta-pg:5432/lakesoul_test?
stringtype=unspecified', 'minio.endpoint'='http://minio:9000', 'minio.access_key'='admin', 'minio.secret_key'='password');
Query OK, 0 rows affected (0.01 sec)

Doris> show tables;
+---------------------+
| Tables_in_tpch      |
+---------------------+
| customer            |
| customer_from_spark |
+---------------------+
2 rows in set (0.00 sec)

Doris> select c_custkey, c_name, c_address, c_nationkey , c_phone, c_acctbal , c_mktsegment , c_comment from lakesoul.tpch.customer
```

```
where c_custkey < 10;
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------
+-----------------------------------------------------------------------------------------------------------------------+
| c_custkey | c_name             | c_address                             | c_nationkey | c_phone         | c_acctbal | c_mktsegment
| c_comment                                                                                                             |
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------
+-----------------------------------------------------------------------------------------------------------------------+
|         1 | Customer#000000001 | IVhzIApeRb ot,c,E                     |          15 | 25-989-741-2988 |    711.56 | BUILDING
| to the even, regular platelets. regular, ironic epitaphs nag e                                                        |
|         3 | Customer#000000003 | MG9kdTD2WBHm                          |           1 | 11-719-748-3364 |   7498.12 | AUTOMOBILE
|  deposits eat slyly ironic, even instructions. express foxes detect slyly. blithely even accounts abov               |
|         7 | Customer#000000007 | TcGe5gaZNgVePxU5kRrvXBfkasDTea        |          18 | 28-190-982-9759 |   9561.95 | AUTOMOBILE
| ainst the ironic, express theodolites. express, even pinto beans among the exp                                        |
|         8 | Customer#000000008 | I0B10bB0AymmC, 0PrRYBCP1yGJ8xcBPmWhl5 |          17 | 27-147-574-9335 |   6819.74 | BUILDING
| among the slyly regular theodolites kindle blithely courts. carefully even theodolites haggle slyly along the ide |
|         2 | Customer#000000002 | XSTf4,NCwDVaWNe6tEgvwfmRchLXak        |          13 | 23-768-687-3665 |    121.65 | AUTOMOBILE
| l accounts. blithely ironic theodolites integrate boldly: caref                                                       |
|         4 | Customer#000000004 | XxVSJsLAGtn                           |           4 | 14-128-190-5944 |   2866.83 | MACHINERY
|  requests. final, regular ideas sleep final accou                                                                     |
|         5 | Customer#000000005 | KvpyuHCplrB84WgAiGV6sYpZq7Tj          |           3 | 13-750-942-6364 |    794.47 | HOUSEHOLD
| n accounts will have to unwind. foxes cajole accor                                                                    |
|         6 | Customer#000000006 | sKZz0CsnMD7mp4Xd0YrBvx,LREYKUWAh yVn  |          20 | 30-114-968-4951 |   7638.57 | AUTOMOBILE
| tions. even deposits boost according to the slyly bold packages. final accounts cajole requests. furious          |
|         9 | Customer#000000009 | xKiAFTjUsCuxfeleNqefumTrjS            |           8 | 18-338-906-3675 |   8324.07 | FURNITURE
| r theodolites according to the requests wake thinly excuses: pending requests haggle furiousl                     |
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------
+-----------------------------------------------------------------------------------------------------------------------+
9 rows in set (1.09 sec)
```

Enter the `mysql client` and try to modify data.

```
bash start_mysql_client.sh
```

Try update row from `mysql client`.

```
mysql> update customer set c_acctbal=2211.26 where c_custkey=1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Back to `doris client` and check the data changing.

```
Doris> select c_custkey, c_name, c_address, c_nationkey , c_phone, c_acctbal , c_mktsegment , c_comment from lakesoul.tpch.customer
where c_custkey < 10;
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------
+-----------------------------------------------------------------------------------------------------------------------+
| c_custkey | c_name             | c_address                             | c_nationkey | c_phone         | c_acctbal | c_mktsegment
| c_comment                                                                                                             |
+-----------+--------------------+---------------------------------------+-------------+-----------------+-----------+-------------
+-----------------------------------------------------------------------------------------------------------------------+
|         2 | Customer#000000002 | XSTf4,NCwDVaWNe6tEgvwfmRchLXak        |          13 | 23-768-687-3665 |    121.65 | AUTOMOBILE
| l accounts. blithely ironic theodolites integrate boldly: caref                                                       |
|         4 | Customer#000000004 | XxVSJsLAGtn                           |           4 | 14-128-190-5944 |   2866.83 | MACHINERY
|  requests. final, regular ideas sleep final accou                                                                     |
|         5 | Customer#000000005 | KvpyuHCplrB84WgAiGV6sYpZq7Tj          |           3 | 13-750-942-6364 |    794.47 | HOUSEHOLD
| n accounts will have to unwind. foxes cajole accor                                                                    |
|         6 | Customer#000000006 | sKZz0CsnMD7mp4Xd0YrBvx,LREYKUWAh yVn  |          20 | 30-114-968-4951 |   7638.57 | AUTOMOBILE
| tions. even deposits boost according to the slyly bold packages. final accounts cajole requests. furious          |
|         9 | Customer#000000009 | xKiAFTjUsCuxfeleNqefumTrjS            |           8 | 18-338-906-3675 |   8324.07 | FURNITURE
| r theodolites according to the requests wake thinly excuses: pending requests haggle furiousl                     |
|         1 | Customer#000000001 | IVhzIApeRb ot,c,E                     |          15 | 25-989-741-2988 |   2211.26 | BUILDING
```

```
| to the even, regular platelets. regular, ironic epitaphs nag e                                                
```

Integration with Glue + AWS S3 Tables

AWS S3 Tables is a special type of S3 Bucket that provides read and write interfaces compatible with Apache Iceberg table format standards, built on Amazon S3, offering the same durability, availability, scalability, and performance characteristics as S3 itself. Additionally, S3 Tables provides the following features:

- Compared to Iceberg tables stored in regular S3 Buckets, S3 Tables can deliver up to 3x higher query performance and up to 10x higher transactions per second.
- Automated table management. S3 Tables automatically optimizes Iceberg table data, including small file compaction, snapshot management, and garbage file cleanup.

The release of S3 Tables further simplifies Lakehouse architecture and brings more possibilities for cloud-native lake-warehouse systems. This includes cold-hot separation, data archiving, data backup, and compute-storage separation architectures, all of which could evolve into entirely new architectures based on S3 Tables.

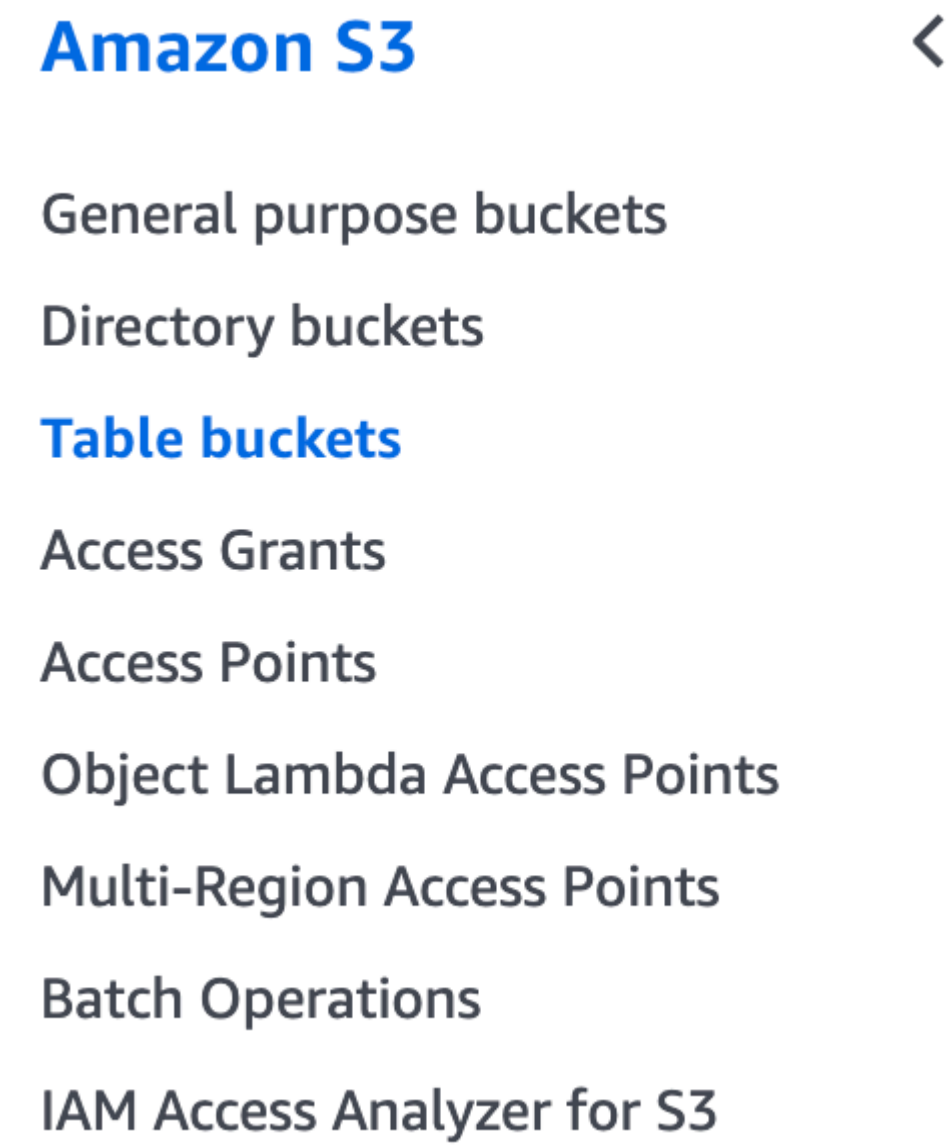
Thanks to Amazon S3 Tables' high compatibility with the Iceberg API, Apache Doris can quickly integrate with S3 Tables. This article will demonstrate how to connect Apache Doris with S3 Tables and perform data analysis and processing.

> **This feature is supported since Doris 3.1**

Usage Guide

01 Create S3 Table Bucket

S3 Table Bucket is the third type of Bucket launched by S3, on par with the previous General purpose bucket and Directory bucket.



AWS S3 Table Bucket

Here we create a Table Bucket named doris-s3-table-bucket. After creation, we will get a Table Bucket represented by an ARN.



AWS S3 Table Bucket Create

02 Create Iceberg Catalog

- Create an Iceberg Catalog of type `s3tables`

```
CREATE CATALOG iceberg_s3 PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 's3tables',
    'warehouse' = 'arn:aws:s3tables:<region>:<acount_id>:bucket/<s3_table_bucket_name>',
    's3.region' = '<region>',
```

```
    's3.endpoint' = 's3.<region>.amazonaws.com',
    's3.access_key' = '<ak>',
    's3.secret_key' = '<sk>'
);
```

- Connecting to `s3 tables` using Glue Rest Catalog

```
CREATE CATALOG glue_s3 PROPERTIES (
    'type' = 'iceberg',
    'iceberg.catalog.type' = 'rest',
    'iceberg.rest.uri' = 'https://glue.<region>.amazonaws.com/iceberg',
    'warehouse' = '<acount_id>:s3tablescatalog/<s3_table_bucket_name>',
    'iceberg.rest.sigv4-enabled' = 'true',
    'iceberg.rest.signing-name' = 'glue',
    'iceberg.rest.access-key-id' = '<ak>',
    'iceberg.rest.secret-access-key' = '<sk>',
    'iceberg.rest.signing-region' = '<region>'
);
```

03 Access S3Tables

```
Doris > SWITCH aws_s3_tables;

Doris > SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| my_namespace       |
| mysql              |
+--------------------+

Doris > USE my_namespace;

Doris > SHOW TABLES;
+------------------------+
| Tables_in_my_namespace |
+------------------------+
| my_table               |
+------------------------+

Doris > SELECT * FROM my_table;
+------+------+-------+
| id   | name | value |
+------+------+-------+
|    1 | ABC  |   100 |
|    2 | XYZ  |   200 |
+------+------+-------+
```

04 Create S3Tables Table and Write Data

```
Doris > CREATE TABLE partition_table (
    ->   `ts` DATETIME COMMENT 'ts',
    ->   `id` INT COMMENT 'col1',
    ->   `pt1` STRING COMMENT 'pt1',
    ->   `pt2` STRING COMMENT 'pt2'
    -> )
    -> PARTITION BY LIST (day(ts), pt1, pt2) ();
```

```
Doris > INSERT INTO partition_table VALUES
    -> ("2024-01-01 08:00:00", 1000, "us-east", "PART1"),
    -> ("2024-01-02 10:00:00", 1002, "us-sout", "PART2");
Query OK, 2 rows affected
{'status':'COMMITTED', 'txnId':'1736935786473'}

Doris > SELECT * FROM partition_table;
+---------------------------+------+---------+-------+
| ts                        | id   | pt1     | pt2   |
+---------------------------+------+---------+-------+
| 2024-01-02 10:00:00.000000 | 1002 | us-sout | PART2 |
| 2024-01-01 08:00:00.000000 | 1000 | us-east | PART1 |
+---------------------------+------+---------+-------+
```

05 Time Travel

We can insert another batch of data, then use the `$snapshots` system table to view Iceberg Snapshots:

```
Doris > INSERT INTO partition_table VALUES
    -> ("2024-01-03 08:00:00", 1000, "us-east", "PART1"),
    -> ("2024-01-04 10:00:00", 1002, "us-sout", "PART2");
Query OK, 2 rows affected (9.76 sec)
{'status':'COMMITTED', 'txnId':'1736935786474'}
```

```
Doris > SELECT * FROM partition_table$snapshots\G
*************************** 1. row ***************************
 committed_at: 2025-01-15 23:27:01
  snapshot_id: 6834769222601914216
    parent_id: -1
    operation: append
manifest_list: s3://80afcb3f-6edf-46f2-7fhehwj6cengfwc7n6iz7ipzakd7quse1b--table-s3/metadata/snap-6834769222601914216-1-a6b2230d-fc0d-4c1d-8f20-94bb798f27b1.avro
      summary: {"added-data-files":"2","added-records":"2","added-files-size":"5152","changed-partition-count":"2","total-records":"2","total-files-size":"5152","total-data-files":"2","total-delete-files":"0","total-position-deletes":"0","total-equality-deletes":"0","iceberg-version":"Apache Iceberg 1.6.1 (commit 8e9d59d299be42b0bca9461457cd1e95dbaad086)"}
*************************** 2. row ***************************
 committed_at: 2025-01-15 23:30:00
  snapshot_id: 5670090782912867298
    parent_id: 6834769222601914216
    operation: append
manifest_list: s3://80afcb3f-6edf-46f2-7fhehwj6cengfwc7n6iz7ipzakd7quse1b--table-s3/metadata/snap-5670090782912867298-1-beeed339-be96-4710-858b-f39bb01cc3ff.avro
      summary: {"added-data-files":"2","added-records":"2","added-files-size":"5152","changed-partition-count":"2","total-records":"4","total-files-size":"10304","total-data-files":"4","total-delete-files":"0","total-position-deletes":"0","total-equality-deletes":"0","iceberg-version":"Apache Iceberg 1.6.1 (commit 8e9d59d299be42b0bca9461457cd1e95dbaad086)"}
```

Use the `VERSION AS OF` syntax to query different snapshots:

```
Doris > SELECT * FROM partition_table FOR VERSION AS OF 5670090782912867298;
+---------------------------+------+---------+-------+
| ts                        | id   | pt1     | pt2   |
+---------------------------+------+---------+-------+
| 2024-01-04 10:00:00.000000 | 1002 | us-sout | PART2 |
| 2024-01-03 08:00:00.000000 | 1000 | us-east | PART1 |
| 2024-01-01 08:00:00.000000 | 1000 | us-east | PART1 |
| 2024-01-02 10:00:00.000000 | 1002 | us-sout | PART2 |
+---------------------------+------+---------+-------+
```

```
Doris > SELECT * FROM partition_table FOR VERSION AS OF 6834769222601914216;
+---------------------------+------+---------+-------+
| ts                        | id   | pt1     | pt2   |
+---------------------------+------+---------+-------+
| 2024-01-02 10:00:00.000000 | 1002 | us-sout | PART2 |
| 2024-01-01 08:00:00.000000 | 1000 | us-east | PART1 |
+---------------------------+------+---------+-------+
```

06 Access S3 Tables Using EMR Spark

Data written using Doris can also be accessed using Spark:

```
spark-shell --jars /usr/share/aws/iceberg/lib//iceberg-spark-runtime-3.5_2.12-1.6.1-amzn-1.jar \
--packages software.amazon.s3tables:s3-tables-catalog-for-iceberg-runtime:0.1.3 \
--conf spark.sql.catalog.s3tablesbucket=org.apache.iceberg.spark.SparkCatalog \
--conf spark.sql.catalog.s3tablesbucket.catalog-impl=software.amazon.s3tables.iceberg.S3TablesCatalog \
--conf spark.sql.catalog.s3tablesbucket.warehouse=arn:aws:s3tables:us-east-1:169698000000:bucket/doris-s3-table-bucket \
--conf spark.sql.defaultCatalog=s3tablesbucket \
--conf spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

```
scala> spark.sql("SELECT * FROM s3tablesbucket.my_namespace.`partition_table` ").show()
+-------------------+----+-------+-----+
|                 ts|  id|    pt1|  pt2|
+-------------------+----+-------+-----+
|2024-01-02 10:00:00|1002|us-sout|PART2|
|2024-01-01 08:00:00|1000|us-east|PART1|
|2024-01-04 10:00:00|1002|us-sout|PART2|
|2024-01-03 08:00:00|1000|us-east|PART1|
+-------------------+----+-------+-----+
```

Integration with Aliyun DLF Rest Catalog

Aliyun Data Lake Formation (DLF) serves as a core component of cloud-native data lake architecture, helping users quickly build cloud-native data lake architectures. Data Lake Formation provides unified metadata management on the lake, enterprise-level permission control, and seamlessly integrates with multiple computing engines to break data silos and uncover business value.

- Unified Metadata and Storage

  Computing engines share a unified set of lake metadata and storage, enabling data flow between lake ecosystem products.

- Unified Permission Management

  Computing engines share a unified set of lake table permission configurations, achieving one-time configuration with multi-location effectiveness.

- Storage Optimization

  Provides optimization strategies including small file merging, expired snapshot cleanup, partition organization, and obsolete file cleanup to improve storage efficiency.

- Comprehensive Cloud Ecosystem Support

  Deep integration with Alibaba Cloud products, including streaming and batch computing engines, enabling out-of-the-box functionality and enhancing user experience and operational convenience.

Starting from DLF version 2.5, Paimon Rest Catalog is supported. Doris, beginning from version 3.1.0, supports integration with DLF 2.5+ Paimon Rest Catalog, enabling seamless connection to DLF for accessing and analyzing Paimon table data. This document demonstrates how to use Apache Doris to connect to DLF 2.5+ and access Paimon table data.

> **This feature is supported since Doris 3.1**

Usage Guide

01 Enable DLF Service

Please refer to the DLF official documentation to enable the DLF service and create corresponding Catalog, Database, and Table.

02 Access DLF Using EMR Spark SQL

- Connection

```
spark-sql --master yarn \
    --conf spark.driver.memory=5g \
    --conf spark.sql.defaultCatalog=paimon \
    --conf spark.sql.catalog.paimon=org.apache.paimon.spark.SparkCatalog \
    --conf spark.sql.catalog.paimon.metastore=rest \
    --conf spark.sql.extensions=org.apache.paimon.spark.extensions.PaimonSparkSessionExtensions \
    --conf spark.sql.catalog.paimon.uri=http://<region>-vpc.dlf.aliyuncs.com \
    --conf spark.sql.catalog.paimon.warehouse=<your-catalog-name> \
    --conf spark.sql.catalog.paimon.token.provider=dlf \
    --conf spark.sql.catalog.paimon.dlf.token-loader=ecs
```

> Replace the corresponding `warehouse` and `uri` address.

- Write Data

```
USE <your-catalog-name>;

CREATE TABLE users_samples
(
    user_id INT,
    age_level STRING,
    final_gender_code STRING,
    clk BOOLEAN
);

INSERT INTO users_samples VALUES
(1, '25-34', 'M', true),
(2, '18-24', 'F', false);

INSERT INTO users_samples VALUES
(3, '25-34', 'M', true),
(4, '18-24', 'F', false);

INSERT INTO users_samples VALUES
(5, '25-34', 'M', true),
(6, '18-24', 'F', false);
```

If you encounter the following error, please try removing `paimon-jindo-x.y.z.jar` from `/opt/apps/PAIMON/paimon-dlf-2.5/lib/spark3` and restart the Spark service before retrying.

```
Ambiguous FileIO classes are:
org.apache.paimon.jindo.JindoLoader
org.apache.paimon.oss.OSSLoader
```

03 Connect Doris to DLF

- Create Paimon Catalog

```
CREATE CATALOG paimon_dlf_test PROPERTIES (
    'type' = 'paimon',
    'paimon.catalog.type' = 'rest',
    'uri' = 'http://<region>-vpc.dlf.aliyuncs.com',
    'warehouse' = '<your-catalog-name>',
    'paimon.rest.token.provider' = 'dlf',
```

```
    'paimon.rest.dlf.access-key-id' = '<ak>',
    'paimon.rest.dlf.access-key-secret' = '<sk>'
);
```

  - Doris will use temporary credentials returned by DLF to access OSS object storage, without requiring additional OSS credential information.
  - Only supports accessing DLF within the same VPC, ensure you provide the correct uri address.

- Query Data

```
SELECT * FROM users_samples ORDER BY user_id;
+---------+-----------+-------------------+------+
| user_id | age_level | final_gender_code | clk  |
+---------+-----------+-------------------+------+
|       1 | 25-34     | M                 |    1 |
|       2 | 18-24     | F                 |    0 |
|       3 | 25-34     | M                 |    1 |
|       4 | 18-24     | F                 |    0 |
|       5 | 25-34     | M                 |    1 |
|       6 | 18-24     | F                 |    0 |
+---------+-----------+-------------------+------+
```

- Query System Tables

```
SELECT snapshot_id, commit_time, total_record_count FROM users_samples$snapshots;
+-------------+-------------------------+--------------------+
| snapshot_id | commit_time             | total_record_count |
+-------------+-------------------------+--------------------+
|           1 | 2025-08-09 05:56:02.906 |                  2 |
|           2 | 2025-08-13 03:41:32.732 |                  4 |
|           3 | 2025-08-13 03:41:35.218 |                  6 |
+-------------+-------------------------+--------------------+
```

- Batch Incremental Reading

```
SELECT * FROM users_samples@incr('startSnapshotId'=1, 'endSnapshotId'=2) ORDER BY user_id;
+---------+-----------+-------------------+------+
| user_id | age_level | final_gender_code | clk  |
+---------+-----------+-------------------+------+
|       3 | 25-34     | M                 |    1 |
|       4 | 18-24     | F                 |    0 |
+---------+-----------+-------------------+------+
```

From MaxCompute to Doris

This document explains how to quickly import data from Alibaba Cloud MaxCompute into Apache Doris using the MaxCompute Catalog.

This document is based on Apache Doris version 2.1.9.

Environment Preparation

01 Enable MaxCompute Open Storage API

In the left navigation bar of the MaxCompute Console -> `Tenant Management` -> `Tenant Properties` -> Turn on the `Open Storage (Storage API) switch`.

02 Enable MaxCompute Permissions

Doris uses AK/SK to access MaxCompute services. Please ensure that the IAM user corresponding to the AK/SK has the following roles or permissions for the corresponding MaxCompute services:

```
{
    "Statement": [{
            "Action": ["odps:List",
                "odps:Usage"],
            "Effect": "Allow",
            "Resource": ["acs:odps:*:regions/*/quotas/pay-as-you-go"]}],
    "Version": "1"
}
```

03 Confirm Doris and MaxCompute Network Environment

It is strongly recommended that the Doris cluster and MaxCompute service are in the same VPC and ensure that the correct security group is set.

The examples in this document are tested in the same VPC network environment.

Import MaxCompute Data

01 Create Catalog

```
CREATE CATALOG mc PROPERTIES (
  "type" = "max_compute",
  "mc.default.project" = "xxx",
  "mc.access_key" = "AKxxxxx",
  "mc.secret_key" = "SKxxxxx",
  "mc.endpoint" = "xxxxx"
);
```

Support Schema Level (3.1.3+):

```
CREATE CATALOG mc PROPERTIES (
  "type" = "max_compute",
  "mc.default.project" = "xxx",
  "mc.access_key" = "AKxxxxx",
  "mc.secret_key" = "SKxxxxx",
  "mc.endpoint" = "xxxxx",
  'mc.enable.namespace.schema' = 'true'
);
```

Please refer to the MaxCompute Catalog documentation for details.

02 Import TPCH Dataset

We use the TPCH 100 dataset from the public datasets in MaxCompute as an example (data has already been imported into MaxCompute), and use the `CREATE TABLE AS SELECT` statement to import MaxCompute data into Doris.

This dataset contains 7 tables. The largest table, `lineitem`, has 16 columns and 600,037,902 rows. It occupies about 30GB of disk space.

```
-- switch catalog
SWITCH internal;
-- create database
CREATE DATABASE tpch_100g;
-- ingest data
CREATE TABLE tpch_100g.lineitem AS SELECT * FROM mc.selectdb_test.lineitem;
CREATE TABLE tpch_100g.nation AS SELECT * FROM mc.selectdb_test.nation;
CREATE TABLE tpch_100g.orders AS SELECT * FROM mc.selectdb_test.orders;
CREATE TABLE tpch_100g.part AS SELECT * FROM mc.selectdb_test.part;
CREATE TABLE tpch_100g.partsupp AS SELECT * FROM mc.selectdb_test.partsupp;
CREATE TABLE tpch_100g.region AS SELECT * FROM mc.selectdb_test.region;
CREATE TABLE tpch_100g.supplier AS SELECT * FROM mc.selectdb_test.supplier;
```

In a Doris cluster with a single BE of 16C 64G specification, the above operations take about 6-7 minutes to execute serially.

03 Import Github Event Dataset

We use the Github Event dataset from the public datasets in MaxCompute as an example (data has already been imported into MaxCompute), and use the `CREATE TABLE AS SELECT` statement to import MaxCompute data into Doris.

Here we select data from the `dwd_github_events_odps` table for the 365 partitions from '2015-01-01' to '2016-01-01'. The data has 32 columns and 212,786,803 rows. It occupies about 10GB of disk space.

```
-- switch catalog
SWITCH internal;
-- create database
CREATE DATABASE github_events;
-- ingest data
CREATE TABLE github_events.dwd_github_events_odps
AS SELECT * FROM mc.github_events.dwd_github_events_odps
WHERE ds BETWEEN '2015-01-01' AND '2016-01-01';
```

In a Doris cluster with a single BE of 16C 64G specification, the above operation takes about 2 minutes.

Generating TPC-H on Hive/Iceberg

Doris supports using the Trino Connector compatible framework to quickly build TPCH test sets with the TPCH Connector.

By combining the data write-back function of Hive/Iceberg tables, you can quickly build TPCH test datasets for Doris, Hive, and Iceberg tables through Doris.

This document mainly introduces how to deploy and use the TPCH Connector to build test datasets.

> **This feature is supported starting from Doris version 3.0.0.**

Compile TPCH Connector

> Requires JDK version 17.

```
git clone https://github.com/trinodb/trino.git
git checkout 435
cd trino/plugin/trino-tpch
mvn clean install -DskipTests
```

After compilation, you will get the `trino-tpch-435/` directory under `trino/plugin/trino-tpch/target/`.

You can also directly download the precompiled trino-tpch-435.tar.gz and extract it.

Deploy TPCH Connector

Place the `trino-tpch-435/` directory into the `connectors/` directory of all FE and BE deployment paths. (If not present, you can create it manually).

```
├─── bin
├─── conf
├─── connectors
|    ├─── trino-tpch-435
...
```

After deployment, it is recommended to restart the FE and BE nodes to ensure the Connector can be loaded correctly.

Create TPCH Catalog

```
CREATE CATALOG `tpch` PROPERTIES (
    "type" = "trino-connector",
    "trino.connector.name" = "tpch",
```

```
    "trino.tpch.column-naming" = "STANDARD",
    "trino.tpch.splits-per-node" = "32"
);
```

The `tpch.splits-per-node` is the concurrency number, which is recommended to be set to 2 times the number of cores per BE machine to achieve optimal concurrency. This improves data generation efficiency.

When `"tpch.column-naming" = "STANDARD"`, the column names in the TPCH table will start with the table name abbreviation, such as `l_orderkey`, otherwise, it is `orderkey`.

Use TPCH Catalog

The TPCH Catalog has pre-configured TPCH datasets of different Scale Factors, which can be viewed using the `SHOW DATABASES` and `SHOW TABLES` commands.

```
mysql> SWITCH tpch;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| sf1                |
| sf100              |
| sf1000             |
| sf10000            |
| sf100000           |
| sf300              |
| sf3000             |
| sf30000            |
| tiny               |
+--------------------+

mysql> USE sf1;
mysql> SHOW TABLES;
+---------------+
| Tables_in_sf1 |
+---------------+
| customer      |
| lineitem      |
| nation        |
| orders        |
| part          |
| partsupp      |
| region        |
| supplier      |
+---------------+
```

You can directly query these tables using the SELECT statement.

> **The data in these pre-configured datasets is not actually stored but is generated in real-time during queries. Therefore, these pre-configured datasets are not suitable for direct Benchmark testing. They are suitable for writing datasets into other target tables (such as Doris internal**

> **tables, Hive, Iceberg, and all other data sources that Doris supports writing to) through `INSERT INTO SELECT`, and then performing performance testing on the target tables.**

Build TPCH Test Dataset

The following example quickly builds a TPCH test dataset on Hive using the CTAS statement:

```
CREATE TABLE hive.tpch100.customer PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.customer  ;
CREATE TABLE hive.tpch100.lineitem PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.lineitem  ;
CREATE TABLE hive.tpch100.nation   PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.nation    ;
CREATE TABLE hive.tpch100.orders   PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.orders    ;
CREATE TABLE hive.tpch100.part     PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.part      ;
CREATE TABLE hive.tpch100.partsupp PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.partsupp  ;
CREATE TABLE hive.tpch100.region   PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.region    ;
CREATE TABLE hive.tpch100.supplier PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpch.sf100.supplier  ;
```

> **In a Doris cluster with 3 16C BE nodes, creating a TPCH 1000 Hive dataset takes about 25 minutes, and TPCH 10000 takes about 4 to 5 hours.**

Generating TPC-DS on Hive/Iceberg

Doris supports using the TPCDS Connector to quickly build TPCDS test sets through the Trino Connector compatible framework.

Combined with the data write-back function of Hive/Iceberg tables, you can quickly build TPCDS test data sets for Doris, Hive, and Iceberg tables through Doris.

This document mainly introduces how to deploy and use the TPCDS Connector to build test data sets.

> **This feature is supported starting from Doris version 3.0.0.**

Compile TPCDS Connector

> Requires JDK version 17.

```
git clone https://github.com/trinodb/trino.git
git checkout 435
cd trino/plugin/trino-tpcds
mvn clean install -DskipTests
```

After compilation, you will get the `trino-tpcds-435/` directory under `trino/plugin/trino-tpcds/target/`.

You can also directly download the precompiled trino-tpcds-435.tar.gz and extract it.

Deploy TPCDS Connector

Place the `trino-tpcds-435/` directory under the `connectors/` directory of all FE and BE deployment paths. (If not present, you can create it manually).

```
├─── bin
├─── conf
├─── connectors
|    ├─── trino-tpcds-435
...
```

After deployment, it is recommended to restart the FE and BE nodes to ensure the Connector can be loaded correctly.

Create TPCDS Catalog

```
CREATE CATALOG `tpcds` PROPERTIES (
    "type" = "trino-connector",
    "trino.connector.name" = "tpcds",
```

```
    "trino.tpcds.split-count" = "32"
);
```

The `tpcds.split-count` is the concurrency number, which is recommended to be set to 2 times the number of cores per BE machine to achieve optimal concurrency. Improve data generation efficiency.

Use TPCDS Catalog

The TPCDS Catalog has pre-built TPCDS datasets of different Scale Factors, which can be viewed using the `SHOW DATABASES` and `SHOW TABLES` commands.

```
mysql> SWITCH tpcds;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| sf1                |
| sf100              |
| sf1000             |
| sf10000            |
| sf100000           |
| sf300              |
| sf3000             |
| sf30000            |
| tiny               |
+--------------------+

mysql> USE sf1;
mysql> SHOW TABLES;
+------------------------+
| Tables_in_sf1          |
+------------------------+
| call_center            |
| catalog_page           |
| catalog_returns        |
| catalog_sales          |
| customer               |
| customer_address       |
| customer_demographics  |
| date_dim               |
| dbgen_version          |
| household_demographics |
| income_band            |
| inventory              |
| item                   |
| promotion              |
| reason                 |
| ship_mode              |
| store                  |
| store_returns          |
| store_sales            |
| time_dim               |
| warehouse              |
| web_page               |
| web_returns            |
| web_sales              |
```

```
| web_site               |
+------------------------+
```

You can directly query these tables using the SELECT statement.

> **The data of these pre-built datasets is not actually stored but is generated in real-time during queries. Therefore, these pre-built datasets are not suitable for direct Benchmark testing. They are suitable for writing datasets into other target tables (such as Doris internal tables, Hive, Iceberg, and other data sources that Doris supports writing to) through `INSERT INTO SELECT`, and then performing performance testing on the target tables.**

Build TPCDS Test Data Set

The following example quickly builds a TPCDS test data set on Hive using the CTAS statement:

```
CREATE TABLE hive.tpcds100.call_center            PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.call_center
;
CREATE TABLE hive.tpcds100.catalog_page           PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.catalog_page
;
CREATE TABLE hive.tpcds100.catalog_returns        PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.catalog_returns
;
CREATE TABLE hive.tpcds100.catalog_sales          PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.catalog_sales
;
CREATE TABLE hive.tpcds100.customer               PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.customer
;
CREATE TABLE hive.tpcds100.customer_address       PROPERTIES("file_format" = "parquet") AS SELECT * FROM
tpcds.sf100.customer_address      ;
CREATE TABLE hive.tpcds100.customer_demographics  PROPERTIES("file_format" = "parquet") AS SELECT * FROM
tpcds.sf100.customer_demographics ;
CREATE TABLE hive.tpcds100.date_dim               PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.date_dim
;
CREATE TABLE hive.tpcds100.dbgen_version          PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.dbgen_version
;
CREATE TABLE hive.tpcds100.household_demographics PROPERTIES("file_format" = "parquet") AS SELECT * FROM
tpcds.sf100.household_demographics;
CREATE TABLE hive.tpcds100.income_band            PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.income_band
;
CREATE TABLE hive.tpcds100.inventory              PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.inventory
;
CREATE TABLE hive.tpcds100.item                   PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.item
;
CREATE TABLE hive.tpcds100.promotion              PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.promotion
;
CREATE TABLE hive.tpcds100.reason                 PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.reason
;
CREATE TABLE hive.tpcds100.ship_mode              PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.ship_mode
;
CREATE TABLE hive.tpcds100.store                  PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.store
;
CREATE TABLE hive.tpcds100.store_returns          PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.store_returns
;
CREATE TABLE hive.tpcds100.store_sales            PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.store_sales
;
CREATE TABLE hive.tpcds100.time_dim               PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.time_dim
;
CREATE TABLE hive.tpcds100.warehouse              PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.warehouse
;
```

```
CREATE TABLE hive.tpcds100.web_page               PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.web_page
;
CREATE TABLE hive.tpcds100.web_returns            PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.web_returns
;
CREATE TABLE hive.tpcds100.web_sales              PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.web_sales
;
CREATE TABLE hive.tpcds100.web_site               PROPERTIES("file_format" = "parquet") AS SELECT * FROM tpcds.sf100.web_site
;
```

> **In a Doris cluster with 3 16C BE nodes, creating a TPCDS 1000 Hive dataset takes about 3 to 4 hours.**

Overview | Observability

Overview

What Is Observability?

Observability refers to the ability to infer a system's internal state through its external output data. An observability platform collects, stores, and visualizes three core data: Logs, Traces, and Metrics. This helps teams gain a comprehensive understanding of the operational status of distributed systems, supports resource optimization, fault prediction, root cause analysis, improves system reliability, and enhances user experience.

Why Observability Is Becoming Increasingly Important

Observability platforms have several critical use cases that are vital for improving system stability, optimizing operations efficiency, and enabling business innovation.

1. **Fault Diagnosis and Root Cause Analysis**: Real-time monitoring, anomaly detection, and tracing capabilities enable quick identification and analysis of faults. For example, in the financial industry, combining observability with transaction tracing and AI technologies can shorten recovery time and ensure business continuity. It also supports chaos engineering to simulate failure scenarios and validate system fault tolerance.

2. **Performance Optimization and Resource Planning**: Analyzing system resource utilization and response times helps identify performance bottlenecks and dynamically adjust configurations (e.g., load balancing, auto-scaling). Historical data can be used to predict resource needs, optimize cloud resource allocation, and reduce costs.

3. **Business Decision Support**: Correlating IT performance data with business outcomes (such as user retention rates and transaction volumes) helps formulate business strategies. For instance, analyzing user experience metrics can guide product feature improvements.

4. **Security and Compliance Monitoring**: Detects abnormal behaviors (e.g., zero-day attacks) and triggers automated responses to enhance system security. At the same time, log auditing ensures compliance with regulatory requirements.

5. **DevOps Collaboration**: During canary releases, traffic tagging enables tracking of new version behavior. Combined with call chain analysis, it informs release progression and helps developers optimize code performance, reducing production incidents.

The growing importance of observability in recent years is mainly driven by two factors:

1. **Increasing Complexity of Business and IT Systems**: With the development of cloud computing and microservices, business systems are becoming increasingly complex. For example, a GenAI application request might involve dozens of services such as App, service gateway, authentication service, billing service, RAG engine, Agent engine, vector database, business database, distributed cache, message queue, and large model APIs. Traditional methods like checking server status via SSH and analyzing

logs are no longer effective in such complex environments. Observability platforms unify Log, Trace, and Metric data collection and storage, providing centralized visualization and rapid issue investigation.

2. **Higher Requirements for Business Reliability**: System failures have increasingly high impacts on user experience. Therefore, the efficiency of fault detection and recovery has become more critical. Observability provides full data visibility and panoramic analytics, allowing teams to quickly locate root causes, reduce downtime, and ensure service availability. Moreover, with global data analytics and forecasting, potential resource bottlenecks can be identified early, preventing failures before they occur.

How to Choose an Observability Solution

Observability data has several characteristics, and addressing the challenges of massive data storage and analysis is key to any observability solution.

1. **High Storage Volume and Cost Sensitivity**: Observability data, especially Logs and Traces, are typically enormous in volume and generated continuously. In medium-to-large enterprises, daily data generation often reaches terabytes or even petabytes. To meet business or regulatory requirements, data must often be stored for months or even years, leading to storage volumes reaching the PB or EB scale and resulting in significant storage costs. Over time, the value of this data diminishes, making cost efficiency increasingly important.

2. **High Throughput Writes with Real-Time Requirements**: Handling daily ingestion of TB or PB-scale data offen requires write throughput ranging from 1–10 GB/s or millions to tens of millions of records per second. Simultaneously, due to the need for real-time troubleshooting and security investigations, platforms must support sub-second write latencies to ensure real-time data availability.

3. **Real-Time Analysis and Full-Text Search Capabilities**: Logs and Traces contain large amounts of textual data. Quickly searching for keywords and phrases is essential. Traditional full-scan and string-matching approaches often fail to deliver real-time performance, especially at this scale—especially under high-throughput, low-latency ingestion conditions. Thus, building inverted indexes tailored for text becomes crucial for achieving sub-second query responsiveness.

4. **Dynamic Data Schema and Frequent Expansion Needs**: Logs originally existed as unstructured free-text logs but evolved into semi-structured JSON formats. Producers frequently modify JSON fields, making schema flexibility essential. Traditional databases and data warehouses struggle to handle such dynamic schemas efficiently, while datalake systems offer storage flexibility but fall short in real-time analytical performance.

5. **Integration with Multiple Data Sources and Analysis Tools**: There are many observability ecosystem tools for data collection and visualization. The storage and analysis engine must integrate seamlessly with these diverse tools.

Given options like Elasticsearch, ClickHouse, Doris, and logging services provided by Cloud vendors, how should one choose? Here are the key evaluation criteria:

1. Performance: Includes Write and Query Performance

Since observability is often used in urgent situations like troubleshooting, queries must respond quickly—especially for textual content in Logs and Traces, which require real-time full-text search to support iterative

exploration. Additionally, users must be able to query near real-time data—queries limited to data from hours or minutes ago are insufficient; fresh data from the past few seconds is needed.

- **Elasticsearch** is known for inverted indexing and full-text search, offering sub-second retrieval. However, it struggles with high-throughput writes, often rejecting writes or experiencing high latency during peak loads. Its aggregation and statistical analysis performance is also relatively weak.
- **Cloud Logging Services** provide sufficient performance through rich resources but come with higher costs.
- **ClickHouse** delivers high write throughput and high aggregation query performance using columnar storage and vectorized execution. However, its full-text search performance lags behind Elasticsearch and Doris by multiples and remains experimental and unsuitable for production use.
- **Doris**, leveraging columnar storage and vectorized execution, optimizes inverted indexing for observability scenarios. It offers better performance than Elasticsearch, with ~5x faster writes and ~2x faster queries. Aggregation performance is up to 6–21x better than Elasticsearch.

2. Cost: Includes Storage and Compute Costs

Observability data volumes are huge, especially Logs and Traces. Medium-to-large enterprises generate TBs or even PBs of data daily. Due to business or regulatory needs, data must be retained for months or years, pushing storage requirements into the PB or even EB range. Compared to business-critical data, observability data has lower value density, and its value decreases over time, making cost sensitivity critical. Additionally, processing massive volumes of data incurs substantial compute costs.

- **Elasticsearch** suffers from high costs. Its storage model combines row-based raw data, inverted indexes, and docvalue columnar storage, with typical compression ratios around 1.5:1. High CPU overhead from JVM and index construction further increases compute costs.
- **Doris** includes numerous optimizations for observability scenarios. Compared to Elasticsearch, it reduces total cost by 50–80%. These include simplified inverted indexing, columnar storage with ZSTD compression (5:1–10:1), cold-hot tiered storage, single-replica writes, time-series compaction to reduce write amplification, and vectorized index building.
- **ClickHouse** uses columnar storage and vectorized engines, delivering lower storage and write costs.
- **Cloud Logging Services** are expensive as Elasticsearch.

3. Openness: Includes Open Source and Multi-Cloud Neutrality

When selecting an observability platform, consider openness, including whether it’s open source and multi-cloud neutral.

- **Elasticsearch** is an open-source project maintained by Elastic, available on multiple clouds. Its ELK ecosystem is self-contained and difficult to integrate with other ecosystems, eg. Kibana only supports Elasticsearch and is hard to extend.
- **Doris** is an Apache Top-Level open-source project, supported by major global cloud providers. It integrates well with OpenTelemetry, Grafana, and ELK, maintaining openness and neutrality.
- **ClickHouse** is an open-source project maintained by ClickHouse Inc., available across clouds. While it supports OpenTelemetry and Grafana, its acquisition of an observability company raises concerns about future neutrality.
- **Cloud Logging Services** are tied to their respective clouds, not open source, and differ between vendors, limiting consistent experiences and migration flexibility.

4. Ease of Use: Includes Manageability and Usability

Due to the volume of data, observability platforms usually adopt distributed architectures. Ease of deployment, scaling, upgrades, and other management tasks significantly affects scalability. The interface provided by the system determines developer efficiency and user experience.

- **Elasticsearch**'s Kibana web UI is very user-friendly and manageable. However, its DSL query language is complex and hard to learn, posing integration and development challenges.
- **Doris** provides an interactive analysis interface similar to Kibana and integrates natively with Grafana and Kibana (comming soon). Its SQL is standard and MySQL-compatible, making it developer- and analyst-friendly. Doris has a simple architecture that's easy to deploy and maintain, supports online scaling without service interruption, automatic load balancing, and includes a visual Cluster Manager.
- **ClickHouse** provides SQL interfaces but uses its own syntax. Maintenance is challenging due to exposed concepts like local tables vs. distributed tables and lack of automatic rebalancing during scaling. Typically, developing a custom cluster management system is required.
- **Cloud Logging Services** offer SaaS convenience—users don't manage infrastructure and enjoy ease of use.

Based on the above analysis, **Doris** achieves high-performance ingestion and queries while keeping costs low. Its SQL interface is easy to use, and its architecture is simple to maintain and scale. It also ensures consistent experiences across multiple clouds, making it an optimal choice for building an observability platform.

Observability Solution Based on Doris

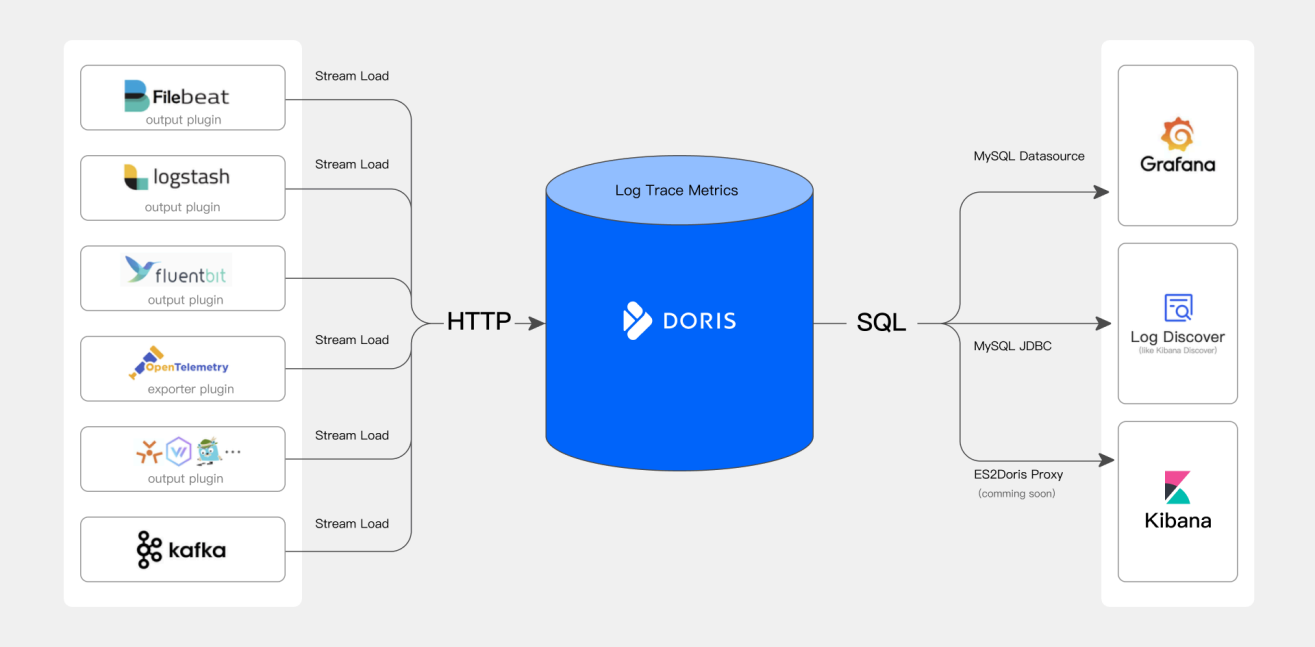
System Architecture

Apache Doris is a modern data warehouse with an MPP distributed architecture, integrating vectorized execution engines, CBO optimizers, advanced indexing, and materialized views. It supports ultra-fast querying and analysis on large-scale real-time datasets, delivering an exceptional analytical experience. Through continuous technical innovation, Doris has achieved top rankings in authoritative benchmarks such as ClickBench (single table), TPC-H, and TPC-DS (multi tables).

For observability scenarios, Doris introduces inverted indexing and ultra-fast full-text search capabilities, achieving optimized write performance and storage efficiency. This allows users to build high-performance, low-cost, and open observability platforms based on Doris.

A Doris-based observability platform consists of three core components:

- **Data Collection and Preprocessing**: Supports various observability data collection tools, including OpenTelemetry and ELK ecosystem tools like Logstash and Filebeat. Log, Trace, and Metric data are ingested into Doris via HTTP APIs.
- **Data Storage and Analysis Engine**: Doris provides unified, high-performance, low-cost storage for observability data and exposes powerful search and analysis capabilities via SQL interfaces.
- **Query Analysis and Visualization**: Integrates with popular observability visualization tools such as Grafana and Kibana (from the ELK stack), offering intuitive interfaces for searching, analyzing, alerting, and achieving real-time monitoring and rapid response.



doris-observabiltiy-architecture

Key Features and Advantages

High Performance

- **High Throughput, Low Latency Writes**: Supports stable ingestion of PB-scale (10GB/s) Log, Trace, and Metric data daily with sub-second latency.
- **High-Performance Inverted Index and Full-Text Search**: Supports inverted indexing and full-text search, delivering sub-second response times for common log keyword searches—3–10x faster than ClickHouse.
- **High-Performance Aggregation Analysis**: Utilizing MPP distributed architecture and vectorized pipeline execution engines, Doris excels in trend analysis and alerting in observability scenarios, leading globally in ClickBench tests.

Low Cost

- **High Compression Ratio and Low-Cost Storage**: Supports PB-scale storage with compression ratios of 5:1 – 10:1 (including indexes), reducing storage costs by 50–80% compared to Elasticsearch. Cold data can be offloaded to S3/HDFS, cutting storage costs by another 50%.
- **Low-Cost Writes**: Consumes 70% less CPU than Elasticsearch for the same write throughput.

Flexible Schema

- **Schema Changes at the Top Level**: Users can use Light Schema Change to add or drop columns or indexes (ADD/DROP COLUMN/INDEX), and schema modifications can be completed in seconds. When designing an observability platform, users only need to consider which fields and indexes are needed at the current stage.
- **Internal Field Changes**: A semi-structured data type called VARIANT is specially designed for scalable JSON data. It can automatically identify field names and types within JSON, and further split frequently

occurring fields into columnar storage, improving compression ratio and analytical performance. Compared to Elasticsearch's Dynamic Mapping, VARIANT allows changes in the data type of a single field.

User-Friendly

- **Standard SQL Interface**: Doris supports standard SQL and is compatible with MySQL protocols and syntax, making it accessible to engineers and analysts.
- **Integration with Observability Ecosystems**: Compatible with OpenTelemetry and ELK ecosystems, supporting Grafana and Kibana (comming soon) visualization tools for seamless data collection and analysis.
- **Easy Operations**: Supports online scaling, automatic load balancing, and visual management via Cluster Manager.

Openness

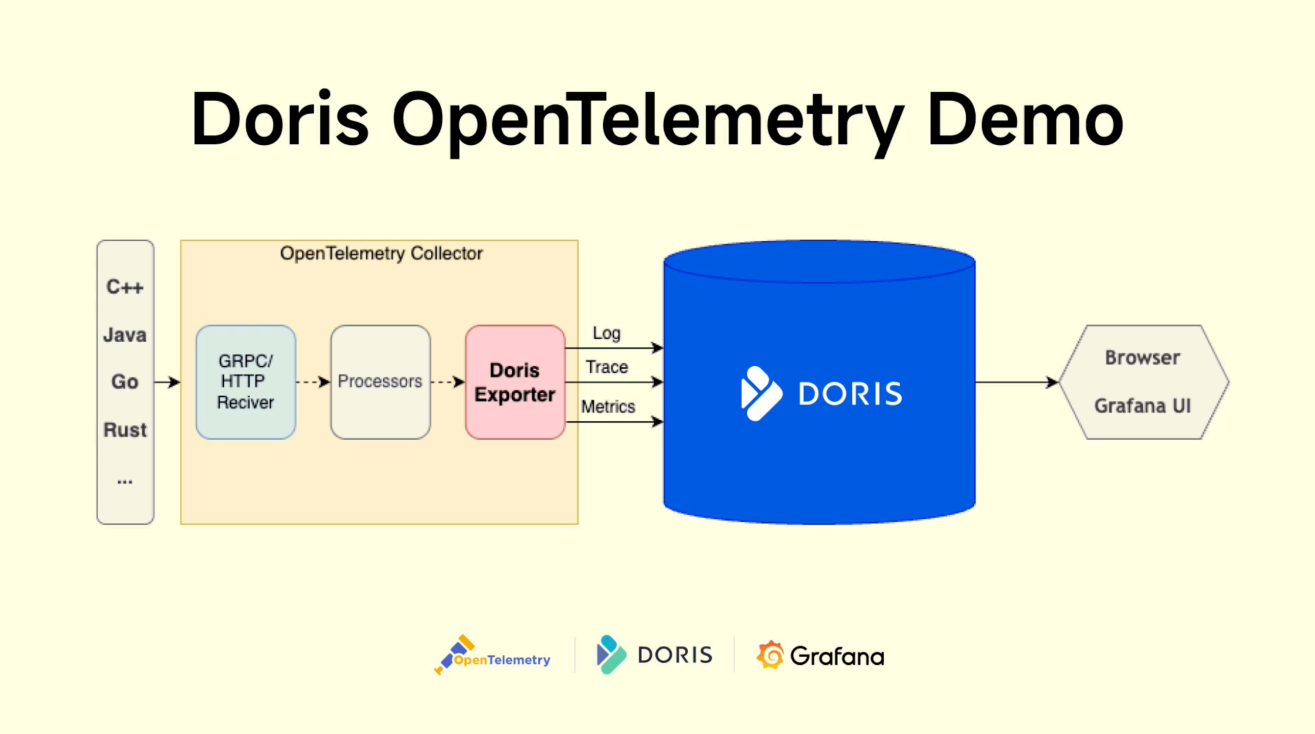
- **Open Source**: Apache Doris is a top-level open-source project adopted by over 5000 companies worldwide, supporting OpenTelemetry, Grafana, and other observability ecosystems.
- **Multi-Cloud Neutral**: Major cloud providers offer Doris SaaS services, ensuring consistent experiences across clouds.

Demo & Screenshots

We demonstrate the Doris-based observability platform using a comprehensive demo from the OpenTelemetry community.

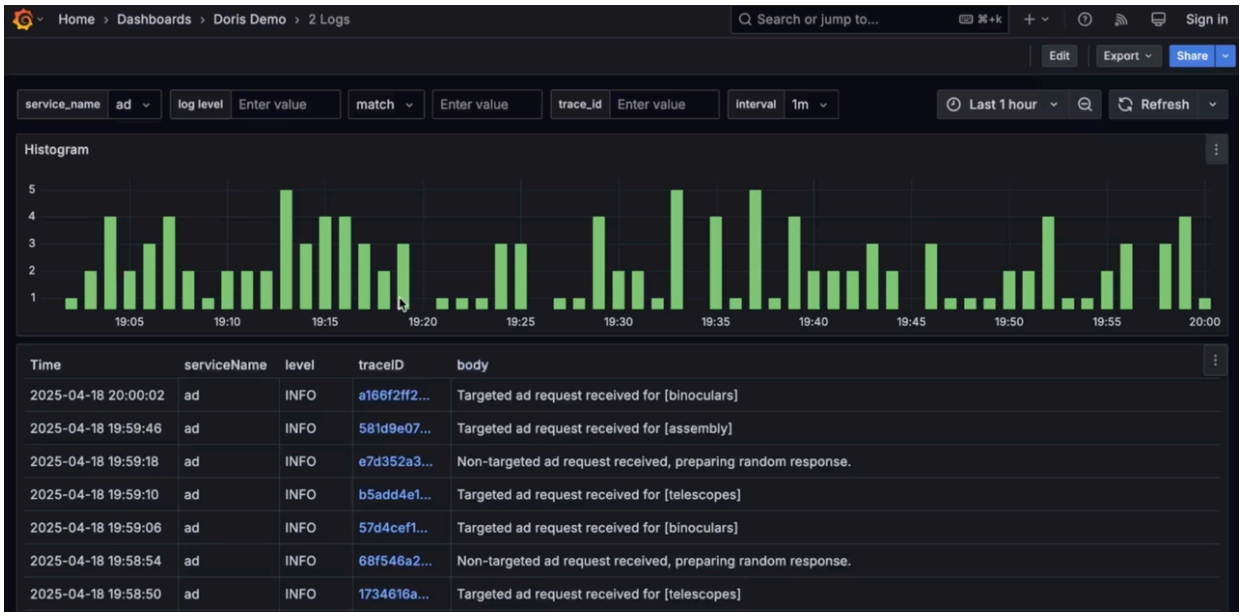
The observed business system simulates an [e-commerce website](https://opentelemetry.io/docs/demo/architecture/) composed of frontend, authentication, cart, payment, logistics, advertising, recommendation, risk control, and more than ten modules, reflecting a high level of system complexity, thus presenting significant challenges for observability data collection, storage, and analysis.

The Load Generator tool sends continuous requests to the entry service, generating vast volumes of observability data (Logs, Traces, Metrics). These data are collected using OpenTelemetry SDKs in various languages, sent to the OpenTelemetry Collector, preprocessed by Processors, and finally written into Doris via the OpenTelemetry Doris Exporter. Observability visualization tools such as Grafana connects to Doris through the MySQL interface, providing visualized query and analysis capabilities.

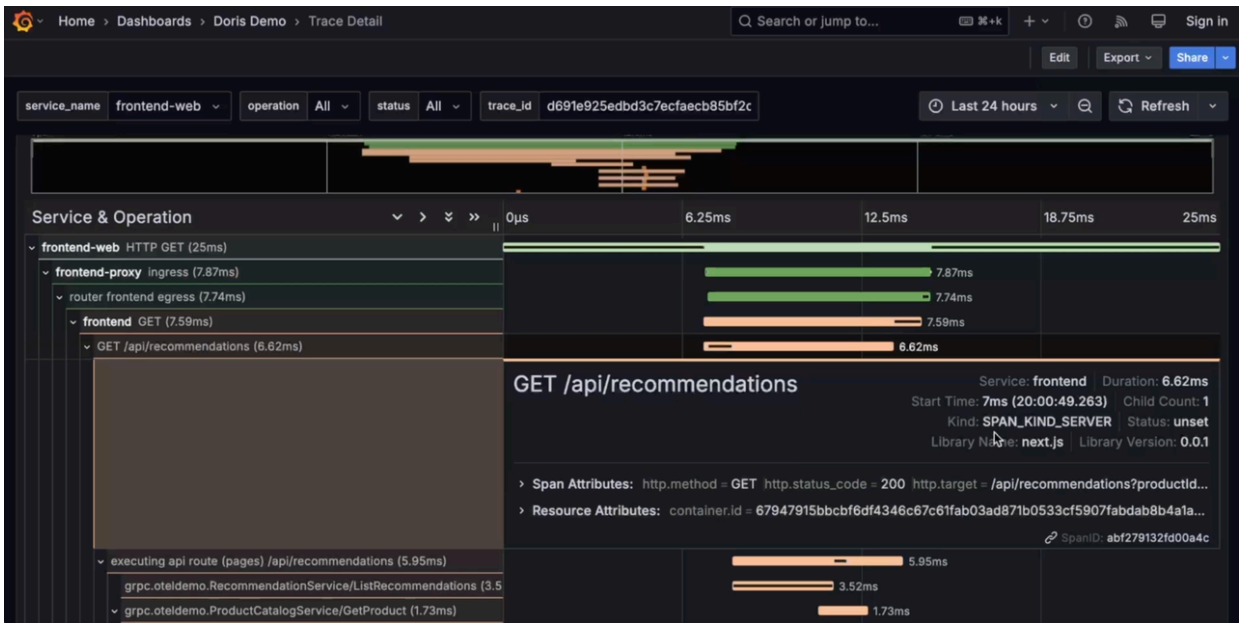


Grafana connects to Doris via MySQL datasource, offering unified visualization and analysis of Logs, Traces, and Metrics, including cross-analysis between Logs and Traces.

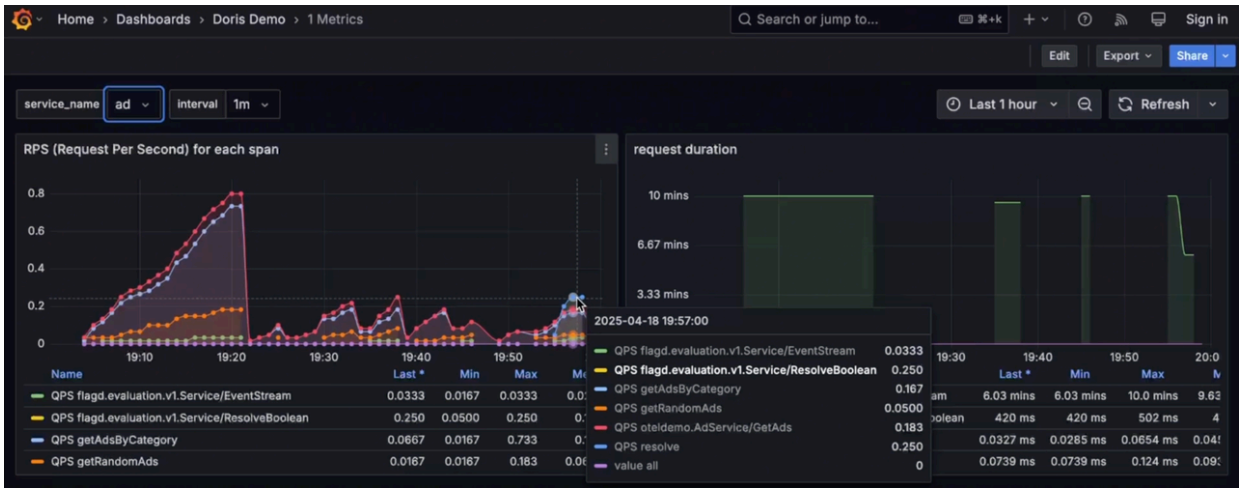
- **Log**



- **Trace**



- **Metrics**



While Grafana's log visualization and analysis capabilities are relatively basic compared to Kibana, third-party vendors have implemented Kibana-like Discover features. These will soon be integrated into Grafana's Doris datasource, enhancing unified observability visualization. Future enhancements will include Elasticsearch protocol compatibility, enabling native Kibana connections to Doris. For ELK users, replacing Elasticsearch with Doris maintains existing logging and visualization habits while significantly reducing costs and improving efficiency.



studio-visualization

Log

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

This document introduces the storage and analysis practices of Logs, one of the core observability components. For an overview of the complete observability solution, please refer to Overview.

Step 1: Estimate resources

Before deploying the cluster, you need to estimate the hardware resources required for the servers. Follow the steps below:

1. Estimate the resources for data writing by the following calculation formulas:

- `Average write throughput = Daily data increment / 86400 s`
- `Peak write throughput = Average write throughput \* Ratio of the peak write throughput to the average write throughput`
- `Number of CPU cores for the peak write throughput = Peak write throughput / Write throughput of a single-core CPU`

1. Estimate the resources for data storage by the calculation formula: `Storage space = Daily data increment / Data compression ratio * Number of data copies * Data storage duration`.
2. Estimate the resources for data querying. The resources for data querying depend on the query volume and complexity. It is recommended to reserve 50% of CPU resources for data query initially and then adjust according to the actual test results.
3. Integrate the calculation results as follows:
    1. Divide the number of CPU cores calculated in Step 1 and Step 3 by the number of CPU cores of a BE server, and you can get the number of BE servers.
    2. Based on the number of BE servers and the calculation result of Step 2, estimate the storage space required for each BE server.
    3. Allocate the storage space required for each BE server to 4 to 12 data disks, and you can get the storage capacity required for a single data disk.

For example, suppose that the daily data increment is 100 TB, the data compression ratio is 5, the number of data copies is 2, the storage duration of hot data is 3 days, the storage duration of cold data is 30 days, the ratio of the peak write throughput to the average write throughput is 200%, the write throughput of a single-core CUP is 10 MB/s, and 50% of CPU resources are reserved for data querying, one can estimate that:

compute-storage-integrated mode - 3 FE servers are required, each configured with a 16-core CPU, 64 GB memory, and an 1 100 GB SSD disk. - 30 BE servers are required, each configured with a 32-core CPU, 256 GB memory, and 8 625 GB SSD disks. - S3 object storage space 540 TB

compute-storage-decoupled mode - 3 FE servers are required, each configured with a 16-core CPU, 64 GB memory, and an 1 100 GB SSD disk. - 15 BE servers are required, each configured with a 32-core CPU, 256 GB memory, and 8 680 GB SSD disks. - S3 object storage space 600 TB

Using the storage-compute separation mode, write operations and hot data storage require only 1 replica, which can significantly reduce costs.

Refer to the following table to learn about the values of indicators in the example above and how they are calculated.

Indicator (Unit)	compute-storage-decoupled	compute-storage-integrated	Description
Daily data increment (TB)	100	100	Specify the value according to your actual needs.
Data compression ratio	5	5	Specify the value according to your actual needs, which is often between 3 to 10. Note that the data contains index data.
Number of data copies	1	2	Specify the value according to your actual needs, which can be 1, 2, or 3. The default value is 1.
Storage duration of hot data (day)	3	3	Specify the value according to your actual needs.
Storage duration of cold data (day)	30	27	Specify the value according to your actual needs.
Data storage duration	30	30	Calculation formula: `Storage duration of hot data + Storage duration of cold data`
Estimated storage space for hot data (TB)	60	120	Calculation formula: `Daily data increment / Data compression ratios * Number of data copies * Storage duration of hot data`
Estimated storage space for cold data (TB)	600	540	Calculation formula: `Daily data increment / Data compression ratios * Number of data copies * Storage duration of cold data`

Indicator (Unit)	compute-storage-decoupled	compute-storage-integrated	Description
Ratio of the peak write throughput to the average write throughput	200%	200%	Specify the value according to your actual needs. The default value is 200%.
Number of CPU cores of a BE server	32	32	Specify the value according to your actual needs. The default value is 32.
Average write throughput (MB/s)	1214	2427	Calculation formula: `Daily data increment / 86400 s`
Peak write throughput (MB/s)	2427	4855	Calculation formula: `Average write throughput * Ratio of the peak write throughput to the average write throughput`
Number of CPU cores for the peak write throughput	242.7	485.5	Calculation formula: `Peak write throughput / Write throughput of a single-core CPU`
Percent of CPU resources reserved for data querying	50%	50%	Specify the value according to your actual needs. The default value is 50%.
Estimated number of BE servers	15.2	30.3	Calculation formula: `Number of CPU cores for the peak write throughput / Number of CPU cores of a BE server /(1 - Percent of CPU resources reserved for data querying)`
Rounded number of BE servers	15	30	Calculation formula: `MAX(Number of data copies, Estimated number of BE servers)`
Estimated data storage space for each BE server (TB)	5.33	5.33	Calculation formula: `Estimated storage space for hot data / Estimated number of BE servers /(1 - 30%)`, where 30% represents the percent of reserved storage space. It is recommended to mount 4 to 12 data disks on each BE server to enhance I/O capabilities.

Step 2: Deploy the cluster

After estimating the resources, you need to deploy the cluster. It is recommended to deploy in both physical and virtual environments manually. For manual deployment, refer to Manual Deployment.

Step 3: Optimize FE and BE configurations

After completing the cluster deployment, it is necessary to optimize the configuration parameters for both the front-end and back-end separately, so as to better suit the scenario of log storage and analysis.

Optimize FE configurations

You can find FE configuration fields in `fe/conf/fe.conf`. Refer to the following table to optimize FE configurations.

Configuration fields to be optimized	Description
`max_running_txn_num_per_db = 10000`	Increase the parameter value to adapt to high-concurrency import transactions.
`streaming_label_keep_max_second = 3600 label_keep_max_second = 7200`	Increase the retention time to handle high-frequency import transactions with high memory usage.
`enable_round_robin_create_tablet = true`	When creating Tablets, use a Round Robin strategy to distribute evenly.
`tablet_rebalancer_type = partition`	When balancing Tablets, use a strategy to evenly distribute within each partition.
`autobucket_min_buckets = 10`	Increase the minimum number of automatically bucketed buckets from 1 to 10 to avoid insufficient buckets when the log volume increases.
`max_backend_heartbeat_failure_tolerance_count = 10`	In log scenarios, the BE server may experience high pressure, leading to short-term timeouts, so increase the tolerance count from 1 to 10.

For more information, refer to FE Configuration.

Optimize BE configurations

You can find BE configuration fields in `be/conf/be.conf`. Refer to the following table to optimize BE configurations.

Module	Configuration fields to be optimized	Description
Storage	`storage_root_path = /path/to/dir1;/path/to/dir2;...;/path/to/dir12`	Configure the storage path for hot data on disk directories.
-	`enable_file_cache = true`	Enable file caching.

Module	Configuration fields to be optimized	Description
-	`file_cache_path = [{"path": "/mnt/datadisk0/file_cache", "total_size":53687091200, "query_limit": "10737418240"}, {"path": "/mnt/datadisk1/file_cache", "total_size":53687091200,"query_limit": "10737418240"}]`	Configure the cache path and related settings for cold data with the following specific configurations: `path`: cache path `total_size`: total size of the cache path in bytes, where 53687091200 bytes equals 50 GB `query_limit`: maximum amount of data that can be queried from the cache path in one query in bytes, where 10737418240 bytes equals 10 GB
Write	`write_buffer_size = 1073741824`	Increase the file size of the write buffer to reduce small files and random I/O operations, improving performance.
-	`max_tablet_version_num = 20000`	In coordination with the time_series compaction strategy for table creation, allow more versions to remain temporarily unmerged
Compaction	`max_cumu_compaction_threads = 8`	Set to CPU core count / 4, indicating that 1/4 of CPU resources are used for writing, 1/4 for background compaction, and 2/1 for queries and other operations.
-	`inverted_index_compaction_enable = true`	Enable inverted index compaction to reduce CPU consumption during compaction.
-	`enable_segcompaction = false enable_ordered_data_compaction = false`	Disable two compaction features that are unnecessary for log scenarios.
-	`enable_compaction_priority_scheduling = false`	Low-priority compaction is limited to 2 tasks on a single disk, which can affect the speed of compaction.
-	`total_permits_for_compaction_score = 200000`	The parameter is used to control memory, under the memory time series strategy, the parameter itself can control memory.
Cache	`disable_storage_page_cache = true inverted_index_searcher_cache_limit = 30%`	Due to the large volume of log data and limited caching effect, switch from data caching to index caching.
-	`inverted_index_cache_stale_sweep_time_sec = 3600 index_cache_entry_stay_time_after_lookup_s = 3600`	Maintain index caching in memory for up to 1 hour.
-	`enable_inverted_index_cache_on_cooldown = true enable_write_index_searcher_cache = false`	Enable automatic caching of cold data storage during index uploading.
-	`tablet_schema_cache_recycle_interval = 3600 segment_cache_capacity = 20000`	Reduce memory usage by other caches.
-	`inverted_index_ram_dir_enable = true`	Reduce the IO overhead caused by writing to index files temporarily.
Thread	`pipeline_executor_size = 24 doris_scanner_thread_pool_thread_num = 48`	Configure computing threads and I/O threads for a 32-core CPU in proportion to core count.
-	`scan_thread_nice_value = 5`	Lower the priority of query I/O threads to ensure writing performance and timeliness.
Other	`string_type_length_soft_limit_bytes = 10485760`	Increase the length limit of string-type data to 10 MB.
-	`trash_file_expire_time_sec = 300 path_gc_check_interval_second = 900 path_scan_interval_second = 900`	Accelerate the recycling of trash files.

For more information, refer to BE Configuration.

Step 4: Create tables

Due to the distinct characteristics of both writing and querying log data, it is recommended to configure tables with targeted settings to enhance performance.

Configure data partitioning and bucketing

- For data partitioning:
    - Enable range partitioning (`PARTITION BY RANGE(ts)`) with dynamic partitions (`"dynamic_partition.enable" = "true"`) managed automatically by day.
    - Use a field in the DATETIME type as the sort key (`DUPLICATE KEY(ts)`) for accelerated retrieval of the latest N log entries.
- For data bucketing:
    - Configure the number of buckets to be roughly three times the total number of disks in the cluster, with each bucket containing approximately 5GB of data after compression.
    - Use the Random strategy (`DISTRIBUTED BY RANDOM BUCKETS 60`) to optimize batch writing efficiency when paired with single tablet imports.

For more information, refer to Data Partitioning.

Configure compression parameters

Use the zstd compression algorithm (“compression” = “zstd”) to improve data compression efficiency.

Configure compaction parameters

Configure compaction fields as follows:

- Use the time_series strategy (`"compaction_policy" = "time_series"`) to reduce write amplification, which is crucial for high-throughput log writes.

Configure index parameters

Configuring index fields as follows:

- Create indexes for fields that are frequently queried (`USING INVERTED`).
- For fields that require full-text search, specify the parser field as unicode, which satisfies most requirements. If there is a need to support phrase queries, set the support_phrase field to true; if not needed, set it to false to reduce storage space.

Configure storage parameters

Configure storage policies as follows:

- For storage of hot data, if using cloud storage, configure the number of data copies as 1; if using physical disks, configure the number of data copies as at least 2 (`"replication_num" = "2"`).

- Configure the storage location for log_s3 (`CREATE RESOURCE "log_s3"`) and set the log_policy_3day policy (`CREATE STORAGE POLICY log_policy_3day`), where the data is cooled and moved to the specified storage location of log_s3 after 3 days. Refer to the SQL below.

```
CREATE DATABASE log_db;
USE log_db;

-- unneccessary for the compute-storage-decoupled mode
CREATE RESOURCE "log_s3"
PROPERTIES
(
    "type" = "s3",
    "s3.endpoint" = "your_endpoint_url",
    "s3.region" = "your_region",
    "s3.bucket" = "your_bucket",
    "s3.root.path" = "your_path",
    "s3.access_key" = "your_ak",
    "s3.secret_key" = "your_sk"
);

-- unneccessary for the compute-storage-decoupled mode
CREATE STORAGE POLICY log_policy_3day
PROPERTIES(
    "storage_resource" = "log_s3",
    "cooldown_ttl" = "259200"
);

CREATE TABLE log_table
(
  `ts` DATETIME,
  `host` TEXT,
  `path` TEXT,
  `message` TEXT,
  INDEX idx_host (`host`) USING INVERTED,
  INDEX idx_path (`path`) USING INVERTED,
  INDEX idx_message (`message`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true")
)
ENGINE = OLAP
DUPLICATE KEY(`ts`)
PARTITION BY RANGE(`ts`) ()
DISTRIBUTED BY RANDOM BUCKETS 60
PROPERTIES (
  "compression" = "zstd",
  "compaction_policy" = "time_series",
  "dynamic_partition.enable" = "true",
  "dynamic_partition.create_history_partition" = "true",
  "dynamic_partition.time_unit" = "DAY",
  "dynamic_partition.start" = "-30",
  "dynamic_partition.end" = "1",
  "dynamic_partition.prefix" = "p",
  "dynamic_partition.buckets" = "60",
  "dynamic_partition.replication_num" = "2", -- unneccessary for the compute-storage-decoupled mode
  "replication_num" = "2", -- unneccessary for the compute-storage-decoupled mode
  "storage_policy" = "log_policy_3day" -- unneccessary for the compute-storage-decoupled mode
);
```

Step 5: Collect logs

After completing table creation, you can proceed with log collection.

Apache Doris provides open and versatile Stream HTTP APIs, through which you can connect with popular log collectors such as Logstash, Filebeat, Kafka, and others to carry out log collection work. This section explains how to integrate these log collectors using the Stream HTTP APIs.

Integrating Logstash

Follow these steps:

1. Download and install the Logstash Doris Output plugin. You can choose one of the following two methods:
    - Click to download and install.
    - Compile from the source code and run the following command to install:

```
./bin/logstash-plugin install logstash-output-doris-1.2.0.gem
```

2. Configure Logstash. Specify the following fields:

- `logstash.yml`: Used to configure Logstash batch processing log sizes and timings for improved data writing performance.

```
Plain Text pipeline.batch.size: 1000000 pipeline.batch.delay: 10000
```

- `logstash_demo.conf`: Used to configure the specific input path of the collected logs and the settings for output to Apache Doris.

```
input {
    file {
    path => "/path/to/your/log"
  }
}

output {
  doris {
    http_hosts => [ "<http://fehost1:http_port>", "<http://fehost2:http_port>", "<http://fehost3:http_port>" ]
    user => "your_username"
    password => "your_password"
    db => "your_db"
    table => "your_table"

    # doris stream load http headers
    headers => {
    "format" => "json"
    "read_json_by_line" => "true"
    "load_to_single_tablet" => "true"
    }

    # field mapping: doris fileld name => logstash field name
    # %{} to get a logstash field, [] for nested field such as [host][name] for host.name
    mapping => {
    "ts" => "%{@timestamp}"
    "host" => "%{[host][name]}"
    "path" => "%{[log][file][path]}"
    "message" => "%{message}"
    }
    log_request => true
    log_speed_interval => 10
```

```
  }
}
    ```

3. Run Logstash according to the command below, collect logs, and output to Apache Doris.

```shell
./bin/logstash -f logstash_demo.conf
```

For more information about the Logstash Doris Output plugin, see Logstash Doris Output Plugin.

Integrating Filebeat

Follow these steps:

1. Obtain the Filebeat binary file that supports output to Apache Doris. You can click to download or compile it from the Apache Doris source code.

2. Configure Filebeat. Specify the filebeat_demo.yml field that is used to configure the specific input path of the collected logs and the settings for output to Apache Doris.

```
# input
filebeat.inputs:
- type: log
enabled: true
paths:
    - /path/to/your/log
multiline:
    type: pattern
    pattern: '^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}'
    negate: true
    match: after
    skip_newline: true

processors:
- script:
    lang: javascript
    source: >
        function process(event) {
            var msg = event.Get("message");
            msg = msg.replace(/\t/g, "  ");
            event.Put("message", msg);
        }
- dissect:
    # 2024-06-08 18:26:25,481 INFO (report-thread|199) [ReportHandler.cpuReport():617] begin to handle
    tokenizer: "%{day} %{time} %{log_level} (%{thread}) [%{position}] %{content}"
    target_prefix: ""
    ignore_failure: true
    overwrite_keys: true

# queue and batch
queue.mem:
events: 1000000
flush.min_events: 100000
flush.timeout: 10s

# output
output.doris:
fenodes: [ "http://fehost1:http_port", "http://fehost2:http_port", "http://fehost3:http_port" ]
user: "your_username"
password: "your_password"
```

```
database: "your_db"
table: "your_table"
# output string format
codec_format_string: '{"ts": "%{[day]} %{[time]}", "host": "%{[agent][hostname]}", "path": "%{[log][file][path]}", "message": "%
{[message]}"}'
headers:
    format: "json"
    read_json_by_line: "true"
    load_to_single_tablet: "true"
```

3. Run Filebeat according to the command below, collect logs, and output to Apache Doris.

```
chmod +x filebeat-doris-2.1.1
./filebeat-doris-2.1.1 -c filebeat_demo.yml
```

For more information about Filebeat, refer to Beats Doris Output Plugin.

Integrating Kafka

Write JSON formatted logs to Kafka's message queue, create a Kafka Routine Load, and allow Apache Doris to actively pull data from Kafka.

You can refer to the example below, where `property.*` represents Librdkafka client-related configurations and needs to be adjusted according to the actual Kafka cluster situation.

```
CREATE ROUTINE LOAD load_log_kafka ON log_db.log_table
COLUMNS(ts, host, path, message)
PROPERTIES (
"max_batch_interval" = "60",
"max_batch_rows" = "20000000",
"max_batch_size" = "1073741824",
"load_to_single_tablet" = "true",
"format" = "json"
)
FROM KAFKA (
"kafka_broker_list" = "host:port",
"kafka_topic" = "log__topic_",
"property.group.id" = "your_group_id",
"property.security.protocol"="SASL_PLAINTEXT",
"property.sasl.mechanism"="GSSAPI",
"property.sasl.kerberos.service.name"="kafka",
"property.sasl.kerberos.keytab"="/path/to/xxx.keytab",
"property.sasl.kerberos.principal"="<xxx@yyy.com>"
);
<br />SHOW ROUTINE LOAD;
```

For more information about Kafka, see Routine Load.

Using customized programs to collect logs

In addition to integrating common log collectors, you can also customize programs to import log data into Apache Doris using the Stream Load HTTP API. Refer to the following code:

```
curl
--location-trusted
-u username:password
-H "format:json"
```

```
-H "read_json_by_line:true"
-H "load_to_single_tablet:true"
-H "timeout:600"
-T logfile.json
http://fe_host:fe_http_port/api/log_db/log_table/_stream_load
```

When using custom programs, pay attention to the following key points:

- Use Basic Auth for HTTP authentication, calculate using the command echo -n 'username:password' | base64.
- Set HTTP header "format:json" to specify the data format as JSON.
- Set HTTP header "read_json_by_line:true" to specify one JSON per line.
- Set HTTP header "load_to_single_tablet:true" to import data into one bucket at a time to reduce small file imports.
- It is recommended to write batches whose sizes are between 100MB to 1GB on the client side. For Apache Doris version 2.1 and higher, you need to reduce batch sizes on the client side through the Group Commit function.

Step 6: Query and analyze logs

Query logs

Apache Doris supports standard SQL, so you can connect to the cluster through MySQL client or JDBC to execute SQL for log queries.

Plain Text `mysql -h fe_host -P fe_mysql_port -u your_username -Dyour_db_name`

Here are 5 common SQL query commands for reference:

- View the latest 10 log entries

```
SELECT * FROM your_table_name ORDER BY ts DESC LIMIT 10;
```

- Query the latest 10 log entries with the host as 8.8.8.8

```
SELECT * FROM your_table_name WHERE host = '8.8.8.8' ORDER BY ts DESC LIMIT 10;
```

- Retrieve the latest 10 log entries with error or 404 in the request field. In the command below, MATCH_ANY is a full-text search SQL syntax used by Apache Doris for matching any keyword in the fields.

```
SELECT * FROM your_table_name WHERE message **MATCH_ANY** 'error 404'
ORDER BY ts DESC LIMIT 10;
```

- Retrieve the latest 10 log entries with image and faq in the request field. In the command below, MATCH_ALL is a full-text search SQL syntax used by Apache Doris for matching all keywords in the fields.

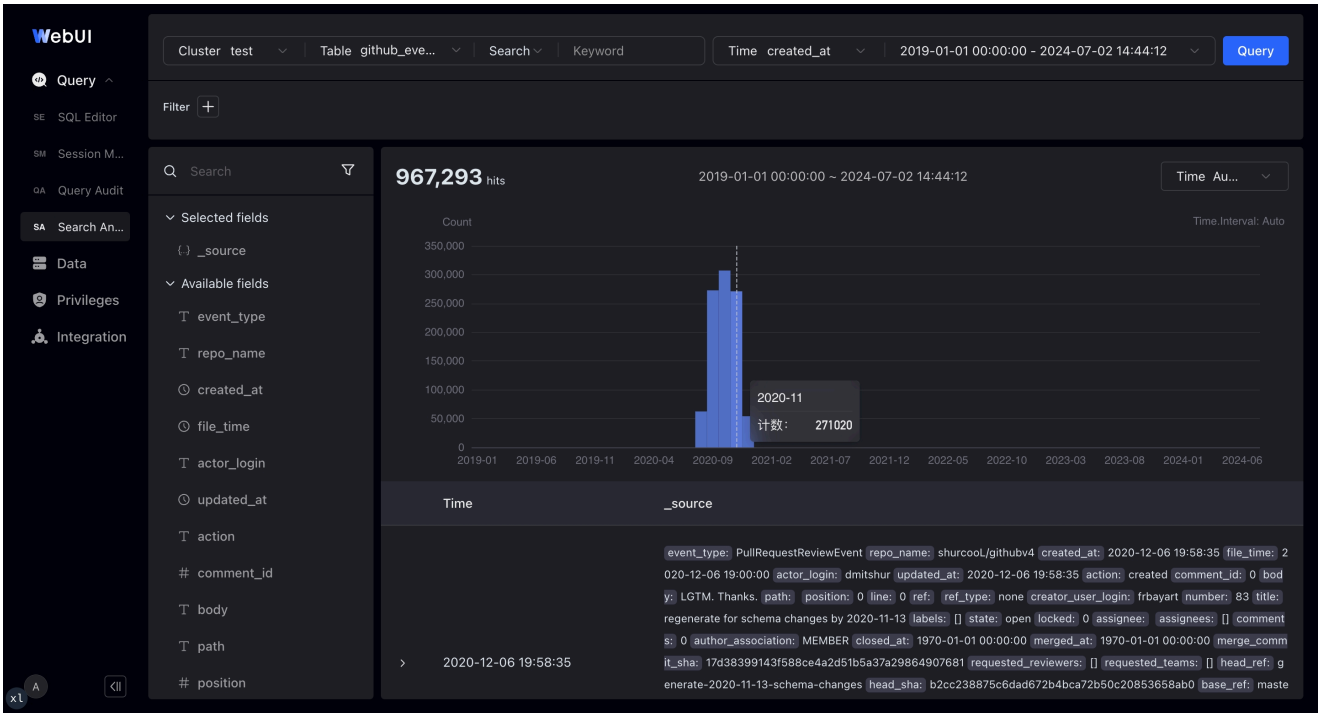
```
SELECT * FROM your_table_name WHERE message **MATCH_ALL** 'image faq'
ORDER BY ts DESC LIMIT 10;
```

- Retrieve the latest 10 entries with image and faq in the request field. In the following command, MATCH_PHRASE is a full-text search SQL syntax used by Apache Doris for matching all keywords in the fields and requiring consistent order. In the example below, a image faq b can match, but a faq image b cannot match because the order of image and faq does not match the syntax.

```
SELECT * FROM your_table_name WHERE message **MATCH_PHRASE** 'image faq'
ORDER BY ts DESC LIMIT 10;
```

Analyze logs visually

Some third-party vendors offer visual log analysis development platforms based on Apache Doris, which include a log search and analysis interface similar to Kibana Discover. These platforms provide an intuitive and user-friendly exploratory log analysis interaction.



WebUI-a log search and analysis interface similar to Kibana

- Support for full-text search and SQL modes
- Support for selecting query log timeframes with time boxes and histograms
- Display of detailed log information, expandable into JSON or tables
- Interactive clicking to add and remove filter conditions in the log data context
- Display of top field values in search results for finding anomalies and further drilling down for analysis

Please contact dev@doris.apache.org to find more.

Trace

Trace

This article introduces the storage and analysis practices of Trace, one of the core observability data. For an overview of the complete observability solution, please refer to Overview. For resource evaluation, cluster deployment, and optimization, please refer to Log.

1. Table Creation

Trace data has distinct characteristics in terms of writing and querying patterns. Targeted configurations during table creation can significantly improve performance. Create your table based on the key guidelines below:

Partitioning and Sorting - Use RANGE partitioning on the time field, enable dynamic partitioning to manage partitions automatically by day. - Use `service_name` and a time field of type DATETIME as keys; this provides multiple times acceleration when querying traces for a specific service over a certain period.

Bucketing - The number of buckets should be approximately three times the total number of disks in the cluster. - Use the RANDOM bucketing strategy. Combined with single-tablet ingestion during writes, it improves batch write efficiency.

Compaction - Use the time_series compaction strategy to reduce write amplification, which is crucial for optimizing resources under high-throughput ingestion.

VARIANT Data Type - Use the semi-structured VARIANT data type for extended Trace fields like `span_attributes` and `resource_attributes`. This automatically splits JSON data into sub-columns for storage, improving compression rates and reducing storage space while also enhancing filtering and sub-column analysis performance.

Indexing - Build indexes on frequently queried fields. - For fields requiring full-text search, specify the parser parameter. Unicode tokenization generally meets most needs. Enable the `support_phrase` option to support phrase queries. If not needed, set it to false to reduce storage usage.

Storage - For hot data, configure 1 replica if using cloud disks or at least 2 replicas if using physical disks. - Use hot-cold tiered storage configuration with `log_s3` object storage and `log_policy_3day` policy to move data older than 3 days to S3.

```
CREATE DATABASE log_db;
USE log_db;

-- Not required for compute-storage separation mode
CREATE RESOURCE "log_s3"
PROPERTIES
(
    "type" = "s3",
    "s3.endpoint" = "your_endpoint_url",
    "s3.region" = "your_region",
    "s3.bucket" = "your_bucket",
    "s3.root.path" = "your_path",
    "s3.access_key" = "your_ak",
```

```
    "s3.secret_key" = "your_sk"
);

-- Not required for compute-storage separation mode
CREATE STORAGE POLICY log_policy_3day
PROPERTIES(
    "storage_resource" = "log_s3",
    "cooldown_ttl" = "259200"
);

CREATE TABLE trace_table
(
    service_name          VARCHAR(200),
    timestamp             DATETIME(6),
    service_instance_id   VARCHAR(200),
    trace_id              VARCHAR(200),
    span_id               STRING,
    trace_state           STRING,
    parent_span_id        STRING,
    span_name             STRING,
    span_kind             STRING,
    end_time              DATETIME(6),
    duration              BIGINT,
    span_attributes       VARIANT,
    events                ARRAY<STRUCT<timestamp:DATETIME(6), name:STRING, attributes:MAP<STRING, STRING>>>,
    links                 ARRAY<STRUCT<trace_id:STRING, span_id:STRING, trace_state:STRING, attributes:MAP<STRING, STRING>>>,
    status_message        STRING,
    status_code           STRING,
    resource_attributes   VARIANT,
    scope_name            STRING,
    scope_version         STRING,
    INDEX idx_timestamp(timestamp) USING INVERTED,
    INDEX idx_service_instance_id(service_instance_id) USING INVERTED,
    INDEX idx_trace_id(trace_id) USING INVERTED,
    INDEX idx_span_id(span_id) USING INVERTED,
    INDEX idx_trace_state(trace_state) USING INVERTED,
    INDEX idx_parent_span_id(parent_span_id) USING INVERTED,
    INDEX idx_span_name(span_name) USING INVERTED,
    INDEX idx_span_kind(span_kind) USING INVERTED,
    INDEX idx_end_time(end_time) USING INVERTED,
    INDEX idx_duration(duration) USING INVERTED,
    INDEX idx_span_attributes(span_attributes) USING INVERTED,
    INDEX idx_status_message(status_message) USING INVERTED,
    INDEX idx_status_code(status_code) USING INVERTED,
    INDEX idx_resource_attributes(resource_attributes) USING INVERTED,
    INDEX idx_scope_name(scope_name) USING INVERTED,
    INDEX idx_scope_version(scope_version) USING INVERTED
)
ENGINE = OLAP
DUPLICATE KEY(service_name, timestamp)
PARTITION BY RANGE(timestamp) ()
DISTRIBUTED BY RANDOM BUCKETS 250
PROPERTIES (
"compression" = "zstd",
"compaction_policy" = "time_series",
"inverted_index_storage_format" = "V2",
"dynamic_partition.enable" = "true",
"dynamic_partition.create_history_partition" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-30",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "250",
"dynamic_partition.replication_num" = "2", -- Not required for compute-storage separation
"replication_num" = "2", -- Not required for compute-storage separation
```

```
"storage_policy" = "log_policy_3day" -- Not required for compute-storage separation
);
```

2. Trace Collection

Doris provides open and general-purpose Stream HTTP APIs that can integrate with Trace collection systems like OpenTelemetry.

OpenTelemetry Integration

1. **Application-side Integration with OpenTelemetry SDK**

Here we use a Spring Boot example application integrated with the OpenTelemetry Java SDK. The example application comes from the official demo, which returns a simple “Hello World!” string for requests to the path “/”.
Download the OpenTelemetry Java Agent. The advantage of using the Java Agent is that no modifications are needed to existing application. For other languages and integration methods, see the OpenTelemetry official website Language APIs & SDKs or Zero-code Instrumentation.

1. **Deploy and Configure OpenTelemetry Collector**

Download and extract OpenTelemetry Collector. You need to download the package starting with “otelcol-contrib”, which includes the Doris Exporter.

Create the `otel_demo.yaml` configuration file as follows. For more details, refer to the Doris Exporter documentation.

```
receivers:
  otlp: # OTLP protocol, receiving data sent by the OpenTelemetry Java Agent
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

processors:
  batch:
    send_batch_size: 100000 # Number of records per batch; recommended batch size between 100MB-1GB
    timeout: 10s

exporters:
  doris:
    endpoint: http://localhost:8030 # FE HTTP address
    database: doris_db_name
    username: doris_username
    password: doris_password
    table:
      traces: doris_table_name
    create_schema: true # Whether to auto-create schema; manual table creation is needed if set to false
    mysql_endpoint: localhost:9030  # FE MySQL address
    history_days: 10
    create_history_days: 10
    timezone: Asia/Shanghai
    timeout: 60s # Timeout for HTTP stream load client
    log_response: true
    sending_queue:
      enabled: true
```

```
    num_consumers: 20
    queue_size: 1000
  retry_on_failure:
    enabled: true
    initial_interval: 5s
    max_interval: 30s
  headers:
    load_to_single_tablet: "true"
```

1. **Run OpenTelemetry Collector**

```
./otelcol-contrib --config otel_demo.yaml
```

4. **Start the Spring Boot Example Application**

Before starting the application, simply add a few environment variables without modifying any code.

```
export JAVA_TOOL_OPTIONS="${JAVA_TOOL_OPTIONS} -javaagent:/your/path/to/opentelemetry-javaagent.jar" # Path to OpenTelemetry Java
Agent
export OTEL_JAVAAGENT_LOGGING="none" # Disable Otel logs to prevent interference with application logs
export OTEL_SERVICE_NAME="myproject"
export OTEL_TRACES_EXPORTER="otlp" # Send trace data using OTLP protocol
export OTEL_EXPORTER_OTLP_ENDPOINT="http://localhost:4317" # Address of the OpenTelemetry Collector

java -jar myproject-0.0.1-SNAPSHOT.jar
```

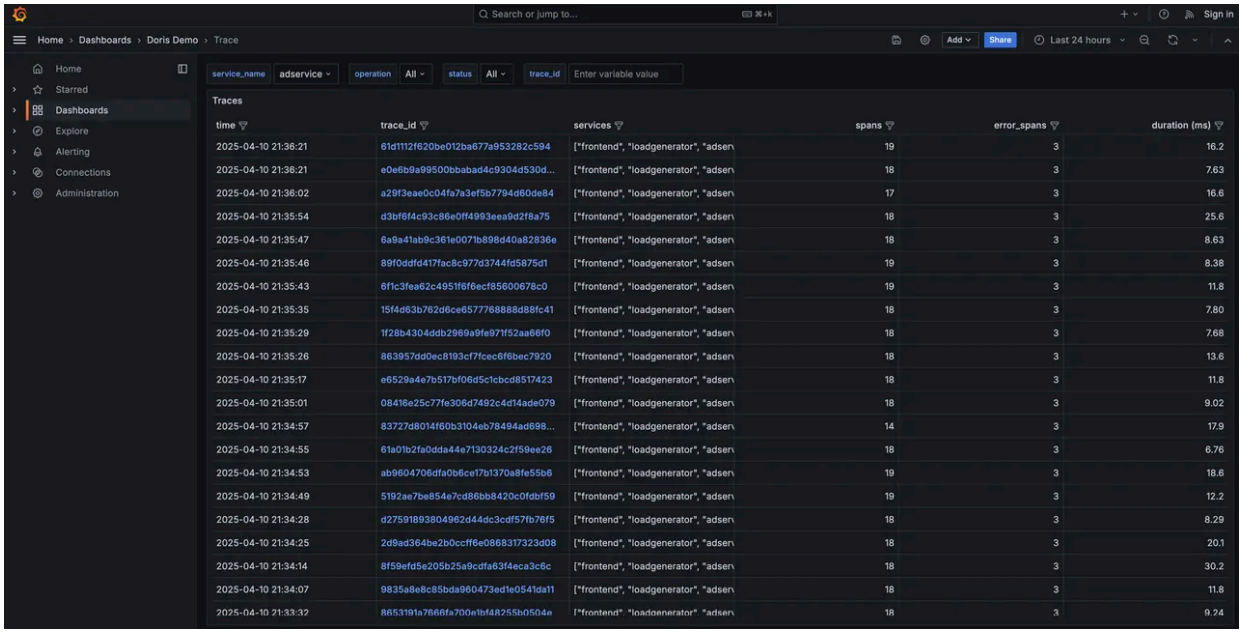
5. **Access the Spring Boot Example Service to Generate Trace Data**

Running `curl localhost:8080` will trigger a call to the `hello` service. The OpenTelemetry Java Agent will automatically generate Trace data and send it to the OpenTelemetry Collector, which then writes the Trace data to the Doris table (default is `otel.otel_traces`) via the configured Doris Exporter.

3. Trace Querying

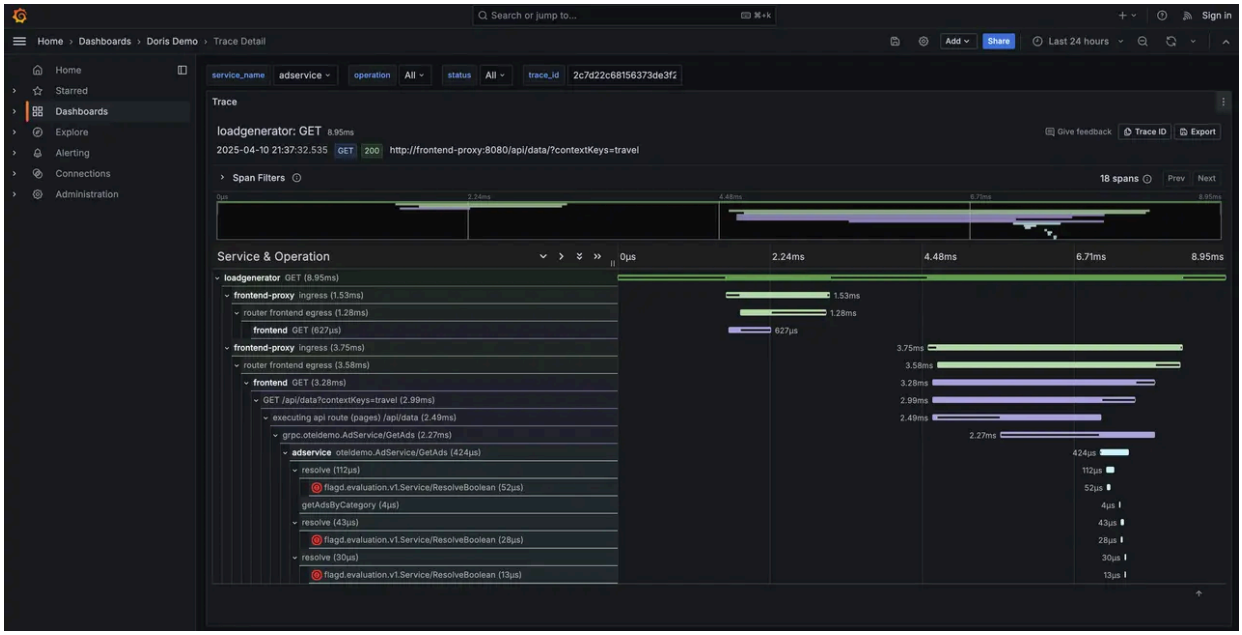
Trace querying typically uses visual query interfaces such as Grafana.

- Filter by time range and service name to display Trace summaries, including latency distribution charts and detailed individual Traces.



Trace List

- Click on the link to view the Trace detail.



Trace Detail

Logstash

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

Logstash Doris output plugin

Introduction

Logstash is a log ETL framework (collect, preprocess, send to storage systems) that supports custom output plugins to write data into storage systems. The Logstash Doris output plugin is a plugin for outputting data to Doris.

The Logstash Doris output plugin calls the Doris Stream Load HTTP interface to write data into Doris in real-time, offering capabilities such as multi-threaded concurrency, failure retries, custom Stream Load formats and parameters, and output write speed.

Using the Logstash Doris output plugin mainly involves three steps: 1. Install the plugin into Logstash 2. Configure the Doris output address and other parameters 3. Start Logstash to write data into Doris in real-time

Installation

Obtaining the Plugin

You can download the plugin from the official website or compile it from the source code yourself.

- Download from the official website

```
# Installation package with dependencies
wget https://assets-cdn.selectdb.com/downloads/releases/extension/logstash-output-doris-1.2.0-java.gem
```

- Compile from source code

```
cd extension/logstash/

gem build logstash-output-doris.gemspec
```

Installing the Plugin

- Standard Installation

`${LOGSTASH_HOME}` is the installation directory of Logstash. Run the `bin/logstash-plugin` command under it to install the plugin.

```
${LOGSTASH_HOME}/bin/logstash-plugin install logstash-output-doris-1.2.0.gem

Validating logstash-output-doris-1.2.0.gem
Installing logstash-output-doris
Installation successful
```

The standard installation mode will automatically install the ruby modules that the plugin depends on. In cases where the network is not available, it will get stuck and cannot complete. In such cases, you can download the zip installation package with dependencies for a completely offline installation, noting that you need to use `file://` to specify the local file system.

- Offline Installation

```
export JARS_SKIP="true"

${LOGSTASH_HOME}/bin/logstash-plugin install logstash-output-doris-1.2.0.gem
```

Configuration

The configuration for the Logstash Doris output plugin is as follows:

Configuration	Description
`http_hosts`	Stream Load HTTP address, formatted as a string array, can have one or more elements, each element is host:port. For example: ["http://fe1:8030", "http://fe2:8030"]
`user`	Doris username, this user needs to have import permissions for the corresponding Doris database and table
`password`	Password for the Doris user
`db`	The Doris database name to write into
`table`	The Doris table name to write into
`label_prefix`	Doris Stream Load Label prefix, the final generated Label is *{label_prefix}{db}{table}{yyyymmdd_hhmmss}{uuid}*, the default value is logstash
`headers`	Doris Stream Load headers parameter, the syntax format is a ruby map, for example: headers => { "format" => "json", "read_json_by_line" => "true" }
`mapping`	Mapping from Logstash fields to Doris table fields, refer to the usage examples in the subsequent sections
`message_only`	A special form of mapping, only outputs the Logstash @message field to Doris, default is false
`max_retries`	Number of retries for Doris Stream Load requests on failure, default is -1 for infinite retries to ensure data reliability

Configuration	Description
log_request	Whether to output Doris Stream Load request and response metadata in logs for troubleshooting, default is false
log_speed_interval	Time interval for outputting speed in logs, unit is seconds, default is 10, setting to 0 can disable this type of logging

Usage Example

TEXT Log Collection Example

This example demonstrates TEXT log collection using Doris FE logs as an example.

1. Data

FE log files are typically located at the fe/log/fe.log file under the Doris installation directory. They are typical Java program logs, including fields such as timestamp, log level, thread name, code location, and log content. Not only do they contain normal logs, but also exception logs with stacktraces, which are multiline. Log collection and storage need to combine the main log and stacktrace into a single log entry.

```
2024-07-08 21:18:01,432 INFO (Statistics Job Appender|61) [StatisticsJobAppender.runAfterCatalogReady():70] Stats table not
available, skip
2024-07-08 21:18:53,710 WARN (STATS_FETCH-0|208) [StmtExecutor.executeInternalQuery():3332] Failed to run internal SQL:
OriginStatement{originStmt='SELECT * FROM __internal_schema.column_statistics WHERE part_id is NULL  ORDER BY update_time DESC LIMIT
500000', idx=0}
org.apache.doris.common.UserException: errCode = 2, detailMessage = tablet 10031 has no queryable replicas. err: replica 10032's
backend 10008 does not exist or not alive
        at org.apache.doris.planner.OlapScanNode.addScanRangeLocations(OlapScanNode.java:931) ~[doris-fe.jar:1.2-SNAPSHOT]
        at org.apache.doris.planner.OlapScanNode.computeTabletInfo(OlapScanNode.java:1197) ~[doris-fe.jar:1.2-SNAPSHOT]
```

2. Table Creation

The table structure includes fields such as the log's creation time, collection time, hostname, log file path, log type, log level, thread name, code location, and log content.

```
CREATE TABLE `doris_log` (
  `log_time` datetime NULL COMMENT 'log content time',
  `collect_time` datetime NULL COMMENT 'log agent collect time',
  `host` text NULL COMMENT 'hostname or ip',
  `path` text NULL COMMENT 'log file path',
  `type` text NULL COMMENT 'log type',
  `level` text NULL COMMENT 'log level',
  `thread` text NULL COMMENT 'log thread',
  `position` text NULL COMMENT 'log code position',
  `message` text NULL COMMENT 'log message',
  INDEX idx_host (`host`) USING INVERTED COMMENT '',
  INDEX idx_path (`path`) USING INVERTED COMMENT '',
  INDEX idx_type (`type`) USING INVERTED COMMENT '',
  INDEX idx_level (`level`) USING INVERTED COMMENT '',
  INDEX idx_thread (`thread`) USING INVERTED COMMENT '',
  INDEX idx_position (`position`) USING INVERTED COMMENT '',
  INDEX idx_message (`message`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true") COMMENT ''
) ENGINE=OLAP
DUPLICATE KEY(`log_time`)
COMMENT 'OLAP'
PARTITION BY RANGE(`log_time`) ()
```

```
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"dynamic_partition.enable" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-7",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.create_history_partition" = "true",
"compaction_policy" = "time_series"
);
```

3. Logstash Configuration

Logstash mainly has two types of configuration files: one for the entire Logstash system and another for a specific log collection.

The configuration file for the entire Logstash system is usually located at config/logstash.yml. To improve performance when writing to Doris, it is necessary to modify the batch size and batch delay. For logs with an average size of a few hundred bytes per line, a batch size of 1,000,000 lines and a batch delay of 10 seconds are recommended.

```
pipeline.batch.size: 1000000
pipeline.batch.delay: 10000
```

The configuration file for a specific log collection, such as logstash_doris_log.conf, mainly consists of three parts corresponding to the various stages of ETL: 1. Input is responsible for reading the raw data. 2. Filter is responsible for data transformation. 3. Output is responsible for sending the data to the output destination.

```
# 1. input is responsible for reading raw data
# File input is an input plugin that can be configured to read the log file of the configured path. It uses the multiline codec to
concatenate lines that do not start with a timestamp to the end of the previous line, achieving the effect of merging stacktraces
with the main log. File input saves the log content in the @message field, and there are also some metadata fields such as host,
log.file.path. Here, we manually add a field named type through add_field, with its value set to fe.log.
input {
    file {
        path => "/mnt/disk2/xiaokang/opt/doris_master/fe/log/fe.log"
        add_field => {"type" => "fe.log"}
        codec => multiline {
            # valid line starts with timestamp
            pattern => "^%{TIMESTAMP_ISO8601} "
            # any line not starting with a timestamp should be merged with the previous line
            negate => true
            what => "previous"
        }
    }
}

# 2. filter section is responsible for data transformation
# grok is a commonly used data transformation plugin that has some built-in patterns, such as TIMESTAMP_ISO8601 for parsing
timestamps, and also supports writing regular expressions to extract fields.
filter {
    grok {
        match => {
            # parse log_time, level, thread, position fields from message
            "message" => "%{TIMESTAMP_ISO8601:log_time} (?<level>[A-Z]+) \((?<thread>[^\[]*)\) \[(?<position>[^\]]*)\]"
        }
    }
```

```
}

# 3. output section is responsible for data output
# Doris output sends data to Doris using the Stream Load HTTP interface. The data format for Stream Load is specified as JSON through the headers parameter, and the mapping parameter specifies the mapping from Logstash fields to JSON fields. Since headers specify "format" => "json", Stream Load will automatically parse the JSON fields and write them into the corresponding fields of the Doris table.
output {
    doris {
        http_hosts => ["http://localhost:8630"]
        user => "root"
        password => ""
        db => "log_db"
        table => "doris_log"
        headers => {
          "format" => "json"
          "read_json_by_line" => "true"
          "load_to_single_tablet" => "true"
        }
        mapping => {
          "log_time" => "%{log_time}"
          "collect_time" => "%{@timestamp}"
          "host" => "%{[host][name]}"
          "path" => "%{[log][file][path]}"
          "type" => "%{type}"
          "level" => "%{level}"
          "thread" => "%{thread}"
          "position" => "%{position}"
          "message" => "%{message}"
        }
        log_request => true
    }
}
```

4. Running Logstash

```
${LOGSTASH_HOME}/bin/logstash -f config/logstash_doris_log.conf

# When log_request is set to true, the log will output the request parameters and response results of each Stream Load.
[2024-07-08T22:35:34,772][INFO ][logstash.outputs.doris   ][main][e44d2a24f17d764647ce56f5fed24b9bbf08d3020c7fddcc3298800daface80a]
doris stream load response:
{
    "TxnId": 45464,
    "Label": "logstash_log_db_doris_log_20240708_223532_539_6c20a0d1-dcab-4b8e-9bc0-76b46a929bd1",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 452,
    "NumberLoadedRows": 452,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 277230,
    "LoadTimeMs": 1797,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 18,
    "ReadDataTimeMs": 9,
    "WriteDataTimeMs": 1758,
    "CommitAndPublishTimeMs": 18
}

# By default, speed information is logged every 10 seconds, including the amount of data since startup (in MB and ROWS), the total
```

```
speed (in MB/s and R/s), and the speed in the last 10 seconds.

[2024-07-08T22:35:38,285][INFO ][logstash.outputs.doris   ][main] total 11 MB 18978 ROWS, total speed 0 MB/s 632 R/s, last 10
seconds speed 1 MB/s 1897 R/s
```

JSON Log Collection Example

This example demonstrates JSON log collection using data from the GitHub events archive.

1. Data

The GitHub events archive contains archived data of GitHub user actions, formatted as JSON. It can be downloaded from here, for example, the data for January 1, 2024, at 3 PM.

```
wget https://data.gharchive.org/2024-01-01-15.json.gz
```

Below is a sample of the data. Normally, each piece of data is on a single line, but for ease of display, it has been formatted here.

```
{
  "id": "37066529221",
  "type": "PushEvent",
  "actor": {
    "id": 46139131,
    "login": "Bard89",
    "display_login": "Bard89",
    "gravatar_id": "",
    "url": "https://api.github.com/users/Bard89",
    "avatar_url": "https://avatars.githubusercontent.com/u/46139131?"
  },
  "repo": {
    "id": 780125623,
    "name": "Bard89/talk-to-me",
    "url": "https://api.github.com/repos/Bard89/talk-to-me"
  },
  "payload": {
    "repository_id": 780125623,
    "push_id": 17799451992,
    "size": 1,
    "distinct_size": 1,
    "ref": "refs/heads/add_mvcs",
    "head": "f03baa2de66f88f5f1754ce3fa30972667f87e81",
    "before": "85e6544ede4ae3f132fe2f5f1ce0ce35a3169d21"
  },
  "public": true,
  "created_at": "2024-04-01T23:00:00Z"
}
```

2. Table Creation

```
CREATE DATABASE log_db;
USE log_db;


CREATE TABLE github_events
(
  `created_at` DATETIME,
```

```
  `id` BIGINT,
  `type` TEXT,
  `public` BOOLEAN,
  `actor.id` BIGINT,
  `actor.login` TEXT,
  `actor.display_login` TEXT,
  `actor.gravatar_id` TEXT,
  `actor.url` TEXT,
  `actor.avatar_url` TEXT,
  `repo.id` BIGINT,
  `repo.name` TEXT,
  `repo.url` TEXT,
  `payload` TEXT,
  `host` TEXT,
  `path` TEXT,
  INDEX `idx_id` (`id`) USING INVERTED,
  INDEX `idx_type` (`type`) USING INVERTED,
  INDEX `idx_actor.id` (`actor.id`) USING INVERTED,
  INDEX `idx_actor.login` (`actor.login`) USING INVERTED,
  INDEX `idx_repo.id` (`repo.id`) USING INVERTED,
  INDEX `idx_repo.name` (`repo.name`) USING INVERTED,
  INDEX `idx_host` (`host`) USING INVERTED,
  INDEX `idx_path` (`path`) USING INVERTED,
  INDEX `idx_payload` (`payload`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true")
)
ENGINE = OLAP
DUPLICATE KEY(`created_at`)
PARTITION BY RANGE(`created_at`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"compaction_policy" = "time_series",
"enable_single_replica_compaction" = "true",
"dynamic_partition.enable" = "true",
"dynamic_partition.create_history_partition" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-30",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.replication_num" = "1"
);
```

3. Logstash Configuration

The configuration file differs from the previous TEXT log collection in the following aspects:

1. The codec parameter for file input is json. Logstash will parse each line of text as JSON format and use the parsed fields for subsequent processing.
2. No filter plugin is used because no additional processing or transformation is needed.

```
input {
    file {
        path => "/tmp/github_events/2024-04-01-23.json"
        codec => json
    }
}

output {
    doris {
        http_hosts => ["http://fe1:8630", "http://fe2:8630", "http://fe3:8630"]
        user => "root"
```

```
        password => ""
        db => "log_db"
        table => "github_events"
        headers => {
          "format" => "json"
          "read_json_by_line" => "true"
          "load_to_single_tablet" => "true"
        }
        mapping => {
          "created_at" => "%{created_at}"
          "id" => "%{id}"
          "type" => "%{type}"
          "public" => "%{public}"
          "actor.id" => "%{[actor][id]}"
          "actor.login" => "%{[actor][login]}"
          "actor.display_login" => "%{[actor][display_login]}"
          "actor.gravatar_id" => "%{[actor][gravatar_id]}"
          "actor.url" => "%{[actor][url]}"
          "actor.avatar_url" => "%{[actor][avatar_url]}"
          "repo.id" => "%{[repo][id]}"
          "repo.name" => "%{[repo][name]}"
          "repo.url" => "%{[repo][url]}"
          "payload" => "%{[payload]}"
          "host" => "%{[host][name]}"
          "path" => "%{[log][file][path]}"
        }
        log_request => true
    }
}
```

4. Running Logstash

```
${LOGSTASH_HOME}/bin/logstash -f logstash_github_events.conf
```

Filebeat

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

Beats Doris output plugin

Beats is a data collection agent that supports custom output plugins to write data into storage systems, with the Beats Doris output plugin being the one for outputting to Doris.

The Beats Doris output plugin supports Filebeat, Metricbeat, Packetbeat, Winlogbeat, Auditbeat, and Heartbeat.

By invoking the Doris Stream Load HTTP interface, the Beats Doris output plugin writes data into Doris in real-time, offering capabilities such as multi-threaded concurrency, failure retries, custom Stream Load formats and parameters, and output write speed.

To use the Beats Doris output plugin, there are three main steps: 1. Download or compile the Beats binary program that includes the Doris output plugin. 2. Configure the Beats output address and other parameters. 3. Start Beats to write data into Doris in real-time.

Installation

Download from the Official Website

https://assets-cdn.selectdb.com/downloads/releases/extension/filebeat-doris-2.1.1

Compile from Source Code

Execute the following commands in the `extension/beats/` directory:

```
cd doris/extension/beats

go build -o filebeat-doris filebeat/filebeat.go
go build -o metricbeat-doris metricbeat/metricbeat.go
go build -o winlogbeat-doris winlogbeat/winlogbeat.go
go build -o packetbeat-doris packetbeat/packetbeat.go
go build -o auditbeat-doris auditbeat/auditbeat.go
go build -o heartbeat-doris heartbeat/heartbeat.go
```

Configuration

The configuration for the Beats Doris output plugin is as follows:

Configuration	Description
`http_hosts`	Stream Load HTTP address, formatted as a string array, can have one or more elements, each element is host:port. For example: ["http://fe1:8030", "http://fe2:8030"]
`user`	Doris username, this user needs to have import permissions for the corresponding Doris database and table
`password`	Doris user's password
`database`	The Doris database name to write into
`table`	The Doris table name to write into
`label_prefix`	Doris Stream Load Label prefix, the final generated Label is *{label_prefix}{db}{table}{yyyymmdd_hhmmss}{uuid}*, the default value is beats
`headers`	Doris Stream Load headers parameter, syntax format is YAML map
`codec_format_string`	The format string for outputting to Doris Stream Load, %{[a][b]} represents the a.b field in the input, refer to the usage examples in subsequent sections
`bulk_max_size`	Doris Stream Load batch size, default is 100000
`max_retries`	Number of retries for Doris Stream Load requests on failure, default is -1 for infinite retries to ensure data reliability
`log_request`	Whether to output Doris Stream Load request and response metadata in logs for troubleshooting, default is true
`log_progress_interval`	Time interval for outputting speed in logs, unit is seconds, default is 10, setting to 0 can disable this type of logging

Usage Example

TEXT Log Collection Example

This example demonstrates TEXT log collection using Doris FE logs as an example.

1. Data

FE log files are typically located at the fe/log/fe.log file under the Doris installation directory. They are typical Java program logs, including fields such as timestamp, log level, thread name, code location, and log content. Not only do they contain normal logs, but also exception logs with stacktraces, which are multiline. Log collection and storage need to combine the main log and stacktrace into a single log entry.

```
2024-07-08 21:18:01,432 INFO (Statistics Job Appender|61) [StatisticsJobAppender.runAfterCatalogReady():70] Stats table not
available, skip
2024-07-08 21:18:53,710 WARN (STATS_FETCH-0|208) [StmtExecutor.executeInternalQuery():3332] Failed to run internal SQL:
OriginStatement{originStmt='SELECT * FROM __internal_schema.column_statistics WHERE part_id is NULL  ORDER BY update_time DESC LIMIT
500000', idx=0}
org.apache.doris.common.UserException: errCode = 2, detailMessage = tablet 10031 has no queryable replicas. err: replica 10032's
backend 10008 does not exist or not alive
        at org.apache.doris.planner.OlapScanNode.addScanRangeLocations(OlapScanNode.java:931) ~[doris-fe.jar:1.2-SNAPSHOT]
        at org.apache.doris.planner.OlapScanNode.computeTabletInfo(OlapScanNode.java:1197) ~[doris-fe.jar:1.2-SNAPSHOT]
```

2. Table Creation

The table structure includes fields such as the log's creation time, collection time, hostname, log file path, log type, log level, thread name, code location, and log content.

```
CREATE TABLE `doris_log` (
  `log_time` datetime NULL COMMENT 'log content time',
  `collect_time` datetime NULL COMMENT 'log agent collect time',
  `host` text NULL COMMENT 'hostname or ip',
  `path` text NULL COMMENT 'log file path',
  `type` text NULL COMMENT 'log type',
  `level` text NULL COMMENT 'log level',
  `thread` text NULL COMMENT 'log thread',
  `position` text NULL COMMENT 'log code position',
  `message` text NULL COMMENT 'log message',
  INDEX idx_host (`host`) USING INVERTED COMMENT '',
  INDEX idx_path (`path`) USING INVERTED COMMENT '',
  INDEX idx_type (`type`) USING INVERTED COMMENT '',
  INDEX idx_level (`level`) USING INVERTED COMMENT '',
  INDEX idx_thread (`thread`) USING INVERTED COMMENT '',
  INDEX idx_position (`position`) USING INVERTED COMMENT '',
  INDEX idx_message (`message`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true") COMMENT ''
) ENGINE=OLAP
DUPLICATE KEY(`log_time`)
COMMENT 'OLAP'
PARTITION BY RANGE(`log_time`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"dynamic_partition.enable" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-7",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.create_history_partition" = "true",
"compaction_policy" = "time_series"
);
```

3. Configuration

The filebeat log collection configuration file, such as filebeat_doris_log.yml, is in YAML format and mainly consists of four parts corresponding to the various stages of ETL: 1. Input is responsible for reading the raw data. 2. Processor is responsible for data transformation. 3. queue.mem configures the internal buffer queue of filebeat. 4. Output is responsible for sending the data to the output destination.

```
# 1. input is responsible for reading raw data
# type: log is a log input plugin that can be configured to read the path of the log file. It uses the multiline feature to
concatenate lines that do not start with a timestamp to the end of the previous line, achieving the effect of merging stacktraces
with the main log. The log input saves the log content in the message field, and there are also some metadata fields such as
agent.host, log.file.path.

filebeat.inputs:
- type: log
   enabled: true
   paths:
     - /path/to/your/log
   # multiline can concatenate multi-line logs (e.g., Java stacktraces)
   multiline:
     type: pattern
```

```
    # Effect: Lines starting with yyyy-mm-dd HH:MM:SS are considered as a new log, others are concatenated to the previous log
    pattern: '^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}'
    negate: true
    match: after
    skip_newline: true

# 2. processors section is responsible for data transformation
processors:
# Use the js script plugin to replace \t in logs with spaces to avoid JSON parsing errors
- script:
    lang: javascript
    source: >
        function process(event) {
            var msg = event.Get("message");
            msg = msg.replace(/\t/g, "   ");
            event.Put("message", msg);
        }
# Use the dissect plugin for simple log parsing
- dissect:
    # Example log: 2024-06-08 18:26:25,481 INFO (report-thread|199) [ReportHandler.cpuReport():617] begin to handle
    tokenizer: "%{day} %{time} %{log_level} (%{thread}) [%{position}] %{content}"
    target_prefix: ""
    ignore_failure: true
    overwrite_keys: true

# 3. internal buffer Queue total count, flush batch size, flush interval
queue.mem:
   events: 1000000
   flush.min_events: 100000
   flush.timeout: 10s

# 4. output section is responsible for data output
# The doris output sends data to Doris using the Stream Load HTTP interface. The data format for Stream Load is specified as JSON
through the headers parameter, and the codec_format_string parameter formats the output to Doris in a printf-like manner. For
example, the following example formats a JSON based on filebeat internal fields such as agent.hostname, and fields produced by
processors like dissect, such as day, using %{[a][b]} to reference them. Stream Load will automatically write the JSON fields into
the corresponding fields of the Doris table.

output.doris:
   fenodes: [ "http://fehost1:http_port", "http://fehost2:http_port", "http://fehost3:http_port" ]
   user: "your_username"
   password: "your_password"
   database: "your_db"
   table: "your_table"
   # Output string format
   ## %{[agent][hostname]} %{[log][file][path]} are filebeat自带的metadata
   ## Common filebeat metadata also includes采集时间戳 %{[@timestamp]}
   ## %{[day]} %{[time]} are fields obtained from the above dissect parsing
   codec_format_string: '{"ts": "%{[day]} %{[time]}", "host": "%{[agent][hostname]}", "path": "%{[log][file][path]}", "message": "%
{[message]}" }'
   headers:
     format: "json"
     read_json_by_line: "true"
     load_to_single_tablet: "true"
```

4. Running filebeat

```
./filebeat-doris -f config/filebeat_doris_log.yml

# When log_request is set to true, the log will output the request parameters and response results of each Stream Load.

doris stream load response:
```

```
{
    "TxnId": 45464,
    "Label": "beats_log_db_doris_log_20240708_223532_539_6c20a0d1-dcab-4b8e-9bc0-76b46a929bd1",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 452,
    "NumberLoadedRows": 452,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 277230,
    "LoadTimeMs": 1797,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 18,
    "ReadDataTimeMs": 9,
    "WriteDataTimeMs": 1758,
    "CommitAndPublishTimeMs": 18
}

# By default, speed information is logged every 10 seconds, including the amount of data since startup (in MB and ROWS), the total
speed (in MB/s and R/S), and the speed in the last 10 seconds.

total 11 MB 18978 ROWS, total speed 0 MB/s 632 R/s, last 10 seconds speed 1 MB/s 1897 R/s
```

JSON Log Collection Example

This example demonstrates JSON log collection using data from the GitHub events archive.

1. Data

The GitHub events archive contains archived data of GitHub user actions, formatted as JSON. It can be downloaded from here, for example, the data for January 1, 2024, at 3 PM.

```
wget https://data.gharchive.org/2024-01-01-15.json.gz
```

Below is a sample of the data. Normally, each piece of data is on a single line, but for ease of display, it has been formatted here.

```
{
  "id": "37066529221",
  "type": "PushEvent",
  "actor": {
    "id": 46139131,
    "login": "Bard89",
    "display_login": "Bard89",
    "gravatar_id": "",
    "url": "https://api.github.com/users/Bard89",
    "avatar_url": "https://avatars.githubusercontent.com/u/46139131?"
  },
  "repo": {
    "id": 780125623,
    "name": "Bard89/talk-to-me",
    "url": "https://api.github.com/repos/Bard89/talk-to-me"
  },
  "payload": {
    "repository_id": 780125623,
    "push_id": 17799451992,
    "size": 1,
```

```
    "distinct_size": 1,
    "ref": "refs/heads/add_mvcs",
    "head": "f03baa2de66f88f5f1754ce3fa30972667f87e81",
    "before": "85e6544ede4ae3f132fe2f5f1ce0ce35a3169d21"
  },
  "public": true,
  "created_at": "2024-04-01T23:00:00Z"
}
```

2. Table Creation

```
CREATE DATABASE log_db;
USE log_db;


CREATE TABLE github_events
(
  `created_at` DATETIME,
  `id` BIGINT,
  `type` TEXT,
  `public` BOOLEAN,
  `actor` VARIANT,
  `repo` VARIANT,
  `payload` TEXT,
  INDEX `idx_id` (`id`) USING INVERTED,
  INDEX `idx_type` (`type`) USING INVERTED,
  INDEX `idx_actor` (`actor`) USING INVERTED,
  INDEX `idx_host` (`repo`) USING INVERTED,
  INDEX `idx_payload` (`payload`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true")
)
ENGINE = OLAP
DUPLICATE KEY(`created_at`)
PARTITION BY RANGE(`created_at`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"compaction_policy" = "time_series",
"enable_single_replica_compaction" = "true",
"dynamic_partition.enable" = "true",
"dynamic_partition.create_history_partition" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-30",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.replication_num" = "1"
);
```

3. Filebeat Configuration

This configuration file differs from the previous TEXT log collection in the following aspects:

1. Processors are not used because no additional processing or transformation is needed.
2. The codec_format_string in the output is simple, directly outputting the entire message, which is the raw content.

```
# input
filebeat.inputs:
- type: log
  enabled: true
```

```
  paths:
    - /path/to/your/log

# queue and batch
queue.mem:
  events: 1000000
  flush.min_events: 100000
  flush.timeout: 10s

# output
output.doris:
  fenodes: [ "http://fehost1:http_port", "http://fehost2:http_port", "http://fehost3:http_port" ]
  user: "your_username"
  password: "your_password"
  database: "your_db"
  table: "your_table"
  # output string format
  ## Directly outputting the raw message of each line from the original file. Since headers specify format: "json", Stream Load will
automatically parse the JSON fields and write them into the corresponding fields of the Doris table.
  codec_format_string: '%{[message]}'
  headers:
    format: "json"
    read_json_by_line: "true"
    load_to_single_tablet: "true"
```

4. Running Filebeat

```
./filebeat-doris -f config/filebeat_github_events.yml
```

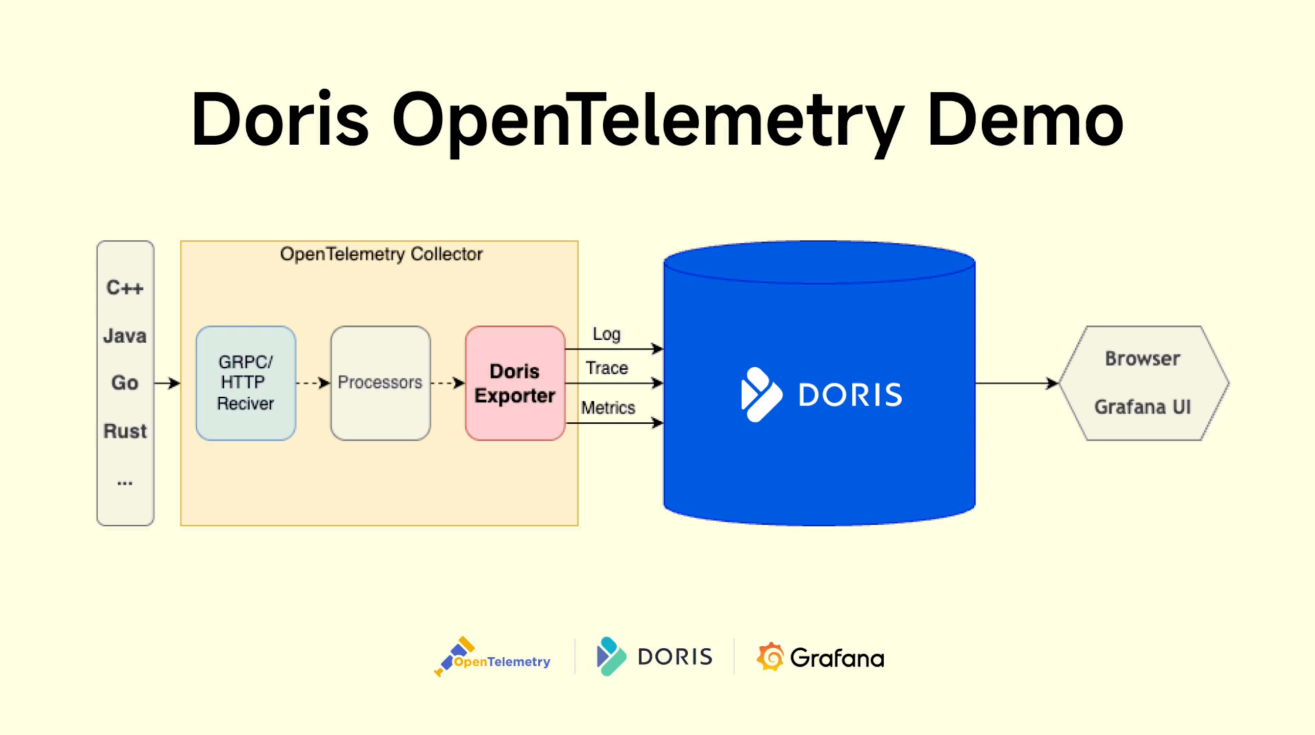
OpenTelemetry

Doris OpenTelemetry Integration

Introduction

OpenTelemetry, also known as OTel, is a vendor-neutral open source Observability framework for instrumenting, generating, collecting, and exporting telemetry data such as traces, metrics, and logs. OpenTelemetry defines a set of standards and protocols for observability, widely adopted by the observability community and vendors, gradually becoming the de facto standard in the field of observability.

A major goal of OpenTelemetry is to enable easy instrumentation of your applications and systems, regardless of the programming language, infrastructure, and runtime environments used. The backend (storage) and the frontend (visualization) of telemetry data are intentionally left to other tools. Doris, as a storage backend integrated with OpenTelemetry, provides high-performance, low-cost, unified observability data storage and analysis capabilities. The overall architecture is as follows:



Installation

Download the OpenTelemetry Collector Contrib installation package from the OpenTelemetry official Release page, for example: https://github.com/open-telemetry/opentelemetry-collector-releases/releases/download/v0.132.2/otelcol-contrib_0.132.2_linux_amd64.tar.gz

Extract the package to obtain the `otelcol-contrib` executable file.

Parameter Configuration

The core configuration of the OpenTelemetry Collector Doris Exporter is as follows:

Configuration	Description
`endpoint`	Doris FE HTTP address in the format `host:port`, e.g., `"127.0.0.1:8030"`
`mysql_endpoint`	Doris FE MySQL address in the format `host:port`, e.g., `"127.0.0.1:9030"`
`username`	Doris username with write permissions for the corresponding database tables
`password`	Password for the Doris user
`database`	Target Doris database name
`table.logs`	Doris table name for logs data (default: `otel_logs`)
`table.traces`	Doris table name for traces data (default: `otel_traces`)
`table.metrics`	Doris table name for metrics data (default: `otel_metrics`)
`create_schema`	Whether to automatically create Doris database tables (default: `true`)
`history_days`	Retention days for historical data in auto-created tables (default: `0`, meaning permanent retention)
`create_history_days`	Initial partition days for auto-created tables (default: `0`, meaning no partitions created)
`label_prefix`	Doris Stream Load Label prefix. The final label is `{label_prefix}_{db}_{table}_{yyyymmdd_hhmmss}_{uuid}` (default: `open_telemetry`)
`headers`	Headers for Doris Stream Load in YAML map format
`log_progress_interval`	Interval (in seconds) for logging throughput (default: `10`, set to `0` to disable)

For more configurations, refer to:
https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/exporter/dorisexporter

Usage Example

Usage Example

TEXT Log Collection Example

This example demonstrates TEXT log collection using Doris FE logs.

1. Data

FE log files are typically located in the `fe/log/fe.log` file under the Doris installation directory. These are standard Java program logs, including fields such as timestamps, log levels, thread names, code locations, and log content. In addition to normal logs, there are also exception logs with stack traces. Since stack traces span multiple lines, the log collection and storage process must combine the main log and stack trace into a single log entry.

Example logs:

```
2024-07-08 21:18:01,432 INFO (Statistics Job Appender|61) [StatisticsJobAppender.runAfterCatalogReady():70] Stats table not
available, skip
```

```
2024-07-08 21:18:53,710 WARN (STATS_FETCH-0|208) [StmtExecutor.executeInternalQuery():3332] Failed to run internal SQL:
OriginStatement{originStmt='SELECT * FROM __internal_schema.column_statistics WHERE part_id is NULL  ORDER BY update_time DESC LIMIT
500000', idx=0}
org.apache.doris.common.UserException: errCode = 2, detailMessage = tablet 10031 has no queryable replicas. err: replica 10032's
backend 10008 does not exist or not alive
        at org.apache.doris.planner.OlapScanNode.addScanRangeLocations(OlapScanNode.java:931) ~[doris-fe.jar:1.2-SNAPSHOT]
        at org.apache.doris.planner.OlapScanNode.computeTabletInfo(OlapScanNode.java:1197) ~[doris-fe.jar:1.2-SNAPSHOT]
```

2. OpenTelemetry Configuration

The log collection configuration file (e.g., `opentelemetry_java_log.yml`) consists of three main parts, corresponding to the ETL process:
1. **Receivers** – Read raw data. 2. **Processors** – Transform data. 3. **Exporters** – Output data.

Example configuration (`opentelemetry_java_log.yml`):

```
# 1. Receivers – Read raw data
# The `filelog` receiver reads log files from the local filesystem.
# The `multiline` configuration merges stack traces with the main log by matching timestamps.
receivers:
  filelog:
    include:
      - /path/to/fe.log
    start_at: beginning
    multiline:
      line_start_pattern: '^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}'  # Match timestamp as new log start
    operators:
      # Parse logs
      - type: regex_parser
        regex: '^(?P<time>\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}) (?P<severity>INFO|WARN|ERROR) (?P<message>.*)'
        timestamp:
          parse_from: attributes.time
          layout: '%Y-%m-%d %H:%M:%S,%f'
        severity:
          parse_from: attributes.severity
          trace: TRACE
          debug: DEBUG
          info: INFO
          warn: WARN
          error: ERROR
          fatal: FATAL

# 2. Processors – Transform data
# The `batch` processor groups logs before sending.
processors:
  batch:
    send_batch_size: 100000  # Number of logs per batch (recommended batch size: 100MB–1GB)
    timeout: 10s

# 3. Exporters – Output data
# The `doris` exporter sends data to Doris using the Stream Load HTTP interface (JSON format by default).
exporters:
  doris:
    endpoint: http://localhost:8030  # FE HTTP address
    mysql_endpoint: localhost:9030   # FE MySQL address
    database: doris_db_name
    username: doris_username
    password: doris_password
    table:
      logs: otel_logs
    create_schema: true  # Automatically create schema (if false, tables must be created manually)
    history_days: 10
```

```
    create_history_days: 10
    timezone: Asia/Shanghai
    timeout: 60s  # HTTP Stream Load client timeout
    log_response: true
    sending_queue:
      enabled: true
      num_consumers: 20
      queue_size: 1000
    retry_on_failure:
      enabled: true
      initial_interval: 5s
      max_interval: 30s
    headers:
      load_to_single_tablet: "true"

service:
  pipelines:
    logs:
      receivers: [filelog]
      processors: [batch]
      exporters: [doris]
```

3. Running OpenTelemetry

```
./otelcol-contrib --config config/opentelemetry_java_log.yml

# When `log_response` is true, logs will display Stream Load request/response details:
2025-08-18T00:33:22.543+0800    info    dorisexporter@v0.132.0/exporter_logs.go:181 log response:
{
    "TxnId": 52,
    "Label": "open_telemetry_otel_otel_logs_20250818003321_498bb8ec-040c-4982-9eb4-452b15129782",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 50355,
    "NumberLoadedRows": 50355,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 31130235,
    "LoadTimeMs": 680,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 3,
    "ReadDataTimeMs": 106,
    "WriteDataTimeMs": 653,
    "ReceiveDataTimeMs": 11,
    "CommitAndPublishTimeMs": 23
}

# Progress logs (every 10s) show throughput:
2025-08-18T00:05:00.017+0800    info    dorisexporter@v0.132.0/progress_reporter.go:63  [LOG] total 11 MB 18978 ROWS, total speed 0
MB/s 632 R/s, last 10 seconds speed 1 MB/s 1897 R/s
```

JSON Log Collection Example

This example demonstrates JSON log collection using GitHub Events Archive data.

1. Data

GitHub Events Archive contains archived user activity events in JSON format, downloadable from https://www.gharchive.org/. For example, to download data from January 1, 2024, at 15:00:

```
wget https://data.gharchive.org/2024-01-01-15.json.gz
```

Below is a sample entry (formatted for readability; actual data is one JSON object per line):

```
{
  "id": "37066529221",
  "type": "PushEvent",
  "actor": {
    "id": 46139131,
    "login": "Bard89",
    "display_login": "Bard89",
    "gravatar_id": "",
    "url": "https://api.github.com/users/Bard89",
    "avatar_url": "https://avatars.githubusercontent.com/u/46139131?"
  },
  "repo": {
    "id": 780125623,
    "name": "Bard89/talk-to-me",
    "url": "https://api.github.com/repos/Bard89/talk-to-me"
  },
  "payload": {
    "repository_id": 780125623,
    "push_id": 17799451992,
    "size": 1,
    "distinct_size": 1,
    "ref": "refs/heads/add_mvcs",
    "head": "f03baa2de66f88f5f1754ce3fa30972667f87e81",
    "before": "85e6544ede4ae3f132fe2f5f1ce0ce35a3169d21"
  },
  "public": true,
  "created_at": "2024-04-01T23:00:00Z"
}
```

2. OpenTelemetry Configuration

The key difference from the TEXT log configuration is the `json_parser` operator in the `filelog` receiver, which parses each line as JSON. Extracted fields are used for subsequent processing.

Example configuration (`opentelemetry_json_log.yml`):

```
receivers:
  filelog:
    include:
      - /path/to/2024-01-01-15.json
    start_at: beginning
    operators:
      - type: json_parser
        timestamp:
          parse_from: attributes.created_at
          layout: '%Y-%m-%dT%H:%M:%SZ'

processors:
  batch:
    send_batch_size: 100000  # Logs per batch (recommended: 100MB-1GB total size)
    timeout: 10s
```

```
exporters:
  doris:
    endpoint: http://localhost:8030  # FE HTTP address
    mysql_endpoint: localhost:9030   # FE MySQL address
    database: doris_db_name
    username: doris_username
    password: doris_password
    table:
      logs: otel_logs
    create_schema: true  # Auto-create schema (if false, manually create tables)
    history_days: 10
    create_history_days: 10
    timezone: Asia/Shanghai
    timeout: 60s  # HTTP Stream Load client timeout
    log_response: true
    sending_queue:
      enabled: true
      num_consumers: 20
      queue_size: 1000
    retry_on_failure:
      enabled: true
      initial_interval: 5s
      max_interval: 30s
    headers:
      load_to_single_tablet: "true"

service:
  pipelines:
    logs:
      receivers: [filelog]
      processors: [batch]
      exporters: [doris]
```

3. Running OpenTelemetry

```
./otelcol-contrib --config config/opentelemetry_json_log.yml
```

Trace Collection Example

1. OpenTelemetry Configuration

Create the configuration file `otel_trace.yml` as follows:

```
receivers:
  otlp:  # OTLP protocol to receive data from OpenTelemetry Java Agent
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

processors:
  batch:
    send_batch_size: 100000  # Number of traces per batch (recommended batch size: 100MB–1GB)
    timeout: 10s

exporters:
  doris:
    endpoint: http://localhost:8030  # FE HTTP address
    database: doris_db_name
```

```
    username: doris_username
    password: doris_password
    table:
      traces: doris_table_name
    create_schema: true  # Auto-create schema (if false, tables must be created manually)
    mysql_endpoint: localhost:9030  # FE MySQL address
    history_days: 10
    create_history_days: 10
    timezone: Asia/Shanghai
    timeout: 60s  # HTTP Stream Load client timeout
    log_response: true
    sending_queue:
      enabled: true
      num_consumers: 20
      queue_size: 1000
    retry_on_failure:
      enabled: true
      initial_interval: 5s
      max_interval: 30s
    headers:
      load_to_single_tablet: "true"

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [doris]
```

2. Run OpenTelemetry

```
./otelcol-contrib --config otel_trace.yaml
```

3. Application Integration with OpenTelemetry SDK

Here we demonstrate integration using a Spring Boot example application (from the official demo), which returns “Hello World!” for the “/” endpoint.

1. Download the OpenTelemetry Java Agent.
    - Advantage: No code changes required for existing applications.
    - For other languages/integration methods, see OpenTelemetry docs:
        - Language APIs & SDKs
        - Zero-code Instrumentation
2. Before starting the application, set these environment variables (no code modification needed):

```
export JAVA_TOOL_OPTIONS="${JAVA_TOOL_OPTIONS} -javaagent:/your/path/to/opentelemetry-javaagent.jar"  # Path to OpenTelemetry Java
Agent
export OTEL_JAVAAGENT_LOGGING="none"  # Disable OTEL logs to avoid interference with application logs
export OTEL_SERVICE_NAME="myproject"
export OTEL_TRACES_EXPORTER="otlp"  # Use OTLP protocol to send trace data
export OTEL_EXPORTER_OTLP_ENDPOINT="http://localhost:4317"  # OpenTelemetry Collector address

java -jar myproject-0.0.1-SNAPSHOT.jar
```

FluentBit

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

Fluent Bit is a fast log processor and forwarder that supports custom output plugins to write data into storage systems, with the Fluent Bit Doris output plugin being the one for outputting to Doris.

By invoking the Doris Stream Load HTTP interface, the Fluent Bit Doris output plugin writes data into Doris in real-time, offering capabilities such as multi-threaded concurrency, failure retries, custom Stream Load formats and parameters, and output write speed.

To use the Fluent Bit Doris output plugin, there are three main steps: 1. Download or compile the Fluent Bit binary program that includes the Doris output plugin. 2. Configure the Fluent Bit output address and other parameters. 3. Start Fluent Bit to write data into Doris in real-time.

Installation (alpha)

Download

https://assets-cdn.selectdb.com/downloads/releases/integrations/fluent-bit-doris-3.1.9

Compile from Source Code

Clone the dev branch of https://github.com/joker-star-l/fluent-bit and run the following commands in the build/ directory

```
cmake -DFLB_RELEASE=ON ..
make
```

The build output is build/bin/fluent-bit.

Configuration

The configuration for the Fluent Bit Doris output plugin is as follows:

Configuration	Description
`host`	Stream Load HTTP host
`port`	Stream Load HTTP port
`user`	Doris username, this user needs to have import permissions for the corresponding Doris database and table

Configuration	Description
password	Doris user's password
database	The Doris database name to write into
table	The Doris table name to write into
label_prefix	Doris Stream Load Label prefix，the final generated Label is *{label_prefix}_{timestamp}_{uuid}*，the default value is fluentbit. If set to false, no Label will be added
time_key	The name of the timestamp column to add to the data. The default value is date. If set to false, the column will not be added
header	Doris Stream Load headers parameter, can be set more than one
log_request	Whether to output Doris Stream Load request and response metadata in logs for troubleshooting, default is true
log_progress_interval	Time interval for outputting speed in logs, unit is seconds, default is 10, setting to 0 can disable this type of logging
retry_limit	Doris Stream Load request failure retry number, default value is 1, if set to false will not limit the number of retries
workers	Number of workers to perform Doris Stream Load, default value is 2

Usage Example

TEXT Log Collection Example

This example demonstrates TEXT log collection using Doris FE logs as an example.

1. Data

FE log files are typically located at the fe/log/fe.log file under the Doris installation directory. They are typical Java program logs, including fields such as timestamp, log level, thread name, code location, and log content. Not only do they contain normal logs, but also exception logs with stacktraces, which are multiline. Log collection and storage need to combine the main log and stacktrace into a single log entry.

```
2024-07-08 21:18:01,432 INFO (Statistics Job Appender|61) [StatisticsJobAppender.runAfterCatalogReady():70] Stats table not
available, skip
2024-07-08 21:18:53,710 WARN (STATS_FETCH-0|208) [StmtExecutor.executeInternalQuery():3332] Failed to run internal SQL:
OriginStatement{originStmt='SELECT * FROM __internal_schema.column_statistics WHERE part_id is NULL  ORDER BY update_time DESC LIMIT
500000', idx=0}
org.apache.doris.common.UserException: errCode = 2, detailMessage = tablet 10031 has no queryable replicas. err: replica 10032's
backend 10008 does not exist or not alive
        at org.apache.doris.planner.OlapScanNode.addScanRangeLocations(OlapScanNode.java:931) ~[doris-fe.jar:1.2-SNAPSHOT]
        at org.apache.doris.planner.OlapScanNode.computeTabletInfo(OlapScanNode.java:1197) ~[doris-fe.jar:1.2-SNAPSHOT]
```

2. Table Creation

The table structure includes fields such as the log's creation time, collection time, hostname, log file path, log type, log level, thread name, code location, and log content.

```
CREATE TABLE `doris_log` (
  `log_time` datetime NULL COMMENT 'log content time',
  `collect_time` datetime NULL COMMENT 'log agent collect time',
  `host` text NULL COMMENT 'hostname or ip',
  `path` text NULL COMMENT 'log file path',
  `type` text NULL COMMENT 'log type',
  `level` text NULL COMMENT 'log level',
  `thread` text NULL COMMENT 'log thread',
  `position` text NULL COMMENT 'log code position',
  `message` text NULL COMMENT 'log message',
  INDEX idx_host (`host`) USING INVERTED COMMENT '',
  INDEX idx_path (`path`) USING INVERTED COMMENT '',
  INDEX idx_type (`type`) USING INVERTED COMMENT '',
  INDEX idx_level (`level`) USING INVERTED COMMENT '',
  INDEX idx_thread (`thread`) USING INVERTED COMMENT '',
  INDEX idx_position (`position`) USING INVERTED COMMENT '',
  INDEX idx_message (`message`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true") COMMENT ''
) ENGINE=OLAP
DUPLICATE KEY(`log_time`)
COMMENT 'OLAP'
PARTITION BY RANGE(`log_time`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"dynamic_partition.enable" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-7",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.create_history_partition" = "true",
"compaction_policy" = "time_series"
);
```

3. Configuration

The configuration file of Fluent Bit log collection is as follows, doris_log.conf is used to define various parts of ETL components, and parsers.conf is used to define different log parsers.

doris_log.conf:

```
# config for Fluent Bit service
[SERVICE]
    log_level info
    # parsers file
    parsers_file parsers.conf

# use input tail
[INPUT]
    name tail
    path /path/to/your/log
    # add log file name to the record, key is 'path'
    path_key path
    # set multiline parser
    multiline.parser multiline_java

# parse log
[FILTER]
    match *
    name parser
    key_name log
    parser fe_log
```

```
    reserve_data true

# add host info
[FILTER]
    name sysinfo
    match *
    # add hostname to the record, key is 'host'
    hostname_key host

# output to doris
[OUTPUT]
    name doris
    match *
    host fehost
    port feport
    user your_username
    password your_password
    database your_db
    table your_table
    # add 'collect_time' to the record
    time_key collect_time
    # 'collect_time' is timestamp, change it to datatime
    header columns collect_time=from_unixtime(collect_time)
    log_request true
    log_progress_interval 10
```

parsers.conf:

```
[MULTILINE_PARSER]
    name          multiline_java
    type          regex
    flush_timeout 1000
    # Regex rules for multiline parsing
    # ---------------------------------
    #
    # configuration hints:
    #
    #  - first state always has the name: start_state
    #  - every field in the rule must be inside double quotes
    #
    # rules   |   state name   | regex pattern | next state name
    # --------|----------------|---------------|-----------------
    rule         "start_state"   "/(^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2})(.*)/"  "cont"
    rule         "cont"          "/(^(?![0-9]{4}-[0-9]{2}-[0-9]{2}))(.*)/"     "cont"


[PARSER]
    name        fe_log
    format      regex
    # parse and add 'log_time', 'level', 'thread', 'position', 'message' to the record
    regex       ^(?<log_time>[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2},[0-9]{3}) (?<level>[^ ]+) \((?<thread>[^\)]+)\) \
[(?<position>[^\]]+)\] (?<message>(\n|.)*)\n$
```

4. Running Fluent Bit

```
fluent-bit -c doris_log.conf

# log stream load response

[2024/10/31 18:39:55] [ info] [output:doris:doris.1] 127.0.0.1:8040, HTTP status=200
{
    "TxnId": 32155,
```

```
    "Label": "fluentbit_1730371195_91cca1aa-c15f-45d2-b503-fe7d2e839c2a",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 1,
    "NumberLoadedRows": 1,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 836,
    "LoadTimeMs": 298,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 3,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 268,
    "CommitAndPublishTimeMs": 25
}

# log speed info

[2024/10/31 18:40:13] [ info] [output:doris:doris.1] total 0 MB 2 ROWS, total speed 0 MB/s 0 R/s, last 10 seconds speed 0 MB/s 0 R/s
```

JSON Log Collection Example

This example demonstrates JSON log collection using data from the GitHub events archive.

1. Data

The GitHub events archive contains archived data of GitHub user actions, formatted as JSON. It can be downloaded from here, for example, the data for January 1, 2024, at 3 PM.

```
wget https://data.gharchive.org/2024-01-01-15.json.gz
```

Below is a sample of the data. Normally, each piece of data is on a single line, but for ease of display, it has been formatted here.

```
{
  "id": "37066529221",
  "type": "PushEvent",
  "actor": {
    "id": 46139131,
    "login": "Bard89",
    "display_login": "Bard89",
    "gravatar_id": "",
    "url": "https://api.github.com/users/Bard89",
    "avatar_url": "https://avatars.githubusercontent.com/u/46139131?"
  },
  "repo": {
    "id": 780125623,
    "name": "Bard89/talk-to-me",
    "url": "https://api.github.com/repos/Bard89/talk-to-me"
  },
  "payload": {
    "repository_id": 780125623,
    "push_id": 17799451992,
    "size": 1,
    "distinct_size": 1,
    "ref": "refs/heads/add_mvcs",
    "head": "f03baa2de66f88f5f1754ce3fa30972667f87e81",
```

```
    "before": "85e6544ede4ae3f132fe2f5f1ce0ce35a3169d21"
  },
  "public": true,
  "created_at": "2024-04-01T23:00:00Z"
}
```

2. Table Creation

```
CREATE DATABASE log_db;
USE log_db;


CREATE TABLE github_events
(
  `created_at` DATETIME,
  `id` BIGINT,
  `type` TEXT,
  `public` BOOLEAN,
  `actor` VARIANT,
  `repo` VARIANT,
  `payload` TEXT,
  INDEX `idx_id` (`id`) USING INVERTED,
  INDEX `idx_type` (`type`) USING INVERTED,
  INDEX `idx_actor` (`actor`) USING INVERTED,
  INDEX `idx_host` (`repo`) USING INVERTED,
  INDEX `idx_payload` (`payload`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true")
)
ENGINE = OLAP
DUPLICATE KEY(`created_at`)
PARTITION BY RANGE(`created_at`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"compaction_policy" = "time_series",
"enable_single_replica_compaction" = "true",
"dynamic_partition.enable" = "true",
"dynamic_partition.create_history_partition" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-30",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.replication_num" = "1"
);
```

3. Configuration

In contrast to the previous TEXT log collection, this configuration does not use FILTER because no additional processing transformations are required.

github_events.conf:

```
[SERVICE]
    log_level info
    parsers_file github_parsers.conf

[INPUT]
    name tail
    parser github
    path /path/to/your/log
```

```
[OUTPUT]
    name doris
    match *
    host fehost
    port feport
    user your_username
    password your_password
    database your_db
    table your_table
    time_key false
    log_request true
    log_progress_interval 10
```

github_parsers.conf:

```
[PARSER]
    name github
    format json
```

4. Running Fluent Bit

```
fluent-bit -c github_events.conf
```

LoongCollector (iLogtail) Doris Flusher

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

LoongCollector (iLogtail) Doris Flusher

Introduction

LoongCollector (iLogtail) is an open-source, high-performance log collection and processing framework originating from Alibaba Cloud. Before version 3.0, it was named Logtail/iLogtail. It supports custom output plugins to write data into storage systems, and the LoongCollector Doris Flusher is the plugin for outputting data to Doris.

The Doris Flusher calls the Doris Stream Load HTTP interface to write data into Doris in real-time, providing capabilities such as multi-threaded concurrency, failure retries, custom Stream Load formats and parameters, and output write speed monitoring.

There are three main steps to use the Doris Flusher: 1. Install LoongCollector 2. Configure the Doris output address and other parameters 3. Start LoongCollector to write data into Doris in real-time

Installation

Download from Official Website

```
wget https://assets-cdn.selectdb.com/downloads/releases/extension/loongcollector-linux-amd64.tar.gz
```

Compile from Source Code

```
# Clone the repository
git clone https://github.com/alibaba/loongcollector.git
cd loongcollector
git submodule update --init

# Build LoongCollector
make all
cd output
```

Configuration

The configuration for the LoongCollector Doris Flusher Plugin is as follows:

Configuration	Description
`Addresses`	Stream Load HTTP addresses, formatted as a string array with one or more elements, each element being host:port. For example: [“http://fe1:8030”, “http://fe2:8030”]
`Database`	The Doris database name to write into
`Table`	The Doris table name to write into
`Authentication.PlainText.Username`	Doris username, this user needs to have import permissions for the corresponding Doris database and table
`Authentication.PlainText.Password`	Doris user’s password
`LoadProperties`	Doris Stream Load header parameters, formatted as a map. For example: `LoadProperties: {"format": "json", "read_json_by_line": "true"}`
`LogProgressInterval`	Time interval for outputting speed in logs, unit is seconds, default is 10, setting to 0 can disable this type of logging
`GroupCommit`	Group commit mode, optional values are “sync”, “async”, or “off”, default is “off”
`Concurrency`	Number of goroutines for concurrent data sending, default is 1 (synchronous mode)
`QueueCapacity`	Task queue capacity in asynchronous mode, default is 1024
`Convert.Protocol`	Data conversion protocol, default is custom_single
`Convert.Encoding`	Data conversion encoding, default is json
`Convert.TagFieldsRename`	Rename one or more fields from tags
`Convert.ProtocolFieldsRename`	Rename protocol fields, protocol field options can only be: contents, tags, time

Usage Examples

TEXT Log Collection Example

This example demonstrates TEXT log collection using Doris FE logs as an example.

1. Data

FE log files are typically located at the fe/log/fe.log file under the Doris installation directory. They are typical Java program logs, including fields such as timestamp, log level, thread name, code location, and log content. Not only do they contain normal logs, but also exception logs with stacktraces, which are multiline. Log collection and storage need to combine the main log and stacktrace into a single log entry.

```
2024-07-08 21:18:01,432 INFO (Statistics Job Appender|61) [StatisticsJobAppender.runAfterCatalogReady():70] Stats table not
available, skip
2024-07-08 21:18:53,710 WARN (STATS_FETCH-0|208) [StmtExecutor.executeInternalQuery():3332] Failed to run internal SQL:
OriginStatement{originStmt='SELECT * FROM __internal_schema.column_statistics WHERE part_id is NULL  ORDER BY update_time DESC LIMIT
500000', idx=0}
org.apache.doris.common.UserException: errCode = 2, detailMessage = tablet 10031 has no queryable replicas. err: replica 10032's
backend 10008 does not exist or not alive
```

```
        at org.apache.doris.planner.OlapScanNode.addScanRangeLocations(OlapScanNode.java:931) ~[doris-fe.jar:1.2-SNAPSHOT]
        at org.apache.doris.planner.OlapScanNode.computeTabletInfo(OlapScanNode.java:1197) ~[doris-fe.jar:1.2-SNAPSHOT]
```

2. Table Creation

The table structure includes fields such as the log's creation time, collection time, hostname, log file path, log type, log level, thread name, code location, and log content.

```
CREATE TABLE `doris_log` (
  `log_time` datetime NULL COMMENT 'log content time',
  `collect_time` datetime NULL COMMENT 'log agent collect time',
  `host` text NULL COMMENT 'hostname or ip',
  `path` text NULL COMMENT 'log file path',
  `type` text NULL COMMENT 'log type',
  `level` text NULL COMMENT 'log level',
  `thread` text NULL COMMENT 'log thread',
  `position` text NULL COMMENT 'log code position',
  `message` text NULL COMMENT 'log message',
  INDEX idx_host (`host`) USING INVERTED COMMENT '',
  INDEX idx_path (`path`) USING INVERTED COMMENT '',
  INDEX idx_type (`type`) USING INVERTED COMMENT '',
  INDEX idx_level (`level`) USING INVERTED COMMENT '',
  INDEX idx_thread (`thread`) USING INVERTED COMMENT '',
  INDEX idx_position (`position`) USING INVERTED COMMENT '',
  INDEX idx_message (`message`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true") COMMENT ''
) ENGINE=OLAP
DUPLICATE KEY(`log_time`)
COMMENT 'OLAP'
PARTITION BY RANGE(`log_time`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"dynamic_partition.enable" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-7",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.create_history_partition" = "true",
"compaction_policy" = "time_series"
);
```

3. LoongCollector Configuration

The LoongCollector configuration file consists of 3 main parts: 1. inputs - responsible for reading raw data 2. processors - responsible for data transformation 3. flushers - responsible for data output

Configuration file location: `conf/continuous_pipeline_config/local/` Create configuration file: `loongcollector_doris_log.yaml`

```
enable: true

inputs:
  # 1. inputs section is responsible for reading raw data
  # file_log input is an input plugin that can configure the log file path to read
  # Using multiline configuration to append lines not starting with timestamp to the previous line,
  # achieving the effect of merging stacktrace with the main log
  - Type: input_file
    FilePaths:
      - /path/fe.log
```

```
    Multiline:
      Mode: custom
      StartPattern: '^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}'

processors:
  # 2. processors section is responsible for data transformation
  # processor_regex is a commonly used data transformation plugin that extracts fields using regular expressions
  - Type: processor_regex
    SourceKey: content
    Regex: '(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}) ([A-Z]+) \(([^\)]*)\) \[([^\]]*)\] (.*)'
    Keys:
      - log_time
      - level
      - thread
      - position
      - message
  # Add extra fields
  - Type: processor_add_fields
    Fields:
      type: fe.log
    IgnoreIfExist: false

flushers:
  # 3. flushers section is responsible for data output
  # flusher_doris outputs data to Doris using the Stream Load HTTP interface
  # The LoadProperties parameter specifies the Stream Load data format as JSON
  - Type: flusher_doris
    Addresses:
      - "http://fe_ip:http_port"
    Database: log_db
    Table: doris_log
    Authentication:
      PlainText:
        Username: root
        Password: ""
    LoadProperties:
      format: json
      read_json_by_line: "true"
      load_to_single_tablet: "true"
      columns:
"log_time,collect_time,host,path,type,level,thread,position,message,log_time=replace(log_time,',','.'),collect_time=from_unixtime(co
llect_time)"
    Convert:
      Protocol: custom_single_flatten
      Encoding: json
      TagFieldsRename:
        host.ip: host
        log.file.path: path
      ProtocolFieldsRename:
        time: collect_time
    LogProgressInterval: 10
```

4. Running LoongCollector

```
nohup ./loongcollector > stdout.log 2> stderr.log &

# By default, speed information is logged every 10 seconds, including data volume since startup (MB and ROWS), total speed (MB/s and
R/s), and speed for the last 10 seconds
total 11 MB 18978 ROWS, total speed 0 MB/s 632 R/s, last 10 seconds speed 1 MB/s 1897 R/s
```

JSON Log Collection Example

This example demonstrates JSON log collection using data from the GitHub events archive.

1. Data

The GitHub events archive contains archived data of GitHub user actions, formatted as JSON. It can be downloaded from https://www.gharchive.org/, for example, the data for January 1, 2024, at 3 PM.

```
wget https://data.gharchive.org/2024-01-01-15.json.gz
```

Below is a sample of the data. Normally, each piece of data is on a single line, but for ease of display, it has been formatted here.

```
{
  "id": "37066529221",
  "type": "PushEvent",
  "actor": {
    "id": 46139131,
    "login": "Bard89",
    "display_login": "Bard89",
    "gravatar_id": "",
    "url": "https://api.github.com/users/Bard89",
    "avatar_url": "https://avatars.githubusercontent.com/u/46139131?"
  },
  "repo": {
    "id": 780125623,
    "name": "Bard89/talk-to-me",
    "url": "https://api.github.com/repos/Bard89/talk-to-me"
  },
  "payload": {
    "repository_id": 780125623,
    "push_id": 17799451992,
    "size": 1,
    "distinct_size": 1,
    "ref": "refs/heads/add_mvcs",
    "head": "f03baa2de66f88f5f1754ce3fa30972667f87e81",
    "before": "85e6544ede4ae3f132fe2f5f1ce0ce35a3169d21"
  },
  "public": true,
  "created_at": "2024-04-01T23:00:00Z"
}
```

2. Table Creation

```
CREATE DATABASE log_db;
USE log_db;

CREATE TABLE github_events
(
  `created_at` DATETIME,
  `id` BIGINT,
  `type` TEXT,
  `public` BOOLEAN,
  `actor` VARIANT,
  `repo` VARIANT,
  `payload` TEXT,
  INDEX `idx_id` (`id`) USING INVERTED,
  INDEX `idx_type` (`type`) USING INVERTED,
```

```
  INDEX `idx_actor` (`actor`) USING INVERTED,
  INDEX `idx_host` (`repo`) USING INVERTED,
  INDEX `idx_payload` (`payload`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true")
)
ENGINE = OLAP
DUPLICATE KEY(`created_at`)
PARTITION BY RANGE(`created_at`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"inverted_index_storage_format"= "v2",
"compaction_policy" = "time_series",
"enable_single_replica_compaction" = "true",
"dynamic_partition.enable" = "true",
"dynamic_partition.create_history_partition" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-30",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.replication_num" = "1"
);
```

3. LoongCollector Configuration

This configuration differs from the previous TEXT log collection in the following ways:

1. input_file uses JSON mode for parsing, LoongCollector will parse each line of text as JSON format
2. No complex processor plugins are used because JSON data already has structured fields

Configuration file location: `conf/continuous_pipeline_config/local/` Create configuration file: `loongcollector_doris_log.yaml`

```
enable: true

inputs:
  # file_log input reads JSON format log files
  - Type: input_file
    FilePaths:
      - /path/2024-01-01-15.json

processors:
  # Parse content, only expand the first level (actor, repo remain as JSON strings for VARIANT type usage)
  - Type: processor_json
    SourceKey: content
    KeepSource: false
    ExpandDepth: 1
    ExpandConnector: ""

flushers:
  # flusher_doris outputs data to Doris
  - Type: flusher_doris
    Addresses:
      - "http://fe_ip:http_port"
    Database: log_db
    Table: github_events
    Authentication:
      PlainText:
        Username: root
        Password: ""
    LoadProperties:
      format: json
```

```
      read_json_by_line: "true"
      load_to_single_tablet: "true"
    Convert:
      Protocol: custom_single_flatten
      Encoding: json
    LogProgressInterval: 10
    Concurrency: 3
```

4. Running LoongCollector

```
nohup ./loongcollector > stdout.log 2> stderr.log &
```

Langfuse on Doris

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

Langfuse on Doris

About Langfuse

Langfuse is an open-source LLM engineering platform that provides comprehensive observability solutions for large language model applications. It offers the following core features:

- **Tracing**: Complete recording of LLM application call chains and execution flows
- **Evaluation**: Multi-dimensional model performance evaluation and quality analysis
- **Prompt Management**: Centralized management and version control of prompt templates
- **Metrics Monitoring**: Real-time monitoring of application performance, cost, and quality metrics

This document provides detailed instructions on how to deploy a Langfuse solution using Apache Doris as the analytics backend, fully leveraging Doris's powerful OLAP analytics capabilities to process large-scale LLM application data.

System Architecture

The Langfuse on Doris solution uses a microservices architecture with the following core components:

Component	Ports	Description
Langfuse Web	3000	Web interface and API service for user interaction and data ingestion
Langfuse Worker	3030	Async task processing for data processing and analytics tasks
PostgreSQL	5432	Transactional data storage for user configuration and metadata
Redis	6379	Cache layer and message queue for improved system response performance
MinIO	9000	Object storage service for raw events and multi-modal attachments
Doris Fe	9030 8030	Doris Frontend, part of the Doris architecture, responsible for receiving user requests, query parsing and planning, metadata management, and node management
Doris Be	8040 8050	Doris Backends, part of the Doris architecture, responsible for data storage and query plan execution. Data is split into shards and stored with multiple replicas in BE nodes.

When deploying Apache Doris, you can choose between integrated compute-storage architecture or disaggregated compute-storage architecture based on your hardware environment and business requirements.

For Langfuse deployment, Docker Doris is not recommended for production environments. The FE and BE components included in Docker are intended for users to quickly experience the Langfuse on Doris capabilities.

```
flowchart TB
    User["UI, API, SDKs"]
    subgraph vpc["VPC"]
        Web["Web Server<br/>(langfuse/langfuse)"]
        Worker["Async Worker<br/>(langfuse/worker)"]
        Postgres["Postgres - OLTP<br/>(Transactional Data)"]
        Cache["Redis/Valkey<br/>(Cache, Queue)"]
        Doris["Doris - OLAP<br/>(Observability Data)"]
        S3["S3 / Blob Storage<br/>(Raw events, multi-modal attachments)"]
    end
    LLM["LLM API/Gateway<br/>(optional)"]

    User --> Web
    Web --> S3
    Web --> Postgres
    Web --> Cache
    Web --> Doris
    Web -.->|"optional for playground"| LLM

    Cache --> Worker
    Worker --> Doris
    Worker --> Postgres
    Worker --> S3
    Worker -.->|"optional for evals"| LLM
```

Deployment Requirements

Software Environment

Component	Version	Description
Docker	20.0+	Container runtime environment
Docker Compose	2.0+	Container orchestration tool
Apache Doris	2.1.10+	Analytics database, requires separate deployment

Hardware Resources

Resource Type	Minimum	Recommended	Description
Memory	8GB	16GB+	Supports multi-service concurrent operation
Disk	50GB	100GB+	Storage for container data and logs
Network	1Gbps	10Gbps	Ensures data transfer performance

Prerequisites

1. **Doris Cluster Preparation**

- Ensure the Doris cluster is running properly with stable performance
- Verify that FE HTTP port (default 8030) and query port (default 9030) are network accessible
- Langfuse will automatically create the required database and table structures in Doris after startup

2. **Network Connectivity**
   - Deployment environment can access Docker Hub to pull images
   - Langfuse services can access the relevant ports of the Doris cluster
   - Clients can access the Langfuse Web service port

> **Deployment Recommendation**
>
> It is recommended to use Docker to deploy Langfuse service components (Web, Worker, Redis, PostgreSQL), but Doris is recommended to be deployed separately for better performance and stability. Please refer to the official documentation for detailed Doris deployment guide.

Configuration Parameters

Langfuse services require multiple environment variables to support the proper operation of each component:

Doris Analytics Backend Configuration

Parameter	Example Value	Description
`LANGFUSE_ANALYTICS_BACKEND`	`doris`	Specify Doris as the analytics backend
`DORIS_FE_HTTP_URL`	`http://localhost:8030`	Doris FE HTTP service address
`DORIS_FE_QUERY_PORT`	`9030`	Doris FE query port
`DORIS_DB`	`langfuse`	Doris database name
`DORIS_USER`	`root`	Doris username
`DORIS_PASSWORD`	`123456`	Doris password
`DORIS_MAX_OPEN_CONNECTIONS`	`100`	Maximum database connections
`DORIS_REQUEST_TIMEOUT_MS`	`300000`	Request timeout in milliseconds

Basic Service Configuration

Parameter	Example Value	Description
`DATABASE_URL`	`postgresql://postgres:postgres@langfuse-postgres:5432/postgres`	PostgreSQL database connection URL
`NEXTAUTH_SECRET`	`your-debug-secret-key-here-must-be-long-enough`	NextAuth authentication key for session encryption
`SALT`	`your-super-secret-salt-with-at-least-32-characters-for-encryption`	Data encryption salt (at least 32 characters)

Parameter	Example Value	Description
ENCRYPTION_KEY	0000000000000000000000000000000000000000000000000000000000000000	Data encryption key (64 characters)
NEXTAUTH_URL	http://localhost:3000	Langfuse Web service address
TZ	UTC	System timezone setting

Redis Cache Configuration

Parameter	Example Value	Description
REDIS_HOST	langfuse-redis	Redis service host address
REDIS_PORT	6379	Redis service port
REDIS_AUTH	myredissecret	Redis authentication password
REDIS_TLS_ENABLED	false	Whether to enable TLS encryption
REDIS_TLS_CA	-	TLS CA certificate path
REDIS_TLS_CERT	-	TLS client certificate path
REDIS_TLS_KEY	-	TLS private key path

Data Migration Configuration

Parameter	Example Value	Description
LANGFUSE_ENABLE_BACKGROUND_MIGRATIONS	false	Disable background migrations (must be disabled when using Doris)
LANGFUSE_AUTO_DORIS_MIGRATION_DISABLED	false	Enable Doris auto migration

Docker Compose Deployment

Pre-deployment Preparation

Here we provide a compose example that can be started directly. Modify the configuration according to your requirements.

Download docker compose

```
wget https://assets-cdn.selectdb.com/downloads/releases/extension/docker-langfuse-doris.tar.gz
```

The directory structure for compose file and configuration file is as follows:

```
docker-langfuse-doris
├── docker-compose.yml
└── doris-config
    └── fe_custom.conf
```

Deployment Steps

1 . Start compose

```
docker compose up -d
```

```
# Check
$ docker compose up -d
[+] Running 9/9
 ✔ Network docker-langfuse-doris_doris_internal  Created
0.1s
 ✔ Network docker-langfuse-doris_default         Created
0.1s
 ✔ Container doris_fe                            Healthy
13.8s
 ✔ Container langfuse-postgres                   Healthy
13.8s
 ✔ Container langfuse-redis                      Healthy
13.8s
 ✔ Container langfuse-minio                      Healthy
13.8s
 ✔ Container doris_be                            Healthy
54.3s
 ✔ Container langfuse-worker                     Started
54.8s
 ✔ Container langfuse-web                        Started
```

3. Verify Deployment

Check service status:

When all service statuses show as Healthy, the compose has started successfully.

```
$ docker compose ps
NAME                IMAGE                               COMMAND                  SERVICE           CREATED         STATUS
PORTS
doris_be            apache/doris:be-2.1.11              "bash entry_point.sh"    doris_be          2 minutes ago   Up 2 minutes
(healthy)        0.0.0.0:8040->8040/tcp, :::8040->8040/tcp, 0.0.0.0:8060->8060/tcp, :::8060->8060/tcp, 0.0.0.0:9050->9050/tcp,
:::9050->9050/tcp, 0.0.0.0:9060->9060/tcp, :::9060->9060/tcp
doris_fe            apache/doris:fe-2.1.11              "bash init_fe.sh"        doris_fe          2 minutes ago   Up 2 minutes
(healthy)        0.0.0.0:8030->8030/tcp, :::8030->8030/tcp, 0.0.0.0:9010->9010/tcp, :::9010->9010/tcp, 0.0.0.0:9030->9030/tcp,
:::9030->9030/tcp
langfuse-minio      minio/minio                         "sh -c 'mkdir -p /da..."   minio             2 minutes ago   Up 2 minutes
(healthy)        0.0.0.0:19090->9000/tcp, :::19090->9000/tcp, 127.0.0.1:19091->9001/tcp
langfuse-postgres   postgres:latest                     "docker-entrypoint.s..."   postgres          2 minutes ago   Up 2 minutes
(healthy)        127.0.0.1:5432->5432/tcp
langfuse-redis      redis:7                             "docker-entrypoint.s..."   redis             2 minutes ago   Up 2 minutes
(healthy)        127.0.0.1:16379->6379/tcp
langfuse-web        selectdb/langfuse-web:latest        "dumb-init -- ./web/..."   langfuse-web      2 minutes ago   Up About a minute
(healthy)   0.0.0.0:13000->3000/tcp, :::13000->3000/tcp
```

```
langfuse-worker     selectdb/langfuse-worker:latest   "dumb-init -- ./work..."   langfuse-worker   2 minutes ago   Up About a minute
(healthy)   0.0.0.0:3030->3030/tcp, :::3030->3030/tcp
```

4. Service Initialization

After deployment is complete, access and initialize the service as follows:

Access Langfuse Web Interface: - URL: http://localhost:3000

Initialization Steps: 1. Open your browser and navigate to http://localhost:3000 2. Create an administrator account and log in 3. Create a new organization and project 4. Obtain the project's API Keys (Public Key and Secret Key) 5. Configure the authentication information required for SDK integration

Examples

Using Langfuse SDK

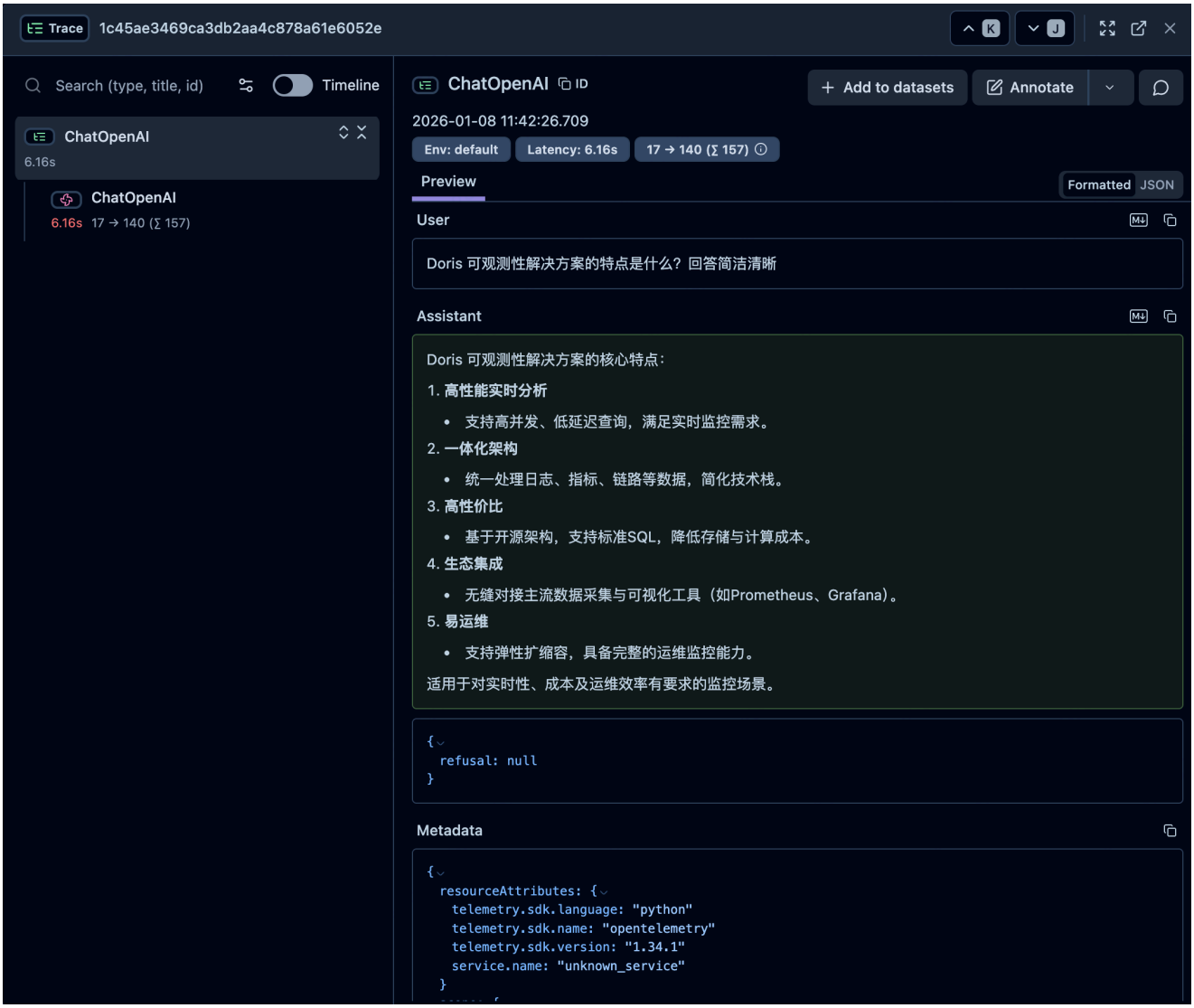
```
import os
# Instead of: import openai
from langfuse.openai import OpenAI
# from langfuse import observe

# Langfuse config
os.environ["LANGFUSE_SECRET_KEY"] = "sk-lf-******-******"
os.environ["LANGFUSE_PUBLIC_KEY"] = "pk-lf-******-******"
os.environ["LANGFUSE_HOST"] = "http://localhost:3000"


# use OpenAI client
client = OpenAI()


# ask a question
question = "What are the key features of the Doris observability solution? Please answer concisely."
print(f"question: {question}")

completion = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "user", "content": question}
    ]
)
response = completion.choices[0].message.content
print(f"response: {response}")
```



Using LangChain SDK

```
import os
from langfuse.langchain import CallbackHandler
from langchain_openai import ChatOpenAI

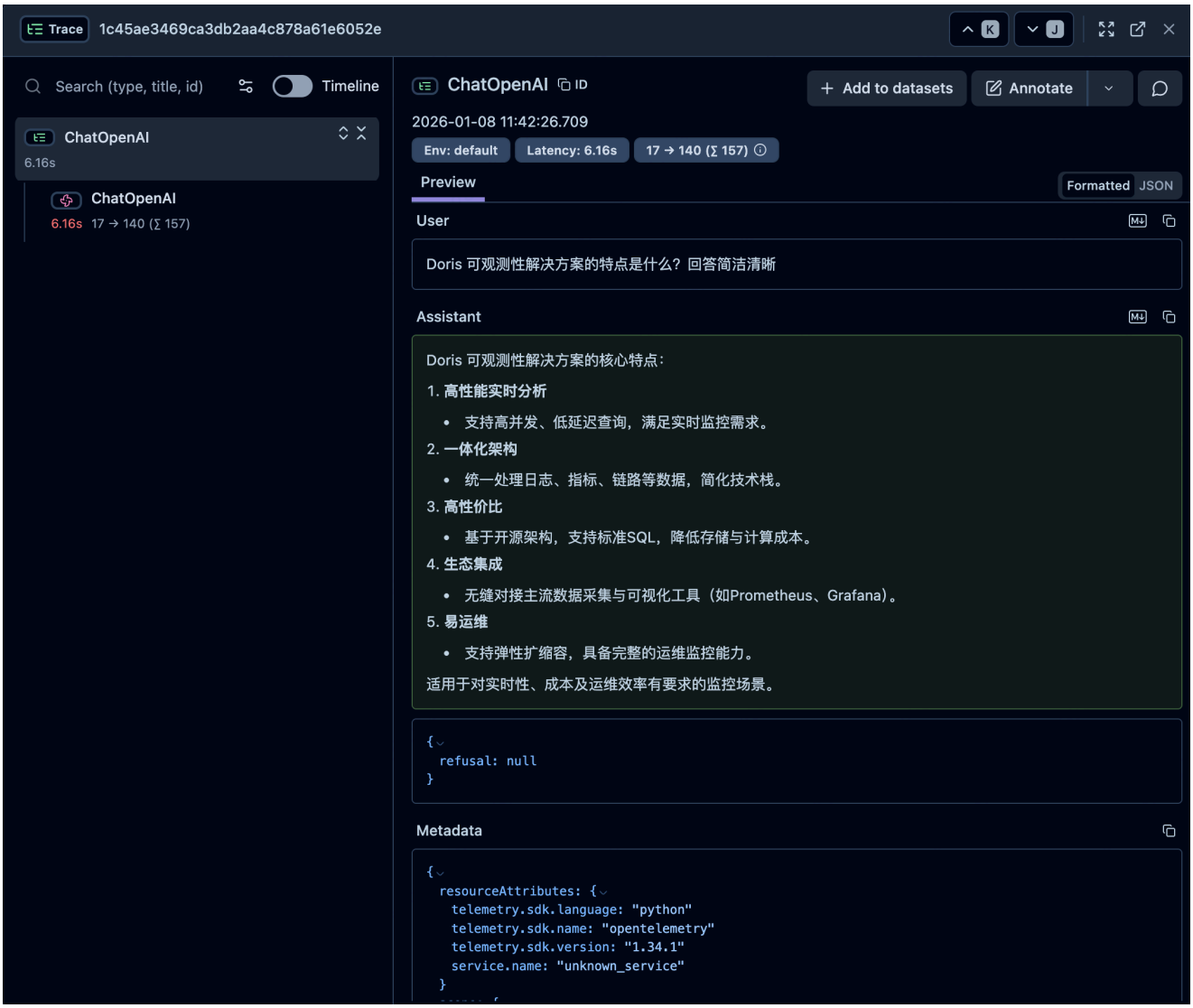
# Langfuse config
os.environ["LANGFUSE_SECRET_KEY"] = "sk-lf-******-******"
os.environ["LANGFUSE_PUBLIC_KEY"] = "pk-lf-******-******"
os.environ["LANGFUSE_HOST"] = "http://localhost:3000"

# Create your LangChain components (using OpenAI API)
llm = ChatOpenAI(
    model="gpt-4o"
)

# ask a question
question = "What are the key features of the Doris observability solution? Please answer concisely."
print(f"question: {question} \n")

# Run your chain with Langfuse tracing
try:
    # Initialize the Langfuse handler
    langfuse_handler = CallbackHandler()
    response = llm.invoke(question, config={"callbacks": [langfuse_handler]})
```

```
    print(f"response: {response.content}")
except Exception as e:
    print(f"Error during chain execution: {e}")
```



Using LlamaIndex SDK

```
import os
from langfuse import get_client
from openinference.instrumentation.llama_index import LlamaIndexInstrumentor
from llama_index.llms.openai import OpenAI

# Langfuse config
os.environ["LANGFUSE_SECRET_KEY"] = "sk-lf-******-*******"
os.environ["LANGFUSE_PUBLIC_KEY"] = "pk-lf-******-*******"
os.environ["LANGFUSE_HOST"] = "http://localhost:3000"

langfuse = get_client()

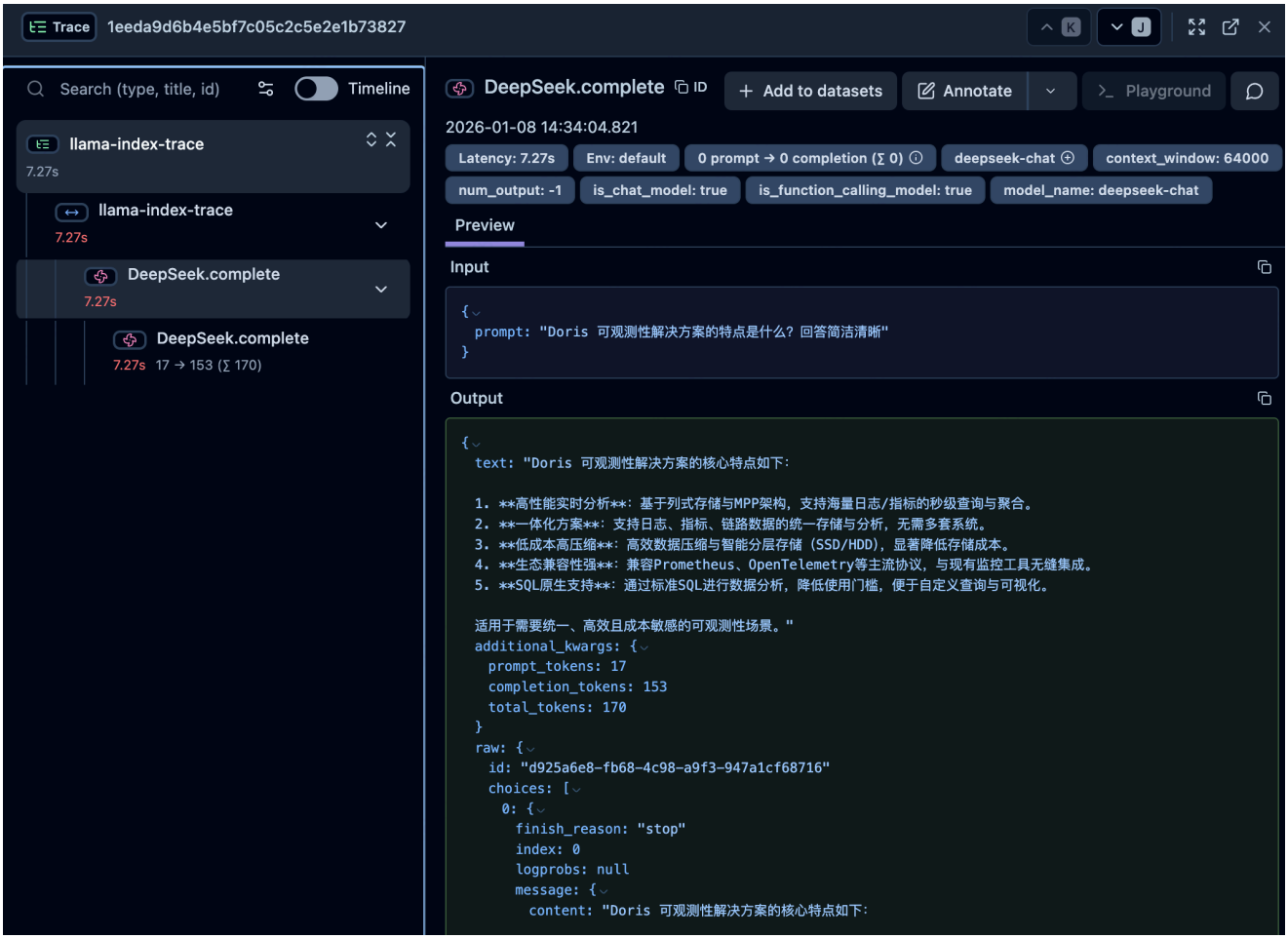

# Initialize LlamaIndex instrumentation
LlamaIndexInstrumentor().instrument()


# Set up the OpenAI class with the required model
```

```
llm = OpenAI(model="gpt-4o")


# ask a question
question = "What are the key features of the Doris observability solution? Please answer concisely."
print(f"question: {question} \n")

with langfuse.start_as_current_span(name="llama-index-trace"):
    response = llm.complete(question)
    print(f"response: {response}")
```



Integrating Vector with Doris

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

Integrating Vector with Doris

About Vector

Vector is a high-performance observability data pipeline written in Rust, specifically designed for collecting, transforming, and routing logs, metrics, and traces. To better support the Doris ecosystem, we have developed a dedicated Doris Sink component for Vector, enabling efficient data ingestion from various data sources into Doris for analysis.

Installation

Download Installation Package

```
wget https://assets-cdn.selectdb.com/downloads/releases/extension/vector-x86_64-unknown-linux-gnu.tar.gz
```

Build from Source

```
cd ${Vector_HOME}

## Choose the appropriate option based on your deployment environment. Multiple options are available in the Makefile.
make package-x86_64-unknown-linux-gnu
```

Configuration Parameters

Doris Sink supports extensive configuration options to meet data writing requirements in different scenarios:

Basic Configuration

Parameter	Type	Default	Description
`type`	string	-	Fixed as `doris`

Parameter	Type	Default	Description
`inputs`	array	-	List of upstream data source names
`endpoints`	array<string>	-	Doris FE HTTP/HTTPS addresses, must include protocol and port, e.g., `["https://fe1:8030"]`
`database`	string/template	-	Target database name, supports Template
`table`	string/template	-	Target table name, supports template
`label_prefix`	string	`"vector"`	Stream Load label prefix, final label format is `{label_prefix}_{database}_{table}_{timestamp}_{uuid}`

Authentication Configuration

Parameter	Type	Default	Description
`auth.strategy`	string	`"basic"`	Authentication strategy, Doris currently only supports Basic Auth
`auth.user`	string	-	Doris username
`auth.password`	string	-	Doris password, can be used with environment variables or secret management systems

Request and Concurrency Configuration

Parameter	Type	Default	Description
`request.concurrency`	string/integer	`"adaptive"`	Controls concurrency strategy, supports `"adaptive"`, `"none"` (serial), or a positive integer for concurrency limit
`request.timeout_secs`	integer	`60`	Timeout for a single Stream Load request (seconds)
`request.rate_limit_duration_secs`	integer	`1`	Rate limit time window (seconds)
`request.rate_limit_num`	integer	`i64::MAX`	Number of requests allowed per time window, default is virtually unlimited
`request.retry_attempts`	integer	`usize::MAX`	Maximum retry attempts for Tower middleware, default means unlimited retries
`request.retry_initial_backoff_secs`	integer	`1`	Wait time before the first retry (seconds), subsequent retries use Fibonacci backoff
`request.retry_max_duration_secs`	integer	`30`	Maximum wait time for a single retry backoff (seconds)
`request.retry_jitter_mode`	string	`"full"`	Retry jitter mode, supports `full` or `none`

Adaptive Concurrency (`request.adaptive_concurrency`, only effective when `request.concurrency = "adaptive"`)

Parameter	Type	Default	Description
`request.adaptive_concurrency.initial_concurrency`	integer	`1`	Initial value for adaptive concurrency
`request.adaptive_concurrency.max_concurrency_limit`	integer	`200`	Upper limit for adaptive concurrency to prevent overload
`request.adaptive_concurrency.decrease_ratio`	float	`0.9`	Reduction ratio used when triggering slowdown
`request.adaptive_concurrency.ewma_alpha`	float	`0.4`	Exponential moving average weight for RTT metrics

Parameter	Type	Default	Description
request.adaptive_concurrency.rtt_deviation_scale	float	2.5	RTT deviation amplification factor, used to ignore normal fluctuations

Encoding and Data Format

Doris Sink uses the encoding block to control event serialization behavior, defaulting to NDJSON (newline-delimited JSON):

Parameter	Type	Default	Description
encoding.codec	string	"json"	Serialization encoding, options include json, text, csv, etc.
encoding.timestamp_format	string	-	Adjust timestamp output format, supports rfc3339, unix, etc.
encoding.only_fields / encoding.except_fields	array<string>	-	Control field whitelist or blacklist
encoding.framing.method	string	auto-inferred	Set when custom framing format is needed, e.g., newline_delimited, character_delimited

Stream Load Headers (headers)

headers is a key-value pair mapping that is passed directly as HTTP headers for Doris Stream Load. You can use all parameters available in stream load headers. Common settings are as follows (all values must be strings):

Parameter	Type	Default	Description
headers.format	string	"json"	Data format, supports json, csv, parquet, etc.
headers.read_json_by_line	string	"true"	Whether to read JSON line by line (NDJSON)
headers.strip_outer_array	string	"false"	Whether to remove the outermost array
headers.column_separator	string	-	CSV column separator (effective when format = csv)
headers.columns	string	-	Column order for CSV/JSON mapping, e.g., timestamp,client_ip,status_code

Parameter	Type	Default	Description
`headers.where`	string	-	Stream Load `where` filter condition

Batch Configuration

Parameter	Type	Default	Description
`batch.max_bytes`	integer	`10485760`	Maximum bytes per batch (10 MB)
`batch.max_events`	integer/`null`	`null`	Maximum events per batch, default is unlimited, primarily controlled by byte count
`batch.timeout_secs`	float	`1`	Maximum wait time for a batch (seconds)

Reliability and Security Configuration

Parameter	Type	Default	Description
`max_retries`	integer	`-1`	Maximum retries at Sink level, `-1` means unlimited
`log_request`	boolean	`false`	Whether to print each Stream Load request and response (enable as needed in production)
`compression`	-	`Not supported`	-
`distribution.retry_initial_backoff_secs`	integer	`1`	Initial backoff time for endpoint health check recovery (seconds)
`distribution.retry_max_duration_secs`	integer	`3600`	Maximum health check backoff duration (seconds)
`tls.verify_certificate`	boolean	`true`	Enable/disable upstream certificate verification
`tls.verify_hostname`	boolean	`true`	Enable/disable hostname verification
`tls.ca_file` / `tls.crt_file` / `tls.key_file` / `tls.key_pass` / `tls.alpn_protocols` / `tls.server_name`	various	-	Standard Vector TLS client configuration options for custom CA, mutual authentication, or SNI
`acknowledgements.enabled`	boolean	`false`	Enable end-to-end acknowledgements for use with Sources that support acknowledgements

Usage Examples

TEXT Log Collection Example

This example demonstrates TEXT log collection using Doris FE logs as an example.

1. Data

FE log files are typically located at fe/log/fe.log under the Doris installation directory. This is a typical Java application log containing fields such as timestamp, log level, thread name, code position, and log message. In addition to regular logs, there are exception logs with stack traces that span multiple lines. Log collection and storage need to combine the main log and stack trace into a single log entry.

```
2024-07-08 21:18:01,432 INFO (Statistics Job Appender|61) [StatisticsJobAppender.runAfterCatalogReady():70] Stats table not
available, skip
2024-07-08 21:18:53,710 WARN (STATS_FETCH-0|208) [StmtExecutor.executeInternalQuery():3332] Failed to run internal SQL:
OriginStatement{originStmt='SELECT * FROM __internal_schema.column_statistics WHERE part_id is NULL  ORDER BY update_time DESC LIMIT
500000', idx=0}
org.apache.doris.common.UserException: errCode = 2, detailMessage = tablet 10031 has no queryable replicas. err: replica 10032's
backend 10008 does not exist or not alive
        at org.apache.doris.planner.OlapScanNode.addScanRangeLocations(OlapScanNode.java:931) ~[doris-fe.jar:1.2-SNAPSHOT]
        at org.apache.doris.planner.OlapScanNode.computeTabletInfo(OlapScanNode.java:1197) ~[doris-fe.jar:1.2-SNAPSHOT]
```

2. Create Table

The table structure includes fields for log generation time, collection time, hostname, log file path, log type, log level, thread name, code position, and log message.

```
CREATE TABLE `doris_log` (
  `log_time` datetime NULL COMMENT 'log content time',
  `collect_time` datetime NULL COMMENT 'log agent collect time',
  `host` text NULL COMMENT 'hostname or ip',
  `path` text NULL COMMENT 'log file path',
  `type` text NULL COMMENT 'log type',
  `level` text NULL COMMENT 'log level',
  `thread` text NULL COMMENT 'log thread',
  `position` text NULL COMMENT 'log code position',
  `message` text NULL COMMENT 'log message',
  INDEX idx_host (`host`) USING INVERTED COMMENT '',
  INDEX idx_path (`path`) USING INVERTED COMMENT '',
  INDEX idx_type (`type`) USING INVERTED COMMENT '',
  INDEX idx_level (`level`) USING INVERTED COMMENT '',
  INDEX idx_thread (`thread`) USING INVERTED COMMENT '',
  INDEX idx_position (`position`) USING INVERTED COMMENT '',
  INDEX idx_message (`message`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true") COMMENT ''
) ENGINE=OLAP
DUPLICATE KEY(`log_time`)
COMMENT 'OLAP'
PARTITION BY RANGE(`log_time`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"dynamic_partition.enable" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-7",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
```

```
"dynamic_partition.create_history_partition" = "true",
"compaction_policy" = "time_series"
);
```

3. Vector Configuration

```
# ==================== Sources ====================
[sources.fe_log_input]
  type = "file"
  include = ["/path/fe/log/fe.log"]
  start_at_beginning = true
  max_line_bytes = 102400
  ignore_older_secs = 0
  fingerprint.strategy = "device_and_inode"

  # Multi-line log handling - corresponds to Logstash's multiline codec
  # Lines starting with a timestamp are new logs, other lines are merged with the previous line (handling stack traces)
  multiline.start_pattern = "^\\d{4}-\\d{2}-\\d{2} \\d{2}:\\d{2}:\\d{2},\\d{3}"
  multiline.mode = "halt_before"
  multiline.condition_pattern = "^\\d{4}-\\d{2}-\\d{2} \\d{2}:\\d{2}:\\d{2},\\d{3}"
  multiline.timeout_ms = 10000

# ==================== Transforms ====================
# Use grok to parse log content
[transforms.parse_log]
  inputs = ["fe_log_input"]
  type = "remap"
  source = '''
    # Add type field (corresponds to Logstash's add_field)
    .type = "fe.log"

    # Add collect_time (corresponds to Logstash's @timestamp)
    # Use Asia/Shanghai timezone, consistent with log_time
    .collect_time = format_timestamp!(.timestamp, format: "%Y-%m-%d %H:%M:%S", timezone: "Asia/Shanghai")

    # Parse log format: 2024-01-01 12:00:00,123 INFO (thread-name) [position] message
    # Use (?s) to enable DOTALL mode, allowing .* to match newlines (handling multi-line logs)
    parsed, err = parse_regex(.message, r'(?s)^(?P<log_time>\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2},\d{3}) (?P<level>[A-Z]+) \((?
P<thread>[^\)]+)\) \[(?P<position>[^\]]+)\] (?P<content>.*)')

    # Extract parsed fields
    if err == null {
      .log_time = parsed.log_time
      .level = parsed.level
      .thread = parsed.thread
      .position = parsed.position
      # Keep the complete original message (including multi-line stack traces)
    } else {
      # If parsing fails, set default values to avoid NULL (avoid partition errors)
      .log_time = .collect_time
      .level = "UNKNOWN"
      .thread = ""
      .position = ""
    }

    # Extract host and path (Vector automatically adds these metadata)
    .host = .host
    .path = .file
  '''

# ==================== Sinks ====================
[sinks.doris]
  inputs = ["parse_log"]
```

```
  type = "doris"
  endpoints = ["http://fe_ip:http_port"]
  database = "log_db"
  table = "doris_log"
  label_prefix = "vector_fe_log"
  log_request = true

[sinks.doris.auth]
  user = "root"
  password = ""
  strategy = "basic"

[sinks.doris.encoding]
  codec = "json"

[sinks.doris.framing]
  method = "newline_delimited"

[sinks.doris.request]
  concurrency = 10

[sinks.doris.headers]
  format = "json"
  read_json_by_line = "true"
  load_to_single_tablet = "true"

[sinks.doris.batch]
  max_events = 10000
  timeout_secs = 3
  max_bytes = 100000000
```

4. Run Vector

```
${VECTOR_HOME}/bin/vector --config vector_fe_log.toml

# When log_request is true, the log will output the request parameters and response results of each Stream Load
2025-11-19T10:14:40.822071Z  INFO sink{component_kind="sink" component_id=doris component_type=doris}:request{request_id=82}:
vector::sinks::doris::service: Doris stream load response received. status_code=200 OK stream_load_status=Successful response={
  "TxnId": 169721,
  "Label": "vector_fe_log_log_db_doris_log_1763547280791_e2e619ee-4067-4fe8-974e-9f35f0d4e48e",
  "Comment": "",
  "TwoPhaseCommit": "false",
  "Status": "Success",
  "Message": "OK",
  "NumberTotalRows": 10,
  "NumberLoadedRows": 10,
  "NumberFilteredRows": 0,
  "NumberUnselectedRows": 0,
  "LoadBytes": 7301,
  "LoadTimeMs": 30,
  "BeginTxnTimeMs": 0,
  "StreamLoadPutTimeMs": 1,
  "ReadDataTimeMs": 0,
  "WriteDataTimeMs": 8,
  "ReceiveDataTimeMs": 2,
  "CommitAndPublishTimeMs": 18
} internal_log_rate_limit=true
```

JSON Log Collection Example

This example demonstrates JSON log collection using GitHub events archive data.

1. Data

GitHub events archive contains archived data of GitHub user operation events in JSON format. You can download it from https://www.gharchive.org/, for example, downloading data from 15:00 on January 1, 2024.

```
wget https://data.gharchive.org/2024-01-01-15.json.gz
```

Below is a sample data entry. The actual data is one entry per line; formatting is added here for display purposes.

```
{
  "id": "37066529221",
  "type": "PushEvent",
  "actor": {
    "id": 46139131,
    "login": "Bard89",
    "display_login": "Bard89",
    "gravatar_id": "",
    "url": "https://api.github.com/users/Bard89",
    "avatar_url": "https://avatars.githubusercontent.com/u/46139131?"
  },
  "repo": {
    "id": 780125623,
    "name": "Bard89/talk-to-me",
    "url": "https://api.github.com/repos/Bard89/talk-to-me"
  },
  "payload": {
    "repository_id": 780125623,
    "push_id": 17799451992,
    "size": 1,
    "distinct_size": 1,
    "ref": "refs/heads/add_mvcs",
    "head": "f03baa2de66f88f5f1754ce3fa30972667f87e81",
    "before": "85e6544ede4ae3f132fe2f5f1ce0ce35a3169d21"
  },
  "public": true,
  "created_at": "2024-04-01T23:00:00Z"
}
```

2. Create Doris Table

```
CREATE DATABASE log_db;
USE log_db;

CREATE TABLE github_events
(
  `created_at` DATETIME,
  `id` BIGINT,
  `type` TEXT,
  `public` BOOLEAN,
  `actor` VARIANT,
  `repo` VARIANT,
  `payload` TEXT,
  INDEX `idx_id` (`id`) USING INVERTED,
  INDEX `idx_type` (`type`) USING INVERTED,
```

```
  INDEX `idx_actor` (`actor`) USING INVERTED,
  INDEX `idx_host` (`repo`) USING INVERTED,
  INDEX `idx_payload` (`payload`) USING INVERTED PROPERTIES("parser" = "unicode", "support_phrase" = "true")
)
ENGINE = OLAP
DUPLICATE KEY(`created_at`)
PARTITION BY RANGE(`created_at`) ()
DISTRIBUTED BY RANDOM BUCKETS 10
PROPERTIES (
"replication_num" = "1",
"inverted_index_storage_format"= "v2",
"compaction_policy" = "time_series",
"enable_single_replica_compaction" = "true",
"dynamic_partition.enable" = "true",
"dynamic_partition.create_history_partition" = "true",
"dynamic_partition.time_unit" = "DAY",
"dynamic_partition.start" = "-30",
"dynamic_partition.end" = "1",
"dynamic_partition.prefix" = "p",
"dynamic_partition.buckets" = "10",
"dynamic_partition.replication_num" = "1"
);
```

3. Vector Configuration

```
# ==================== Sources ====================
[sources.github_events_reload]
type = "file"
include = ["/path/2024-01-01-15.json"]
read_from = "beginning"
ignore_checkpoints = true
max_line_bytes = 10485760
ignore_older_secs = 0
line_delimiter = "\n"
fingerprint.strategy = "device_and_inode"

# ==================== Transforms ====================
# Parse JSON format GitHub Events data, VARIANT type can directly store nested objects
[transforms.parse_json]
inputs = ["github_events_reload"]
type = "remap"
source = '''
    # Parse JSON data (each line is a complete JSON object)
    . = parse_json!(.message)

    # Convert payload field to JSON string (TEXT type)
    .payload = encode_json(.payload)

    # Keep only the fields needed for the table
    . = {
      "created_at": .created_at,
      "id": .id,
      "type": .type,
      "public": .public,
      "actor": .actor,
      "repo": .repo,
      "payload": .payload
    }
  '''

# ==================== Sinks ====================
[sinks.doris]
inputs = ["parse_json"]
```

```
type = "doris"
endpoints = ["http://fe_ip:http_port"]
database = "log_db"
table = "github_events"
label_prefix = "vector_github_events"
log_request = true

[sinks.doris.auth]
user = "root"
password = ""
strategy = "basic"

[sinks.doris.encoding]
codec = "json"

[sinks.doris.framing]
method = "newline_delimited"

[sinks.doris.request]
concurrency = 10

[sinks.doris.headers]
format = "json"
read_json_by_line = "true"
load_to_single_tablet = "true"

[sinks.doris.batch]
max_events = 10000
timeout_secs = 3
max_bytes = 100000000
```

Start Vector

Use the following command to start the Vector service:

```
vector --config vector_config.toml
```

Security Overview

Doris provides the following mechanisms to manage data security:

Authentication: Doris supports both username/password and LDAP authentication methods.

- **Built-in Authentication:** Doris includes a built-in username/password authentication method, allowing customization of password policies.
- **LDAP Authentication:** Doris can centrally manage user credentials through LDAP services, simplifying access control and enhancing system security.

Permission Management: Doris supports role-based access control (RBAC) or can inherit Ranger to achieve centralized permission management.

- **Role-Based Access Control (RBAC):** Doris can restrict users' access to and operations on database resources based on their roles and permissions.
- **Ranger Permission Management:** By integrating with Ranger, Doris enables centralized permission management, allowing administrators to set fine-grained access control policies for different users and groups.

Audit and Logging: Doris can enable audit logs to record all user actions, including logins, queries, data modifications, and more, facilitating post-audit and issue tracking.

Data Encryption and Masking: Doris supports encryption and masking of data within tables to prevent unauthorized access and data leakage.

Data Transmission Encryption: Doris supports SSL encryption protocols to ensure secure data transmission between clients and Doris servers, preventing data from being intercepted or tampered with during transfer.

Fine-Grained Access Control: Doris allows configuring data row and column access permissions based on rules to control user access at a granular level.

JAVA-UDF Security: Doris supports user-defined function functionality, so root administrators need to review the implementation of user UDFs to ensure the operations in the logic are safe and prevent high-risk actions in UDFs, such as data deletion and system disruption.

Third-Party Packages: When using Doris features like JDBC Catalog or UDFs, administrators must ensure that any third-party packages are from trusted and secure sources. To reduce security risks, it is recommended to use dependencies only from official or reputable community sources.

Authentication and Authorization

The Doris permission management system is modeled after the MySQL permission management mechanism. It supports fine-grained permission control at the row and column level, role-based access control, and also supports a whitelist mechanism.

Glossary

1. User Identity

   Within a permission system, a user is identified as a User Identity. A User Identity consists of two parts: `username` and `host`. The `username` is the user's name, consisting of English letters (both uppercase and lowercase). `host` represents the IP from which the user connection originates. User Identity is represented as `username@'host'`, indicating `username` from `host`.

   Another representation of User Identity is `username@['domain']`, where `domain` refers to a domain name that can be resolved into a set of IPs through DNS. Eventually, this is represented as a set of `username@'host'`, hence moving forward, we uniformly use `username@'host'` to denote it.

2. Privilege

   Privileges apply to nodes, data directories, databases, or tables. Different privileges represent different operation permissions.

3. Role

   Doris allows the creation of custom-named roles. A role can be viewed as a collection of privileges. Newly created users can be assigned a role, automatically inheriting the privileges of that role. Subsequent changes to the role's privileges will also reflect on the permissions of all users associated with that role.

4. User Property

   User properties are directly affiliated with a user, not the User Identity. Meaning, both `user@'192.%'` and `user@['domain']` share the same set of user properties, which belong to the user `user`, not to `user@'192.%'` or `user@['domain']`.

   User properties include but are not limited to: maximum number of user connections, import cluster configurations, etc.

Authentication and Authorization Framework

The process of a user logging into Apache Doris is divided into two parts: **Authentication** and **Authorization**.

- Authentication: Identity verification is conducted based on the credentials provided by the user (such as username, client IP, password). Once verified, the individual user is mapped to a system-defined User Identity.

- Authorization: Based on the acquired User Identity, it checks whether the user has the necessary permissions for the intended operations, according to the privileges associated with that User Identity.

Authentication

Doris supports built-in authentication schemes as well as LDAP authentication.

Doris Built-in Authentication Scheme

Authentication is based on usernames, passwords, and other information stored within Doris itself.

Administrators create users with the `CREATE USER` command and view all created users with the `SHOW ALL GRANTS` command.

When a user logs in, the system verifies whether the username, password, and client IP address are correct.

Password Policy

Doris supports the following password policies to assist users in better password management.

1. `PASSWORD_HISTORY`

   Determines whether a user can reuse a historical password when resetting their current password. For example, `PASSWORD_HISTORY 10` means the last 10 passwords cannot be reused as a new password. Setting `PASSWORD_HISTORY DEFAULT` will use the value from the global variable `PASSWORD_HISTORY`. A setting of 0 disables this feature. The default is 0.

   Examples:

   - Set a global variable: `SET GLOBAL password_history = 10`
   - Set for a user: `ALTER USER user1@'ip' PASSWORD_HISTORY 10`

2. `PASSWORD_EXPIRE`

   Sets the expiration time for the current user's password. For instance, `PASSWORD_EXPIRE INTERVAL 10 DAY` means the password will expire after 10 days. `PASSWORD_EXPIRE NEVER` indicates the password never expires. Setting `PASSWORD_EXPIRE DEFAULT` will use the value from the global variable `default_password_lifetime` (in days). The default is NEVER (or 0), indicating it does not expire.

   Examples:

   - Set a global variable: `SET GLOBAL default_password_lifetime = 1`
   - Set for a user: `ALTER USER user1@'ip' PASSWORD_EXPIRE INTERVAL 10 DAY`

3. `FAILED_LOGIN_ATTEMPTS` and `PASSWORD_LOCK_TIME`

   Configures the number of incorrect password attempts after which the user account will be locked and sets the lock duration. For example, `FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1 DAY` means if there are 3 incorrect logins, the account will be locked for one day. Administrators can unlock the account using the `ALTER USER` statement.

   Example:

- Set for a user: `ALTER USER user1@'ip' FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1 DAY`

4. Password Strength

   This is controlled by the global variable `validate_password_policy`. The default is `NONE/0`, which means no password strength checking. If set to `STRONG/2`, the password must include at least three of the following: uppercase letters, lowercase letters, numbers, and special characters, and must be at least 8 characters long.

   Example:

   - `SET validate_password_policy=STRONG`

For more help, please refer to ALTER USER.

LDAP-based Authentication Scheme

Please refer to LDAP-based Authentication Scheme.

Authorization

Permission Operations

- Create user: CREATE USER
- Modify user: ALTER USER
- Delete user: DROP USER
- Grant/Assign role: GRANT
- Revoke/Withdraw role: REVOKE
- Create role: CREATE ROLE
- Delete role: DROP ROLE
- Modify role: ALTER ROLE
- View current user's permissions and roles: SHOW GRANTS
- View all users' permissions and roles: SHOW ALL GRANTS
- View created roles: SHOW ROLES
- Set user property: SET PROPERTY
- View user property: SHOW PROPERTY
- Change password: SET PASSWORD
- View all supported privileges: [SHOW PRIVILEGES]
- View row policy: [SHOW ROW POLICY]
- Create row policy: [CREATE ROW POLICY]

Types of Permissions

Doris currently supports the following permissions:

1. `Node_priv`

   Node modification permission. Includes adding, deleting, and offlining FE, BE, BROKER nodes.

Root users have this permission by default. Users who possess both `Grant_priv` and `Node_priv` can grant this permission to other users.

This permission can only be granted at the Global level.

2. `Grant_priv`

   Permission modification authority. Allows execution of operations including granting, revoking, adding/deleting/modifying users/roles.

   Before version 2.1.2, when granting permissions to other users/roles, the current user only needed the respective level's `Grant_priv` permission. After version 2.1.2, the current user also needs permission for the resource they wish to grant.

   When assigning roles to other users, Global level `Grant_priv` permission is required.

3. `Select_priv`

   Read-only permission for data directories, databases, and tables.

4. `Load_priv`

   Write permission for data directories, databases, and tables. Includes Load, Insert, Delete, etc.

5. `Alter_priv`

   Alteration permissions for data directories, databases, and tables. Includes renaming libraries/tables, adding/deleting/modifying columns, adding/deleting partitions, etc.

6. `Create_priv`

   Permission to create data directories, databases, tables, and views.

7. `Drop_priv`

   Permission to delete data directories, databases, tables, and views.

8. `Usage_priv`

   Usage permissions for Resources and Workload Groups.

9. `Show_view_priv`

   Permission to execute `SHOW CREATE VIEW`.

Permission Levels

Global Permissions

Permissions granted through the GRANT statement with `*.*.*` scope. These permissions apply to any table within any catalog.

Catalog Permissions

Permissions granted through the GRANT statement with `ctl.*.*` scope. These permissions apply to any table within the specified catalog.

Database Permissions

Permissions granted through the GRANT statement with `ctl.db.*` scope. These permissions apply to any table within the specified database.

Table Permissions

Permissions granted through the GRANT statement with `ctl.db.tbl` scope. These permissions apply to any column within the specified table.

Column Permissions

Column permissions are primarily used to restrict user access to certain columns within a table. Specifically, column permissions allow administrators to set viewing, editing, and other rights for certain columns, controlling user access and manipulation of specific column data.

Permissions for specific columns of a table can be granted with `GRANT Select_priv(col1,col2) ON ctl.db.tbl TO user1`.

Currently, column permissions support only `Select_priv`.

Row-Level Permissions

Row Policies enable administrators to define access policies based on fields within the data, controlling which users can access which rows.

Specifically, Row Policies allow administrators to create rules that can filter or restrict user access to rows based on actual values stored in the data.

From version 1.2, row-level permissions can be created with the `CREATE ROW POLICY` command.

From version 2.1.2, support for setting row-level permissions through Apache Ranger's `Row Level Filter` is available.

Usage Permissions

- Resource Permissions

  Resource permissions are set specifically for Resources, unrelated to permissions for databases or tables, and can only assign `Usage_priv` and `Grant_priv`.

  Permissions for all Resources can be granted with the `GRANT USAGE_PRIV ON RESOURCE '%' TO user1`.

- Workload Group Permissions

> Workload Group permissions are set specifically for Workload Groups, unrelated to permissions for databases or tables, and can only assign `Usage_priv` and `Grant_priv`.
>
> Permissions for all Workload Groups can be granted with `GRANT USAGE_PRIV ON WORKLOAD GROUP '%' TO user1`.

Data Masking

Data masking is a method to protect sensitive data by modifying, replacing, or hiding the original data, such that the masked data retains certain formats and characteristics while no longer containing sensitive information.

For example, administrators may choose to replace part or all of the digits of sensitive fields like credit card numbers or ID numbers with asterisks * or other characters, or replace real names with pseudonyms.

From version 2.1.2, support for setting data masking policies for certain columns through Apache Ranger's Data Masking is available, currently only configurable via Apache Ranger.

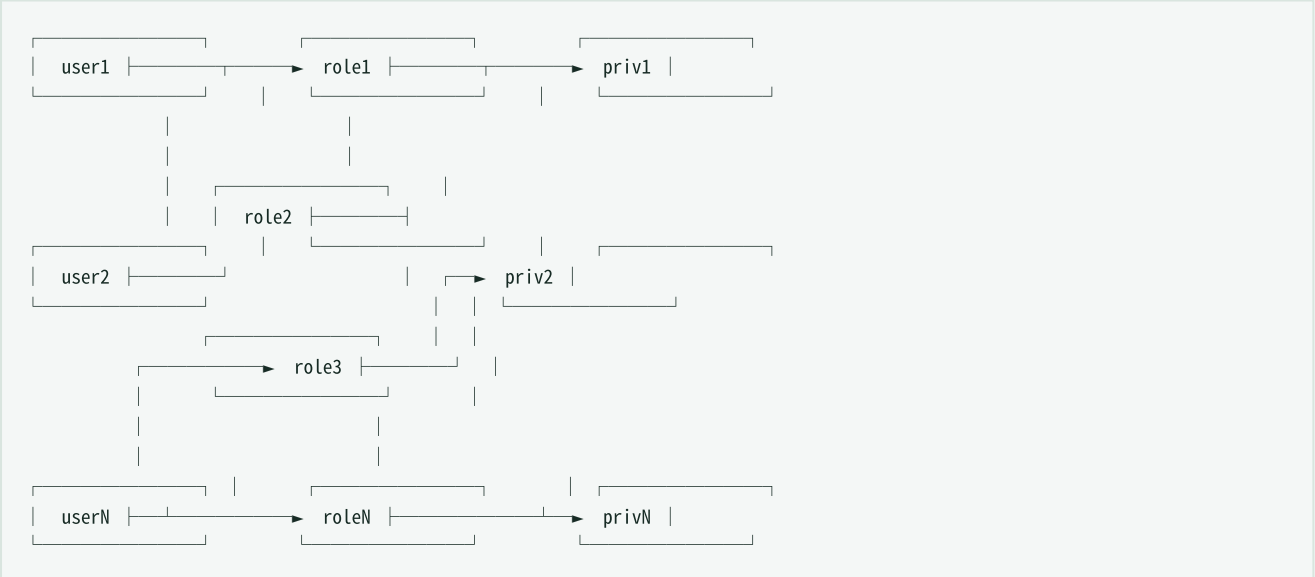
Doris Built-in Authorization Scheme

Doris's permission design is based on the RBAC (Role-Based Access Control) model, where users are associated with roles, and roles are associated with permissions. Users are indirectly linked to permissions through their roles.

When a role is deleted, users automatically lose all permissions associated with that role.

When a user is disassociated from a role, they automatically lose all permissions of that role.

When permissions are added to or removed from a role, the permissions of the users associated with that role change accordingly.



As shown above:

User1 and user2 both have permission `priv1` through `role1`.

UserN has permission `priv1` through `role3`, and permissions `priv2` and `privN` through `roleN`. Thus, userN has permissions `priv1`, `priv2`, and `privN` simultaneously.

For ease of user operations, it is possible to directly grant permissions to a user. Internally, a unique default role is created for each user. When permissions are granted to a user, it is essentially granting permissions to the user's default role.

The default role cannot be deleted, nor can it be assigned to someone else. When a user is deleted, their default role is automatically deleted as well.

Authorization Scheme Based on Apache Ranger

Please refer to Authorization Scheme Based on Apache Ranger.

Common Questions

Explanation of Permissions

1. Users with ADMIN privileges or GRANT privileges at the GLOBAL level can perform the following operations:
    - CREATE USER
    - DROP USER
    - ALTER USER
    - SHOW GRANTS
    - CREATE ROLE
    - DROP ROLE
    - ALTER ROLE
    - SHOW ROLES
    - SHOW PROPERTY FOR USER
2. GRANT/REVOKE
    - Users with ADMIN privileges can grant or revoke permissions for any user.
    - Users with ADMIN or GLOBAL level GRANT privileges can assign roles to users.
    - Users who have the corresponding level of GRANT privilege and the permissions to be assigned can distribute those permissions to users/roles.
3. SET PASSWORD
    - Users with ADMIN privileges or GLOBAL level GRANT privileges can set passwords for non-root users.
    - Ordinary users can set the password for their corresponding User Identity. Their corresponding User Identity can be viewed with the `SELECT CURRENT_USER()` command.
    - ROOT users can change their own password.

Additional Information

1. When Doris is initialized, the following users and roles are automatically created:

- operator role: This role has `Node_priv` and `Admin_priv`, i.e., all permissions in Doris.
- admin role: This role has `Admin_priv`, i.e., all permissions except for node changes.
- root@‘%’: root user, allowed to log in from any node, with the operator role.
- admin@‘%’: admin user, allowed to log in from any node, with the admin role.

2. Deleting or altering the permissions of default created users, roles, or users is not supported.

    - Deleting the users root@‘%’ and admin@‘%’ is not supported, but creating and deleting root@‘xxx’ and admin@‘xxx’ users (where xxx refers to any host except %) is allowed (Doris treats these users as regular users).
    - Revoking the default roles of root@‘%’ and admin@‘%’ is not supported.
    - Deleting the roles operator and admin is not supported.
    - Modifying the permissions of the roles operator and admin is not supported.

3. There is only one user with the operator role, which is Root. There can be multiple users with the admin role.

4. Some potentially conflicting operations are explained as follows:

    1. Domain and IP conflict:

        Suppose the following user is created:

        ```
        CREATE USER user1@['domain'];
        ```

        And granted:

        ```
        GRANT SELECT_PRIV ON *.* TO user1@['domain']
        ```

        This domain is resolved to two IPs: ip1 and ip2.

        Suppose later, we grant a separate permission to `user1@'ip1'`:

        ```
        GRANT ALTER_PRIV ON . TO user1@'ip1';
        ```

        Then `user1@'ip1'` will have permissions for both Select_priv and Alter_priv. And when we change the permissions for `user1@['domain']` again, `user1@'ip1'` will not follow the change.

    2. Duplicate IP conflict:

        Suppose the following users are created:

        ```
        CREATE USER user1@'%' IDENTIFIED BY "12345";
        CREATE USER user1@'192.%' IDENTIFIED BY "abcde";
        ```

        In terms of priority, `'192.%'` takes precedence over `'%'`, so when user `user1` from machine `192.168.1.1` tries to log into Doris using password `'12345'`, access will be denied.

5. Forgotten Password

    If you forget the password and cannot log into Doris, you can add `skip_localhost_auth_check=true` to the FE’s config file and restart the FE, thus logging into Doris as root without a password from the local machine.

After logging in, you can reset the password using the `SET PASSWORD` command.

6. No user can reset the root user's password except for the root user themselves.

7. `Admin_priv` permissions can only be granted or revoked at the GLOBAL level.

8. `current_user()` and `user()`

   Users can view their `current_user` and `user` by executing `SELECT current_user()` and `SELECT user()` respectively. Here, `current_user` indicates the identity the user authenticated with, while `user` is the actual User Identity at the moment.

   For example:

   Suppose `user1@'192.%'` is created, and then user `user1` logs in from `192.168.10.1`, then the `current_user` would be `user1@'192.%'`, and `user` would be `user1@'192.168.10.1'`.

   All permissions are granted to a specific `current_user`, and the real user has all the permissions of the corresponding `current_user`.

Best Practices

Here are some examples of use cases for the Doris permission system.

1. Scenario 1

   Users of the Doris cluster are divided into administrators (Admin), development engineers (RD), and users (Client). Administrators have all permissions over the entire cluster, primarily responsible for cluster setup and node management. Development engineers are responsible for business modeling, including creating databases and tables, importing, and modifying data. Users access different databases and tables to retrieve data.

   In this scenario, administrators can be granted ADMIN or GRANT privileges. RDs can be granted CREATE, DROP, ALTER, LOAD, and SELECT permissions for any or specific databases and tables. Clients can be granted SELECT permissions for any or specific databases and tables. Additionally, different roles can be created to simplify the authorization process for multiple users.

2. Scenario 2

   A cluster may contain multiple businesses, each potentially using one or more datasets. Each business needs to manage its users. In this scenario, an administrative user can create a user with DATABASE-level GRANT privileges for each database. This user can only authorize users for the specified database.

3. Blacklist

   Doris itself does not support a blacklist, only a whitelist, but we can simulate a blacklist through certain means. Suppose a user named `user@'192.%'` is created, allowing users from `192.*` to log in. If you want to prohibit a user from `192.168.10.1` from logging in, you can create another user `cmy@'192.168.10.1'` with a new password. Since `192.168.10.1` has higher priority than `192.%`, the user from `192.168.10.1` will no longer be able to log in with the old password.

Built-in Authentication

Key Concepts

Users

In Doris, a user_identity uniquely identifies a user. user_identity consists of two parts: user_name and host, where username is the user name. host identifies the host address where the user client connection is located. The host part can use % for fuzzy matching. If host is not specified, it defaults to '%', which means the user can connect to Doris from any host.

User Attributes

User attributes are directly attached to user_name, not user_identity. This means user@'192.%' and user@['domain'] both have the same set of user attributes. These attributes belong to user, not to user@'192.%' or user@['domain'].

User attributes include but are not limited to: maximum user connections, import cluster configuration, etc.

Built-in Users

Built-in users are users created by default in Doris and have certain default permissions, including root and admin. The initial passwords are empty. After FE starts, you can modify them using the password change command. Built-in users cannot be deleted.

- root@'%': root user, allowed to log in from any node, with operator role.
- admin@'%': admin user, allowed to log in from any node, with admin role.

Password

User login credentials, set by administrators when creating users, or can be changed by users themselves after creation.

Password Policies

Doris supports the following password policies to help users better manage passwords.

- `PASSWORD_HISTORY`

  Whether to allow the current user to use historical passwords when resetting passwords. For example, `PASSWORD_HISTORY` 10 means prohibiting the use of the last 10 passwords as new passwords. If set to `PASSWORD_HISTORY DEFAULT`, it will use the value in the global variable `password_history`. 0 means this feature is disabled. Default is 0.

  Examples:

    - Set global variable: `SET GLOBAL password_history = 10`
    - Set for user: `ALTER USER user1@'ip' PASSWORD_HISTORY 10`

- `PASSWORD_EXPIRE`

  Set the expiration time for the current user's password. For example, `PASSWORD_EXPIRE INTERVAL 10 DAY` means the password will expire in 10 days. `PASSWORD_EXPIRE NEVER` means the password never expires. If set to `PASSWORD_EXPIRE DEFAULT`, it will use the value in the global variable `default_password_lifetime` (in days). Default is NEVER (or 0), meaning it will not expire.

  Examples:

    - Set global variable: `SET GLOBAL default_password_lifetime = 1`
    - Set for user: `ALTER USER user1@'ip' PASSWORD_EXPIRE INTERVAL 10 DAY`

- `FAILED_LOGIN_ATTEMPTS` and `PASSWORD_LOCK_TIME`

  Set that when the current user logs in with wrong password n times, the account will be locked and set a lock time. For example, `FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1 DAY` means if there are 3 wrong logins, the account will be locked for one day. Administrators can actively unlock locked accounts through ALTER USER statements.

  Examples:

    - Set for user: `ALTER USER user1@'ip' FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1 DAY`

- Password Strength

  This feature is controlled by the global variable `validate_password_policy`. Default is NONE/0, meaning password strength is not checked. If set to STRONG/2, the password must contain 3 of the following: "uppercase letters", "lowercase letters", "numbers", and "special characters", and the length must be greater than or equal to 8.

  Examples:

    - `SET validate_password_policy=STRONG`

After setting the above policies, you can view them with the following command:

```
SHOW PROC "/auth/'<user>'@'<host>'";
```

Note that you need to wrap the user and host parts with single quotes separately. For example:

```
SHOW PROC "/auth/'root'@'%'";
SHOW PROC "/auth/'user1'@'127.0.0.1'";
```

Authentication Mechanism

1. Client authentication information sending: The client packages and sends user information (such as username, password, database, etc.) to the Doris server. This information is used to prove the client's identity and request access to the database.

2. Server authentication: After Doris receives the client's authentication information, it performs verification. If the username, password, and client IP are correct, and the user has permission to access the selected database, authentication succeeds, and Doris will map the user individual to the User Identity in the system. Otherwise, authentication fails and returns the corresponding error message to the client.

Blacklist and Whitelist

Doris itself does not support blacklists, only whitelist functionality, but we can simulate blacklists in certain ways. Suppose we first created a user named `user@'192.%'`, indicating that users from `192.*` are allowed to log in. If we want to prohibit users from `192.168.10.1` from logging in, we can create another user `cmy@'192.168.10.1'` and set a new password. Because `192.168.10.1` has higher priority than `192.%`, users from `192.168.10.1` will no longer be able to log in using the old password.

Related Commands

- Create user: CREATE USER
- View users: SHOW ALL GRANTS
- Modify user: ALTER USER
- Change password: SET PASSWORD
- Delete user: DROP USER
- Set user attributes: SET PROPERTY
- View user attributes: SHOW PROPERTY

Other Notes

1. Priority selection of user_identity during login

   As mentioned above, `user_identity` consists of `user_name` and `host`, but when users log in, they only need to input `user_name`, so Doris matches the corresponding `host` based on the client's IP to determine which `user_identity` to use for login.

   If only one `user_identity` can be matched based on the client IP, then this `user_identity` will undoubtedly be matched. However, when multiple `user_identity` can be matched, there will be the following priority issues.

   1. Priority between domain and IP:

      Suppose the following users were created:

      ```
      CREATE USER user1@['domain1'] IDENTIFIED BY "12345";
      CREATE USER user1@'ip1'IDENTIFIED BY "abcde";
      ```

      domain1 is resolved to two IPs: ip1 and ip2.

      In terms of priority, IP takes precedence over domain, so when user user1 tries to log in to Doris from ip1 using password '12345', it will be rejected.

   2. Priority between specific IP and IP range:

   Suppose the following users were created:

```
CREATE USER user1@'%' IDENTIFIED BY "12345";
CREATE USER user1@'192.%' IDENTIFIED BY "abcde";
```

In terms of priority, ‘192.%’ takes precedence over ‘%’, so when user user1 tries to log in to Doris from 192.168.1.1 using password ‘12345’, it will be rejected.

2. Forgot password

   If you forget your password and cannot log in to Doris, you can add the skip_localhost_auth_check=true parameter in the FE config file and restart FE, so you can log in to Doris as root on the FE local machine without a password.

   After logging in, you can reset the password using the SET PASSWORD command.

3. No user can reset the root user’s password except the root user itself.

4. `current_user()` and `user()`

   Users can view `current_user` and `user` through `SELECT current_user()` and `SELECT user()` respectively. `current_user` represents the identity through which the current user passed the authentication system, while `user` is the user’s actual User Identity.

   Example:

   Suppose we created the user `user1@'192.%'`, then user `user1` from `192.168.10.1` logged into the system. In this case, `current_user` would be `user1@'192.%'`, while `user` would be `user1@'192.168.10.1'`.

   All permissions are granted to a specific `current_user`, and the actual user has all the permissions of the corresponding `current_user`.

Federated Authentication

LDAP

Integrate third-party LDAP services to provide login authentication and group authorization services for Doris.

LDAP Login Authentication

LDAP login authentication refers to supplementing Doris's login authentication by integrating password verification from LDAP services. Doris prioritizes using LDAP to verify user passwords. If the user does not exist in the LDAP service, Doris continues to use its own password verification. If the LDAP password is correct but there is no corresponding account in Doris, a temporary user is created to log in to Doris.

After enabling LDAP, users can exist in the following scenarios in Doris and LDAP:

LDAP User	Doris User	Password	Login Status	User Logged into Doris
Exists	Exists	LDAP Password	Login Successful	Doris User
Exists	Exists	Doris Password	Login Failed	None
Does Not Exist	Exists	Doris Password	Login Successful	Doris User
Exists	Does Not Exist	LDAP Password	Login Successful	Ldap Temporary User

After enabling LDAP, when users log in using the MySQL client, Doris first verifies the user password through the LDAP service. If the user exists in LDAP and the password is correct, Doris logs in with that user; if Doris has a corresponding account, it logs into that account directly. If there is no corresponding account, a temporary account is created for the user to log in. Temporary accounts have corresponding permissions (see LDAP Group Authorization) and are only valid for the current connection. Doris will not create the user or generate user creation metadata. If the login user does not exist in the LDAP service, Doris's password authentication is used.

Assuming LDAP authentication is enabled, configured with `ldap_user_filter = (&(uid={login}))`, and other configurations are correct, the client sets the environment variables accordingly.

For example:

1. Both Doris and LDAP have the account:

   Doris account: `jack@'172.10.1.10'`, password: `123456`

   LDAP user node has attributes: `uid: jack` user password: `abcdef`

   Using the following command to log in to Doris can log in to the `jack@'172.10.1.10'` account:

   ```
   mysql -hDoris_HOST -PDoris_PORT -ujack -p abcdef
   ```

   Using the following command will log in failed:

```
mysql -hDoris_HOST -PDoris_PORT -ujack -p 123456
```

2. LDAP has a user, but Doris does not have a corresponding account:

   LDAP user node has attributes: `uid: jack` user password: `abcdef`

   Using the following command to create a temporary user and log in to `jack@'%'`, the temporary user has basic permissions DatabasePrivs: Select_priv, and the user will be deleted after logging out:

   ```
   mysql -hDoris_HOST -PDoris_PORT -ujack -p abcdef
   ```

3. LDAP does not have a user:

   Doris account: `jack@'172.10.1.10'`, password: `123456`

   Using Doris's password to log in to the account, successful:

   ```
   mysql -hDoris_HOST -PDoris_PORT -ujack -p 123456
   ```

LDAP Group Authorization

LDAP group authorization is to map LDAP groups to Doris roles and grant all corresponding role permissions to the logged-in user. After logging out, Doris will revoke the corresponding role permissions. Before using LDAP group authorization, you should create the corresponding role in Doris and grant permissions to the role.

The permissions of the logged-in user are related to the Doris user and group permissions, as shown in the following table:

LDAP User	Doris User	Login User's Permissions
Exists	Exists	LDAP Group Permissions + Doris User Permissions
Does Not Exist	Exists	Doris User Permissions
Exists	Does Not Exist	LDAP Group Permissions

If the logged-in user is a temporary user and does not have group permissions, the user has the select_priv permission of the information_schema by default.

For example:

LDAP user dn is the "member" attribute of the LDAP group node, and Doris considers the user to belong to that group. Doris takes the first Rdn of the group dn as the group name.

For example, the user dn is `uid=jack,ou=aidp,dc=domain,dc=com`, and the group information is as follows:

```
dn: cn=doris_rd,ou=group,dc=domain,dc=com
objectClass: groupOfNames
member: uid=jack,ou=aidp,dc=domain,dc=com
```

Then the group name is `doris_rd`.

Assuming `jack` also belongs to LDAP groups `doris_qa` and `doris_pm`; and Doris has roles: `doris_rd`, `doris_qa`, `doris_pm`, after logging in using LDAP authentication, the user will not only have the original permissions of the account but also gain the permissions of roles `doris_rd`, `doris_qa`, and `doris_pm`.

> Note:
>
> The group a user belongs to is unrelated to the organizational structure of the LDAP tree. User2 in the example does not necessarily belong to group2.

LDAP Example

Modify Doris Configuration

1. In the `fe/conf/fe.conf` file, configure the authentication method as ldap: `authentication_type=ldap`.
2. In the `fe/conf/ldap.conf` file, configure the basic LDAP information.
3. Set the LDAP administrator password: After configuring the `ldap.conf` file, start the fe, log in to Doris using the root or admin account, and execute the SQL

```
set ldap_admin_password = password('ldap_admin_password');
```

Log in Using the MySQL Client

```
mysql -hDORIS_HOST -PDORIS_PORT -u user -p --enable-cleartext-plugin
Enter the LDAP password
```

Note: To log in using other clients, refer to the section “How Clients Use Clear Text Login” below.

LDAP Information Cache

To avoid frequent access to the LDAP service, Doris caches LDAP information in memory. You can configure the `ldap_user_cache_timeout_s` parameter in the `ldap.conf` file to specify the cache time for LDAP users, which defaults to 12 hours. After modifying the information in the LDAP service or modifying the corresponding role permissions in Doris, the changes may not take effect immediately due to caching. You can use the `refresh ldap` statement to refresh the cache. For details, see REFRESH-LDAP.

Limitations of LDAP Verification

- Currently, Doris’s LDAP function only supports clear text password verification, which means that the password is transmitted in clear text between the client and fe, and between fe and the LDAP service.

Common Issues

- How to determine which roles an LDAP user has in Doris?

Log in to Doris using the LDAP user and execute `show grants;` to view the roles of the current user. `ldapDefaultRole` is the default role that each LDAP user has in Doris.

- Why does an LDAP user have fewer roles in Doris than expected?

    1. Use `show roles;` to check if the expected role exists in Doris. If it does not exist, create the role using `CREATE ROLE rol_name;`.
    2. Check if the expected group is under the organizational structure corresponding to `ldap_group_basedn`.
    3. Check if the expected group has the member attribute.
    4. Check if the member attribute of the expected group contains the current user.

LDAP Concepts

In LDAP, data is organized in a tree structure.

Example (The following explanations are based on this example)

```
- dc=example,dc=com
 - ou = ou1
   - cn = group1
   - cn = user1
 - ou = ou2
   - cn = group2
     - cn = user2
 - cn = user3
```

LDAP Terminology

- dc (Domain Component): Can be understood as the domain name of an organization, serving as the root node of the tree.
- dn (Distinguished Name): Equivalent to a unique name, such as the dn of user1 is `cn=user1,ou=ou1,dc=example,dc=com`, and the dn of user2 is `cn=user2,cn=group2,ou=ou2,dc=example,dc=com`.
- rdn (Relative Distinguished Name): A part of the dn, such as the four rdns of user1 are `cn=user1`, `ou=ou1`, `dc=example`, and `dc=com`.
- ou (Organization Unit): Can be understood as a sub-organization, users can be placed under ou, or directly under the example.com domain.
- cn (common name): Name.
- group: Group, can be understood as a role in Doris.
- user: User, equivalent to a user in Doris.
- objectClass: Can be understood as the type of each line of data, such as how to distinguish group1 as a group or a user, each type of data requires different attributes, such as group requires cn and member (user list), user requires cn, password, uid, etc.

How Clients Use Clear Text Login

MySQL Client

To use LDAP verification, the client needs to enable the MySQL client clear text verification plugin. To log in to Doris using the command line, you can use one of the following methods to enable the MySQL clear text verification plugin:

- Set the environment variable `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN` to 1

  For example, in a Linux or Mac environment, you can use:

  ```
  echo "export LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN=1" >> ~/.bash_profile && source ~/.bash_profile
  ```

- Add the parameter `--enable-cleartext-plugin` when logging in to Doris each time

  ```
  mysql -hDORIS_HOST -PDORIS_PORT -u user -p --enable-cleartext-plugin

  Enter the LDAP password
  ```

JDBC Client

To log in to Doris using the JDBC client, you need to customize the plugin.

First, create a class named `MysqlClearPasswordPluginWithoutSSL` that inherits from `MysqlClearPasswordPlugin`. In this class, override the `requiresConfidentiality()` method and return false.

```
public class MysqlClearPasswordPluginWithoutSSL extends MysqlClearPasswordPlugin {
@Override
public boolean requiresConfidentiality() {
    return false;
  }
}
```

When getting a database connection, you need to configure the customized plugin in the properties.

That is, (xxx is the package name of the customized class)

- authenticationPlugins=xxx.xxx.xxx.MysqlClearPasswordPluginWithoutSSL
- defaultAuthenticationPlugin=xxx.xxx.xxx.MysqlClearPasswordPluginWithoutSSL
- disabledAuthenticationPlugins=com.mysql.jdbc.authentication.MysqlClearPasswordPlugin

For example:

```
jdbcUrl = "jdbc:mysql://localhost:9030/mydatabase?
authenticationPlugins=xxx.xxx.xxx.MysqlClearPasswordPluginWithoutSSL&defaultAuthenticationPlugin=xxx.xxx.xxx.MysqlClearPasswordPlugi
nWithoutSSL&disabledAuthenticationPlugins=com.mysql.jdbc.authentication.MysqlClearPasswordPlugin";
```

Built-in Authorization

Key Concepts

Authorization refers to the mechanism by which user identities are restricted in accessing and operating Doris resources.

Doris uses a Role-Based Access Control (RBAC) model for permission management.

Permissions

Permissions apply to nodes, catalogs, databases, or tables. Different permissions represent different operation allowances.

All Permissions

Permission	Object Type	Description
Admin_priv	Global	Super admin permissions.
Node_priv	Global	Node change permissions. Includes adding, deleting, and decommissioning FE, BE, and BROKER nodes.
Grant_priv	Global, Catalog, Db, Table, Resource, WorkloadGroup	Permission change permissions. Allows operations such as granting, revoking, adding/removing/changing users/roles. When granting permissions to other users/roles, prior to version 2.1.2, the current user only needs the corresponding level of Grant_priv permission. After version 2.1.2, the current user also needs the permissions of the resources they want to grant. To assign roles to other users, Global level Grant_priv permission is required.

Permission	Object Type	Description
Select_priv	Global, Catalog, Db, Table, Column	Select permission. Allows querying data.
Load_priv	Global, Catalog, Db, Table	Load permission. Includes Load, Insert, Delete, etc.
Alter_priv	Global, Catalog, Db, Table	Alter permission. Includes renaming databases/tables, adding/deleting/changing columns, adding/deleting partitions, etc.
Create_priv	Global, Catalog, Db, Table	Create permission. Allows creating catalogs, databases, tables, and views.

Permission	Object Type	Description
Drop_priv	Global, Catalog, Db, Table	Drop permission. Allows dropping catalogs, databases, tables, and views.
Usage_priv	Resource, WorkloadGroup	Usage permission for Resources and Workload Groups.
Show_view_priv	Global, Catalog, Db, Table	Execute SHOW CREATE VIEW permission.

Roles

Doris allows creating custom-named roles. Roles can be seen as a collection of permissions. New users can be assigned a role, and they will automatically be granted the permissions of that role. Subsequent changes to the role's permissions will also affect the permissions of all users belonging to that role.

Built-in Roles

Built-in roles are default roles created by Doris, with default permissions, including operator and admin.

- operator: Has Admin_priv and Node_priv
- admin: Has Admin_priv

Users

In Doris, a `user_identity` uniquely identifies a user. `user_identity` consists of two parts: `user_name` and `host`, where `username` is the username. `host` identifies the host address from which the user connects.

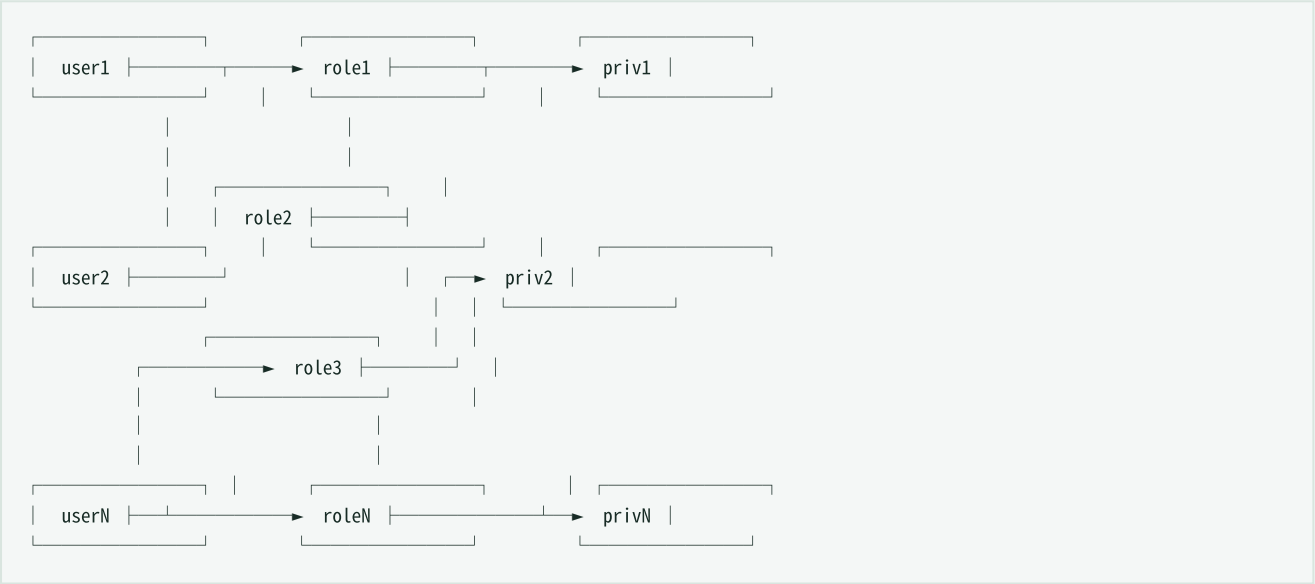
Authorization Mechanism

Doris's permission design is based on the RBAC (Role-Based Access Control) model, associating users with roles, roles with permissions, and users indirectly with permissions through roles.

When a role is deleted, users automatically lose all permissions of that role.

When a user and role are unassociated, the user automatically loses all permissions of that role.

When a role's permissions are added or deleted, the user's permissions also change.



As shown above:

user1 and user2 both have priv1 permission through role1.

userN has priv1 permission through role3 and priv2 and privN permissions through roleN, so userN has priv1, priv2, and privN permissions.

Notes

- For convenience, users can be directly granted permissions. Under the hood, a default role is created for each user, and granting permissions to the user is equivalent to granting permissions to the default role.
- Default roles cannot be deleted, cannot be assigned to other users, and are automatically deleted when the user is deleted.

Related Commands

- Grant/assign role: GRANT
- Revoke/revoke role: REVOKE
- Create role: CREATE ROLE

- Delete role: DROP ROLE
- Modify role: ALTER ROLE
- View current user permissions and roles: SHOW GRANTS
- View all user permissions and roles: SHOW ALL GRANTS
- View created roles: SHOW ROLES
- View supported permissions: SHOW PRIVILEGES

Best Practices

Here are some examples of using Doris's permission system.

1. Scenario 1

   Doris cluster users are divided into administrators (Admin), development engineers (RD), and users (Client). Administrators have all permissions for the entire cluster, mainly responsible for cluster setup, node management, etc. Development engineers are responsible for business modeling, including database and table creation, data import and modification, etc. Users access different databases and tables to retrieve data.

   In this scenario, administrators can be granted ADMIN or GRANT permissions. RD can be granted CREATE, DROP, ALTER, LOAD, and SELECT permissions for any or specified databases and tables. Client can be granted SELECT permission for any or specified databases and tables. At the same time, different roles can be created to simplify permission management for multiple users.

2. Scenario 2

   A cluster has multiple businesses, each business may use one or more datasets. Each business needs to manage its own users. In this scenario, the administrator can create a user with DATABASE-level GRANT permission for each database. This user can only grant permissions for the specified database.

Ranger Authorization

Apache Ranger is a security framework used for monitoring, enabling services, and comprehensive data security access management on the Hadoop platform. After using Ranger, permissions configured on the Ranger side replace the execution of Grant statements in Doris for authorization. For Ranger installation and configuration, see below: Installing and Configuring Doris Ranger Plugin.

Ranger Example

Change Doris Configuration

1. In the `fe/conf/fe.conf` file, configure the authorization method as `ranger access_controller_type=ranger-doris`.

2. Create a `ranger-doris-security.xml` file in the conf directory of all FEs with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
        <name>ranger.plugin.doris.policy.cache.dir</name>
        <value>/path/to/ranger/cache/</value>
    </property>
    <property>
        <name>ranger.plugin.doris.policy.pollIntervalMs</name>
        <value>30000</value>
    </property>
    <property>
        <name>ranger.plugin.doris.policy.rest.client.connection.timeoutMs</name>
        <value>60000</value>
    </property>
    <property>
        <name>ranger.plugin.doris.policy.rest.client.read.timeoutMs</name>
        <value>60000</value>
    </property>
    <property>
        <name>ranger.plugin.doris.policy.rest.url</name>
        <value>http://172.21.0.32:6080</value>
    </property>
    <property>
        <name>ranger.plugin.doris.policy.source.impl</name>
        <value>org.apache.ranger.admin.client.RangerAdminRESTClient</value>
    </property>
    <property>
        <name>ranger.plugin.doris.service.name</name>
        <value>doris</value>
    </property>
</configuration>
```

   Among which, the `ranger.plugin.doris.policy.cache.dir` and `ranger.plugin.doris.policy.rest.url` need to be modified to their actual values.

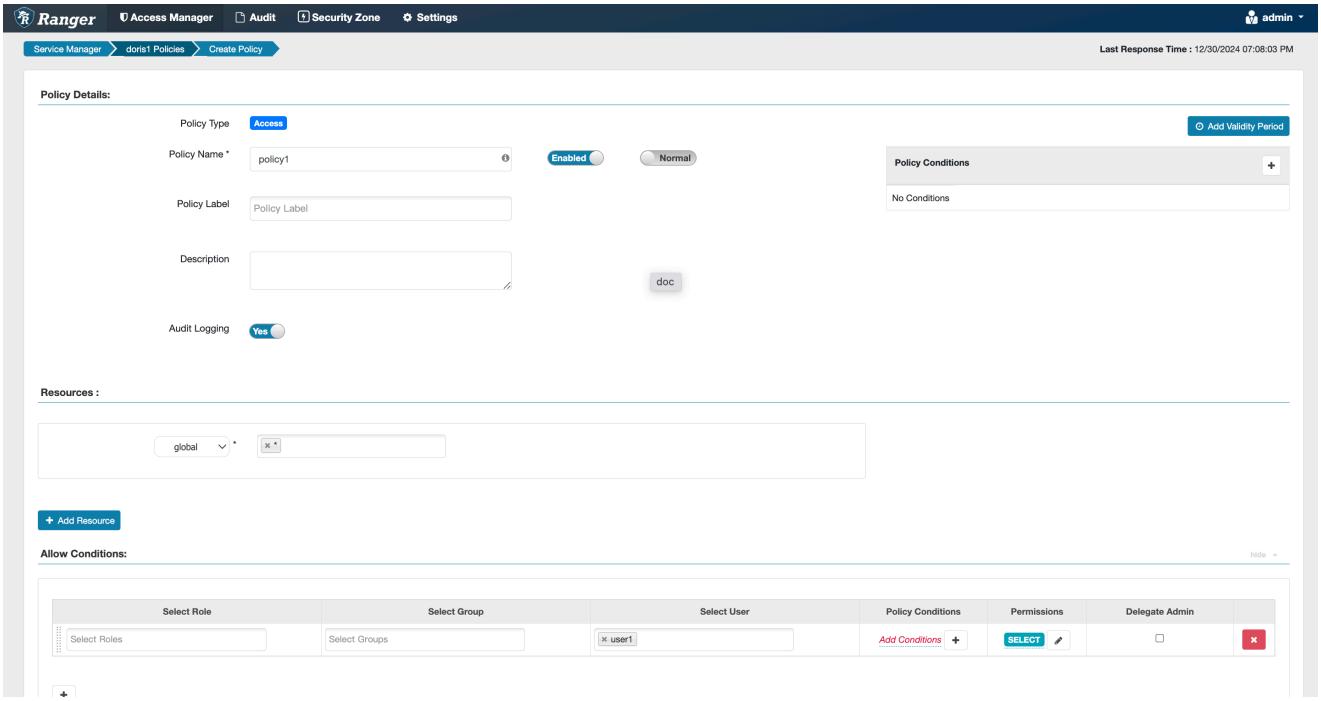
3. Start the cluster.

Permission Example

1. Create `user1` in Doris.
2. In Doris, first use the `admin` user to create a Catalog: `hive`.
3. Create `user1` in Ranger.

Global Permissions

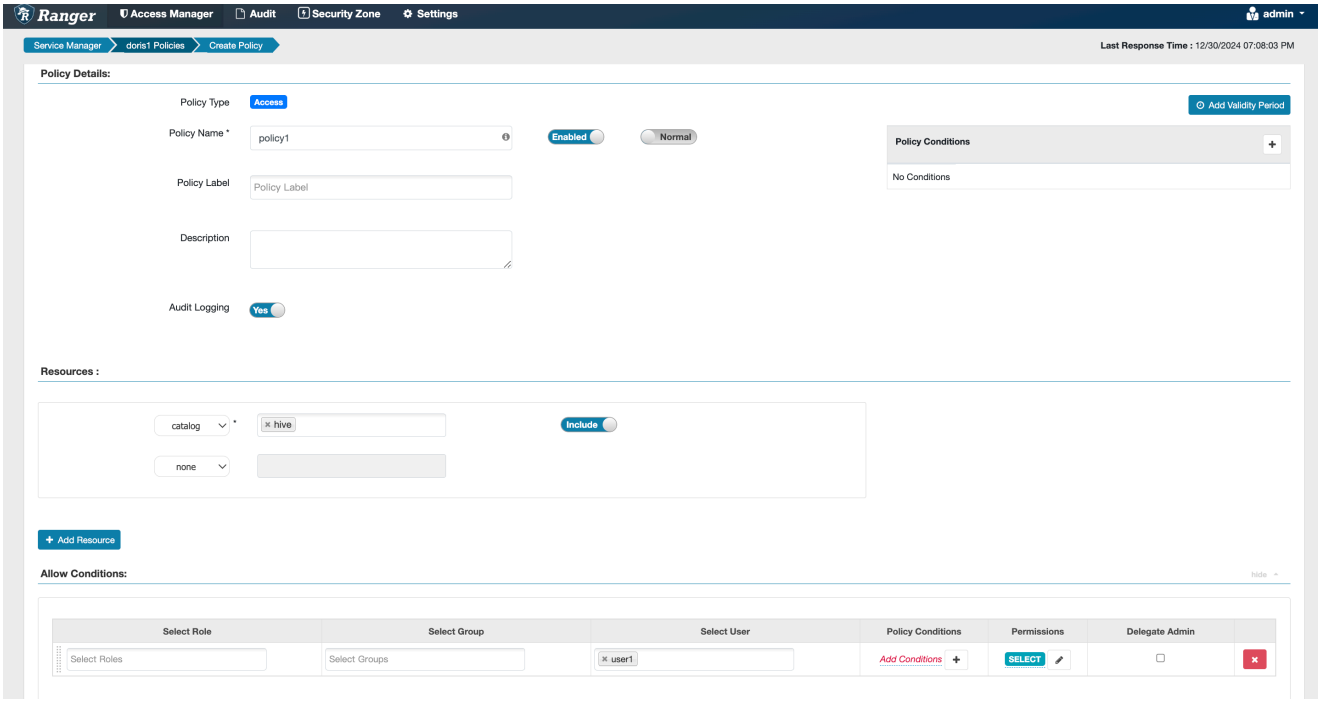
Equivalent to the internal Doris authorization statement `grant select_priv on *.*.* to user1`; - The global option can be found in the dropdown box at the same level as the catalog. - Only * can be entered in the input box.



global

Catalog Permissions

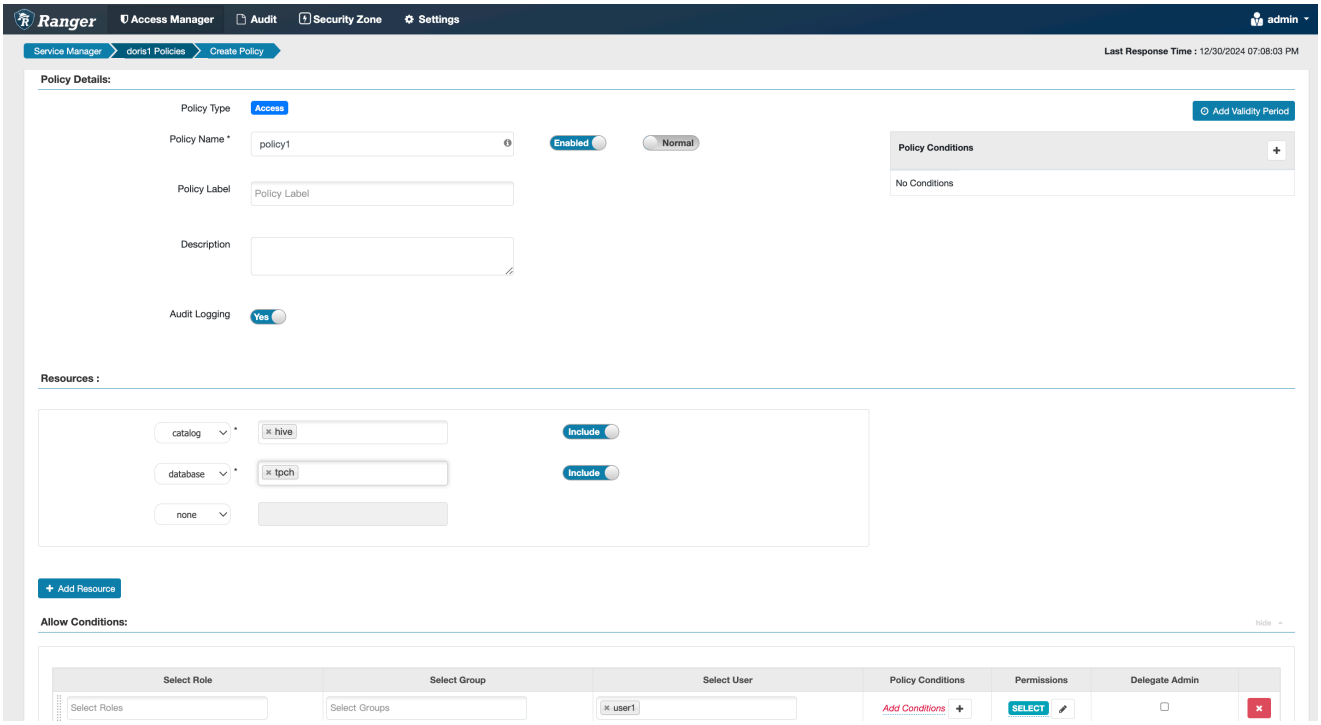
Equivalent to the internal Doris authorization statement `grant select_priv on hive.*.* to user1`;



catalog

Database Permissions

Equivalent to the internal Doris authorization statement `grant select_priv on hive.db1.* to user1;`

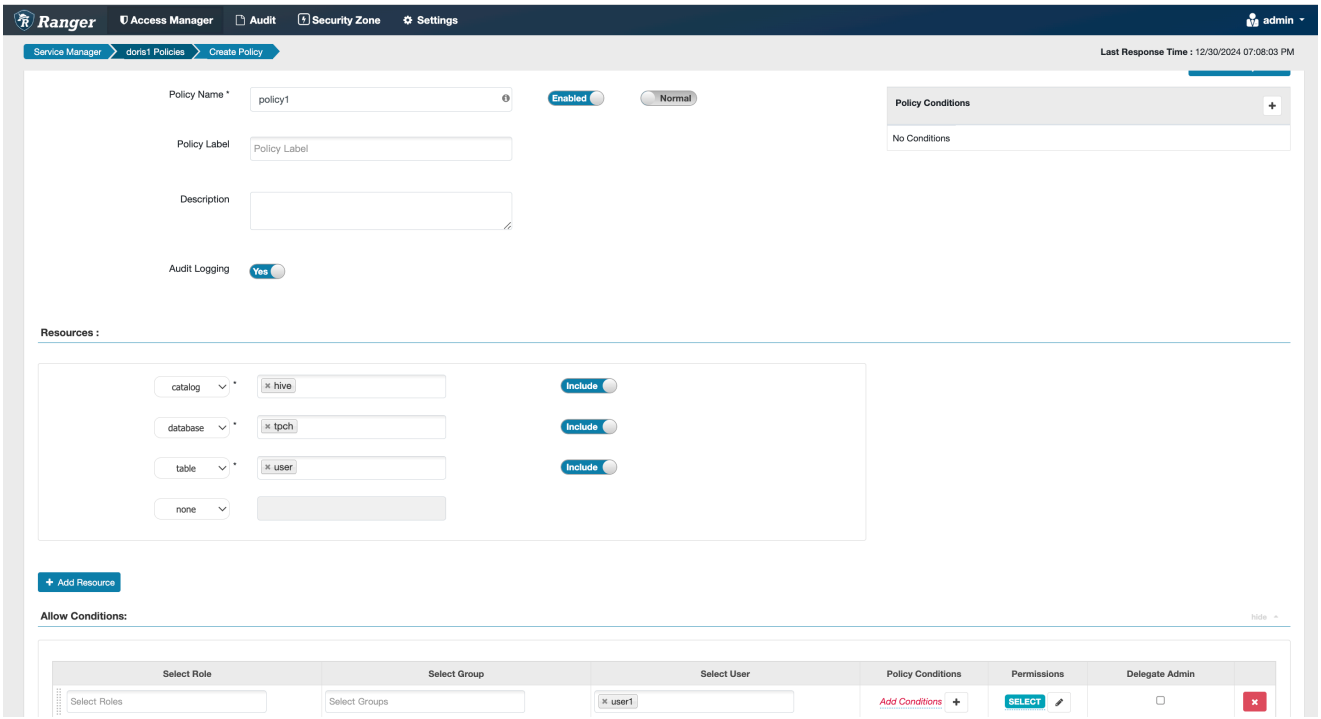


database

Table Permissions

> Here, the term “table” generally refers to tables, views, and asynchronous materialized views.

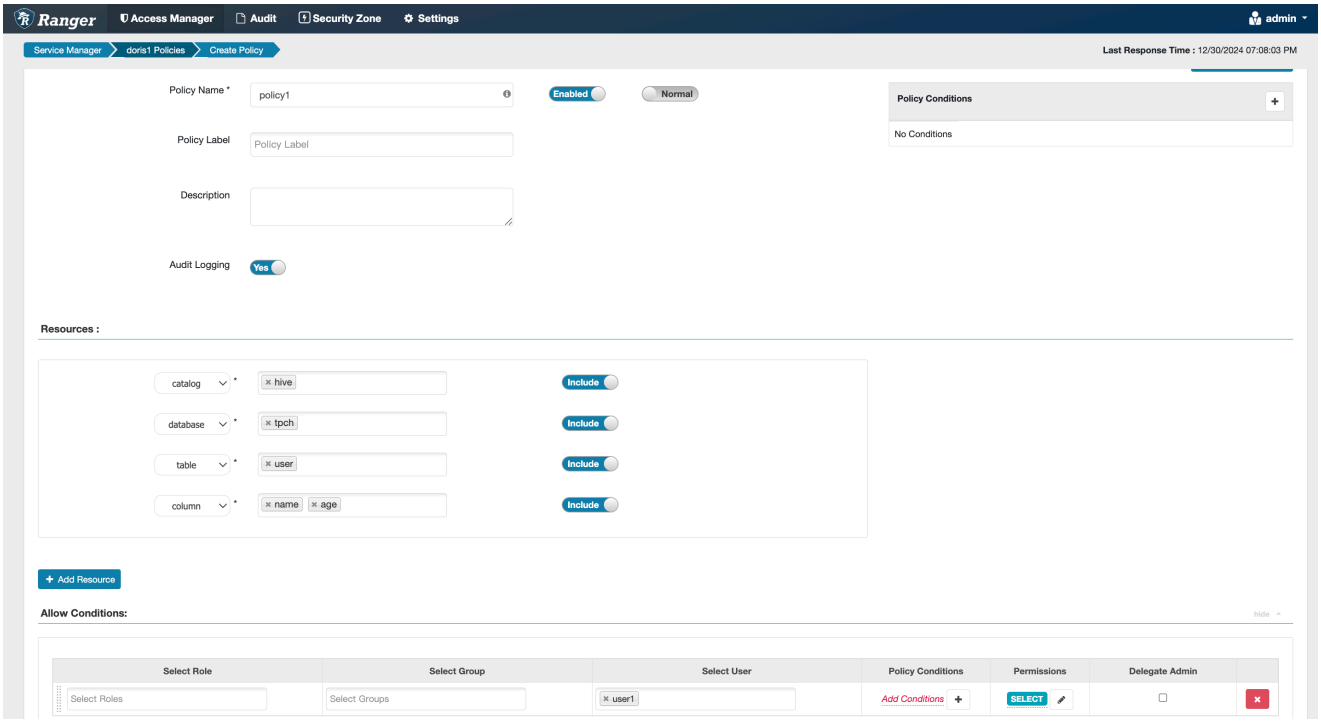
Equivalent to the internal Doris authorization statement `grant select_priv on hive.db1.tbl1 to user1;`



table

Column Permissions

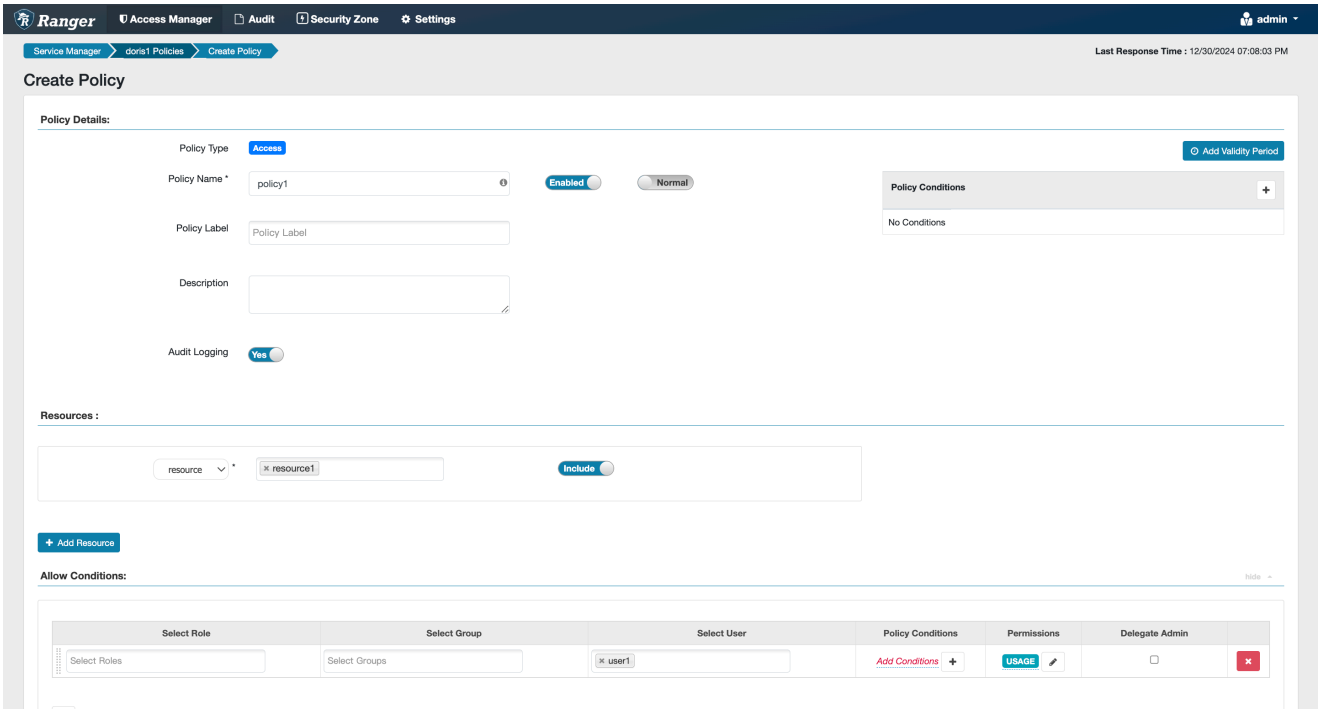
Equivalent to the internal Doris authorization statement `grant select_priv(col1,col2) on hive.db1.tbl1 to user1;`



column

Resource Permissions

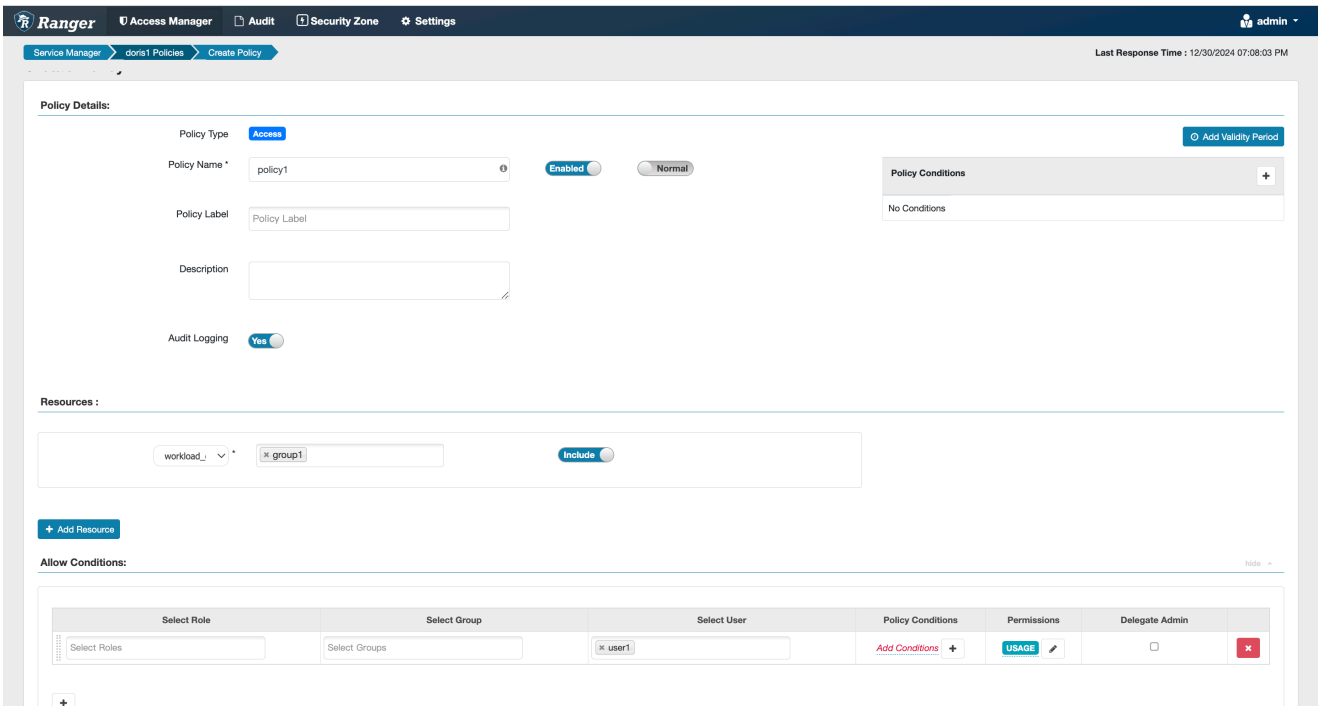
Equivalent to the internal Doris authorization statement `grant usage_priv on resource 'resource1' to user1;` - The resource option can be found in the dropdown box at the same level as the catalog.



resource

Workload Group Permissions

Equivalent to the internal Doris authorization statement `grant usage_priv on workload group 'group1' to user1;` - The workload group option can be found in the dropdown box at the same level as the catalog.



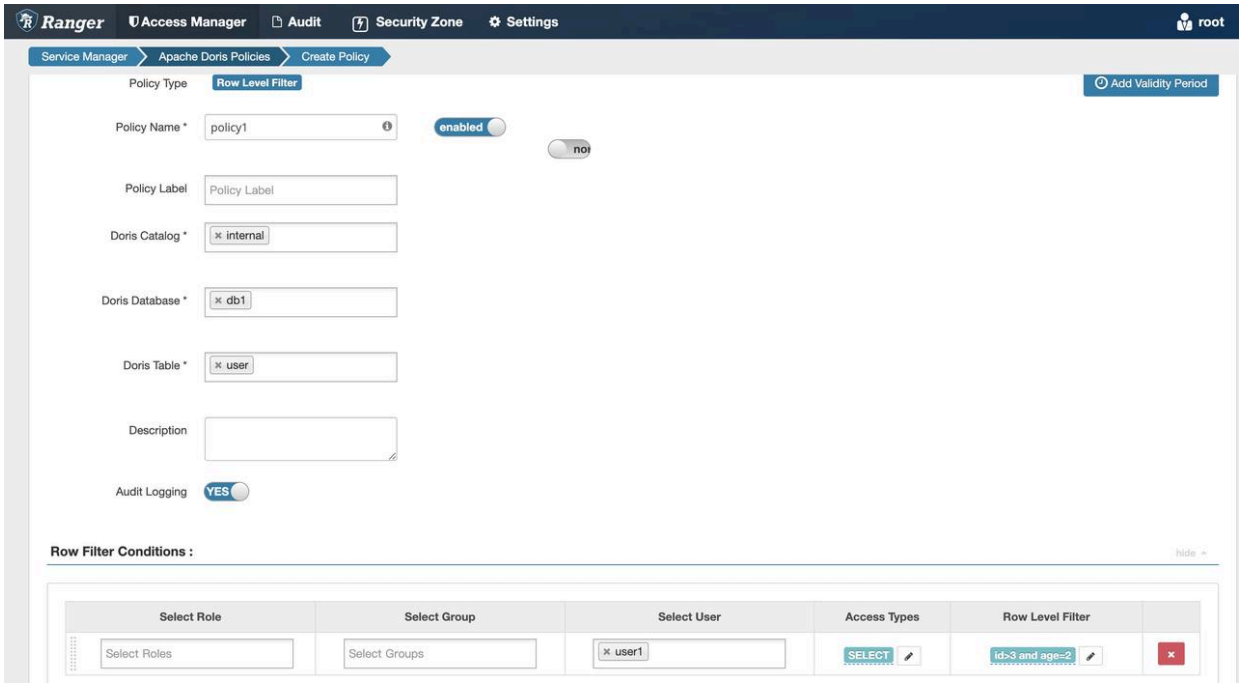
group1

Row-Level Permissions Example

> Supported in version 2.1.3

1. Refer to the permission example to grant `user1` the select permission on the `internal.db1.user` table.

2. In Ranger, add a Row Level Filter policy



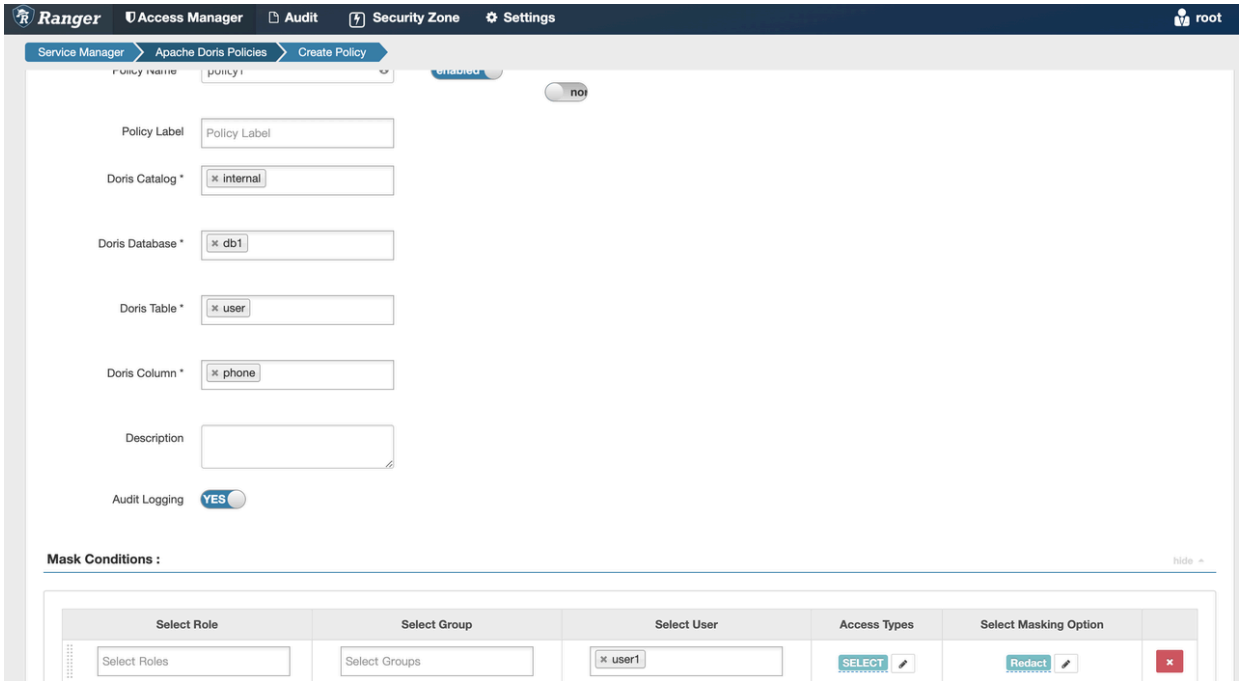
Row Policy Example

3. Log in to Doris with `user1`. Execute `select * from internal.db1.user`, and only see the data that meets the condition `id > 3` and `age = 2`.

Data Masking Example

> Supported in version 2.1.3

1. Refer to the permission example to grant `user1` the select permission on the `internal.db1.user` table.
2. In Ranger, add a Masking policy



Data Mask Example

3. Log in to Doris with `user1`. Execute `select * from internal.db1.user`, and see the phone number is masked according to the specified rule.

Frequently Asked Questions

1. How to view the log when Ranger access fails?

   Create a `log4j.properties` file in the `conf` directory of all FEs, with the following content:

   ```
   log4j.rootLogger = warn,stdout,D

   log4j.appender.stdout = org.apache.log4j.ConsoleAppender
   log4j.appender.stdout.Target = System.out
   log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
   log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n

   log4j.appender.D = org.apache.log4j.DailyRollingFileAppender
   log4j.appender.D.File = /path/to/fe/log/ranger.log
   log4j.appender.D.Append = true
   log4j.appender.D.Threshold = INFO
   log4j.appender.D.layout = org.apache.log4j.PatternLayout
   log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss}  [ %t:%r ] - [ %p ]  %m%n
   ```

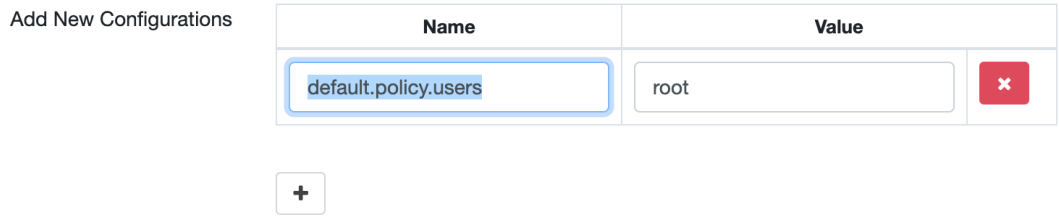
   Change `log4j.appender.D.File` to the actual path, which is used to store the Ranger plugin log.

2. A Row Level Filter policy has been configured, but the user encounters a permission denied error when querying.

   The Row Level Filter policy is solely used to restrict users from accessing specific records within a table's data; authorization for the user must still be granted through an ACCESS POLICY.

3. After creating the service, only the 'admin' user has permission by default` Root 'user does not have permission

As shown in the image, when creating the service, add the configuration `default.policy.users`. If you need to configure multiple users with full permissions, separate them with `,`.



4. After using Ranger for authentication, is internal authorization still effective?

   Currently, Ranger does not support configuring global-level permissions, so internal authorization like `GRANT priv ON *.*.*` remains effective, while permissions at other levels are not functional.

Install and Configure Doris Ranger Plugin

Install Plugin

1. Download the following files

   - ranger-doris-plugin-3.0.0-SNAPSHOT.jar
   - mysql-connector-java-8.0.25.jar

2. Place the downloaded files in the `ranger-plugins/doris` directory of the Ranger service, such as:

```
/usr/local/service/ranger/ews/webapp/WEB-INF/classes/ranger-plugins/doris/ranger-doris-plugin-3.0.0-SNAPSHOT.jar
/usr/local/service/ranger/ews/webapp/WEB-INF/classes/ranger-plugins/doris/mysql-connector-java-8.0.25.jar
```

3. Restart the Ranger service.

4. Download ranger-servicedef-doris.json

5. Execute the following command to upload the definition file to the Ranger service:

```
curl -u user:password -X POST \
    -H "Accept: application/json" \
    -H "Content-Type: application/json" \
    http://172.21.0.32:6080/service/plugins/definitions \
    -d@ranger-servicedef-doris.json
```

   Replace the username and password with the actual login credentials for the Ranger WebUI.

   The service address and port can be found in the `ranger-admin-site.xml` configuration file, in the `ranger.service.http.port` configuration item.

   If the execution is successful, a JSON-formatted service definition will be returned, such as:

```
{
  "id": 207,
  "guid": "d3ff9e41-f9dd-4217-bb5f-3fa9996454b6",
```

```
  "isEnabled": true,
  "createdBy": "Admin",
  "updatedBy": "Admin",
  "createTime": 1705817398112,
  "updateTime": 1705817398112,
  "version": 1,
  "name": "doris",
  "displayName": "Apache Doris",
  "implClass": "org.apache.ranger.services.doris.RangerServiceDoris",
  "label": "Doris",
  "description": "Apache Doris",
  "options": {
    "enableDenyAndExceptionsInPolicies": "true"
  },
  ...
}
```

If you want to recreate the service definition, you can use the following command to delete the service definition and then re-upload it:

```
curl -v -u user:password -X DELETE \
http://172.21.0.32:6080/service/plugins/definitions/207
```

Replace `207` with the actual ID returned when creating the service definition.

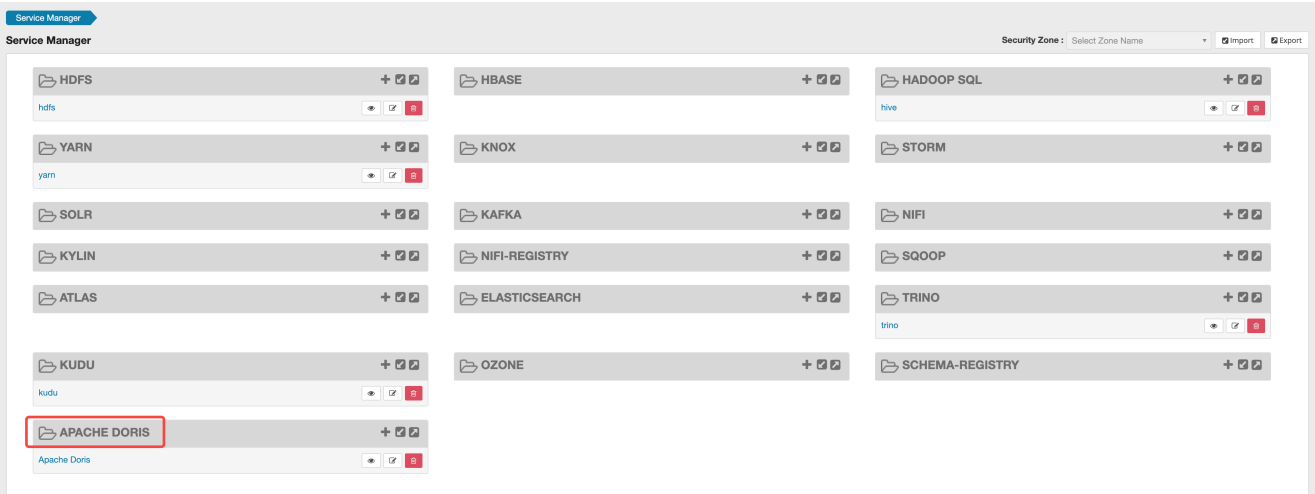
Before deleting, you need to delete the Doris service created in the Ranger WebUI.

You can also use the following command to list the current service definitions and get the ID:

```
curl -v -u user:password -X GET \
http://172.21.0.32:6080/service/plugins/definitions/
```

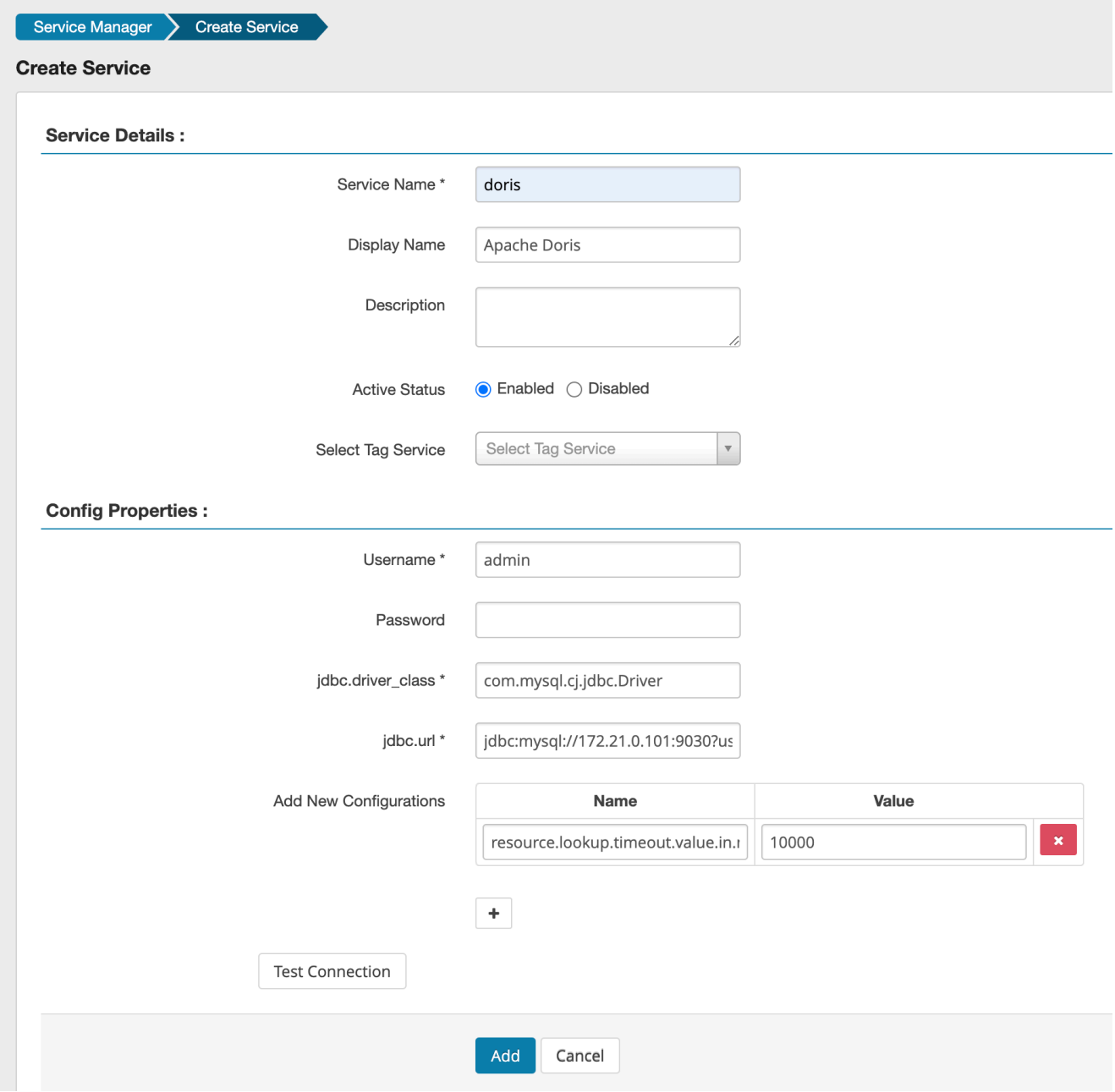
Configure Plugin

After installation, open the Ranger WebUI, and you can see the Apache Doris plugin in the Service Manager interface:



ranger

Click the + button next to the plugin to add a Doris service:



ranger2

The Config Properties section has the following parameters:

- `Username/Password`: The username and password of the Doris cluster. It is recommended to use the Admin user.
- `jdbc.driver_class`: The JDBC driver used to connect to Doris. `com.mysql.cj.jdbc.Driver`
- `jdbc.url`: The JDBC URL connection string of the Doris cluster. `jdbc:mysql://172.21.0.101:9030?useSSL=false`
- Additional parameters:
  - `resource.lookup.timeout.value.in.ms`: The timeout for getting metadata, recommended to set to `10000`, which is 10 seconds.

You can click `Test Connection` to check if the connection is successful.

After clicking `Add`, you can see the created service in the Service Manager interface of the Apache Doris plugin. Click the service to start configuring Ranger.

Data Access Control

Row Permissions

With row-level policies in Doris, you can achieve fine-grained access control over sensitive data. You can decide which users or roles can access specific records in a table based on security policies defined at the table level.

Mechanism

Equivalent to automatically adding the predicate set in the Row Policy for users configured with Row Policy when querying.

Limitations

Row Policy cannot be set for default users root and admin.

Related Commands

- View Row Permission Policies SHOW ROW POLICY
- Create Row Permission Policy CREATE ROW POLICY

Row Permission Example

1. Restrict the test user to only query data in table1 where c1='a'

```
CREATE ROW POLICY test_row_policy_1 ON test.table1
AS RESTRICTIVE TO test USING (c1 = 'a');
```

Column Permissions

With column permissions in Doris, you can achieve fine-grained access control over tables. You can grant permissions to specific columns in a table to decide which users or roles can access specific columns in a table.

Currently, column permissions only support Select_priv.

Related Commands

- Grant: GRANT
- Revoke: REVOKE

Column Permission Example

1. Grant user1 the permission to query columns col1 and col2 in table tbl.

```
GRANT Select_priv(col1,col2) ON ctl.db.tbl TO user1
```

Data Masking

Data masking is a method to protect sensitive data by modifying, replacing, or hiding the original data, making the masked data no longer contain sensitive information while maintaining certain formats and characteristics.

For example, administrators can choose to replace part or all of the digits of sensitive fields such as credit card numbers or ID numbers with asterisks * or other characters, or replace real names with pseudonyms.

Starting from version 2.1.2, data masking is supported through Apache Ranger's Data Masking to set masking policies for certain columns, currently only through Apache Ranger.

> Data Masking settings for admin/root users will not take effect.

Audit Log

Doris provides auditing capabilities for database operations, allowing the recording of user logins, queries, and modification operations on the database. In Doris, audit logs can be queried directly through built-in system tables or by viewing Doris's audit log files.

Enabling Audit Logs

The audit log plugin can be enabled or disabled at any time using the global variable `enable_audit_plugin` (disabled by default), for example:

```
set global enable_audit_plugin = true;
```

Once enabled, Doris will write the audit logs to the `audit_log` table.

You can disable the audit log plugin at any time:

```
set global enable_audit_plugin = false;
```

After disabling, Doris will stop writing to the `audit_log` table. The already written audit logs will remain unchanged.

Viewing the Audit Log Table

> **Note**
>
> Before version 2.1.8, as the system version was upgraded, the audit log table fields may have increased. After upgrading, you need to add fields to the `audit_log` table using the `ALTER TABLE` command based on the fields in the audit log table.

Starting from Doris version 2.1, Doris can write user behavior operations to the `audit_log` table in the `__internal_schema` database by enabling the audit log feature.

The audit log table is a dynamically partitioned table, partitioned daily by default, retaining the most recent 30 days of data. You can adjust the retention period of dynamic partitions by modifying the `dynamic_partition.start` property using the `ALTER TABLE` statement.

Audit Log Files

In `fe.conf`, `LOG_DIR` defines the storage path for FE logs. All database operations executed by this FE node are recorded in `${LOG_DIR}/fe.audit.log`. To view all operations in the cluster, you need to traverse the audit logs of each FE node.

Audit Log Configuration

Global Variables:

Audit log variables can be modified using `set [global] <var_name> = <var_value>`.

Variable	Default Value	Description
`audit_plugin_max_batch_interval_sec`	60 seconds	Maximum write interval for the audit log table.
`audit_plugin_max_batch_bytes`	50MB	Maximum data volume per batch for the audit log table.
`audit_plugin_max_sql_length`	4096	Maximum length of SQL statements recorded in the audit log table.
`audit_plugin_load_timeout`	600 seconds	Default timeout for audit log import jobs.

FE Configuration Items:

FE configuration items can be modified by editing the `fe.conf` directory.

Configuration Item	Description
`skip_audit_user_list`	If you do not want operations of certain users to be recorded in the audit logs, you can modify this configuration (supported since version 3.0.01). For example, use the config to exclude `user1` and `user2` from audit log recording: `skip_audit_user_list=user1,user2`

MySQL Client Certificate

Communicate with the server over an encrypted connection

Doris supports SSL-based encrypted connections. It currently supports TLS1.2 and TLS1.3 protocols. Doris' SSL mode can be enabled through the following configuration: Modify the FE configuration file `conf/fe.conf` and add `enable_ssl = true`.

Next, connect to Doris through `mysql` client, mysql supports three SSL modes:

1. `mysql -uroot -P9030 -h127.0.0.1` is the same as `mysql --ssl-mode=PREFERRED -uroot -P9030 -h127.0.0.1`, both try to establish an SSL encrypted connection at the beginning, if it fails , a normal connection is attempted.
2. `mysql --ssl-mode=DISABLE -uroot -P9030 -h127.0.0.1`, do not use SSL encrypted connection, use normal connection directly.
3. `mysql --ssl-mode=REQUIRED -uroot -P9030 -h127.0.0.1`, force the use of SSL encrypted connections.

> Note: `--ssl-mode` parameter is introduced by mysql5.7.11 version, please refer to here for mysql client version lower than this version。 Doris needs a key certificate file to verify the SSL encrypted connection. The default key certificate file is located at `Doris/fe/mysql_ssl_default_certificate/certificate.p12`, and the default password is `doris`. You can modify the FE configuration file `conf/fe. conf`, add `mysql_ssl_default_certificate = /path/to/your/certificate` to modify the key certificate file, and you can also add the password corresponding to your custom key book file through `mysql_ssl_default_certificate_password = your_password`.

Doris also supports mTLS: Modify the FE configuration file `conf/fe.conf` and add `ssl_force_client_auth=true`.

Then you can connect to Doris via the `mysql` client:

```
mysql -ssl-mode=VERIFY_CA -uroot -P9030 -h127.0.0.1 --tls-version=TLSv1.2 --ssl-ca=/path/to/your/ca --ssl-cert=/path/to/your/cert --ssl-key=/path/to/your/key
```

The default ca, cert, and key files are located in `Doris/conf/mysql_ssl_default_certificate/client_certificate/`, named `ca.pem`, `client-cert.pem`, and `client-key.pem` respectively.

You can also generate your own certificate files using openssl or keytool.

Key Certificate Configuration

Enabling SSL functionality in Doris requires configuring both a CA key certificate and a server-side key certificate. To enable mutual authentication, a client-side key certificate must also be generated:

- The default CA key certificate file is located at `Doris/fe/mysql_ssl_default_certificate/ca_certificate.p12`, with a default password of `doris`. You can

modify the FE configuration file `conf/fe.conf` to add `mysql_ssl_default_ca_certificate = /path/to/your/certificate` to change the CA key certificate file. You can also add `mysql_ssl_default_ca_certificate_password = your_password` to specify the password for your custom key certificate file.

- The default server-side key certificate file is located at `Doris/fe/mysql_ssl_default_certificate/server_certificate.p12`, with a default password of `doris`. You can modify the FE configuration file `conf/fe.conf` to add `mysql_ssl_default_server_certificate = /path/to/your/certificate` to change the server-side key certificate file. You can also add `mysql_ssl_default_server_certificate_password = your_password` to specify the password for your custom key certificate file.
- By default, a client-side key certificate is also generated and stored in `Doris/fe/mysql_ssl_default_certificate/client-key.pem` and `Doris/fe/mysql_ssl_default_certificate/client_certificate/`.

Custom key certificate file

In addition to the Doris default certificate file, you can also generate a custom certificate file through `openssl`. Here are the steps (refer to Creating SSL Certificates and Keys Using OpenSSL):

1. Generate the CA, server-side, and client-side keys and certificates:

```
# Generate the CA certificate
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600 \
        -key ca-key.pem -out ca.pem

# Generate the server certificate and sign it with the above CA
# server-cert.pem = public key, server-key.pem = private key
openssl req -newkey rsa:2048 -days 3600 \
        -nodes -keyout server-key.pem -out server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
openssl x509 -req -in server-req.pem -days 3600 \
        -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem

# Generate the client certificate and sign it with the above CA
# client-cert.pem = public key, client-key.pem = private key
openssl req -newkey rsa:2048 -days 3600 \
        -nodes -keyout client-key.pem -out client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -req -in client-req.pem -days 3600 \
        -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

2. Verify the created certificates:

```
openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

3. Combine your key and certificate in a PKCS#12 (P12) bundle. You can also specify a certificate format (PKCS12 by default). You can modify the conf/fe.conf configuration file and add parameter ssl_trust_store_type to specify the certificate format.

```
# Package the CA key and certificate
openssl pkcs12 -inkey ca-key.pem -in ca.pem -export -out ca_certificate.p12

# Package the server-side key and certificate
openssl pkcs12 -inkey server-key.pem -in server-cert.pem -export -out server_certificate.p12
```

> **Note**
>
> reference documents

FE SSL Certificate

To enable SSL function on Doris FE interface, you need to configure key certificate as follows:

1.Purchase or generate a self-signed SSL certificate. It is advised to use CA certificate in Production environment

2.Copy the SSL certificate to specified path. The default path is `${DORIS_HOME}/conf/ssl/`, and user can also specify their own path

3.Modify FE configuration file `conf/fe.conf`, and note that the following parameters are consistent with purchased or generated SSL certificate Set `enable_https = true` to enable https function, default is `false` Set certificate path `key_store_path`, default is `${DORIS_HOME}/conf/ssl/doris_ssl_certificate.keystore` Set certificate password `key_store_password`, default is null Set certificate type `key_store_type`, default is `JKS` Set certificate alias `key_store_alias`, default is `doris_ssl_certificate`

Encryption and Masking Function

Doris provides the following built-in encryption and masking functions. For detailed usage, please refer to the SQL manual.

- AES_ENCRYPT
- AES_DECRYPT
- SM4_ENCRYPT
- SM4_DECRYPT
- MD5
- MD5SUM
- SM3
- SM3SUM
- SHA
- SHA2
- DIGITAL_MASKING

Star Schema Benchmark

Star Schema Benchmark

Star Schema Benchmark(SSB) is a lightweight performance test set in the data warehouse scenario. SSB provides a simplified star schema data based on TPC-H, which is mainly used to test the performance of multi-table JOIN query under star schema. In addition, the industry usually flattens SSB into a wide table model (Referred as: SSB flat) to test the performance of the query engine.

This document mainly introduces the performance of Doris on the SSB 1000G test set.

We tested 13 queries on the SSB standard test dataset based on Apache Doris version 2.0.15.1.

1. Hardware Environment

Hardware	Configuration Instructions
Number of Machines	4 Aliyun Virtual Machine (1FE，3BEs)
CPU	Intel Xeon (Ice Lake) Platinum 8369B 32C
Memory	128G
Disk	Enterprise SSD (PL0)

2. Software Environment

- Doris Deployed 3BEs and 1FE
- Kernel Version: Linux version 5.15.0-101-generic
- OS version: Ubuntu 20.04 LTS (Focal Fossa)
- Doris software version: Apache Doris 2.0.15.1
- JDK: openjdk version “1.8.0_352-352”

3. Test Data Volume

SSB Table Name	Rows	Annotation
lineorder	5,999,989,709	Commodity Order Details
customer	30,000,000	Customer Information
part	2,000,000	Parts Information
supplier	2,000,000	Supplier Information
dates	2,556	Date
lineorder_flat	5,999,989,709	Wide Table after Data Flattening

4. SSB Flat Test Results

Here we use Apache Doris 2.0.15.1 for comparative testing. In the test, we use Query Time(ms) as the main performance indicator. The test results are as follows:

Query	Doris 2.0.15.1 (ms)
q1.1	80
q1.2	10
q1.3	110
q2.1	1680
q2.2	1210
q2.3	1060
q3.1	2010
q3.2	1560
q3.3	600
q3.4	10
q4.1	2380
q4.2	190
q4.3	120
Total	**11020**

5. Standard SSB Test Results

Here we use Apache Doris 2.0.15.1 for comparative testing. In the test, we use Query Time(ms) as the main performance indicator. The test results are as follows:

Query	Doris 2.0.15.1 (ms)
q1.1	330
q1.2	80
q1.3	80
q2.1	1780
q2.2	1970
q2.3	1510
q3.1	4000
q3.2	1720
q3.3	1510
q3.4	160
q4.1	4010

Query	Doris 2.0.15.1 (ms)
q4.2	840
q4.3	400
Total	**19390**

6. Environment Preparation

Please first refer to the official documentation to install and deploy Apache Doris first to obtain a Doris cluster which is working well(including at least 1 FE 1 BE, 1 FE 3 BEs is recommended).

7. Data Preparation

7.1 Download and Install the SSB Data Generation Tool.

Execute the following script to download and compile the ssb-tools tool.

```
sh bin/build-ssb-dbgen.sh
```

After successful installation, the `dbgen` binary will be generated under the `ssb-dbgen/` directory.

7.2 Generate SSB Test Set

Execute the following script to generate the SSB dataset:

```
sh bin/gen-ssb-data.sh -s 1000
```

> Note 1: Check the script help via `sh gen-ssb-data.sh -h`.
>
> Note 2: The data will be generated under the `ssb-data/` directory with the suffix `.tbl`. The total file size is about 600GB and may need a few minutes to an hour to generate.
>
> Note 3: A standard test data set of 100G is generated by default.

7.3 Create Table

7.3.1 Prepare the `doris-cluster.conf` File.

Before import the script, you need to write the FE's ip port and other information in the `doris-cluster.conf` file.

The file is located under `${DORIS_HOME}/tools/ssb-tools/conf/`.

The content of the file includes FE's ip, HTTP port, user name, password and the DB name of the data to be imported:

```
# Any of FE host
export FE_HOST='127.0.0.1'
# http_port in fe.conf
export FE_HTTP_PORT=8030
# query_port in fe.conf
export FE_QUERY_PORT=9030
# Doris username
export USER='root'
# Doris password
export PASSWORD=''
# The database where SSB tables located
export DB='ssb'
```

7.3.2 Execute the Following Script to Generate and Create the SSB Table:

```
sh bin/create-ssb-tables.sh -s 1000
```

Or copy the table creation statements in create-ssb-tables.sql and create-ssb-flat-table.sql and then execute them in the MySQL client.

7.4 Import data

We use the following command to complete all data import of SSB test set and SSB FLAT wide table data synthesis and then import into the table.

```
 sh bin/load-ssb-data.sh
```

7.5 Checking Imported data

```
select count(*) from part;
select count(*) from customer;
select count(*) from supplier;
select count(*) from dates;
select count(*) from lineorder;
select count(*) from lineorder_flat;
```

7.6 Query Test

- SSB-Flat Query Statement: ssb-flat-queries
- Standard SSB Queries: ssb-queries

7.6.1 SSB FLAT Test for SQL

```
--Q1.1
SELECT SUM(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue
FROM lineorder_flat
WHERE
    LO_ORDERDATE >= 19930101
    AND LO_ORDERDATE <= 19931231
    AND LO_DISCOUNT BETWEEN 1 AND 3
```

```
    AND LO_QUANTITY < 25;

--Q1.2
SELECT SUM(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue
FROM lineorder_flat
WHERE
    LO_ORDERDATE >= 19940101
  AND LO_ORDERDATE <= 19940131
  AND LO_DISCOUNT BETWEEN 4 AND 6
  AND LO_QUANTITY BETWEEN 26 AND 35;

--Q1.3
SELECT SUM(LO_EXTENDEDPRICE * LO_DISCOUNT) AS revenue
FROM lineorder_flat
WHERE
    weekofyear(LO_ORDERDATE) = 6
  AND LO_ORDERDATE >= 19940101
  AND LO_ORDERDATE <= 19941231
  AND LO_DISCOUNT BETWEEN 5 AND 7
  AND LO_QUANTITY BETWEEN 26 AND 35;

--Q2.1
SELECT
    SUM(LO_REVENUE), (LO_ORDERDATE DIV 10000) AS YEAR,
    P_BRAND
FROM lineorder_flat
WHERE P_CATEGORY = 'MFGR#12' AND S_REGION = 'AMERICA'
GROUP BY YEAR, P_BRAND
ORDER BY YEAR, P_BRAND;

--Q2.2
SELECT
    SUM(LO_REVENUE), (LO_ORDERDATE DIV 10000) AS YEAR,
    P_BRAND
FROM lineorder_flat
WHERE
    P_BRAND >= 'MFGR#2221'
  AND P_BRAND <= 'MFGR#2228'
  AND S_REGION = 'ASIA'
GROUP BY YEAR, P_BRAND
ORDER BY YEAR, P_BRAND;

--Q2.3
SELECT
    SUM(LO_REVENUE), (LO_ORDERDATE DIV 10000) AS YEAR,
    P_BRAND
FROM lineorder_flat
WHERE
    P_BRAND = 'MFGR#2239'
  AND S_REGION = 'EUROPE'
GROUP BY YEAR, P_BRAND
ORDER BY YEAR, P_BRAND;

--Q3.1
SELECT
    C_NATION,
    S_NATION, (LO_ORDERDATE DIV 10000) AS YEAR,
    SUM(LO_REVENUE) AS revenue
FROM lineorder_flat
WHERE
    C_REGION = 'ASIA'
  AND S_REGION = 'ASIA'
  AND LO_ORDERDATE >= 19920101
  AND LO_ORDERDATE <= 19971231
GROUP BY C_NATION, S_NATION, YEAR
```

```
ORDER BY YEAR ASC, revenue DESC;

--Q3.2
SELECT
    C_CITY,
    S_CITY, (LO_ORDERDATE DIV 10000) AS YEAR,
    SUM(LO_REVENUE) AS revenue
FROM lineorder_flat
WHERE
    C_NATION = 'UNITED STATES'
  AND S_NATION = 'UNITED STATES'
  AND LO_ORDERDATE >= 19920101
  AND LO_ORDERDATE <= 19971231
GROUP BY C_CITY, S_CITY, YEAR
ORDER BY YEAR ASC, revenue DESC;

--Q3.3
SELECT
    C_CITY,
    S_CITY, (LO_ORDERDATE DIV 10000) AS YEAR,
    SUM(LO_REVENUE) AS revenue
FROM lineorder_flat
WHERE
    C_CITY IN ('UNITED KI1', 'UNITED KI5')
  AND S_CITY IN ('UNITED KI1', 'UNITED KI5')
  AND LO_ORDERDATE >= 19920101
  AND LO_ORDERDATE <= 19971231
GROUP BY C_CITY, S_CITY, YEAR
ORDER BY YEAR ASC, revenue DESC;

--Q3.4
SELECT
    C_CITY,
    S_CITY, (LO_ORDERDATE DIV 10000) AS YEAR,
    SUM(LO_REVENUE) AS revenue
FROM lineorder_flat
WHERE
    C_CITY IN ('UNITED KI1', 'UNITED KI5')
  AND S_CITY IN ('UNITED KI1', 'UNITED KI5')
  AND LO_ORDERDATE >= 19971201
  AND LO_ORDERDATE <= 19971231
GROUP BY C_CITY, S_CITY, YEAR
ORDER BY YEAR ASC, revenue DESC;

--Q4.1
SELECT (LO_ORDERDATE DIV 10000) AS YEAR,
    C_NATION,
    SUM(LO_REVENUE - LO_SUPPLYCOST) AS profit
FROM lineorder_flat
WHERE
    C_REGION = 'AMERICA'
  AND S_REGION = 'AMERICA'
  AND P_MFGR IN ('MFGR#1', 'MFGR#2')
GROUP BY YEAR, C_NATION
ORDER BY YEAR ASC, C_NATION ASC;

--Q4.2
SELECT (LO_ORDERDATE DIV 10000) AS YEAR,
    S_NATION,
    P_CATEGORY,
    SUM(LO_REVENUE - LO_SUPPLYCOST) AS profit
FROM lineorder_flat
WHERE
    C_REGION = 'AMERICA'
  AND S_REGION = 'AMERICA'
```

```
  AND LO_ORDERDATE >= 19970101
  AND LO_ORDERDATE <= 19981231
  AND P_MFGR IN ('MFGR#1', 'MFGR#2')
GROUP BY YEAR, S_NATION, P_CATEGORY
ORDER BY
    YEAR ASC,
    S_NATION ASC,
    P_CATEGORY ASC;

--Q4.3
SELECT (LO_ORDERDATE DIV 10000) AS YEAR,
    S_CITY,
    P_BRAND,
    SUM(LO_REVENUE - LO_SUPPLYCOST) AS profit
FROM lineorder_flat
WHERE
    S_NATION = 'UNITED STATES'
  AND LO_ORDERDATE >= 19970101
  AND LO_ORDERDATE <= 19981231
  AND P_CATEGORY = 'MFGR#14'
GROUP BY YEAR, S_CITY, P_BRAND
ORDER BY YEAR ASC, S_CITY ASC, P_BRAND ASC;
```

7.6.2 SSB Standard Test for SQL

```
--Q1.1
SELECT SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM lineorder, dates
WHERE
    lo_orderdate = d_datekey
  AND d_year = 1993
  AND lo_discount BETWEEN 1 AND 3
  AND lo_quantity < 25;

--Q1.2
SELECT SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM lineorder, dates
WHERE
    lo_orderdate = d_datekey
  AND d_yearmonth = 'Jan1994'
  AND lo_discount BETWEEN 4 AND 6
  AND lo_quantity BETWEEN 26 AND 35;

--Q1.3
SELECT
    SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM lineorder, dates
WHERE
    lo_orderdate = d_datekey
  AND d_weeknuminyear = 6
  AND d_year = 1994
  AND lo_discount BETWEEN 5 AND 7
  AND lo_quantity BETWEEN 26 AND 35;

--Q2.1
SELECT SUM(lo_revenue), d_year, p_brand
FROM lineorder, dates, part, supplier
WHERE
    lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
  AND p_category = 'MFGR#12'
```

```
  AND s_region = 'AMERICA'
GROUP BY d_year, p_brand
ORDER BY p_brand;

--Q2.2
SELECT SUM(lo_revenue), d_year, p_brand
FROM lineorder, dates, part, supplier
WHERE
    lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
  AND p_brand BETWEEN 'MFGR#2221' AND 'MFGR#2228'
  AND s_region = 'ASIA'
GROUP BY d_year, p_brand
ORDER BY d_year, p_brand;

--Q2.3
SELECT SUM(lo_revenue), d_year, p_brand
FROM lineorder, dates, part, supplier
WHERE
    lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
  AND p_brand = 'MFGR#2239'
  AND s_region = 'EUROPE'
GROUP BY d_year, p_brand
ORDER BY d_year, p_brand;

--Q3.1
SELECT
    c_nation,
    s_nation,
    d_year,
    SUM(lo_revenue) AS REVENUE
FROM customer, lineorder, supplier, dates
WHERE
    lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_orderdate = d_datekey
  AND c_region = 'ASIA'
  AND s_region = 'ASIA'
  AND d_year >= 1992
  AND d_year <= 1997
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year ASC, REVENUE DESC;

--Q3.2
SELECT
    c_city,
    s_city,
    d_year,
    SUM(lo_revenue) AS REVENUE
FROM customer, lineorder, supplier, dates
WHERE
    lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_orderdate = d_datekey
  AND c_nation = 'UNITED STATES'
  AND s_nation = 'UNITED STATES'
  AND d_year >= 1992
  AND d_year <= 1997
GROUP BY c_city, s_city, d_year
ORDER BY d_year ASC, REVENUE DESC;

--Q3.3
```

```
SELECT
    c_city,
    s_city,
    d_year,
    SUM(lo_revenue) AS REVENUE
FROM customer, lineorder, supplier, dates
WHERE
    lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_orderdate = d_datekey
  AND (
           c_city = 'UNITED KI1'
        OR c_city = 'UNITED KI5'
    )
  AND (
           s_city = 'UNITED KI1'
        OR s_city = 'UNITED KI5'
    )
  AND d_year >= 1992
  AND d_year <= 1997
GROUP BY c_city, s_city, d_year
ORDER BY d_year ASC, REVENUE DESC;

--Q3.4
SELECT
    c_city,
    s_city,
    d_year,
    SUM(lo_revenue) AS REVENUE
FROM customer, lineorder, supplier, dates
WHERE
    lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_orderdate = d_datekey
  AND (
           c_city = 'UNITED KI1'
        OR c_city = 'UNITED KI5'
    )
  AND (
           s_city = 'UNITED KI1'
        OR s_city = 'UNITED KI5'
    )
  AND d_yearmonth = 'Dec1997'
GROUP BY c_city, s_city, d_year
ORDER BY d_year ASC, REVENUE DESC;

--Q4.1
SELECT
    d_year,
    c_nation,
    SUM(lo_revenue - lo_supplycost) AS PROFIT
FROM dates, customer, supplier, part, lineorder
WHERE
    lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_partkey = p_partkey
  AND lo_orderdate = d_datekey
  AND c_region = 'AMERICA'
  AND s_region = 'AMERICA'
  AND (
           p_mfgr = 'MFGR#1'
        OR p_mfgr = 'MFGR#2'
    )
GROUP BY d_year, c_nation
ORDER BY d_year, c_nation;
```

```
--Q4.2
SELECT
    d_year,
    s_nation,
    p_category,
    SUM(lo_revenue - lo_supplycost) AS PROFIT
FROM dates, customer, supplier, part, lineorder
WHERE
    lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_partkey = p_partkey
  AND lo_orderdate = d_datekey
  AND c_region = 'AMERICA'
  AND s_region = 'AMERICA'
  AND (
           d_year = 1997
       OR d_year = 1998
    )
  AND (
           p_mfgr = 'MFGR#1'
       OR p_mfgr = 'MFGR#2'
    )
GROUP BY d_year, s_nation, p_category
ORDER BY d_year, s_nation, p_category;

--Q4.3
SELECT
    d_year,
    s_city,
    p_brand,
    SUM(lo_revenue - lo_supplycost) AS PROFIT
FROM dates, customer, supplier, part, lineorder
WHERE
    lo_custkey = c_custkey
  AND lo_suppkey = s_suppkey
  AND lo_partkey = p_partkey
  AND lo_orderdate = d_datekey
  AND s_nation = 'UNITED STATES'
  AND (
           d_year = 1997
       OR d_year = 1998
    )
  AND p_category = 'MFGR#14'
GROUP BY d_year, s_city, p_brand
ORDER BY d_year, s_city, p_brand;
```

TPC-H Benchmark

TPC-H Benchmark

TPC-H is a decision support benchmark (Decision Support Benchmark), which consists of a set of business-oriented special query and concurrent data modification. The data that is queried and populates the database has broad industry relevance. This benchmark demonstrates a decision support system that examines large amounts of data, executes highly complex queries, and answers key business questions. The performance index reported by TPC-H is called TPC-H composite query performance index per hour (QphH@Size), which reflects multiple aspects of the system's ability to process queries. These aspects include the database size chosen when executing the query, the query processing capability when the query is submitted by a single stream, and the query throughput when the query is submitted by many concurrent users.

This document mainly introduces the performance of Doris on the TPC-H 1000G test set.

On 22 queries on the TPC-H standard test data set, we conducted a comparison test based on Apache Doris 2.1.7-rc03 and Apache Doris 2.0.15.1 versions.

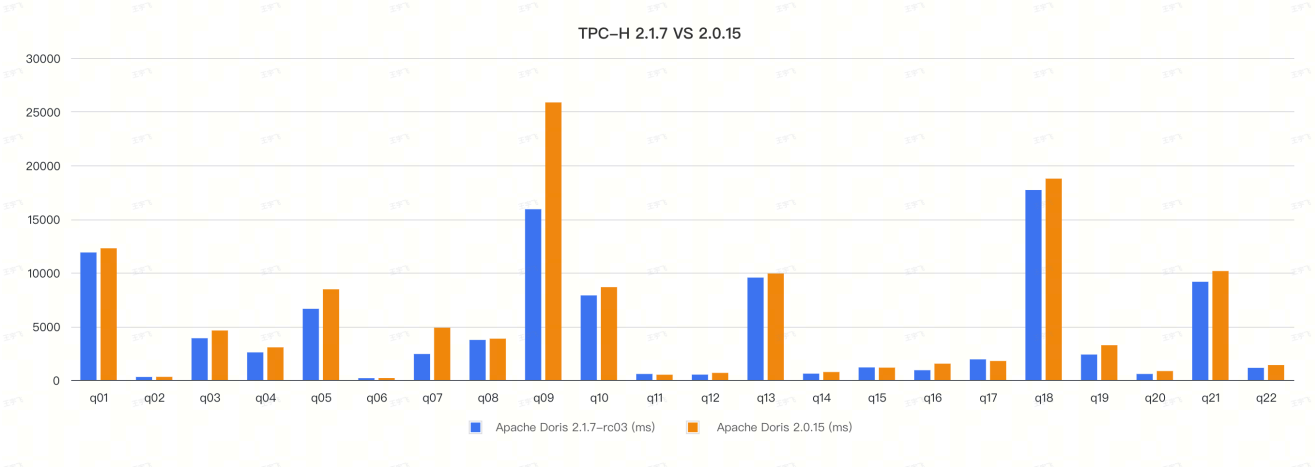


image-20220614114351241

1. Hardware Environment

Hardware	Configuration Instructions
Number of Machines	4 Aliyun Virtual Machine (1FE，3BEs)
CPU	Intel Xeon (Ice Lake) Platinum 8369B 32C
Memory	128G
Disk	Enterprise SSD (PL0)

2. Software Environment

- Doris Deployed 3BEs and 1FE
- Kernel Version: Linux version 5.15.0-101-generic

- OS version: Ubuntu 20.04 LTS (Focal Fossa)
- Doris software version: Apache Doris 2.1.7-rc03, Apache Doris 2.0.15.1
- JDK: openjdk version “1.8.0_352-352”

3. Test Data Volume

The TPC-H 1000G data generated by the simulation of the entire test are respectively imported into Apache Doris 2.1.7-rc03 and Apache Doris 2.0.15.1 for testing. The following is the relevant description and data volume of the table.

TPC-H Table Name	Rows	Annotation
REGION	5	Region
NATION	25	Nation
SUPPLIER	10,000,000	Supplier
PART	200,000,000	Parts
PARTSUPP	800,000,000	Parts Supply
CUSTOMER	150,000,000	Customer
ORDERS	1,500,000,000	Orders
LINEITEM	5,999,989,709	Order Details

4. Test SQL

TPC-H 22 test query statements : TPCH-Query-SQL

5. Test Results

Here we use Apache Doris 2.1.7-rc03 and Apache Doris 2.0.15.1 for comparative testing. In the test, we use Query Time(ms) as the main performance indicator. The test results are as follows:

Query	Apache Doris 2.1.7-rc03 (ms)	Apache Doris 2.0.15.1 (ms)
Q1	11880	12270
Q2	280	290
Q3	3890	4610
Q4	2570	3040
Q5	6630	8450
Q6	170	180
Q7	2420	4870
Q8	3730	3850
Q9	15910	25860
Q10	7880	8650

Query	Apache Doris 2.1.7-rc03 (ms)	Apache Doris 2.0.15.1 (ms)
Q11	560	490
Q12	500	660
Q13	9540	9920
Q14	590	740
Q15	1170	1150
Q16	910	1520
Q17	1920	1770
Q18	17700	18760
Q19	2370	3240
Q20	560	830
Q21	9150	10150
Q22	1130	1390
Total	**101460**	**122690**

6. Environmental Preparation

Please refer to the official document to install and deploy Doris to obtain a normal running Doris cluster (at least 1 FE 1 BE, 1 FE 3 BE is recommended).

7. Data Preparation

7.1 Download and Install TPC-H Data Generation Tool

Execute the following script to download and compile the tpch-tools tool.

```
sh bin/build-tpch-dbgen.sh
```

After successful installation, the `dbgen` binary will be generated under the `TPC-H_Tools_v3.0.0/` directory.

7.2 Generating the TPC-H Test Set

Execute the following script to generate the TPC-H dataset:

```
sh bin/gen-tpch-data.sh -s 1000
```

> Note 1: Check the script help via `sh gen-tpch-data.sh -h`.
>
> Note 2: The data will be generated under the `tpch-data/` directory with the suffix `.tbl`. The total file size is about 1000GB and may need a few minutes to an hour to generate.

> Note 3: A standard test data set of 100G is generated by default.

7.3 Create Table

7.3.1 Prepare the `doris-cluster.conf` File

Before import the script, you need to write the FE's ip port and other information in the `doris-cluster.conf` file.

The file is located under `${DORIS_HOME}/tools/tpch-tools/conf/` .

The content of the file includes FE's ip, HTTP port, user name, password and the DB name of the data to be imported:

```
# Any of FE host
export FE_HOST='127.0.0.1'
# http_port in fe.conf
export FE_HTTP_PORT=8030
# query_port in fe.conf
export FE_QUERY_PORT=9030
# Doris username
export USER='root'
# Doris password
export PASSWORD=''
# The database where TPC-H tables located
export DB='tpch'
```

Execute the Following Script to Generate and Create TPC-H Table

```
sh bin/create-tpch-tables.sh -s 1000
```

Or copy the table creation statement in create-tpch-tables.sql and execute it in Doris.

7.4 Import Data

Please perform data import with the following command:

```
sh bin/load-tpch-data.sh
```

7.5 Check Imported Data

Execute the following SQL statement to check that the imported data is consistent with the above data.

```
select count(*)  from  lineitem;
select count(*)  from  orders;
select count(*)  from  partsupp;
select count(*)  from  part;
select count(*)  from  customer;
select count(*)  from  supplier;
select count(*)  from  nation;
```

```
select count(*)  from  region;
select count(*)  from  revenue0;
```

7.6 Query Test

7.6.1 Executing Query Scripts

Execute the above test SQL or execute the following command

```
sh bin/run-tpch-queries.sh -s 1000
```

7.6.2 Single SQL Execution

The following is the SQL statement used in the test, you can also get the latest SQL from the code base.

```
--Q1
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date '1998-12-01' - interval '90' day
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus;

--Q2
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 15
```

```
    and p_type like '%BRASS'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp,
            supplier,
            nation,
            region
        where
        p_partkey = ps_partkey
        and s_suppkey = ps_suppkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'EUROPE'
)
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey
limit 100;

--Q3
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'BUILDING'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '1995-03-15'
    and l_shipdate > date '1995-03-15'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
limit 10;

--Q4
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date '1993-07-01'
    and o_orderdate < date '1993-07-01' + interval '3' month
    and exists (
        select
            *
        from
            lineitem
```

```
        where
                l_orderkey = o_orderkey
          and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority;

--Q5
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA'
    and o_orderdate >= date '1994-01-01'
    and o_orderdate < date '1994-01-01' + interval '1' year
group by
    n_name
order by
    revenue desc;

--Q6
select
    sum(l_extendedprice * l_discount) as revenue
from
    lineitem
where
    l_shipdate >= date '1994-01-01'
    and l_shipdate < date '1994-01-01' + interval '1' year
    and l_discount between .06 - 0.01 and .06 + 0.01
    and l_quantity < 24;

--Q7
select
    supp_nation,
    cust_nation,
    l_year,
    sum(volume) as revenue
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            extract(year from l_shipdate) as l_year,
            l_extendedprice * (1 - l_discount) as volume
        from
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
```

```
            nation n2
        where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
                or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
            )
            and l_shipdate between date '1995-01-01' and date '1996-12-31'
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
    l_year;

--Q8

select
    o_year,
    sum(case
        when nation = 'BRAZIL' then volume
        else 0
    end) / sum(volume) as mkt_share
from
    (
        select
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) as volume,
            n2.n_name as nation
        from
            part,
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2,
            region
        where
            p_partkey = l_partkey
            and s_suppkey = l_suppkey
            and l_orderkey = o_orderkey
            and o_custkey = c_custkey
            and c_nationkey = n1.n_nationkey
            and n1.n_regionkey = r_regionkey
            and r_name = 'AMERICA'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between date '1995-01-01' and date '1996-12-31'
            and p_type = 'ECONOMY ANODIZED STEEL'
    ) as all_nations
group by
    o_year
order by
    o_year;

--Q9
select
    nation,
```

```
    o_year,
    sum(amount) as sum_profit
from
    (
        select
            n_name as nation,
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
        from
            part,
            supplier,
            lineitem,
            partsupp,
            orders,
            nation
        where
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%green%'
    ) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc;

--Q10
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date '1993-10-01'
    and o_orderdate < date '1993-10-01' + interval '3' month
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc
limit 20;
```

```
--Q11
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value
from
    partsupp,
    supplier,
    nation
where
    ps_suppkey = s_suppkey
    and s_nationkey = n_nationkey
    and n_name = 'GERMANY'
group by
    ps_partkey having
    sum(ps_supplycost * ps_availqty) > (
        select
        sum(ps_supplycost * ps_availqty) * 0.000002
        from
            partsupp,
            supplier,
            nation
        where
            ps_suppkey = s_suppkey
            and s_nationkey = n_nationkey
            and n_name = 'GERMANY'
    )
order by
    value desc;

--Q12
select
    l_shipmode,
    sum(case
        when o_orderpriority = '1-URGENT'
            or o_orderpriority = '2-HIGH'
            then 1
        else 0
    end) as high_line_count,
    sum(case
        when o_orderpriority <> '1-URGENT'
            and o_orderpriority <> '2-HIGH'
            then 1
        else 0
    end) as low_line_count
from
    orders,
    lineitem
where
    o_orderkey = l_orderkey
    and l_shipmode in ('MAIL', 'SHIP')
    and l_commitdate < l_receiptdate
    and l_shipdate < l_commitdate
    and l_receiptdate >= date '1994-01-01'
    and l_receiptdate < date '1994-01-01' + interval '1' year
group by
    l_shipmode
order by
    l_shipmode;

--Q13
select
    c_count,
    count(*) as custdist
from
```

```
    (
        select
            c_custkey,
            count(o_orderkey) as c_count
        from
            customer left outer join orders on
                c_custkey = o_custkey
                and o_comment not like '%special%requests%'
        group by
            c_custkey
    ) as c_orders
group by
    c_count
order by
    custdist desc,
    c_count desc;

--Q14
select
    100.00 * sum(case
        when p_type like 'PROMO%'
            then l_extendedprice * (1 - l_discount)
        else 0
    end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
    lineitem,
    part
where
    l_partkey = p_partkey
    and l_shipdate >= date '1995-09-01'
    and l_shipdate < date '1995-09-01' + interval '1' month;

--Q15
select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
from
    supplier,
    revenue0
where
    s_suppkey = supplier_no
    and total_revenue = (
        select
            max(total_revenue)
        from
            revenue0
    )
order by
    s_suppkey;

--Q16
select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp,
    part
where
    p_partkey = ps_partkey
    and p_brand <> 'Brand#45'
```

```
    and p_type not like 'MEDIUM POLISHED%'
    and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
    and ps_suppkey not in (
        select
            s_suppkey
        from
            supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand,
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size;

--Q17
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );

--Q18
select
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
    sum(l_quantity)
from
    customer,
    orders,
    lineitem
where
    o_orderkey  in  (
        select
            l_orderkey
        from
            lineitem
        group  by
            l_orderkey  having
                sum(l_quantity)  >  300
    )
    and  c_custkey  =  o_custkey
    and  o_orderkey  =  l_orderkey
group  by
    c_name,
```

```
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order  by
    o_totalprice  desc,
    o_orderdate
limit  100;


--Q19
select
    sum(l_extendedprice* (1 - l_discount)) as revenue
from
    lineitem,
    part
where
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#12'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 1 and l_quantity <= 1 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#23'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 10 and l_quantity <= 10 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#34'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 20 and l_quantity <= 20 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    );

--Q20
select
    s_name,
    s_address
from
    supplier,
    nation
where
    s_suppkey in (
        select
            ps_suppkey
        from
            partsupp
        where
            ps_partkey in (
                select
                    p_partkey
                from
```

```
                    part
                where
                        p_name like 'forest%'
            )
            and ps_availqty > (
                select
                    0.5 * sum(l_quantity)
                from
                    lineitem
                where
                    l_partkey = ps_partkey
                    and l_suppkey = ps_suppkey
                    and l_shipdate >= date '1994-01-01'
                    and l_shipdate < date '1994-01-01' + interval '1' year
            )
    )
    and s_nationkey = n_nationkey
    and n_name = 'CANADA'
order by
    s_name;

--Q21
select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
        select
            *
        from
            lineitem l2
        where
                l2.l_orderkey = l1.l_orderkey
          and l2.l_suppkey <> l1.l_suppkey
    )
    and not exists (
        select
            *
        from
            lineitem l3
        where
                l3.l_orderkey = l1.l_orderkey
          and l3.l_suppkey <> l1.l_suppkey
          and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and n_name = 'SAUDI ARABIA'
group by
    s_name
order by
    numwait desc,
    s_name
limit 100;

--Q22
select
```

```
    cntrycode,
    count(*) as numcust,
    sum(c_acctbal) as totacctbal
from
    (
        select
            substring(c_phone, 1, 2) as cntrycode,
            c_acctbal
        from
            customer
        where
            substring(c_phone, 1, 2) in
            ('13', '31', '23', '29', '30', '18', '17')
            and c_acctbal > (
                select
                    avg(c_acctbal)
                from
                    customer
                where
                    c_acctbal > 0.00
                    and substring(c_phone, 1, 2) in
                      ('13', '31', '23', '29', '30', '18', '17')
            )
            and not exists (
                select
                    *
                from
                    orders
                where
                    o_custkey = c_custkey
            )
    ) as custsale
group by
    cntrycode
order by
    cntrycode;
```

TPC-DS Benchmark

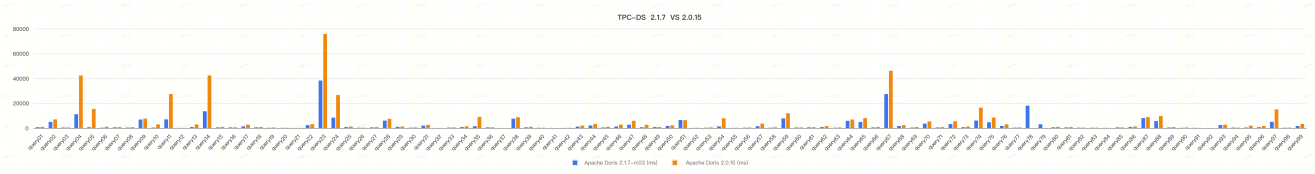
TPC-DS Benchmark

TPC-DS (Transaction Processing Performance Council Decision Support Benchmark) is a benchmark test that focuses on decision support and aims to evaluate the performance of data warehousing and analytics systems. It was developed by the Transaction Processing Performance Council (TPC) organization to compare the capabilities of different systems in handling complex queries and large-scale data analysis.

The design goal of TPC-DS is to simulate complex decision support workloads in the real world. It tests the performance of systems through a series of complex queries and data operations, including joins, aggregations, sorting, filtering, subqueries, and more. These query patterns cover various scenarios ranging from simple to complex, such as report generation, data mining, and OLAP (Online Analytical Processing).

This document mainly introduces the performance of Doris on the TPC-DS 1000G test set.

On 99 queries on the TPC-DS standard test data set, we conducted a comparison test based on Apache Doris 2.1.7-rc03 and Apache Doris 2.0.15.1 versions.



TPCDS_1000G

1. Hardware Environment

Hardware	Configuration Instructions
Number of Machines	4 Aliyun Virtual Machine (1FE，3BEs)
CPU	Intel Xeon (Ice Lake) Platinum 8369B 32C
Memory	128G
Disk	Enterprise SSD (PL0)

2. Software Environment

- Doris Deployed 3BEs and 1FE
- Kernel Version: Linux version 5.15.0-101-generic
- OS version: Ubuntu 20.04 LTS (Focal Fossa)
- Doris software version: Apache Doris 2.1.7-rc03, Apache Doris 2.0.15.1
- JDK: openjdk version “1.8.0_352-352”

3. Test Data Volume

The TPC-DS 1000G data generated by the simulation of the entire test are respectively imported into Apache Doris 2.1.7-rc03 and Apache Doris 2.0.15.1 for testing. The following is the relevant description and data volume of the table.

TPC-DS Table Name	Rows
customer_demographics	1,920,800
reason	65
warehouse	20
date_dim	73,049
catalog_sales	1,439,980,416
call_center	42
inventory	783,000,000
catalog_returns	143,996,756
household_demographics	7,200
customer_address	6,000,000
income_band	20
catalog_page	30,000
item	300,000
web_returns	71,997,522
web_site	54
promotion	1,500
web_sales	720,000,376
store	1,002
web_page	3,000
time_dim	86,400
store_returns	287,999,764
store_sales	2,879,987,999
ship_mode	20
customer	12,000,000

4. Test SQL

TPC-DS 99 test query statements : TPC-DS-Query-SQL

5. Test Results

Here we use Apache Doris 2.1.7-rc03 and Apache Doris 2.0.15.1 for comparative testing. In the test, we use Query Time(ms) as the main performance indicator. The test results are as follows: (Apache Doris 2.0.15.1 q78 q79 failed to execute due to lack of latest memory optimization and was removed when calculating the total sum)

Query	Apache Doris 2.1.7-rc03 (ms)	Apache Doris 2.0.15.1 (ms)
query01	630	890
query02	4930	6930
query03	360	460
query04	11070	42320
query05	620	15360
query06	220	1020
query07	550	750
query08	330	670
query09	6830	7550
query10	370	2900
query11	6960	27380
query12	100	80
query13	790	2860
query14	13470	42340
query15	510	940
query16	520	550
query17	1310	2650
query18	560	820
query19	200	400
query20	100	190
query21	80	80
query22	2300	3070
query23	38240	75260
query24	8340	26580
query25	780	1190
query26	200	220
query27	530	750
query28	5940	7400

Query	Apache Doris 2.1.7-rc03 (ms)	Apache Doris 2.0.15.1 (ms)
query29	940	1250
query30	270	490
query31	1890	2530
query32	60	70
query33	350	450
query34	750	1380
query35	1370	8970
query36	530	570
query37	60	60
query38	7520	8710
query39	560	1010
query40	150	180
query41	50	40
query42	100	140
query43	1150	1960
query44	2020	3220
query45	430	960
query46	1250	2760
query47	2660	5790
query48	630	2570
query49	730	800
query50	1640	2200
query51	6430	6270
query52	110	160
query53	250	490
query54	1280	7790
query55	110	160
query56	290	410
query57	1480	3510
query58	240	550
query59	7760	11870
query60	380	490
query61	540	670

Query	Apache Doris 2.1.7-rc03 (ms)	Apache Doris 2.0.15.1 (ms)
query62	740	1560
query63	210	460
query64	5790	6840
query65	4900	7960
query66	480	810
query67	27320	46110
query68	1600	2380
query69	380	800
query70	3480	5330
query71	460	790
query72	3160	5390
query73	660	1250
query74	5990	16450
query75	4610	8410
query76	1590	2950
query77	300	480
query78	17970	-
query79	3040	-
query80	570	910
query81	460	760
query82	270	330
query83	220	290
query84	130	110
query85	520	470
query86	760	1220
query87	800	8760
query88	5560	9690
query89	430	750
query90	150	400
query91	150	120
query92	40	40
query93	2440	2670
query94	340	310

Query	Apache Doris 2.1.7-rc03 (ms)	Apache Doris 2.0.15.1 (ms)
query95	350	1810
query96	660	1680
query97	5020	14990
query98	190	330
query99	1560	3230
Total	**261320**	**507380**

6. Environmental Preparation

Please refer to the official document to install and deploy Doris to obtain a normal running Doris cluster (at least 1 FE 1 BE, 1 FE 3 BE is recommended).

7. Data Preparation

7.1 Download and Install TPC-DS Data Generation Tool

Execute the following script to download and compile the tpcds-tools tool.

```
sh bin/build-tpcds-dbgen.sh
```

7.2 Generating the TPC-DS Test Set

Execute the following script to generate the TPC-H dataset:

```
sh bin/gen-tpcds-data.sh -s 1000
```

> Note 1: Check the script help via `sh gen-tpcds-data.sh -h`.
>
> Note 2: The data will be generated under the `tpcds-data/` directory with the suffix `.dat`. The total file size is about 1000GB and may need a few minutes to an hour to generate.
>
> Note 3: A standard test data set of 100G is generated by default.

7.3 Create Table

7.3.1 Prepare the `doris-cluster.conf` File

Before import the script, you need to write the FE's ip port and other information in the `doris-cluster.conf` file.

The file is located under `${DORIS_HOME}/tools/tpcds-tools/conf/` .

The content of the file includes FE's ip, HTTP port, user name, password and the DB name of the data to be imported:

```
# Any of FE host
export FE_HOST='127.0.0.1'
# http_port in fe.conf
export FE_HTTP_PORT=8030
# query_port in fe.conf
export FE_QUERY_PORT=9030
# Doris username
export USER='root'
# Doris password
export PASSWORD=''
# The database where TPC-H tables located
export DB='tpcds'
```

Execute the Following Script to Generate and Create TPC-H Table

```
sh bin/create-tpcds-tables.sh -s 1000
```

Or copy the table creation statement in create-tpcds-tables and execute it in Doris.

7.4 Import Data

Please perform data import with the following command:

```
sh bin/load-tpcds-data.sh
```

7.5 Query Test

7.5.1 Executing Query Scripts

Execute the above test SQL or execute the following command

```
sh bin/run-tpcds-queries.sh -s 1000
```

7.5.2 Single SQL Execution

You can also retrieve the latest SQL from the code repository. The address for the latest test query statements of TPC-DS.

Upgrading Cluster

Doris provides the capability for rolling upgrades, enabling step-by-step upgrades of FE and BE nodes, minimizing downtime, and ensuring the system remains operational during the upgrade process.

Version Compatibility

Doris versioning consists of three components: the first digit represents a major milestone version, the second digit indicates a feature version, and the third digit corresponds to a bug fix. New features are not introduced in bug fix versions. For example, in Doris version 2.1.3, “2” indicates the second milestone version, “1” represents the feature version under this milestone, and “3” denotes the third bug fix for this feature version.

During version upgrades, the following rules apply:

- **Three-digit versions:** Versions with the same first two digits can be directly upgraded across three-digit versions. For example, version 2.1.3 can be directly upgraded to version 2.1.7.
- **Two-digit and one-digit versions:** Cross-version upgrades for two-digit versions are not recommended due to compatibility concerns. It is advised to upgrade sequentially through each two-digit version. For example, upgrading from version 3.0 to 3.3 should follow the sequence 3.0 -> 3.1 -> 3.2 -> 3.3.

The detailed version information can be found in the versioning rules.

Upgrade Precautions

When performing an upgrade, pay attention to the following:

- **Behavioral changes between versions:** Review the Release Notes before upgrading to identify compatibility issues.
- **Add retry mechanisms for tasks in the cluster:** Nodes are restarted sequentially during upgrades. Ensure that retry mechanisms are in place for query tasks and Stream Load import jobs to avoid task failures. Routine Load jobs using flink-doris-connector or spark-doris-connector already include retry mechanisms in their code and do not require additional logic.
- **Disable replica repair and balance functions:** Disable these functions during the upgrade process. Regardless of the upgrade outcome, re-enable these functions after the upgrade is complete.

Metadata Compatibility Testing

> **Note**
>
> In a production environment, it is recommended to configure at least three FE nodes for high availability. If there is only one FE node, metadata compatibility testing must be performed before upgrading. Metadata compatibility is critical as incompatibility may cause upgrade failures and data loss. It is recommended to conduct metadata compatibility tests before each upgrade, keeping in mind the following:

> - Perform metadata compatibility testing on a development machine or BE node whenever possible to avoid using FE nodes.
> - If testing must be conducted on an FE node, use a non-Master node and stop the original FE process.

Before upgrading, conduct metadata compatibility testing to prevent failures caused by metadata incompatibility.

1. **Backup metadata information:**

   Before starting the upgrade, back up the metadata of the Master FE node.

   Use the `show frontends` command and refer to the `IsMaster` column to identify the Master FE node. FE metadata can be hot-backed up without stopping the FE node. By default, FE metadata is stored in the `fe/doris-meta` directory. This can be confirmed via the `meta_dir` parameter in the `fe.conf` configuration file.

2. **Modify the `fe.conf` configuration file of the test FE node:**

   ```
   vi ${DORIS_NEW_HOME}/conf/fe.conf
   ```

   Modify the following port information, ensuring all ports are different from those in the production environment, and update the `clusterID` parameter:

   ```
   ...
   ## modify port
   http_port = 18030
   rpc_port = 19020
   query_port = 19030
   arrow_flight_sql_port = 19040
   edit_log_port = 19010

   ## modify clusterID
   clusterId=<a_new_clusterID, such as 123456>
   ...
   ```

3. Copy the backed-up Master FE metadata to the new compatibility testing environment.

   ```
   cp ${DORIS_OLD_HOME}/fe/doris-meta/* ${DORIS_NEW_HOME}/fe/doris-meta
   ```

4. Edit the `VERSION` file in the copied metadata directory to update the `cluster_id` to a new cluster ID, for example, change it to `123456` as shown in the example:

   ```
   vi ${DORIS_NEW_HOME}/fe/doris-meta/image/VERSION
   clusterId=123456
   ```

5. Start the FE process in the testing environment.

   ```
   sh ${DORIS_NEW_HOME}/bin/start_fe.sh --daemon --metadata_failure_recovery
   ```

For versions earlier than 2.0.2, add the `metadata_failure_recovery` parameter to the `fe.conf` file before starting the FE process:

```
echo "metadata_failure_recovery=true" >> ${DORIS_NEW_HOME}/conf/fe.conf
sh ${DORIS_NEW_HOME}/bin/start_fe.sh --daemon
```

6. Verify that the FE process has started successfully by connecting to the current FE using the MySQL command. For example, use the query port `19030` as mentioned above:

```
mysql -uroot -P19030 -h127.0.0.1
```

Upgrade Steps

The detailed process for the upgrade is as follows:

1. Disable replica repair and balance functions
2. Upgrade BE nodes
3. Upgrade FE nodes
4. Enable replica repair and balance functions

During the upgrade process, the principle of upgrading BE nodes first, followed by upgrading FE nodes, should be followed. When upgrading FE, upgrade the Observer FE and Follower FE nodes first, and then upgrade the Master FE node.

> **Note**
>
> In general, only the `/bin` and `/lib` directories under the FE directory and the `/bin` and `/lib` directories under the BE directory need to be upgraded.
>
> For versions 2.0.2 and later, a `custom_lib/` directory has been added under the FE and BE deployment paths (if it doesn't exist, it can be manually created). The `custom_lib/` directory is used to store some user-defined third-party jar files, such as `hadoop-lzo-*.jar`, `orai18n.jar`, etc. This directory does not need to be replaced during the upgrade.

Step 1: Disable Replica Repair and Balance Functions

During the upgrade process, nodes will be restarted, which may trigger unnecessary cluster balancing and replica repair logic. Disable these functions first using the following command:

```
admin set frontend config("disable_balance" = "true");
admin set frontend config("disable_colocate_balance" = "true");
admin set frontend config("disable_tablet_scheduler" = "true");
```

Step 2: Upgrade BE Nodes

> **Note:**
>
> To ensure the safety of your data, please use 3 replicas to store your data to avoid data loss caused by upgrade mistakes or failures.

1. In a multi-replica cluster, you can choose to stop the process on one BE node and perform a gradual upgrade:

   ```
   sh ${DORIS_OLD_HOME}/be/bin/stop_be.sh
   ```

2. Rename the `/bin` and `/lib` directories in the BE directory:

   ```
   mv ${DORIS_OLD_HOME}/be/bin ${DORIS_OLD_HOME}/be/bin_back
   mv ${DORIS_OLD_HOME}/be/lib ${DORIS_OLD_HOME}/be/lib_back
   ```

3. Copy the new version's `/bin` and `/lib` directories to the original BE directory:

   ```
   cp -r ${DORIS_NEW_HOME}/be/bin ${DORIS_OLD_HOME}/be/bin
   cp -r ${DORIS_NEW_HOME}/be/lib ${DORIS_OLD_HOME}/be/lib
   ```

4. Start the BE node:

   ```
   sh ${DORIS_OLD_HOME}/be/bin/start_be.sh --daemon
   ```

5. Connect to the cluster and check the node information:

   ```
   show backends\G
   ```

   If the BE node's `alive` status is `true` and the `Version` value is the new version, the node has been successfully upgraded.

Step 3: Upgrade FE Nodes

1. In a multi-FE node setup, select a non-Master node for the upgrade and stop it first:

   ```
   sh ${DORIS_OLD_HOME}/fe/bin/stop_fe.sh
   ```

2. Rename the `/bin`, `/lib`, and `/mysql_ssl_default_certificate` directories in the FE directory:

   ```
   mv ${DORIS_OLD_HOME}/fe/bin ${DORIS_OLD_HOME}/fe/bin_back
   mv ${DORIS_OLD_HOME}/fe/lib ${DORIS_OLD_HOME}/fe/lib_back
   mv ${DORIS_OLD_HOME}/fe/mysql_ssl_default_certificate ${DORIS_OLD_HOME}/fe/mysql_ssl_default_certificate_back
   ```

3. Copy the new version's `/bin`, `/lib`, and `/mysql_ssl_default_certificate` directories to the original FE directory:

```
cp -r ${DORIS_NEW_HOME}/fe/bin ${DORIS_OLD_HOME}/fe/bin
cp -r ${DORIS_NEW_HOME}/fe/lib ${DORIS_OLD_HOME}/fe/lib
cp -r ${DORIS_NEW_HOME}/fe/mysql_ssl_default_certificate ${DORIS_OLD_HOME}/fe/mysql_ssl_default_certificate
```

4. Start the FE node:

```
sh ${DORIS_OLD_HOME}/fe/bin/start_fe.sh --daemon
```

5. Connect to the cluster and check the node information:

```
show frontends\G
```

   If the FE node's `alive` status is `true` and the `Version` value is the new version, the node has been successfully upgraded.

6. Complete the upgrade of the other FE nodes in sequence, and finally upgrade the Master node.

Step 4: Enable Replica Repair and Balance Functions

After the upgrade is complete and all BE nodes' status is `Alive`, enable the cluster's replica repair and balance functions:

```
admin set frontend config("disable_balance" = "false");
admin set frontend config("disable_colocate_balance" = "false");
admin set frontend config("disable_tablet_scheduler" = "false");
```

Elastic Scaling

Doris supports online elastic scaling, allowing users to dynamically add or remove nodes without interrupting services. This capability ensures businesses can meet growing demands or reduce idle resource waste. Scaling up or down BE nodes does not affect cluster availability but involves data migration, so it is recommended to perform scaling operations during periods of low business activity.

Scale In/Out the FE Clusters

Doris FE nodes are divided into the following three roles, with each FE node containing a full set of metadata:

- Master Node: Responsible for reading and writing metadata. When metadata changes occur on the Master node, they are synchronized to non-Master nodes via the BDB JE protocol. There can be only one Master FE node in the cluster.
- Follower Node: Responsible for reading metadata. In the event of a Master node failure, Follower nodes initiate a leader election to select a new Master node. Within the cluster, the total number of Master and Follower nodes is recommended to be an odd number.
- Observer Node: Responsible for reading metadata but does not participate in leader election. It is used to extend the read service capacity of FE nodes.

Typically, each FE node can handle the load operations of 10-20 BE nodes. A configuration of 3 FE nodes is sufficient to meet the requirements of most business scenarios.

Scale the FE out

> **Note:**
>
> When adding a new FE node, please pay attention to the following:
>
> - The `http_port` of the new FE node must match the `http_port` of all existing FE nodes in the cluster.
> - If adding a Follower node, it is recommended that the total number of Master and Follower nodes in the cluster be an odd number.
> - You can view the ports and roles of the current cluster nodes using the `show frontends` command.

1. Start FE Node:

```
fe/bin/start_fe.sh --helper <leader_fe_host>:<edit_log_port> --daemon
```

- Register FE Node:
    - Register the node as a Follower FE:

```
ALTER SYSTEM ADD FOLLOWER "<follower_host>:<edit_log_port>";
```

- Register the node as an Observer FE:

```
ALTER SYSTEM ADD OBSERVER "<observer_host>:<edit_log_port>";
```

- Check the status of the newly added FE node

```
show frontends;
```

Scale In the FE Cluster

When scaling in FE nodes, ensure that the total number of Master and Follower nodes in the cluster remains an odd number. Use the following commands to remove nodes:

```
ALTER SYSTEM DROP FOLLOWER[OBSERVER] "<fe_host>:<edit_log_port>";
```

After scaling in, you need to manually delete the FE directory.

Scale In/Out the BE Cluster

Scale Out the BE Cluster

1. Start the BE process:

```
be/bin/start_be.sh
```

2. Register the BE node:

```
ALTER SYSTEM ADD backend '<be_host>:<be_heartbeat_service_port>';
```

Scale In the BE Cluster

When scaling in BE nodes, you can choose between the DROP or DECOMMISSION methods:

	DROP	DECOMMISSION
Principle	Directly remove the node, deleting the BE node.	Initiates a command to migrate data on the BE node to other nodes. Once migration is complete, the BE node is automatically removed.
Effective Time	Takes effect immediately after execution.	Takes effect after data migration is completed. Depending on the cluster's existing data volume, this can take hours to up to a day.
Single Replica Table Handling	May result in data loss.	Does not result in data loss.

	DROP	DECOMMISSION
Removing Multiple Nodes Simultaneously	May result in data loss.	Does not result in data loss.
Production Recommendation	Not recommended for production environments.	Recommended for production environments.

- Use the following command to remove a BE node using the DROP method:

```
ALTER SYSTEM DROP backend "<be_host>:<be_heartbeat_service_port>";
```

- Use the following command to remove a BE node using the DECOMMISSION method:

```
ALTER SYSTEM DECOMMISSION backend "<be_host>:<be_heartbeat_service_port>";
```

DECOMMISSION Command Description:

- DECOMMISSION is an asynchronous operation. After execution, you can see the BE node's `SystemDecommissioned` status set to `true` via `SHOW backends;`. This indicates the node is being removed.
- The DECOMMISSION command may fail. For instance, if there is insufficient storage space on the remaining BE nodes to accommodate the data from the BE being removed, or if the remaining nodes do not meet the minimum replication requirements, the command will not complete, and the BE will remain in a `SystemDecommissioned` state set to `true`.
- The progress of DECOMMISSION can be monitored using `SHOW PROC '/backends';`. If the operation is in progress, the `TabletNum` value will decrease continuously.
- You can cancel the operation using the command `CANCEL DECOMMISSION BACKEND "be_host:be_heartbeat_service_port";`. After cancellation, the BE node will retain its current remaining data, and Doris will re-balance the load.
- The data migration rate can be adjusted by modifying the `balance_slot_num_per_path` parameter.

Load Balancing

Users connect to Doris through FE's query port (`query_port`, default 9030) using the MySQL protocol. When deploying multiple FE nodes, users can deploy a load balancing layer on top of multiple FEs to achieve high availability for Doris queries.

This document introduces various load balancing solutions suitable for Doris and explains how to implement client IP passthrough using the Proxy Protocol.

Load Balancing

This article uses the following three FE nodes as examples for demonstrating the steps:

```
192.168.1.101:9030
192.168.1.102:9030
192.168.1.103:9030
```

Proxy server node:

```
192.168.1.100
```

01 JDBC URL

Use the built-in load balancing configuration in JDBC URL.

```
jdbc:mysql:loadbalance://192.168.1.101:9030,192.168.1.102:9030,192.168.1.103:9030/test_db
```

For details, please refer to MySQL Official Documentation

02 Nginx

Use Nginx TCP reverse proxy to implement Doris load balancing.

Install Nginx

Please refer to Nginx official website to install Nginx correctly. Here we demonstrate the Nginx compilation and installation steps using Ubuntu system with Nginx 1.18.0 version as an example.

1. Install compilation dependencies

   ```
   sudo apt-get install build-essential
   sudo apt-get install libpcre3 libpcre3-dev
   sudo apt-get install zlib1g-dev
   sudo apt-get install openssl libssl-dev
   ```

2. Install Nginx

```
sudo wget http://nginx.org/download/nginx-1.18.0.tar.gz
sudo tar zxvf nginx-1.18.0.tar.gz
cd nginx-1.18.0
sudo ./configure --prefix=/usr/local/nginx --with-stream --with-http_ssl_module --with-http_gzip_static_module --with-http_stub_status_module
sudo make && make install
```

Configure Reverse Proxy

Create a new configuration file:

```
vim /usr/local/nginx/conf/default.conf
```

Content as follows:

```
events {
worker_connections 1024;
}
stream {
  upstream mysqld {
      hash $remote_addr consistent;
      server 192.168.1.101:9030 weight=1 max_fails=2 fail_timeout=60s;
      server 192.168.1.102:9030 weight=1 max_fails=2 fail_timeout=60s;
      server 192.168.1.103:9030 weight=1 max_fails=2 fail_timeout=60s;
  }
  server {
   # Proxy port
      listen 6030;
      proxy_connect_timeout 300s;
      proxy_timeout 300s;
      proxy_pass mysqld;
  }
}
```

Start Nginx

Start with specified configuration file:

```
cd /usr/local/nginx
/usr/local/nginx/sbin/nginx -c conf.d/default.conf
```

Verify

Connect using the proxy port:

```
mysql -uroot -P6030 -h192.168.1.100

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
```

```
| test              |
+-------------------+
2 rows in set (0.00 sec)
```

03 HAProxy

HAProxy is a high-performance TCP/HTTP load balancer written in C language.

Install HAProxy

1. Download HAProxy

   Download link: https://src.fedoraproject.org/repo/pkgs/haproxy/

2. Extract

   ```
   tar -zxvf haproxy-2.6.15.tar.gz -C /opt/
   mv haproxy-2.6.15 haproxy
   cd haproxy
   ```

3. Compile

   ```
   yum install gcc gcc-c++ -y
   make TARGET=linux-glibc PREFIX=/usr/local/haproxy
   make install PREFIX=/usr/local/haproxy
   ```

Configure HAProxy

1. Configure haproxy.conf file

   Open configuration file:

   ```
   vim /etc/rsyslog.d/haproxy.conf
   ```

   Content as follows:

   ```
   $ModLoad imudp
   $UDPServerRun 514
   local0.* /usr/local/haproxy/logs/haproxy.log
   &~
   ```

2. Enable remote logging

   ```
   vim /etc/sysconfig/rsyslog
   ```

   Add content:

   ```
   SYSLOGD_OPTIONS="-c 2 -r -m 0"
   ```

Parameter description:

- `-c 2`: Use compatibility mode, default is `-c 5`.
- `-r`: Enable remote logging.
- `-m 0`: Mark timestamps. In minutes, when 0, indicates this feature is disabled.

Apply changes:

```
systemctl restart rsyslog
```

3. Edit load balancing file

```
vim /usr/local/haproxy/haproxy.cfg
```

```
global
    maxconn         2000
    ulimit-n        40075
    log             127.0.0.1 local0 info
    uid             200
    gid             200
    chroot          /var/empty
    daemon
    group           haproxy
    user            haproxy

defaults
    log global
    mode http
    retries 3
    option redispatch

    timeout connect 5000
    timeout client 5000
    timeout server 5000
    timeout check 2000

frontend agent-front
    bind *:6030
    mode tcp
    default_backend forward-fe

backend forward-fe
    mode tcp
    balance roundrobin
    server fe-1 192.168.1.101:9030 weight 1 check inter 3000 rise 2 fall 3
    server fe-2 192.168.1.102:9030 weight 1 check inter 3000 rise 2 fall 3
    server fe-3 192.168.1.103:9030 weight 1 check inter 3000 rise 2 fall 3
```

Start HAProxy

1. Start service

```
/opt/haproxy/haproxy -f /usr/local/haproxy/haproxy.cfg
```

2. Check service status

```
netstat -lnatp | grep -i haproxy
```

Verify

```
mysql -h 192.168.1.100 -uroot -P6030 -p
```

04 ProxySQL

ProxySQL is an open-source MySQL database proxy software written in C language. It can implement connection management, read-write splitting, load balancing, failover, and other functions. It has advantages such as high performance, configurability, and dynamic management, and is commonly used in Web services, big data platforms, cloud databases, and other scenarios.

Install ProxySQL

Please refer to the official documentation to install ProxySQL correctly.

Configure ProxySQL

ProxySQL includes configuration file `/etc/proxysql.cnf` and configuration database file `/var/lib/proxysql/proxysql.db`.

Special attention should be paid that if there is a “proxysql.db” file in the `/var/lib/proxysql` directory, the ProxySQL service only reads and parses `proxysql.cnf` during the first startup, and subsequent startups will no longer read it.

To make `proxysql.cnf` configuration take effect after restart, you need to delete `/var/lib/proxysql/proxysql.db` before restarting the service, which is equivalent to initialization startup and will generate a new `proxysql.db` file, and the original configuration rules will be cleared.

Here is the main content of the configuration file `proxysql.cnf`:

```
datadir="/var/lib/proxysql"         #Data directory
admin_variables=
{
    admin_credentials="admin:admin"  # Admin database username and password.
    mysql_ifaces="0.0.0.0:6032"    # Admin database port, used for connecting admin database of ProxySQL
}
mysql_variables=
{
    threads=4
    max_connections=2048
    default_query_delay=0
    default_query_timeout=36000000
    have_compress=true
    poll_timeout=2000
    interfaces="0.0.0.0:6030"
    default_schema="information_schema"
    stacksize=1048576
    server_version="5.7.99"
    connect_timeout_server=3000
    monitor_username="monitor"
    monitor_password="monitor"
    monitor_history=600000
    monitor_connect_interval=60000
    monitor_ping_interval=10000
```

```
    monitor_read_only_interval=1500
    monitor_read_only_timeout=500
    ping_interval_server_msec=120000
    ping_timeout_server=500
    commands_stats=true
    sessions_sort=true
    connect_retries_on_failure=10
}
mysql_servers =
(
)
mysql_users:
(
)
mysql_query_rules:
(
)
scheduler=
(
)
mysql_replication_hostgroups=
(
)
```

Connect to ProxySQL Admin Database

```
mysql -uadmin -padmin -P6032 -hdoris01
```

```
ProxySQL > show databases;
+-----+---------------+-------------------------------------+
| seq | name          | file                                |
+-----+---------------+-------------------------------------+
| 0   | main          |                                     |
| 2   | disk          | /var/lib/proxysql/proxysql.db       |
| 3   | stats         |                                     |
| 4   | monitor       |                                     |
| 5   | stats_history | /var/lib/proxysql/proxysql_stats.db |
+-----+---------------+-------------------------------------+
5 rows in set (0.000 sec)
ProxySQL > use main;

ProxySQL > show tables;
+--------------------------------------------+
| tables                                     |
+--------------------------------------------+
| global_variables                           |
| mysql_collations                           |
| mysql_group_replication_hostgroups         |
| mysql_query_rules                          |
| mysql_query_rules_fast_routing             |
| mysql_replication_hostgroups               |
| mysql_servers                              |
| mysql_users                                |
| proxysql_servers                           |
| runtime_checksums_values                   |
| runtime_global_variables                   |
| runtime_mysql_group_replication_hostgroups |
| runtime_mysql_query_rules                  |
| runtime_mysql_query_rules_fast_routing     |
| runtime_mysql_replication_hostgroups       |
| runtime_mysql_servers                      |
```

```
| runtime_mysql_users                       |
| runtime_proxysql_servers                  |
| runtime_scheduler                         |
| scheduler                                 |
+-------------------------------------------+
20 rows in set (0.000 sec)
```

Configure Backend Doris FE in ProxySQL

Use INSERT statements to add the FE nodes and ports that need to be proxied to the `mysql_servers` table.

Where: `hostgroup_id` of `10` indicates write group, `20` indicates read group. We don't need read-write splitting here, so it can be set arbitrarily.

```
mysql -uadmin -padmin -P6032 -h127.0.0.1
```

```
ProxySQL > insert into mysql_servers(hostgroup_id,hostname,port) values(10,'192.168.0.101',9030);
Query OK, 1 row affected (0.000 sec)

ProxySQL > insert into mysql_servers(hostgroup_id,hostname,port) values(10,'192.168.0.102',9030);
Query OK, 1 row affected (0.000 sec)

ProxySQL > insert into mysql_servers(hostgroup_id,hostname,port) values(10,'192.168.0.103',9030);
Query OK, 1 row affected (0.000 sec)
```

Check results:

```
ProxySQL > select hostgroup_id,hostname,port,status,weight from mysql_servers;
+--------------+---------------+------+--------+--------+
| hostgroup_id | hostname      | port | status | weight |
+--------------+---------------+------+--------+--------+
| 10           | 192.168.0.101 | 9030 | ONLINE | 1      |
| 20           | 192.168.0.102 | 9030 | ONLINE | 1      |
| 20           | 192.168.0.103 | 9030 | ONLINE | 1      |
+--------------+---------------+------+--------+--------+
3 rows in set (0.000 sec)
```

If you encounter an error during insertion:

```
ERROR 1045 (#2800): UNIQUE constraint failed: mysql_servers.hostgroup_id, mysql_servers.hostname, mysql_servers.port
```

This indicates that other configurations may have been defined previously. You can empty this table or delete the configuration for the corresponding host:

```
ProxySQL > select * from mysql_servers;
ProxySQL > delete from mysql_servers;
Query OK, 6 rows affected (0.000 sec)
```

Save information:

```
ProxySQL > load mysql servers to runtime;
Query OK, 0 rows affected (0.006 sec)
```

```
ProxySQL > save mysql servers to disk;
Query OK, 0 rows affected (0.348 sec)
```

Configure Monitoring for Doris FE Nodes

After adding Doris FE nodes, these backend nodes need to be monitored.

First, create a user for monitoring in Doris:

```
mysql -uroot -P9030 -h192.168.0.101
```

```
Doris > create user monitor@'192.168.0.100' identified by 'P@ssword1!';
Query OK, 0 rows affected (0.03 sec)

Doris > grant ADMIN_PRIV on *.* to monitor@'192.168.0.100';
Query OK, 0 rows affected (0.02 sec)
```

Then go back to the mysql-proxy proxy layer node to configure monitoring

```
mysql -uadmin -padmin -P6032 -h127.0.0.1
```

```
ProxySQL > set mysql-monitor_username='monitor';
Query OK, 1 row affected (0.000 sec)

ProxySQL > set mysql-monitor_password='P@ssword1!';
Query OK, 1 row affected (0.000 sec)
```

Save configuration and exit:

```
ProxySQL > load mysql servers to runtime;
Query OK, 0 rows affected (0.006 sec)

ProxySQL > save mysql servers to disk;
Query OK, 0 rows affected (0.348 sec)
```

Verification monitoring results.

The metrics of the ProxySQL monitoring module are all saved in the `monitor.log` table.

Connection monitoring:

```
ProxySQL > select * from mysql_server_connect_log;
+---------------+------+------------------+-------------------------+---------------+
| hostname      | port | time_start_us    | connect_success_time_us | connect_error |
+---------------+------+------------------+-------------------------+---------------+
| 192.168.0.101 | 9030 | 1548665195883957 | 762                     | NULL          |
| 192.168.0.102 | 9030 | 1548665195894099 | 399                     | NULL          |
| 192.168.0.103 | 9030 | 1548665195904266 | 483                     | NULL          |
| 192.168.0.101 | 9030 | 1548665255883715 | 824                     | NULL          |
| 192.168.0.102 | 9030 | 1548665255893942 | 656                     | NULL          |
| 192.168.0.101 | 9030 | 1548665495884125 | 615                     | NULL          |
| 192.168.0.102 | 9030 | 1548665495894254 | 441                     | NULL          |
| 192.168.0.103 | 9030 | 1548665495904479 | 638                     | NULL          |
```

```
| 192.168.0.101 | 9030 | 1548665512917846 | 487                    | NULL          |
| 192.168.0.102 | 9030 | 1548665512928071 | 994                    | NULL          |
| 192.168.0.103 | 9030 | 1548665512938268 | 613                    | NULL          |
+---------------+------+------------------+------------------------+---------------+
20 rows in set (0.000 sec)
```

Heartbeat monitoring:

```
ProxySQL > select * from mysql_server_ping_log;
+---------------+------+------------------+----------------------+------------+
| hostname      | port | time_start_us    | ping_success_time_us | ping_error |
+---------------+------+------------------+----------------------+------------+
| 192.168.0.101 | 9030 | 1548665195883407 | 98                   | NULL       |
| 192.168.0.102 | 9030 | 1548665195885128 | 119                  | NULL       |
...........
| 192.168.0.102 | 9030 | 1548665415889362 | 106                  | NULL       |
| 192.168.0.103 | 9030 | 1548665562898295 | 97                   | NULL       |
+---------------+------+------------------+----------------------+------------+
110 rows in set (0.001 sec)
```

Client IP Passthrough

In most cases, when connecting to the backend Doris service through a proxy service, client IP information will be lost, and the Doris server can only obtain the IP address information of the proxy server.

Starting from version 2.1.1, Doris supports the Proxy Protocol protocol. Using this protocol, client IP passthrough can be implemented, so that after going through load balancing, Doris can still obtain the client's real IP to implement whitelist and other permission controls.

Below we introduce how to enable Proxy Protocol in Nginx and Haproxy respectively.

Enable Proxy Protocol Support in Doris

Add to `fe.conf` in FE:

```
enable_proxy_protocol = true
```

> **1. Only supports Proxy Protocol V1.**
>
> 2. Only supports and affects MySQL protocol ports, does not support or affect HTTP, ADBC, and other protocol ports.
> 3. Before Doris 3.1 version, after enabling, you must use the Proxy Protocol protocol to connect, otherwise the connection will fail. Starting from version 3.1, after enabling Proxy Protocol, you can still connect using the standard MySQL connection protocol.

01 Nginx

Add `proxy_protocol on;` to the `server` section in the configuration file:

```
events {
worker_connections 1024;
}
stream {
  upstream mysqld {
      hash $remote_addr consistent;
      server 192.168.1.101:9030 weight=1 max_fails=2 fail_timeout=60s;
      server 192.168.1.102:9030 weight=1 max_fails=2 fail_timeout=60s;
      server 192.168.1.103:9030 weight=1 max_fails=2 fail_timeout=60s;
  }
  server {
   # Proxy port
      listen 6030;
      proxy_connect_timeout 300s;
      proxy_timeout 300s;
      proxy_pass mysqld;
      # Enable Proxy Protocol to the upstream server
      proxy_protocol on;
  }
}
```

02 HAProxy

Add `send-proxy` parameter to the `backend` section in `haproxy.cfg`:

```
backend forward-fe
    mode tcp
    balance roundrobin
    server fe-1 192.168.1.101:9030 weight 1 check inter 3000 rise 2 fall 3 send-proxy
    server fe-2 192.168.1.102:9030 weight 1 check inter 3000 rise 2 fall 3 send-proxy
    server fe-3 192.168.1.103:9030 weight 1 check inter 3000 rise 2 fall 3 send-proxy
```

Verify IP Passthrough Success

Connect to Doris through proxy:

```
mysql -uroot -P6030 -h192.168.1.100
```

Verify

```
mysql> show processlist;
+------------------+------+------+------------------+---------------------+----------+------+---------+------+-------+-------------
---------------------+------------------+
| CurrentConnected | Id   | User | Host             | LoginTime           | Catalog  | Db   | Command | Time | State | QueryId
| Info             |
+------------------+------+------+------------------+---------------------+----------+------+---------+------+-------+-------------
---------------------+------------------+
| Yes              |    1 | root | 192.168.1.101:34390 | 2024-03-17 16:32:22 | internal |      | Query   |    0 | OK    |
82edc460d93f4e28-8bbed058a068e259 | show processlist |
+------------------+------+------+------------------+---------------------+----------+------+---------+------+-------+-------------
---------------------+------------------+
1 row in set (0.00 sec)
```

If you see the real client IP in the `Host` column, the verification is successful. Otherwise, you can only see the IP address of the proxy service.

At the same time, the real client IP will also be recorded in fe.audit.log.

Time Zone

Time Zone

Doris supports custom time zone settings

Basic concepts

The following two time zone related parameters exist within Doris:

- `system_time_zone` : When the server starts up, it will be set automatically according to the time zone set by the machine, and cannot be modified after it is set.
- `time_zone` : The current time zone of the cluster. This variable will be set to the same as `system_time_zone` when the cluster starts and will not be changed again unless manually changed by the user.

Specific operations

1. `SHOW VARIABLES LIKE '% time_zone%'`

   View the current time zone related configuration

2. `SET [global] time_zone = 'Asia/Shanghai';`

   This command sets the time zone at the session level. If the `global` keyword is used, Doris FE persists the parameter and it takes effect for all new sessions afterwards.

Data source

The time zone data contains the name of the time zone, the corresponding time offset, and the change of daylight saving time. On the machine where the BE is located, the source of the data is the directory returned by command `TZDIR`. If it was not supported, the directory `/usr/share/zoneinfo`.

Impact of time zone

1. functions

Includes values displayed by time functions such as `NOW()` or `CURTIME()`, and also time values in `show load`, `show backends`.

However, it does not affect the less than value of the time-type partitioned columns in `create table`, nor does it affect the display of values stored as `date/datetime` types.

Functions affected by time zone:

- `FROM_UNIXTIME`: Given a UTC timestamp, return its date and time in the time zone specified by Doris session `time_zone`. For example, when `time_zone` is `CST`, `FROM_UNIXTIME(0)` returns `1970-01-01 08:00:00`.

- `UNIX_TIMESTAMP`: Given a date and time, return its UTC timestamp in the time zone specified by Doris session `time_zone`, such as when `time_zone` is `CST` `UNIX_TIMESTAMP('1970-01-01 08:00:00')` returns `0`.
- `CURTIME`: Returns the time in the time zone specified by the current Doris session `time_zone`.
- `NOW`: Returns the date and time of the current Doris session `time_zone` specified time zone.
- `CONVERT_TZ`: Convert a datetime from one specified time zone to another.

2. Values of time types

For `DATE` and `DATETIME` types, we support time zone conversion when importing data.

- If the data has a time zone, such as "2020-12-12 12:12:12+08:00" with the current Doris `time_zone = +00:00`, then the data is imported into Doris and the actual value is "2020-12-12 04:12:12".
- If the data does not contain a time zone, such as "2020-12-12 12:12:12", the time is considered to be an absolute time and no conversion occurs.

3. Daylight Saving Time

Daylight Saving Time is essentially the actual time offset of a named time zone, which changes on certain dates.

For example, the `America/Los_Angeles` time zone contains a Daylight Saving Time adjustment that begins and ends approximately in March and November of each year. That is, the `America/Los_Angeles` actual time zone offset changes from `-08:00` to `-07:00` at the start of Daylight Savings Time in March, and from `-07:00` to `-08:00` at the end of Daylight Savings Time in November. If you do not want Daylight Saving Time to be turned on, set `time_zone` to `-08:00` instead of `America/Los_Angeles`.

Usage

Time zone values can be given in a variety of formats. The following standard formats are well supported in Doris:

1. standard named time zone formats, such as "Asia/Shanghai", "America/Los_Angeles". This format is derived from time zone data on this machine. "Etc/GMT+3" etc. also belongs to this category.
2. standard offset formats, such as "+02:30", "-10:00".(special offsets such as "+12:03" are not supported)
3. abbreviated time zone formats, currently only support:
    1. "GMT", "UTC", equivalent to "+00:00" time zone
    2. "CST", which is equivalent to the "Asia/Shanghai" time zone
4. single letter Z for Zulu time zone, equivalent to "+00:00" time zone

Besides, all parsing of alphabet is case insensitive.

Note: Some other formats are currently supported in some imports in Doris due to different implementations. **Production environments should not rely on these formats that are not listed here, and their behavior**

may change at any time, so keep an eye on the relevant changelog for version updates.

Best Practices

Time Zone Sensitive Data

The time zone issue involves three main influences:

1. session variable `time_zone` – cluster timezone
2. header `timezone` specified during import(Stream Load, Broker Load etc.) – importing timezone
3. timezone type literal “+08:00” in “2023-12-12 08:00:00+08:00” – data timezone

We can understand it as follows:

Doris is currently compatible with importing data in various time zones into Doris. Since Doris’s own `DATETIME` and other time types do not contain time zone information, and the data will not change with time zone changes after being imported, when time data is imported into Doris, it can be divided into the following two categories:

1. Absolute time

   Absolute time means that the data scene it is associated with has nothing to do with time zones. This type of data should be imported without any time zone suffix and will be stored as-is.

2. Time in a specific time zone

   The time in a specific time zone means that the data scenario it is associated with is related to the time zone. For this type of data, it should be imported with a specific time zone suffix. When imported, they will be converted to the Doris cluster `time_zone` time zone or the header `timezone` specified in Stream Load/Broker Load.

   This type of data is converted to absolute time storage in the time zone specified during import after import, so subsequent imports and queries should maintain this time zone to avoid confusion in the meaning of the data.

- For the Insert statement, we can illustrate it through the following example:

```
Doris > select @@time_zone;
+---------------+
| @@time_zone   |
+---------------+
| Asia/Shanghai |
+---------------+

Doris > insert into dt values('2020-12-12 12:12:12+02:00'); --- The imported data specifies a time zone of +02:00

Doris > select * from dt;
+---------------------+
| dt                  |
+---------------------+
| 2020-12-12 18:12:12 | --- Is converted to the Doris cluster time zone Asia/Shanghai, subsequent imports and queries
should maintain this time zone.
+---------------------+

Doris > set time_zone = 'America/Los_Angeles';
```

```
Doris > select * from dt;
+---------------------+
| dt                  |
+---------------------+
| 2020-12-12 18:12:12 | --- If time_zone is modified, the time value will not change accordingly, and its meaning during
query will be confused.
+---------------------+
```

- For import methods such as Stream Load and Broker Load, we can achieve this by specifying header `timezone`. For example, for Stream Load, we can illustrate it through the following example:

```
cat dt.csv
2020-12-12 12:12:12+02:00

curl --location-trusted -u root: \
 -H "Expect:100-continue" \
 -H "strict_mode: true" \
 -H "timezone: Asia/Shanghai" \
 -T dt.csv -XPUT \
 http://127.0.0.1:8030/api/test/dt/_stream_load
```

```
Doris > select @@time_zone;
+---------------+
| @@time_zone   |
+---------------+
| Asia/Shanghai |
+---------------+

Doris > select * from dt;
+---------------------+
| dt                  |
+---------------------+
| 2020-12-12 18:12:12 | --- Is converted to the Doris cluster time zone Asia/Shanghai, subsequent imports and queries
should maintain this time zone.
+---------------------+
```

  - In import methods such as Stream Load and Broker Load, the header `timezone` will overwrite the Doris cluster `time_zone`, so it should be consistent during import.
  - In import methods such as Stream Load and Broker Load, the header `timezone` will affect the functions used in import conversion.
  - If the header `timezone` is not specified when importing, the East Eighth Zone will be used by default.

To sum up, the best practice for dealing with time zone issues is:

> **Best Practices**
>
> 1. Confirm the time zone represented by the cluster and set `time_zone` before use, and do not change it after that.
> 2. Set header `timezone` to be consistent with cluster `time_zone` when importing.

> 3. For absolute time, import without time zone suffix; for time with time zone, import with specific time zone suffix, and it will be converted to Doris `time_zone` time zone after import.

Daylight Saving Time

The start and end times for Daylight Saving Time are taken from the current time zone data source and may not necessarily correspond exactly to the actual officially recognised times for the current year's time zone location. This data is maintained by ICANN. If you need to ensure that Daylight Saving Time behaves as specified for the current year, please make sure that data source selected by Doris is the latest ICANN-published time zone data. See below for download access.

Information Update

Real-world time zone and daylight saving time data may change from time to time for various reasons, and IANA periodically records these changes and updates the corresponding time zone files. If you want the time zone information in Doris to be up to date with the latest IANA data, do one of the followings:

1. Use the Package Manager to update

Depending on the package manager used by the current operating system, you can update the time zone data directly using the corresponding command:

```
# yum
> sudo yum update tzdata
# apt
> sudo apt update tzdata
```

The data updated in this way is located under the system `$TZDIR` (typically `usr/share/zoneinfo`).

2. pull the IANA time zone database manually (recommended)

Most Linux distributions have a package manager where tzdata is not synchronised in a timely manner. If the accuracy of the time zone data is important, you can pull the data published by IANA on a regular basis:

```
wget https://www.iana.org/time-zones/repository/tzdb-latest.tar.lz
```

Then generate the specific zoneinfo data according the README file in the extracted folder. The generated data should be copied to override `$TZDIR` folder.

Please note that all the above operations **must** be restarted on the corresponding BE to take effect after they are done on the BE machine.

Extended Reading

- List of tz database time zones
- IANA Time Zone Database
- The tz-announce Archives

FQDN

This article introduces how to enable the use of Apache Doris based on FQDN (Fully Qualified Domain Name). FQDN is the complete domain name of a specific computer or host on the Internet.

After Doris supports FQDN, communication between nodes is entirely based on FQDN. When adding various types of nodes, the FQDN should be directly specified. For example, the command to add a BE node is' ALTER SYSTEM ADD BACKEND "be_host:heartbeat_service_port", 'be_host' was previously the IP address of the BE node. After starting the FQDN, be_host should specify the FQDN of the BE node.

Preconditions

1. fe.conf file set `enable_fqdn_mode = true`.
2. All machines in the cluster must be configured with a host name.
3. The IP address and FQDN corresponding to other machines in the cluster must be specified in the '/etc/hosts' file for each machine in the cluster.
4. The /etc/hosts file cannot have duplicate IP addresses.

Best Practices

Enable FQDN for new cluster

1. Prepare machines, for example, if you want to deploy a cluster of 3FE 3BE, you can prepare 6 machines.

2. Each machine returns unique results when executing 'host'. Assuming that the execution results of six machines are fe1, fe2, fe3, be1, be2, and be3, respectively.

3. Configure the real IPs corresponding to 6 FQDNs in the '/etc/hosts' of 6 machines, for example:

```
172.22.0.1 fe1
172.22.0.2 fe2
172.22.0.3 fe3
172.22.0.4 be1
172.22.0.5 be2
172.22.0.6 be3
```

4. Verification: you can 'ping fe2' on FE1, and can resolve the correct IP address and ping it, indicating that the network environment is available.

5. fe.conf settings for each FE node `enable_ fqdn_ mode = true`.

6. Refer toStandard deployment

7. Select several machines to deploy broker on six machines as needed, and execute `ALTER SYSTEM ADD BROKER broker_name "fe1:8000","be1:8000",...;`.

Deployment of Doris for K8S

After an unexpected restart of the Pod, K8s cannot guarantee that the Pod's IP will not change, but it can ensure that the domain name remains unchanged. Based on this feature, when Doris enables FQDN, it can ensure that the Pod can still provide services normally after an unexpected restart.

Please refer to the method for deploying Doris in K8sKubernetes Deployment

Server change IP

After deploying the cluster according to 'Enable FQDN for new cluster', if you want to change the IP of the machine, whether it is switching network cards or replacing the machine, you only need to change the '/etc/hosts' of each machine.

Enable FQDN for old cluster

Precondition: The current program supports the syntax `ALTER SYSTEM MODIFY FRONTEND "<fe_ip>:<edit_log_port>" HOSTNAME "<fe_hostname>"`, If not, upgrade to a version that supports the syntax

> Note that.
>
> At least three followers are required to perform the following operations, otherwise the cluster may not start properly

Next, follow the steps below:

1. Perform the following operations on the Follower and Observer nodes one by one (and finally on the Master node):
    1. Stop the node.
    2. Check if the node has stopped. Execute 'show frontends' through the MySQL client to view the Alive status of the FE node until it becomes false
    3. set FQDN for node: `ALTER SYSTEM MODIFY FRONTEND "<fe_ip>:<edit_log_port>" HOSTNAME "<fe_hostname>"`(After stopping the master, a new master node will be selected and used to execute SQL statements)
    4. Modify node configuration. Modify the 'conf/fe.conf' file in the FE root directory and add the configuration: `enable_fqdn_mode = true`. If the newly stopped node cannot be started properly after adding the configuration to the corresponding fe.config, please add the configuration 'enable_fqdn_mode=true' to all fe.config before starting the just stopped fe node.
    5. Start the node.
2. Enabling FQDN for BE nodes only requires executing the following commands through MySQL, and there is no need to restart BE.

    `ALTER SYSTEM MODIFY BACKEND "<backend_ip>:<HeartbeatPort>" HOSTNAME "<be_hostname>"`,If you don't know the number of HeartbeatPort, please use the command `show backends` to help find this port.

Common problem

- Configuration item enable_ fqdn_ Can the mode be changed freely?

  Cannot be changed arbitrarily. To change this configuration, follow the ‘Enable FQDN for old cluster’ procedure.

Overview | Workload Management

Overview

Workload management is a very important feature of Doris, playing a critical role in the overall system management. Through reasonable workload management strategies, resource utilization can be optimized, system stability enhanced, and response time reduced. It has the following abilities:

- Resource Isolation: By dividing into multiple groups and setting resource (CPU, Memory, IO) limits for each group, it ensures that there is no interference between multiple users or different tasks (such as read and write operations) of the same user.
- Concurrency Control and Queuing: It can limit the number of tasks that can be executed simultaneously in the entire cluster. When threshold is exceeded, tasks will be queued.
- Circuit Breaker: During query planning phase or execution phase, tasks can be automatically cancelled based on conditions such as the estimated number of partitions to be read, the amount of data to be scanned, the allocated memory size, and the execution time. This avoids unreasonable tasks from occupying too many system resources.

Resource Isolation Methods

Doris can divide resource in the following three ways:

- Resource Group: Divide multiple BE processes into groups setting tag for each BE.
- Workload Group: The resource (CPU, Memory, IO) within a BE are divided into multiple resource groups through Cgroup, enabling more fine-grained resource isolation.

The following table records the characteristics and advantageous scenarios of different resource partitioning methods:

Resource Isolation Method	Isolation Granularity	Soft/Hard Limit	Cross Resource Group Query
Resource Group	BE node level, with complete resource isolation, can isolate BE failures	Hard limit	Not support. And it is necessary to ensure that at least one copy of data is stored within the resource group.
Workload Group	Isolation within BE process; cannot isolate BE failures	Both hard and soft limit	Support

Soft Limit and Hard Limit

- Hard Limit: The hard limit refers to the absolute upper limit of resource usage that tenants cannot exceed. Once the hard limit is reached, resource requests for the excess part will be rejected. Hard limit

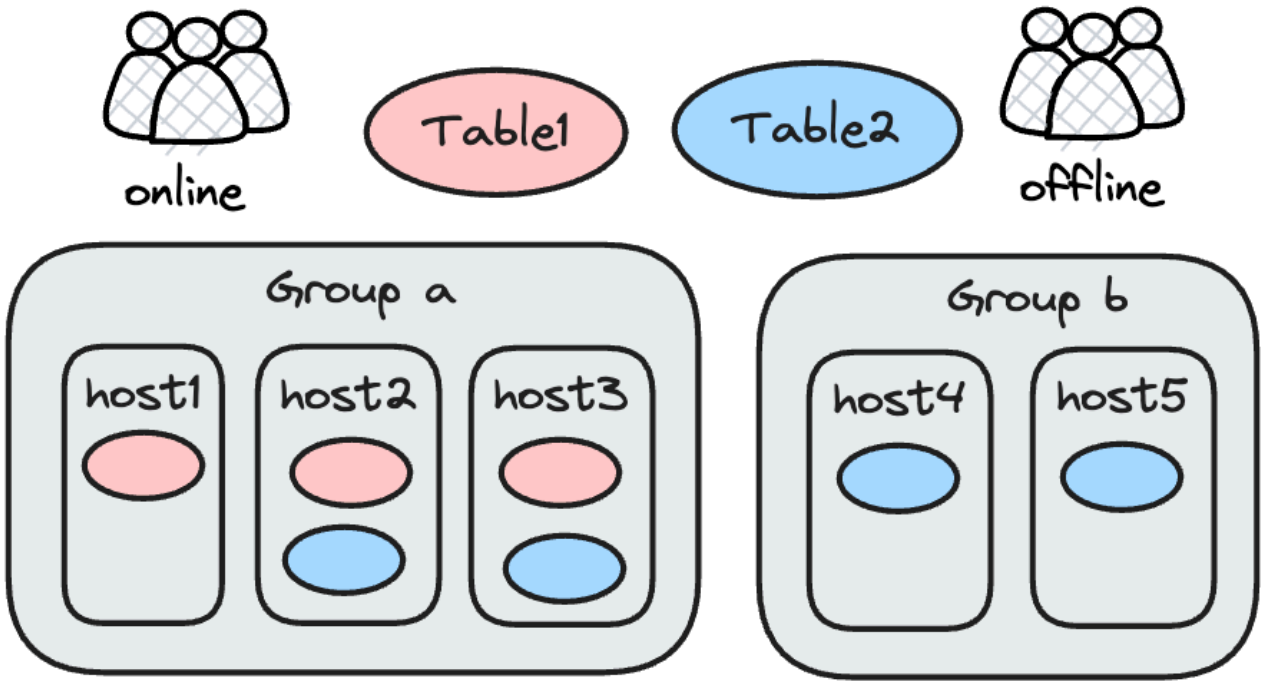
are generally used to prevent the depletion of resources within the cluster or resource preemption between different businesses, ensuring the stability and performance of the cluster.

- Soft Limit: The soft limit is a resource limit that can be exceeded, usually representing the recommended upper limit of resource usage. When the system is not busy, if a tenant requests more resources than the soft limit, it can borrow resources from other groups. When the system is busy and there is resource contention, if a tenant requests resources exceeding the soft limit, it will not be able to obtain additional resources.

When using the Resource Group / Compute Group method to partition resources, only the hard limit mode is supported. When using the Workload Group method to partition resources, both the soft limit and hard limit of Workload Group are supported. The soft limit of Workload Group is usually used for sudden resource control, such as temporary query peaks or short-term increases in data writing.

Resource Group

Resource Group is a mechanism under the compute-storage integration architecture that achieves physical isolation between different workloads. Its basic principle is illustrated in this diagram:



Resource Group

- By using tags, BEs are divided into different groups, each identified by the tag's name. For example, in the diagram above, host1, host2, and host3 are all set to group a, while host4 and host5 are set to group b.
- Different replicas of a table are placed in different groups. For instance, in the diagram above, table1 has 3 replicas, all located in group a, while table2 has 4 replicas, with 2 in group a and 2 in group b.
- During queries, different Resource Groups are used based on the user. For example, online users can only access data on host1, host2, and host3, so they can access both table1 and table2. However, offline users can only access host4 and host5, so they can only access data from table2. Since table1 does not have corresponding replicas in group b, accessing it would result in an error.

Essentially, a Resource Group is a placement strategy for table replicas, so it has the following advantages and limitations:

- Different Resource Groups use different BEs, so they are completely isolated from each other. Even if a BE within a group fails, it will not affect queries in other groups. Since data loading require multiple replicas to succeed, if the remaining number of replicas does not meet the quorum, the data loading will still fail.
- Each Resource Group must have at least one replica of each table. For example, if you want to establish 5 Resource Groups and each group may access all tables, then each table needs 5 replicas, which can

result in significant storage costs.

Typical Use Cases

- Read-write isolation: A cluster can be divided into two Resource Groups, with an Offline Resource Group for executing ETL jobs and an Online Resource Group for handling online queries. Data is stored with 3 replicas, with 2 replicas in the Online Resource Group and 1 replica in the Offline Resource Group. The Online Resource Group is primarily used for high-concurrency, low-latency online data services, while large queries or offline ETL operations can be executed using nodes in the Offline Resource Group. This allows for the provision of both online and offline services within a unified cluster.
- Isolation between different businesses: When data is not shared between multiple businesses, a Resource Group can be assigned to each business, ensuring no interference between them. This effectively consolidates multiple physical clusters into a single large cluster for management.
- Isolation between different users: For example, if there is a business table within a cluster that needs to be shared among all three users, but it is desirable to minimize resource contention between them, we can create 3 replicas of the table, stored in 3 different Resource Groups, and bind each user to a specific Resource Group.

Configure Resource Group

Setting Tags for BEs

Assuming the current Doris cluster has 6 BE nodes, named host[1-6]. Initially, all BE nodes belong to a default resource group (Default).

We can use the following commands to divide these 6 nodes into 3 resource groups: group_a, group_b, and group_c.

```
alter system modify backend "host1:9050" set ("tag.location" = "group_a");
alter system modify backend "host2:9050" set ("tag.location" = "group_a");
alter system modify backend "host3:9050" set ("tag.location" = "group_b");
alter system modify backend "host4:9050" set ("tag.location" = "group_b");
alter system modify backend "host5:9050" set ("tag.location" = "group_c");
alter system modify backend "host6:9050" set ("tag.location" = "group_c");
```

```
Here, we will form Resource Group group_a with host[1-2], Resource Group group_b with host[3-4], and Resource Group group_c with
host[5-6].
```

> Note: A BE can only belong to one Resource Group.

Redistribution data by Resource Group

After dividing the resource groups, you can distribute different replicas of user data across different resource groups. Assuming we have a user table named UserTable, and we want to store one replica in each of the three resource groups. This can be achieved through the following table creation statement:

```
create table UserTable
(k1 int, k2 int)
distributed by hash(k1) buckets 1
properties(
    "replication_allocation"="tag.location.group_a:1, tag.location.group_b:1, tag.location.group_c:1"
)
```

In this way, the data in the UserTable will be stored in three replicas, each located on nodes within the resource groups group_a, group_b, and group_c, respectively.

The following diagram demonstrates the current division of nodes and data distribution:



When a database contains a very large number of tables, modifying the distribution strategy for each table can be cumbersome. Therefore, Doris also supports setting a unified data distribution strategy at the database level, but the settings for individual tables have higher priority than those at the database level. For example, consider a database db1 with four tables: table1 requires a replica distribution strategy of group_a:1,group_b:2, while table2, table3, and table4 require a strategy of group_c:1,group_b:2.

```
To create db1 with a default distribution strategy, you can use the following statement:
```

```
CREATE DATABASE db1 PROPERTIES (
"replication_allocation" = "tag.location.group_c:1, tag.location.group_b:2"
)
```

```
Create table1 with a specific distribution strategy:
```

```
CREATE TABLE table1
(k1 int, k2 int)
distributed by hash(k1) buckets 1
properties(
"replication_allocation"="tag.location.group_a:1, tag.location.group_b:2"
)
```

For table2, table3, and table4, you do not need to specify replication_allocation in their creation statements, as they will inherit the database-level default strategy.

:::caution Changing the replica distribution strategy at the database level will not affect existing tables. :::

Setting Resource Groups for Users

You can use the following statements to restrict users' access to specific resource groups. For example, user1 can only use nodes in the group_a resource group, user2 can only use group_b, and user3 can use all three resource groups:

```
set property for 'user1' 'resource_tags.location' = 'group_a';
set property for 'user2' 'resource_tags.location' = 'group_b';
set property for 'user3' 'resource_tags.location' = 'group_a, group_b, group_c';
```

After setting, when user1 queries the UserTable, it will only access data replicas on nodes in the group_a resource group and use computing resources from this group. User3's queries can use replicas and computing resources from any resource group.

> Note: By default, the resource_tags.location property for users is empty. In versions before 2.0.2, users are not restricted by tags and can use any resource group. In versions 2.0.3 and later, ordinary users can only use the default resource group by default. Root and admin users can use any resource group.

:::caution Caution: After modifying the resource_tags.location property, users need to re-establish connections for the changes to take effect. :::

Resource Group Allocation for Data Loading Jobs

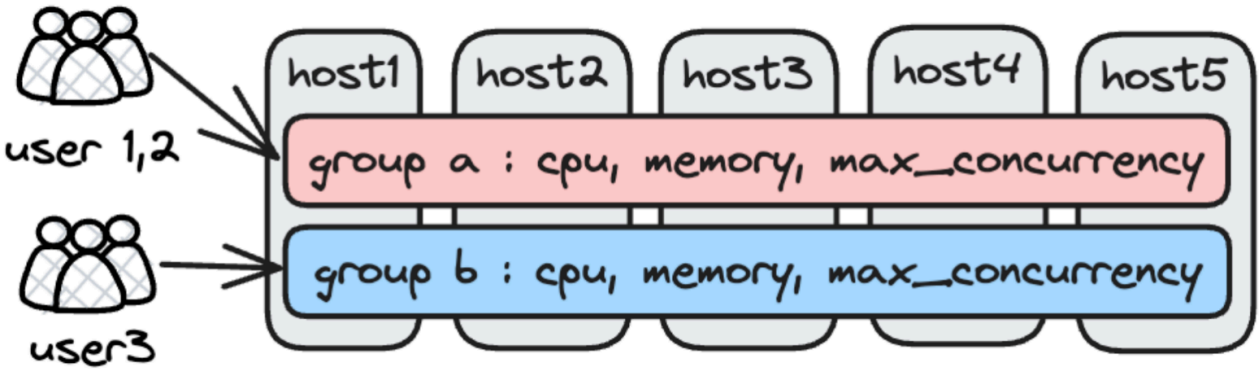
The resource usage for data loading jobs (including insert, broker load, routine load, stream load, etc.) can be divided into two parts:

- Computing Part: responsible for reading data sources, data transformation, and distribution.
- Writing Part: responsible for data encoding, compression, and writing to disk.

Since writing resources must be on nodes where data replicas are located, and computing resources can be allocated from any node, Resource Groups can only restrict the resources used for the computing part in data loading scenarios.

Workload Group

Workload Group is an in-process mechanism for isolating workloads. It achieves resource isolation by finely partitioning or limiting resources (CPU, IO, Memory) within the BE process. Its principle is illustrated in the diagram below:



workload_group

The currently supported isolation capabilities include:

- Managing CPU resources, with support for both cpu hard limit and cpu soft limit;
- Managing memory resources, with support for both memory hard limit and memory soft limit;
- Managing IO resources, including IO generated by reading local and remote files.

> **Workload Group provides in-process resource isolation capabilities, which differ from inter-process resource isolation methods (such as Resource Group and Compute Group) in the following ways:**
>
> 1. In-process resource isolation cannot achieve complete isolation. For example, when high-load and low-load queries run within the same process, even if CPU usage for the high-load group is restricted using Workload Group to keep overall CPU usage within a reasonable range, the latency of the low-load group may still be affected. However, it will perform better compared to having no CPU control at all. This limitation arises because certain shared components within the process, such as common caches and shared RPC thread pools, are difficult to isolate entirely.
> 2. The choice of a resource isolation strategy depends on the trade-off between isolation and cost. If some degree of latency can be tolerated while prioritizing lower costs, the Workload Group isolation approach may be suitable. On the other hand, if complete isolation is required and higher costs are acceptable, an inter-process resource isolation approach (i.e., placing isolated workloads in separate processes) should be considered. For example, using Resource Group or Compute Group to allocate high-priority workloads to independent BE nodes can achieve a more thorough isolation. :::

Version Notes

- The Workload Group feature has been available since Doris 2.0. In Doris 2.0, the Workload Group feature does not rely on CGroup, but starting with Doris 2.1, it requires CGroup.
- Upgrading from Doris 1.2 to 2.0: It is recommended to enable the Workload Group feature only after the entire cluster has been upgraded. If only some follower FE nodes are upgraded, queries on the upgraded follower FE nodes may fail due to the absence of Workload Group metadata on the non-upgraded FE nodes.
- Upgrading from Doris 2.0 to 2.1: Since the Workload Group feature in Doris 2.1 relies on CGroup, you need to configure the CGroup environment before upgrading to Doris 2.1.

Configuring Workload Group

Setting Up the CGroup Environment

Workload Group supports managing CPU, memory, and IO. CPU management relies on the CGroup component. To use Workload Group for CPU resource management, you must first configure the CGroup environment.

The following are the steps for configuring the CGroup environment:

1. First, verify whether the node where the BE is located has CGroup installed. If the output includes cgroup, it indicates that CGroup V1 is installed in the current environment. If it includes cgroup2, it indicates that CGroup V2 is installed. You can determine which version is active in the next step.

```
cat /proc/filesystems | grep cgroup
nodev   cgroup
nodev   cgroup2
nodev   cgroupfs
```

2. The active CGroup version can be confirmed based on the path name.

```
If this path exists, it indicates that CGroup V1 is currently active.
/sys/fs/cgroup/cpu/


If this path exists, it indicates that CGroup V2 is currently active.
/sys/fs/cgroup/cgroup.controllers
```

3. Create a directory named doris under the CGroup path. The directory name can be customized by the user.

```
If using CGroup V1, create the directory under the cpu directory.
mkdir /sys/fs/cgroup/cpu/doris


If using CGroup V2, create the directory directly under the cgroup directory.
mkdir /sys/fs/cgroup/doris
```

4. Ensure that the Doris BE process has read, write, and execute permissions for this directory.

```
// If using CGroup V1, the command is as follows:
// 1. Modify the directory's permissions to be readable, writable, and executable.
chmod 770 /sys/fs/cgroup/cpu/doris
// 2. Change the ownership of this directory to the doris account.
chown -R doris:doris /sys/fs/cgroup/cpu/doris


// If using CGroup V2, the command is as follows:
// 1.Modify the directory's permissions to be readable, writable, and executable.
chmod 770 /sys/fs/cgroup/doris
// 2. Change the ownership of this directory to the doris account.
chown -R doris:doris /sys/fs/cgroup/doris
```

5. If the current environment is using CGroup v2, the following steps are required. If it is CGroup v1, this step can be skipped.

- Modify the permissions of the cgroup.procs file in the root directory. This is necessary because CGroup v2 has stricter permission controls, and write permissions to the cgroup.procs file in the root directory are required to move processes between CGroup directories.

```
chmod a+w /sys/fs/cgroup/cgroup.procs
```

- In CGroup v2, the cgroup.controllers file lists the available controllers for the current directory, and the cgroup.subtree_control file lists the controllers available for the subdirectories. Therefore, it is necessary to check if the doris directory has the cpu controller enabled. If the cgroup.controllers file in the doris directory does not include cpu, it means the cpu controller is not enabled. You can enable it by executing the following command in the doris directory. This command works by modifying the cgroup.subtree_control file in the parent directory to allow the doris directory to use the cpu controller.

```
// After running this command, you should be able to see the cpu.max file in the doris directory,
// and the output of cgroup.controllers should include cpu.
// If the command fails, it means that the parent directory of doris also does not have the cpu controller enabled,
// and you will need to enable the cpu controller for the parent directory.
echo +cpu > ../cgroup.subtree_control
```

6. Modify the BE configuration to specify the path of the cgroup.

```
If using CGroup V1, the configuration path is as follows:
doris_cgroup_cpu_path = /sys/fs/cgroup/cpu/doris

If using CGroup V2, the configuration path is as follows:
doris_cgroup_cpu_path = /sys/fs/cgroup/doris
```

7. Restart the BE, and in the log (be.INFO), the phrase ‘add thread xxx to group’ indicates that the configuration was successful.

> **1. It is recommended to deploy only one BE per machine, as the current Workload Group feature does not support deploying multiple BE instances on a single machine.**

2. After a machine is restarted, all configurations under the CGroup path will be cleared. To persist the CGroup configuration, you can use systemd to set the operation as a custom system service, so that the creation and authorization operations can be automatically performed each time the machine restarts.
3. If using CGroup within a container, the container must have permission to operate on the host machine. :::

Considerations for Using Workload Group in Containers

Workload's CPU management is based on CGroup. If you want to use Workload Group inside a container, the container needs to be started in privileged mode so that the BE process inside the container has permission to read and write CGroup files on the host machine.

When BE runs inside a container, the CPU resource usage for Workload Group is partitioned based on the available resources of the container. For example, if the host machine has 64 cores and the container is allocated 8 cores, and the Workload Group is configured with a 50% CPU hard limit, the actual available CPU cores for the Workload Group will be 4 (8 cores * 50%).

The memory and IO management functions of Workload Group are implemented internally by Doris and do not rely on external components, so there is no difference in deployment between containers and physical machines.

If you want to use Doris on K8S, it is recommended to deploy it using the Doris Operator, which can shield underlying permission issues.

Create Workload Group

```
mysql [information_schema]>create workload group if not exists g1
    -> properties (
    ->     "cpu_share"="1024"
    -> );
Query OK, 0 rows affected (0.03 sec)
```

You can refer to CREATE-WORKLOAD-GROUP。

The CPU limit configured at this point is a soft limit. Since version 2.1, Doris will automatically create a group named normal, which cannot be deleted.

Workload Group Properties

Property	Data type	Default value	Description
cpu_share	int	[1, 10000]	Optional, effective under CPU soft limit mode. The valid range of values depends on the version of CGroup being used, which is described in detail later. cpu_share represents the weight of CPU time that the Workload Group can acquire; the larger the value, the more CPU time it can obtain. For example, if the user creates three Workload Groups, g-a, g-b, and g-c, with cpu_share values of 10, 30, and 40 respectively, and at a certain point g-a and g-b are running tasks while g-c has no tasks, g-a will receive 25% (10 / (10 + 30)) of the CPU resources, and g-b will receive 75% of the CPU resources. If only one Workload Group is running in the system, regardless of the cpu_share value, it will be able to acquire all the CPU resources.
memory_limit	float	(10%, 100%]	Optional. Enabling memory hard limit represents the maximum available memory percentage for the current Workload Group. The default value means no memory limit is applied. The cumulative value of memory_limit for all Workload Groups cannot exceed 100%, and it is typically used in conjunction with the enable_memory_overcommit attribute. For example, if a machine has 64GB of memory and the memory_limit for a Workload Group is set to 50%, the actual physical memory available for that group would be 64GB * 90% * 50% = 28.8GB, where 90% is the default value for the available memory configuration of the BE process.

Property	Default value	Description
enable_memory_overcommit	Bool, true or false	Optional. Used to control whether the memory limit for the current Workload Group is a hard limit or a soft limit, with the default set to true. If set to false, the Workload Group will have hard memory limit, and when the system detects that the memory usage exceeds the limit, it will immediately cancel the tasks with the highest memory usage within the group to release the excess memory. If set to true, the Workload Group will have soft memory limit. If there is free memory available, the Workload Group can continue using system memory even after exceeding the memory_limit. When the system's total memory is under pressure, the system will cancel the tasks with the highest memory usage within the group and release some of the excess memory to alleviate system memory pressure. It is recommended that the total memory_limit of all Workload Groups be kept below 100% to reserve memory for other components of the BE process.
cpu_hard_limit	Int, [1-100%]	Optional. Effective under CPU hard limit mode, it represents the maximum CPU percentage a Workload Group can use. Regardless of whether the machine's CPU resources are fully utilized, the Workload Group's CPU usage cannot exceed the cpu_hard_limit. The cumulative value of cpu_hard_limit for all Workload Groups cannot exceed 100%. This attribute was introduced in version 2.1 and is not supported in version 2.0.

Property	Data type	Default value	Description
max_concurrency	Integer	2[1044, 7241848736487]	Optional. Specifies the maximum query concurrency. The default value is the maximum value of an integer, meaning no concurrency limit. When the number of running queries reaches the maximum concurrency, new queries will enter a queue.
max_queue_size	Integer	0[0, 2147483647]	Optional. Specifies the length of the query waiting queue. When the queue is full, new queries will be rejected. The default value is 0, which means no queuing. If the queue is full, new queries will fail directly.
queue_timeout	Integer	0[0, 2147483647]	Optional. Specifies the maximum waiting time for a query in the waiting queue, in milliseconds. If the query's waiting time in the queue exceeds this value, an exception will be thrown directly to the client. The default value is 0, meaning no queuing; queries will immediately fail upon entering the queue.

Property	Data type	Default value	Description
scan_thread_num	Integer, [-1, 2147483647]	-1	Optional. Specifies the number of threads used for scanning in the current Workload Group. When this property is set to -1, it means it is not active, and the actual scan thread num on the BE will default to the doris_scanner_thread_pool_thread_num configuration in the BE.
max_remote_scan_thread_num	Integer, [-1, 2147483647]	-1	Optional. Specifies the maximum number of threads in the scan thread pool for reading external data sources. When this property is set to -1, the actual number of threads is determined by the BE, typically based on the number of CPU cores.

Property	Default value	Description
min_remote_scan_thread_num	Int type, -1, [1, 2147483647]	Optional. Specifies the minimum number of threads in the scan thread pool for reading external data sources. When this property is set to -1, the actual number of threads is determined by the BE, typically based on the number of CPU cores.

Property	Default value	Description
read_bytes_per_second	Int32 [-1, 9223372036854775807]	Optional. Specifies the maximum I/O throughput when reading internal tables in Doris. The default value is -1, meaning no I/O bandwidth limit is applied. It is important to note that this value is not tied to individual disks but to directories. For example, if Doris is configured with two directories to store internal table data, the maximum read I/O for each directory will not exceed this value. If both directories are placed on the same disk, the maximum throughput will be doubled (i.e., 2 times read_bytes_per_second). The file directory for spill disk is also subject to this limit.

Property	Default value	Description
remote_read_bytes_per_second	Integer, [1, 9223372036854775807]	Optional. Specifies the maximum I/O throughput when reading external tables in Doris. The default value is -1, meaning no I/O bandwidth limit is applied.

1. Currently, the simultaneous use of both cpu hard limit and cpu soft limit is not supported.

At any given time, a cluster can only have either a soft limit or a hard limit. The method for switching between them will be described later.

2. All properties are optional, but at least one property must be specified when creating a Workload Group.

3. It is important to note that the default values for CPU soft limits differ between CGroup v1 and CGroup v2. The default CPU soft limit for CGroup v1 is 1024, with a valid range from 2 to 262144, while the default for CGroup v2 is 100, with a valid range from 1 to 10000. If a value outside the range is set for the soft limit, it may cause the CPU soft limit

modification to fail in BE. If the default value of 100 from CGroup v2 is applied in a CGroup v1 environment, it could result in this Workload Group having the lowest priority on the machine.

4. CPU soft limit is relative weight allocation: it does not restrict the absolute CPU time consumed by a task. Instead, it allocates CPU time slices in proportion to the assigned weights when CPU resources are contended (overall utilization ≥ 100%). If the CPU is idle, tasks can consume all available CPU resources without any restrictions. Therefore, it is extremely difficult to evaluate its actual effectiveness, and it is recommended to use CPU hard limit in production environments. :::

Set Workload Group for user

Before binding a user to a specific Workload Group, it is necessary to ensure that the user has the necessary permissions for the Workload Group. You can use the user to query the information_schema.workload_groups system table, and the result will show the Workload Groups that the current user has permission to access. The following query result indicates that the current user has access to the g1 and normal Workload Groups:

```
SELECT name FROM information_schema.workload_groups;
+--------+
| name   |
+--------+
| normal |
| g1     |
+--------+
```

If the g1 Workload Group is not visible, you can use the ADMIN account to execute the GRANT statement to authorize the user. For example:

```
GRANT USAGE_PRIV ON WORKLOAD GROUP 'g1' TO 'user_1'@'%';
```

This statement means granting the user_1 the permission to use the Workload Group named g1. More details can be found in grant。

Two ways to bind Workload Group to user 1. By setting the user property, you can bind the user to a default Workload Group. The default is normal. It's important to note that the value here cannot be left empty; otherwise, the statement will fail.

```
set property 'default_workload_group' = 'g1';
```

After executing this statement, the current user's queries will default to using the 'g1' Workload Group.

2. By specifying the Workload Group through a session variable, the default is empty:

```
set workload_group = 'g1';
```

When both methods are used to specify a Workload Group for the user, the session variable takes priority over the user property.

Show Workload Group

1. You can use the SHOW statement to view the Workload Group:

```
show workload groups;
```

More details can be found in SHOW-WORKLOAD-GROUPS

2. You can view the Workload Group through the system table:

```
mysql [information_schema]>select * from information_schema.workload_groups where name='g1';
+-------+------+-----------+--------------+-------------------------+-----------------+----------------+-------
--------+----------------+-----------------+----------------------------+----------------------------+----------
------------+-----------------------+------+-----------------------+-----------------------------+
| ID    | NAME | CPU_SHARE | MEMORY_LIMIT | ENABLE_MEMORY_OVERCOMMIT | MAX_CONCURRENCY | MAX_QUEUE_SIZE |
QUEUE_TIMEOUT | CPU_HARD_LIMIT | SCAN_THREAD_NUM | MAX_REMOTE_SCAN_THREAD_NUM | MIN_REMOTE_SCAN_THREAD_NUM |
MEMORY_LOW_WATERMARK | MEMORY_HIGH_WATERMARK | TAG  | READ_BYTES_PER_SECOND | REMOTE_READ_BYTES_PER_SECOND |
+-------+------+-----------+--------------+-------------------------+-----------------+----------------+-------
--------+----------------+-----------------+----------------------------+----------------------------+----------
------------+-----------------------+------+-----------------------+-----------------------------+
| 14009 | g1   |      1024 | -1           | true                    |      2147483647 |              0 |
0 | -1             |              -1 |                         -1 |                          -1 | 50%
| 80%                  |      |                   -1 |                            -1 |
+-------+------+-----------+--------------+-------------------------+-----------------+----------------+-------
--------+----------------+-----------------+----------------------------+----------------------------+----------
------------+-----------------------+------+-----------------------+-----------------------------+
1 row in set (0.05 sec)
```

Alter Workload Group

```
mysql [information_schema]>alter workload group g1 properties('cpu_share'='2048');
Query OK, 0 rows affected (0.00 sec

mysql [information_schema]>select cpu_share from information_schema.workload_groups where name='g1';
+-----------+
| cpu_share |
+-----------+
|      2048 |
+-----------+
1 row in set (0.02 sec)
```

More details can be found in ALTER-WORKLOAD-GROUP

Drop Workload Group

```
mysql [information_schema]>drop workload group g1;
Query OK, 0 rows affected (0.01 sec)
```

More details can be found inDROP-WORKLOAD-GROUP

Explanation of Switching Between CPU Soft and Hard Limit Modes

Currently, Doris does not support running both CPU soft and hard limits simultaneously. At any given time, a Doris cluster can only operate in either CPU soft limit or CPU hard limit mode. Users can switch between these two modes, and the switching method is as follows:

1 If the current cluster configuration is set to the default CPU soft limit and you wish to change it to CPU hard limit, you need to modify the cpu_hard_limit parameter of the Workload Group to a valid value.

```
alter workload group test_group properties ( 'cpu_hard_limit'='20%' );
```

All Workload Groups in the cluster need to be modified, and the cumulative value of cpu_hard_limit for all Workload Groups cannot exceed 100%.

Since CPU hard limits cannot automatically have a valid value, simply enabling the switch without modifying the property will prevent the CPU hard limit from taking effect.

2 Enable the CPU hard limit on all FE nodes

```
1 Modify the configuration in the fe.conf file on the disk.
experimental_enable_cpu_hard_limit = true


2 Modify the configuration in memory.
ADMIN SET FRONTEND CONFIG ("enable_cpu_hard_limit" = "true");
```

If the user wishes to switch from CPU hard limit back to CPU soft limit, they need to set the value of enable_cpu_hard_limit to false on all FE nodes. The CPU soft limit property cpu_share will default to a valid value of 1024 (if it was not previously specified). Users can adjust the cpu_share value based on the priority of the group.

Testing

Memory hard limit

Adhoc-type queries typically have unpredictable SQL inputs and uncertain memory usage, which poses the risk of a few queries consuming a large amount of memory. These types of workloads can be allocated to a separate group, and by using the Workload Group's memory hard limit feature, it helps prevent sudden large queries from consuming all memory, which could cause other queries to run out of available memory or result in OOM (Out of Memory) errors. When the memory usage of this Workload Group exceeds the configured hard limit, the system will kill queries to release memory, preventing the process from running out of memory.

Testing environment

1 FE, 1 BE, with BE configured to 96 cores and 375GB of memory.

The test dataset is clickbench, and the testing method involves using JMeter to run query Q29 with three concurrent executions.

Test without enabling memory hard limit for Workload Group

1. Check the memory usage of the process. The fourth column in the ps command output represents the physical memory usage of the process, in kilobytes (KB). It shows that under the current test load, the process uses approximately 7.7GB of memory.

```
[ ~]$ ps -eo pid,comm,%mem,rss | grep 1407481
1407481 doris_be         2.0 7896792
[ ~]$ ps -eo pid,comm,%mem,rss | grep 1407481
1407481 doris_be         2.0 7929692
[ ~]$ ps -eo pid,comm,%mem,rss | grep 1407481
1407481 doris_be         2.0 8101232
```

2. Use Doris system tables to check the current memory usage of the Workload Group. The memory usage of the Workload Group is approximately 5.8GB.

```
mysql [information_schema]>select MEMORY_USAGE_BYTES / 1024/ 1024 as wg_mem_used_mb from
workload_group_resource_usage where workload_group_id=11201;
+-------------------+
| wg_mem_used_mb    |
+-------------------+
| 5797.524360656738 |
+-------------------+
1 row in set (0.01 sec)

mysql [information_schema]>select MEMORY_USAGE_BYTES / 1024/ 1024 as wg_mem_used_mb from
workload_group_resource_usage where workload_group_id=11201;
+-------------------+
| wg_mem_used_mb    |
+-------------------+
| 5840.246627807617 |
+-------------------+
1 row in set (0.02 sec)

mysql [information_schema]>select MEMORY_USAGE_BYTES / 1024/ 1024 as wg_mem_used_mb from
workload_group_resource_usage where workload_group_id=11201;
+-------------------+
| wg_mem_used_mb    |
+-------------------+
| 5878.394917488098 |
+-------------------+
1 row in set (0.02 sec)
```

Here, we can see that the process memory usage is typically much larger than the memory usage of a Workload Group, even if only one Workload Group is running. This is because the Workload Group only tracks the memory used by queries and loads The memory used by other components within the process, such as metadata and various caches, is not counted as part of the Workload Group's memory usage, nor is it managed by the Workload Group.

Test with the memory hard limit for Workload Group enabled 1. Execute the SQL command to modify the memory configuration.

```
```sql
alter workload group g2 properties('memory_limit'='0.5%');
alter workload group g2 properties('enable_memory_overcommit'='false');
```
```

2. Run the same test and check the memory usage in the system table; the memory usage is around 1.5G.

```
mysql [information_schema]>select MEMORY_USAGE_BYTES / 1024/ 1024 as wg_mem_used_mb from
workload_group_resource_usage where workload_group_id=11201;
+--------------------+
| wg_mem_used_mb     |
+--------------------+
| 1575.3877239227295 |
+--------------------+
1 row in set (0.02 sec)

mysql [information_schema]>select MEMORY_USAGE_BYTES / 1024/ 1024 as wg_mem_used_mb from
workload_group_resource_usage where workload_group_id=11201;
+------------------+
| wg_mem_used_mb   |
+------------------+
| 1668.77405834198 |
+------------------+
1 row in set (0.01 sec)

mysql [information_schema]>select MEMORY_USAGE_BYTES / 1024/ 1024 as wg_mem_used_mb from
workload_group_resource_usage where workload_group_id=11201;
+--------------------+
| wg_mem_used_mb     |
+--------------------+
| 499.96760272979736 |
+--------------------+
1 row in set (0.01 sec)
```

3. Use the ps command to check the memory usage of the process; the memory usage is around 3.8G.

```
[ ~]$ ps -eo pid,comm,%mem,rss | grep 1407481
1407481 doris_be         1.0 4071364
[ ~]$ ps -eo pid,comm,%mem,rss | grep 1407481
1407481 doris_be         1.0 4059012
[ ~]$ ps -eo pid,comm,%mem,rss | grep 1407481
1407481 doris_be         1.0 4057068
```

4. At the same time, the client will observe a significant number of query failures caused by insufficient memory.

```
1724074250162,14126,1c_sql,HY000 1105,"java.sql.SQLException: errCode = 2, detailMessage = (127.0.0.1)
[MEM_LIMIT_EXCEEDED]GC wg for hard limit, wg id:11201, name:g2, used:1.71 GB, limit:1.69 GB,
backend:10.16.10.8. cancel top memory used tracker <Query#Id=4a0689936c444ac8-a0d01a50b944f6e7>
consumption 1.71 GB. details:process memory used 3.01 GB exceed soft limit 304.41 GB or sys available
memory 101.16 GB less than warning water mark 12.80 GB., Execute again after enough memory, details see
be.INFO.",并发 1-3,text,false,,444,0,3,3,null,0,0,0
```

From the error message, it can be observed that the Workload Group used 1.7G of memory, but the Workload Group's limit is 1.69G. The calculation is as follows:1.69G = Physical machine memory (375G) * mem_limit (value from be.conf, default is 0.9) * 0.5% (Workload Group's configuration). This means the memory percentage configured in the Workload Group is calculated based on the memory available to the BE process.

Recommendations

As demonstrated in the tests above, memory hard limits can control the memory usage of a Workload Group but do so by terminating queries to release memory. This approach can lead to a poor user experience and, in extreme cases, may cause all queries to fail.

Therefore, in production environments, it is recommended to use memory hard limits in conjunction with query queuing functionality. This ensures controlled memory usage while maintaining query success rates.

CPU hard limit

Doris workloads can generally be categorized into three types: 1. Core Report Queries: These are typically used by company executives to view reports. While the load may not be very high, the availability requirements are strict. These queries can be assigned to a group with a higher-priority soft limit, ensuring they receive more CPU resources when resources are insufficient. 2. Adhoc queries are typically exploratory and analytical in nature, with random SQL and unpredictable resource consumption. Their priority is usually low. Therefore, CPU hard limits can be used to manage these queries, configuring lower values to prevent excessive CPU resource usage that could reduce cluster availability. 3. ETL queries typically have fixed SQL and stable resource consumption, although there may occasionally be spikes in resource usage due to increased upstream data. Therefore, CPU hard limits can be configured to manage these queries.

Different workloads have varying CPU consumption, and users have different latency requirements. When the BE CPU is fully utilized, availability decreases, and response times increase. For example, an Adhoc analysis query may fully utilize the CPU of the entire cluster, causing core report queries to experience higher latency, which impacts SLA. Therefore, a CPU isolation mechanism is needed to separate different workloads and ensure cluster availability and SLA.

Workload Group supports both CPU soft limits and hard limits. It is currently recommended to configure Workload Groups with hard limits in production environments. This is because CPU soft limits typically only show priority effects when the CPU is fully utilized. However, when the CPU is fully used, internal Doris components (such as the RPC component) and the operating system’s available CPU are reduced, leading to a significant drop in overall cluster availability. Therefore, in production environments, it is essential to avoid CPU resource exhaustion, and the same logic applies to other resources such as memory.

Test environment

1 FE, 1 BE, 96-core machine. The dataset is clickbench, and the test SQL is q29.

Tesing 1. Using JMeter to initiate 3 concurrent queries, the CPU usage of the BE process is pushed to a relatively high usage rate. The test machine has 96 cores, and using the top command, we can see that the BE process’s CPU usage is 7600%, which means the process is currently using 76 cores.

```
  PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  20   0   71.5g   30.4g  66560 S  7660   8.1  40:39.89 doris_be
```

use workload group cpu

2. Modify the CPU hard limit of the currently used Workload Group to 10%.

```
alter workload group g2 properties('cpu_hard_limit'='10%');
```

3. Switch to CPU hard limit mode.

```
ADMIN SET FRONTEND CONFIG ("enable_cpu_hard_limit" = "true");
```

4. Re-run the load test for queries, and you can see that the current process can only use 9 to 10 cores, which is about 10% of the total cores.

```
  PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  20   0   72.6g   21.2g  62056 S 979.7   5.6  57:23.63 doris_be
```

use workload group cpu

It is important to note that this test is best conducted using query workloads, as they are more likely to reflect the effect. If testing load, it may trigger Compaction, causing the actual observed values to be higher than the values configured in the Workload Group. Currently, Compaction workloads are not managed under the Workload Group.

5. In addition to using Linux system commands, you can also observe the current CPU usage of the group through Doris's system tables, where the CPU usage is around 10%.

```
mysql [information_schema]>select CPU_USAGE_PERCENT from workload_group_resource_usage where
WORKLOAD_GROUP_ID=11201;
+-------------------+
| CPU_USAGE_PERCENT |
+-------------------+
|              9.57 |
+-------------------+
1 row in set (0.02 sec)
```

note

1. When configuring, it's better not to set the total CPU allocation of all groups to exactly 100%. This is mainly to ensure the availability of low-latency scenarios, as some resources need to be reserved for other components. However, for scenarios that are not very sensitive to latency and aim for maximum resource utilization, setting the total CPU allocation of all groups to 100% can be considered.
2. Currently, the interval for synchronizing Workload Group metadata from FE to BE is 30 seconds. Therefore, changes to Workload Group settings may take up to 30 seconds to

take effect.

Limit local IO

In OLAP systems, during ETL operations or large Adhoc queries, a significant amount of data needs to be read. To speed up the data analysis process, Doris uses multi-threaded parallel scanning across multiple disk files, which generates substantial disk IO that can impact other queries (such as report analysis). By using Workload Groups, Doris can group offline ETL data processing and online report queries separately, limiting the offline data processing IO bandwidth. This helps reduce the impact of offline data processing on online report analysis.

Test environment

1 FE, 1 BE, 96-core machine. Dataset: clickbench. Test query: q29.

Testing without enabling IO hard limits 1. Clear Cache.

```
```sql
// clear OS cache
sync; echo 3 > /proc/sys/vm/drop_caches

// disable BE page cache
disable_storage_page_cache = true
```
```

2. Perform a full table scan on the clickbench table, and execute a single concurrent query.

```
set dry_run_query = true;
select * from hits.hits;
```

3. Check the maximum throughput of the current Group as 3GB per second through Doris's system table.

```
mysql [information_schema]>select LOCAL_SCAN_BYTES_PER_SECOND / 1024 / 1024 as mb_per_sec from
workload_group_resource_usage where WORKLOAD_GROUP_ID=11201;
+--------------------+
| mb_per_sec         |
+--------------------+
| 1146.6208400726318 |
+--------------------+
1 row in set (0.03 sec)

mysql [information_schema]>select LOCAL_SCAN_BYTES_PER_SECOND / 1024 / 1024 as mb_per_sec from
workload_group_resource_usage where WORKLOAD_GROUP_ID=11201;
+--------------------+
| mb_per_sec         |
+--------------------+
| 3496.2762966156006 |
+--------------------+
1 row in set (0.04 sec)

mysql [information_schema]>select LOCAL_SCAN_BYTES_PER_SECOND / 1024 / 1024 as mb_per_sec from
workload_group_resource_usage where WORKLOAD_GROUP_ID=11201;
+--------------------+
| mb_per_sec         |
+--------------------+
```

```
| 2192.7690029144287 |
+--------------------+
1 row in set (0.02 sec)
```

4. Use the pidstat command to check the process IO. The first column is the process ID, and the second column is the read IO throughput (in kb/s). It can be seen that when IO is not restricted, the maximum throughput is 2GB per second.

```
s be]$ pidstat -d 1 3600 | grep 884431
 884431  14124.00     16.00      0.00       0  doris_be
 884431    224.00      0.00      0.00       0  doris_be
 884431      0.00      8.00      0.00       0  doris_be
 884431 675788.00      8.00      0.00       0  doris_be
 884431 1891976.00      4.00      0.00       0  doris_be
 884431 2101908.00      4.00      0.00       0  doris_be
 884431 2106872.00      4.00      0.00       0  doris_be
 884431 1829000.00      8.00      0.00       0  doris_be
 884431 159376.00      4.00      0.00       0  doris_be
 884431 124836.00      4.00      0.00       0  doris_be
 884431 117048.00      8.00      0.00       0  doris_be
 884431 738164.00     12.00      0.00       0  doris_be
 884431 776224.00      4.00      0.00       0  doris_be
 884431 614728.00      4.00      0.00       0  doris_be
 884431 333784.00      8.00      0.00       0  doris_be
 884431 514560.00      8.00      0.00       0  doris_be
 884431 462176.00      8.00      0.00       0  doris_be
 884431 396720.00      8.00      0.00       0  doris_be
 884431 472272.00      4.00      0.00       0  doris_be
 884431 373544.00      4.00      0.00       0  doris_be
 884431 409288.00      0.00      0.00       0  doris_be
 884431 374816.00      4.00      0.00       0  doris_be
 884431 303184.00     12.00      0.00       0  doris_be
 884431 347520.00      0.00      0.00       0  doris_be
 884431 275096.00      0.00      0.00       0  doris_be
 884431 217592.00      0.00      0.00       0  doris_be
 884431 260428.00      4.00      0.00       0  doris_be
 884431 163272.00      8.00      0.00       0  doris_be
 884431 185872.00      8.00      0.00       0  doris_be
 884431 153172.00      4.00      0.00       0  doris_be
 884431 116780.00      0.00      0.00       0  doris_be
 884431 108364.00      4.00      0.00       0  doris_be
 884431  64804.00     12.00      0.00       0  doris_be
 884431  49116.00      0.00      0.00       0  doris_be
 884431  10700.00      8.00      0.00       0  doris_be
 884431   6188.00      4.00      0.00       0  doris_be
 884431     88.00      4.00      0.00       0  doris_be
 884431      0.00     20.00      0.00       0  doris_be
```

use workload group io

Test after enabling IO hard limit 1. Clear cache.

```
```sql
// Clear OS cache.
sync; echo 3 > /proc/sys/vm/drop_caches
```
```

```
// disable BE page cache
disable_storage_page_cache = true
```

2. Modify the Workload Group configuration to limit the maximum throughput to 100M per second.

```
alter workload group g2 properties('read_bytes_per_second'='104857600');
```

3. Use Doris system tables to check that the maximum IO throughput of the Workload Group is 98M per second.

```
mysql [information_schema]>select LOCAL_SCAN_BYTES_PER_SECOND / 1024 / 1024 as mb_per_sec from
workload_group_resource_usage where WORKLOAD_GROUP_ID=11201;
+--------------------+
| mb_per_sec         |
+--------------------+
| 97.94296646118164  |
+--------------------+
1 row in set (0.03 sec)

mysql [information_schema]>select LOCAL_SCAN_BYTES_PER_SECOND / 1024 / 1024 as mb_per_sec from
workload_group_resource_usage where WORKLOAD_GROUP_ID=11201;
+--------------------+
| mb_per_sec         |
+--------------------+
| 98.37584781646729  |
+--------------------+
1 row in set (0.04 sec)

mysql [information_schema]>select LOCAL_SCAN_BYTES_PER_SECOND / 1024 / 1024 as mb_per_sec from
workload_group_resource_usage where WORKLOAD_GROUP_ID=11201;
+--------------------+
| mb_per_sec         |
+--------------------+
| 98.06641292572021  |
+--------------------+
1 row in set (0.02 sec)
```

4. Use the pid tool to check that the maximum IO throughput of the process is 131M per second.

```
s be]$ pidstat -d 1 3600 | grep 968609
968609      0.00      3.67      0.00       0  doris_be
968609 143028.00     24.00      0.00       0  doris_be
968609 162668.00      0.00      0.00       0  doris_be
968609  80256.00      4.00      0.00       0  doris_be
968609  91800.00      0.00      0.00       0  doris_be
968609 137492.00      0.00      0.00       0  doris_be
968609 125756.00      8.00      0.00       0  doris_be
968609 134624.00      8.00      0.00       0  doris_be
968609 109180.00      0.00      0.00       0  doris_be
968609 114524.00      4.00      0.00       0  doris_be
968609 124952.00      4.00      0.00       0  doris_be
968609 120524.00      8.00      0.00       0  doris_be
968609 103064.00      8.00      0.00       0  doris_be
968609 126904.00      4.00      0.00       0  doris_be
968609 121760.00      8.00      0.00       0  doris_be
968609 119848.00      0.00      0.00       0  doris_be
968609 118012.00      8.00      0.00       0  doris_be
968609 112476.00      0.00      0.00       0  doris_be
968609 118296.00      0.00      0.00       0  doris_be
968609 120792.00      4.00      0.00       0  doris_be
968609 117196.00      0.00      0.00       0  doris_be
968609 122848.00     12.00      0.00       0  doris_be
968609 127760.00      4.00      0.00       0  doris_be
```

use workload group io

Note 1. The LOCAL_SCAN_BYTES_PER_SECOND field in the system table represents the summary value of the current Workload Group's statistics at the process level. For example, if 12 file paths are configured, LOCAL_SCAN_BYTES_PER_SECOND is the maximum IO value of these 12 file paths. If you wish to view the IO throughput for each file path separately, you can check the detailed values in Grafana.

2. Due to the presence of the operating system and Doris's Page Cache, the IO observed through Linux's IO monitoring scripts is typically smaller than the IO seen in the system table.

Limit remote IO

BrokerLoad and S3Load are commonly used methods for large-scale data load. Users can first upload data to HDFS or S3, and then use BrokerLoad and S3Load to load data in parallel. To speed up the load process, Doris uses multi-threading to pull data from HDFS/S3, which can generate significant pressure on HDFS/S3, potentially making other jobs running on HDFS/S3 unstable.

To mitigate the impact on other workloads, the Workload Group's remote IO limit feature can be used to restrict the bandwidth used during the load process from HDFS/S3. This helps reduce the impact on other business operations.

Test environment

1 FE and 1 BE are deployed on the same machine, configured with 16 cores and 64GB of memory. The test data is the clickbench dataset, and before testing, we need to upload the dataset to S3. Considering the upload time, we will only upload 10 million rows of data, and then use the TVF function to query the data from S3.

After the upload is successful, you can use the command to view the schema information.

```
DESC FUNCTION s3 (
    "URI" = "https://bucketname/1kw.tsv",
    "s3.access_key" = "ak",
    "s3.secret_key" = "sk",
    "format" = "csv",
    "use_path_style" = "true"
);
```

Test without restricting remote read IO 1. Initiate a single-threaded test to perform a full table scan on the clickbench table.

```
```sql
// Set the operation to only scan the data without returning results.
set dry_run_query = true;

SELECT * FROM s3(
    "URI" = "https://bucketname/1kw.tsv",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "format" = "csv",
    "use_path_style"="true"
);
```
```

2. Use the system table to check the current remote IO throughput. It shows that the remote IO throughput for this query is 837 MB per second. Note that the actual IO throughput here is highly dependent on the environment. If the machine hosting the BE has limited bandwidth to the external storage, the actual throughput may be lower.

```
MySQL [(none)]> select cast(REMOTE_SCAN_BYTES_PER_SECOND/1024/1024 as int) as read_mb from
information_schema.workload_group_resource_usage;
+---------+
| read_mb |
+---------+
|     837 |
+---------+
1 row in set (0.104 sec)

MySQL [(none)]> select cast(REMOTE_SCAN_BYTES_PER_SECOND/1024/1024 as int) as read_mb from
information_schema.workload_group_resource_usage;
+---------+
| read_mb |
+---------+
|     867 |
+---------+
1 row in set (0.070 sec)

MySQL [(none)]> select cast(REMOTE_SCAN_BYTES_PER_SECOND/1024/1024 as int) as read_mb from
information_schema.workload_group_resource_usage;
+---------+
| read_mb |
```

```
+---------+
|     867 |
+---------+
1 row in set (0.186 sec)
```

3. Use the sar command (sar -n DEV 1 3600) to monitor the machine's network bandwidth. It shows that the maximum network bandwidth at the machine level is 1033 MB per second. The first column of the output represents the number of bytes received per second by a specific network interface on the machine, in KB per second.

```
;    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
)      7.56      7.56      0.00      0.00      0.00      0.00
) 961294.37   2492.58      0.00      0.00      0.00      0.00

;    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
)     19.92     19.92      0.00      0.00      0.00      0.00
) 1058684.12  2975.14       0.00      0.00      0.00      0.00

;    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
)    228.71    228.71      0.00      0.00      0.00      0.00
) 933606.63   2469.83      0.00      0.00      0.00      0.00

;    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
)      0.10      0.10      0.00      0.00      0.00      0.00
) 745845.31   2023.35      0.00      0.00      0.00      0.00
```

use workload group rio

Test limiting remote read IO 1. Modify the Workload Group configuration to limit remote read IO throughput to 100M per second.

```
```sql
alter workload group normal properties('remote_read_bytes_per_second'='104857600');
```
```

2. Initiate a single concurrent full table scan query.

```
set dry_run_query = true;

SELECT * FROM s3(
    "URI" = "https://bucketname/1kw.tsv",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "format" = "csv",
    "use_path_style"="true"
);
```

3. Use the system table to check the current remote read IO throughput. At this time, the IO throughput is around 100M, with some fluctuations. These fluctuations are influenced by the current algorithm design, typically peaking briefly without persisting for long periods, which is considered normal.

```
MySQL [(none)]> select cast(REMOTE_SCAN_BYTES_PER_SECOND/1024/1024 as int) as read_mb from
information_schema.workload_group_resource_usage;
+---------+
| read_mb |
```

```
+---------+
|      56 |
+---------+
1 row in set (0.010 sec)

MySQL [(none)]> select cast(REMOTE_SCAN_BYTES_PER_SECOND/1024/1024 as int) as read_mb from
information_schema.workload_group_resource_usage;
+---------+
| read_mb |
+---------+
|     131 |
+---------+
1 row in set (0.009 sec)

MySQL [(none)]> select cast(REMOTE_SCAN_BYTES_PER_SECOND/1024/1024 as int) as read_mb from
information_schema.workload_group_resource_usage;
+---------+
| read_mb |
+---------+
|     111 |
+---------+
1 row in set (0.009 sec)
```

4. Use the sar command (sar -n DEV 1 3600) to monitor the current network card's received traffic. The first column represents the amount of data received per second. The maximum value observed is now 207M per second, indicating that the read IO limit is effective. However, since the sar command reflects machine-level traffic, the observed value is slightly higher than what Doris reports.

```
   rxkB/s     txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
     3.33       3.33      0.00      0.00      0.00      0.00
140384.12     727.60      0.00      0.00      0.00      0.00

   rxkB/s     txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
     2.01       2.01      0.00      0.00      0.00      0.00
212281.74    1277.05      0.00      0.00      0.00      0.00

   rxkB/s     txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
     1.53       1.53      0.00      0.00      0.00      0.00
 89753.66     495.19      0.00      0.00      0.00      0.00

   rxkB/s     txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
     1.83       1.83      0.00      0.00      0.00      0.00
106358.55     545.90      0.00      0.00      0.00      0.00
```

use workload group rio

Frequently Asked Questions

1. Why is the CPU hard limit configuration not taking effect?

- This is usually caused by the following reasons:
    - Environment initialization failed. You need to check the two configuration files under the Doris CGroup path. Here, we take the CGroup V1 version as an example. If the user has specified the Doris CGroup path as

/sys/fs/cgroup/cpu/doris/, you should first check if the content of /sys/fs/cgroup/cpu/doris/query/1/tasks contains the thread IDs corresponding to the Workload Group. The “1” in the path represents the Workload Group ID, which can be obtained by running the command top -H -b -n 1 -p pid to find the thread IDs of the Workload Group. After confirming, ensure that the thread IDs of the Workload Group are written into the tasks file. Then, check if the value of /sys/fs/cgroup/cpu/doris/query/1/cpu.cfs_quota_us is -1. If it is -1, it means the CPU hard limit configuration has not taken effect.

- The CPU usage of the Doris BE process is higher than the CPU hard limit configured for the Workload Group. This is expected because the CPU managed by the Workload Group is primarily for query threads and memtable flush threads for Load. However, the BE process typically has other components consuming CPU as well, such as Compaction. Therefore, the CPU usage of the process is generally higher than the configured limit for the Workload Group. You can create a test Workload Group that only stresses the query load and then check the CPU usage of the Workload Group through the system table information_schema.workload_group_resource_usage. This table only records the CPU usage of the Workload Group and has been supported since version 2.1.6.
- Some users have configured the cpu_resource_limit.First, execute show property for jack like 'cpu_resource_limit' to check whether this parameter is set in the properties of the user jack. Then, execute show variables like 'cpu_resource_limit' to verify whether this parameter is set in the session variables. The default value of this parameter is -1, which indicates that it is not set. After configuring this parameter, queries are handled by an independent thread pool that is not managed by the Workload Group. Directly modifying this parameter may affect the stability of the production environment. It is recommended to gradually migrate the query loads that are configured with this parameter to be managed by the Workload Group. The current alternative to this parameter is the session variable num_scanner_threads. The main process is as follows: First, divide the users who have configured cpu_resource_limit into several batches. When migrating the first batch of users, modify the session variable num_scanner_threads for these users to 1. Then, assign a Workload Group to these users. After that, change cpu_resource_limit to -1 and observe the cluster’s stability over a period of time. If the cluster remains stable, continue migrating the next batch of users.

2. Why is the default number of Workload Groups limited to 15?

- Workload Groups are primarily used for dividing resources on a single machine. If too many Workload Groups are created on one machine, each Workload Group will only receive a very small portion of the resources. If the user indeed requires creating this many Workload Groups, you can consider dividing the cluster into multiple groups of BEs and then creating different Workload Groups for each group of BEs. You can also temporarily bypass this limit by modifying the FE configuration workload_group_max_num.

3. Why does the error “Resource temporarily unavailable” occur after configuring many Workload Groups?

- Each Workload Group corresponds to an independent thread pool. Creating too many Workload Groups may cause the BE process to attempt to start too many threads, exceeding the maximum number of threads allowed for a process by the operating system. To resolve this issue, you can modify the system environment configuration to allow the BE process to create more threads.

Workload Analysis Diagnosis

The workload analysis for clusters is primarily divided into two stages: - The first stage is runtime workload analysis, when the cluster's availability declines, queries with significant resource consumption can be identified through monitoring and downgraded accordingly. - The second stage involves analyzing historical data, such as audit logs, to identify unreasonable workloads and optimize them.

Runtime Workload Analysis

When the cluster's availability decreases, as detected through monitoring, the following process can be followed: 1. Initially, use monitoring to roughly determine the current cluster's bottleneck, such as excessive memory usage, high CPU usage, or high IO. If all are high, it may be advisable to prioritize addressing the memory issue. 2. Once the cluster's bottleneck is identified, the workload_group_resource_usage table can be consulted to find the Group with the highest current resource usage. For example, if there is a memory bottleneck, the top N Groups with the highest memory usage can be identified. 3. After determining the Group with the highest resource usage, the first step can be to reduce the query concurrency for this Group, as cluster resources are already tight at this point, and new queries should be avoided to prevent exhausting cluster resources. 4. Downgrade the queries for the current Group. Depending on the bottleneck, different approaches can be taken: - If it's a CPU bottleneck, consider setting the Group's CPU to a hard limit and adjusting the cpu_hard_limit to a lower value to voluntarily yield CPU resources. - For an IO bottleneck, limit the Group's maximum IO through the read_bytes_per_second parameter. - In case of a memory bottleneck, set the Group's memory to a hard limit and decrease the memory_limit value to release some memory. Note that this may result in numerous query failures within the current Group. 5. After completing the above steps, the cluster's availability usually recovers to some extent. At this point, further analysis can be conducted to determine the primary cause of the increased resource usage in this Group, whether it's due to an overall increase in query concurrency in this Group or specific large queries. If it's caused by specific large queries, these queries can be quickly killed to restore cluster functionality. 6. The backend_active_tasks table can be used in conjunction with active_queries to identify SQL queries with abnormal resource usage in the cluster and then kill these queries using the kill statement to free up resources.

Workload Analysis Through Historical Data

Currently, Doris's audit logs retain brief information about SQL execution, which can be used to identify unreasonable queries executed in the past and make adjustments. The specific process is as follows: 1. Review monitoring to confirm the cluster's historical resource usage and identify the cluster's bottleneck, whether it's CPU, memory, or IO. 2. Once the cluster's bottleneck is identified, the audit logs can be consulted to find SQL queries with abnormal resource usage during the corresponding period. There are two ways to define abnormal SQL: 1. If users have certain expectations regarding the resource usage of SQL in the cluster, such as most delays being in seconds and scan row counts in the tens of millions, then SQL queries with scan row counts in the hundreds of millions or billions are considered abnormal and require manual intervention. 2. If users do not have expectations regarding SQL resource usage in the cluster, percentile functions can be used to calculate resource usage and identify SQL queries with abnormal resource usage. Taking a CPU bottleneck as an example, first calculate the tp50/tp75/tp99/tp999 for query CPU time over a historical period and use these values as normal. Compare these with the percentile functions for query CPU time during the same

period in the current cluster. For instance, if the tp999 for the historical period is 1 minute but the tp50 for the current cluster during the same period is already 1 minute, it indicates that there are numerous SQL queries with CPU times exceeding 1 minute compared to historical data. Therefore, SQL queries with CPU times greater than 1 minute can be defined as abnormal. The same logic applies to other metrics. 3. Optimize SQL queries with abnormal resource usage, such as rewriting SQL, optimizing table structures, adjusting parallelism to reduce the resource usage per SQL query. 4. If the audit logs reveal that SQL resource usage is normal, monitoring and auditing can be used to check if the number of SQL queries executed during that time has increased compared to historical periods. If so, confirm with upstream businesses whether there has been an increase in upstream access traffic during the corresponding time periods, and decide whether to scale the cluster or implement queuing and rate limiting.

Commonly Used SQL

> **Note that the active_queries table records queries running on the FE, while the backend_active_tasks table records queries running on the BE. Not all queries are registered with the FE during execution; for example, stream loads are not registered with the FE. Therefore, it is normal to get no matching results when performing a LEFT JOIN between backend_active_tasks and active_queries.**
>
> When a query is a SELECT query, the queryId recorded in both active_queries and backend_active_tasks is the same. When a query is a stream load, the queryId in the active_queries table is empty, while the queryId in backend_active_tasks is the ID of the stream load.

1. View all current Workload Groups and display them in descending order of memory/CPU/I/O usage.

```
select be_id,workload_group_id,memory_usage_bytes,cpu_usage_percent,local_scan_bytes_per_second
   from workload_group_resource_usage
order by  memory_usage_bytes,cpu_usage_percent,local_scan_bytes_per_second desc
```

2. Cpu usage TopN Sql. `select t1.query_id as be_query_id, t1.query_type, t2.query_id, t2.workload_group_id, t2.`database`, t1.cpu_time, t2.`sql` from (select query_id, query_type,sum(task_cpu_time_ms) as cpu_time from backend_active_tasks group by query_id, query_type) t1 left join active_queries t2 on t1.query_id = t2.query_id order by cpu_time desc limit 10;`

3. Memory usage TopN Sql. `select t1.query_id as be_query_id, t1.query_type, t2.query_id, t2.workload_group_id, t1.mem_used from (select query_id, query_type, sum(current_used_memory_bytes) as mem_used from backend_active_tasks group by query_id, query_type) t1 left join active_queries t2 on t1.query_id = t2.query_id order by mem_used desc limit 10;`

4. Scan bytes/rows TopN Sql. `select t1.query_id as be_query_id, t1.query_type, t2.query_id, t2.workload_group_id, t1.scan_rows, t1.scan_bytes from (select query_id, query_type, sum(scan_rows) as scan_rows,sum(scan_bytes) as scan_bytes from backend_active_tasks group by query_id,query_type) t1 left join active_queries t2 on t1.query_id = t2.query_id order by scan_rows desc,scan_bytes desc limit 10;`

5. Show workload group's scan rows/bytes. `select t2.workload_group_id, sum(t1.scan_rows) as wg_scan_rows, sum(t1.scan_bytes) as wg_scan_bytes from (select query_id, sum(scan_rows) as`

scan_rows,sum(scan_bytes) as scan_bytes from backend_active_tasks group by query_id) t1 left join active_queries t2 on t1.query_id = t2.query_id group by t2.workload_group_id order by wg_scan_rows desc,wg_scan_bytes desc

6. Show workload group's query queue details. `select workload_group_id, query_id, query_status, now() - queue_start_time as queued_time from active_queries where query_status='queued' order by workload_group_id`

Concurrency Control and Queuing

Concurrency control and queuing is a resource management mechanism. When multiple queries simultaneously request resources and reach the system's concurrency limit, Doris will manage the queries based on predefined strategies and restrictions, ensuring the system can still operate smoothly under high load and avoid issues like OOM (Out of Memory) or system freezes.

Doris's concurrency control and queuing mechanism is primarily implemented through workload groups. A workload group defines the resource usage limits for queries, including maximum concurrency, queue length, and timeout parameters. By properly configuring these parameters, the goal of resource management can be achieved.

Basic usage

```
create workload group if not exists queue_group
properties (
    "max_concurrency" = "10",
    "max_queue_size" = "20",
    "queue_timeout" = "3000"
);
```

Parameter description

Property	Data type	Default value	Value range	Description
max_concurrency	Integer	2147483647	[0, 2147483647]	Optional, the maximum number of concurrent queries. The default value is the maximum integer value, meaning there is no limit on concurrency. When the number of running queries reaches the maximum concurrency, new queries will enter the queuing process.
max_queue_size	Integer	0	[0, 2147483647]	Optional, the length of the query queue. When the queue is full, new queries will be rejected. The default value is 0, meaning no queuing.

Property	Data type	Default value	Value range	Description
queue_timeout	Integer	0	[0, 2147483647]	Optional, the maximum wait time for a query in the queue, in milliseconds. If the query exceeds this time in the queue, an exception will be thrown to the client. The default value is 0, meaning no queuing, and queries will immediately fail upon entering the queue.

If there is currently 1 FE in the cluster, the meaning of this configuration is as follows: The maximum number of concurrent queries in the cluster is limited to 10. When the maximum concurrency is reached, new queries will enter the queue, with the queue length limited to 20. The maximum wait time for a query in the queue is 3 seconds, and queries that exceed 3 seconds in the queue will return a failure directly to the client.

> **The current queuing design does not take into account the number of FEs. The queuing parameters only take effect at the single FE level. For example:**
>
> In a Doris cluster, if a workload group is configured with max_concurrency = 1, If there is 1 FE in the cluster, the workload group will allow only one SQL query to run at a time in the cluster; If there are 3 FEs in the cluster, the maximum number of SQL queries in the cluster could be 3.

Check the queue status

Syntax

```
show workload groups
```

Example

```
mysql [(none)]>show workload groups\G;
*************************** 1. row ***************************
                           Id: 1
                         Name: normal
                    cpu_share: 20
                 memory_limit: 50%
    enable_memory_overcommit: true
              max_concurrency: 2147483647
               max_queue_size: 0
                queue_timeout: 0
               cpu_hard_limit: 1%
              scan_thread_num: 16
  max_remote_scan_thread_num: -1
  min_remote_scan_thread_num: -1
        memory_low_watermark: 50%
       memory_high_watermark: 80%
```

```
                        tag:
       read_bytes_per_second: -1
remote_read_bytes_per_second: -1
            running_query_num: 0
            waiting_query_num: 0
```

`running_query_num`Represents the number of queries currently running, `waiting_query_num`Represents the number of queries in the queue.

Bypass the queuing

In some operational scenarios, the administrator account needs to bypass the queuing logic to execute SQL for system management tasks. This can be done by setting session variables to bypass the queuing:

```
set bypass_workload_group = true;
```

Query Circuit Breaker

Query circuit breaking is a protective mechanism used to prevent long-running or resource-consuming queries from having a negative impact on the system. When a query exceeds predefined resource or time limits, the circuit breaker mechanism automatically terminates the query to avoid adverse effects on system performance, resource usage, and other queries. This mechanism ensures the stability of the cluster in a multi-user environment, preventing a single query from exhausting system resources or slowing down responses, thereby improving overall availability and efficiency.

In Doris, there are two types of circuit breaker strategies:

- **Planning-time circuit breaking**, namely **SQL Block Rule**, which is used to prevent the execution of statements that match specific patterns. Block rules apply to any statements, including DDL and DML. Typically, block rules are configured by database administrators (DBAs) to enhance cluster stability. For example,
    - Preventing a query from scanning too many rows of data
    - Preventing a query from scanning too many partitions
    - Preventing a statement that modifies global variables to avoid accidental changes to cluster configurations.
    - Preventing a query pattern that typically consumes excessive resources
- **Runtime circuit breaking**, namely **Workload Policy**, which monitors in real-time during runtime the execution time of queries, the amount of data scanned, and memory consumption to implement rule-based query circuit breaking.

SQL Block Rule

According to the block pattern, it can be divided into:

- Scan row count block rule
- Scan partition count block rule
- Scan bucket count block rule
- Query statement regex matching block rule
- Query statement hash value matching block rule

Blocking rules can be categorized according to the scope of blocking into:

- Global-level blocking rules
- User-level blocking rules

Usage

Global-level blocking rules

```
CREATE SQL_BLOCK_RULE rule_001
PROPERTIES (
```

```
  "sql"="select \\* from t",
  "global" = "true",
  "enable" = "true"
)
```

In this way, we have created a global-level blocking rule. The rule is named rule_001. A query statement regex matching rule has been configured to block all query statements that match the regex `select \\* from t`.

Since this is a global-level blocking rule, any user executing statements that match the above regex will be blocked. For example:

```
MySQL root@127.0.0.1:test> select * from t;
(1105, 'errCode = 2, detailMessage = errCode = 2, detailMessage = SQL match regex SQL block rule: rule_001')
```

User-level Blocking Rule

```
CREATE SQL_BLOCK_RULE rule_001
PROPERTIES (
  "sql"="select * from t",
  "global" = "false",
  "enable" = "true"
)
```

Unlike global-level blocking rules, user-level blocking rules only apply to specified users. When we create a blocking rule, set the property “global” to “false”. Then, this blocking rule will be considered a user-level blocking rule.

To make user-level blocking rules effective, you also need to set the corresponding properties for the users who need to use this rule. For example:

```
set property for 'root' 'SQL_block_rules' = 'rule_001';
```

Thus, after the above configuration, when the root user executes a query, the blocking rule named rule_001 will be applied.

```
MySQL root@127.0.0.1:test> set property for 'root' 'SQL_block_rules' = '';
Query OK, 0 rows affected
Time: 0.018s
MySQL root@127.0.0.1:test> select * from t;
+----+----+
| id | c1 |
+----+----+
| 1  | 1  |
+----+----+

1 row in set
Time: 0.027s
MySQL root@127.0.0.1:test> set property for 'root' 'SQL_block_rules' = 'rule_001';
Query OK, 0 rows affected
Time: 0.008s
MySQL root@127.0.0.1:test> select * from t;
(1105, 'errCode = 2, detailMessage = errCode = 2, detailMessage = SQL match regex SQL block rule: rule_001')
```

- If you want to add multiple user-level blocking rules for a user, list all the rule names in the rule list, separated by commas.
- If you want to remove all user-level blocking rules for a user, set the rule list to an empty string.

Other Operations

If you need to modify or delete blocking rules, refer to the SQL manual for blocking rules.

Use Cases

You can use the following scenarios:

- Blocking scans that exceed a specified number of rows
- Blocking scans that exceed a specified number of partitions
- Blocking scans that exceed a specified number of buckets
- Blocking queries with specific patterns

Blocking Scans that Exceed a Specified Number of Rows

Scanning data significantly consumes BE's IO and CPU resources. Therefore, unnecessary data scans pose a substantial challenge to cluster stability. In daily use, blind full table scan operations often occur, such as `SELECT * FROM t`. To prevent such queries from damaging the cluster, you can set an upper limit on the number of rows scanned by a single query on a single table.

```
CREATE SQL_BLOCK_RULE rule_card
PROPERTIES
(
   "cardinality" = "1000",
   "global" = "true",
   "enable" = "true"
);
```

When the above rules are set, if a single table scan exceeds 1000 rows, the execution of the query is prohibited. It should be noted that since the calculation of the number of rows scanned is done during the planning phase, not the execution phase. Therefore, when calculating the number of rows, only partition and bucket pruning are considered, and the impact of other filtering conditions on the number of rows scanned is not taken into account. In other words, the worst-case scenario is considered. Therefore, queries where the actual number of rows scanned is less than the set value may also be blocked.

Prevent Scanning Data from Exceeding the Specified Number of Partitions

Scanning too many partitions can significantly increase the CPU consumption of BE. Additionally, if the query is on an external table, it is more likely to incur significant network overhead and metadata retrieval overhead. In daily use, this is often due to forgetting to write filtering conditions on partition columns or writing them incorrectly. To prevent such queries from damaging the cluster, you can set an upper limit on the number of partitions scanned by a single query on a single table.

```
CREATE SQL_BLOCK_RULE rule_part_num
PROPERTIES
```

```
(
   "partition_num" = "30",
   "global" = "true",
   "enable" = "true"
);
```

When the above rules are set, if the number of partitions scanned for a single table exceeds 30, the execution of the query is prohibited. It should be noted that the calculation of the number of scanned partitions is completed during the planning phase, not the execution phase. Therefore, there may be cases where more partitions are retained due to incomplete partition pruning. Thus, queries where the actual number of scanned partitions is less than the set value may also be blocked.

Prevent Scanning Data with Excessive Bucket Counts

Scanning too many buckets can significantly increase the BE's CPU consumption. To prevent such queries from harming the cluster, you can set an upper limit on the number of partitions a single query can scan on a single table.

```
CREATE SQL_BLOCK_RULE rule_teblet_num
PROPERTIES
(
   "tablet_num" = "200",
   "global" = "true",
   "enable" = "true"
);
```

When the above rules are set, if the number of buckets scanned in a single table exceeds 200, the execution of the query is prohibited.

It is important to note that the calculation of the number of scanned buckets is done during the planning phase, not the execution phase. Therefore, it is possible that due to incomplete bucket pruning, more partitions are retained. As a result, queries with an actual number of scanned buckets less than the set value may also be blocked.

Blocking Queries with Specific Patterns

For various reasons, such as high computational complexity or long planning time, you may want to block queries that use certain patterns.

For example, to block the `abs` function. You can use the following regular expression block rule to achieve this purpose.

```
CREATE SQL_BLOCK_RULE rule_abs
PROPERTIES(
  "sql"="(?i)abs\\s*\\(.+\\)",
  "global"="true",
  "enable"="true"
);
```

In the above regular expressions:

- `(?i)` indicates case-insensitive matching.

- `abs` is the target function to be blocked.
- `\s*` signifies that any amount of whitespace is allowed between `abs` and the left parenthesis.
- `\(.+\)` matches the function parameters.

Similarly, similar methods can be used to block `set global` to prevent unintended variable changes, or block `truncate table` to prevent unintended data deletions.

FAQ

Q: Will regular expression blocking rules have side effects on the cluster?

A: Yes. Regular expression matching is computationally intensive. When using complex regular expressions or too many regex blocking rules, it can significantly increase the CPU load on the FE. Therefore, add regex blocking rules cautiously. Avoid using complex regular expressions unless necessary.

Q: Can a blocking rule be temporarily disabled?

A: Yes. Modify the blocking rule by setting its “enable” property to “false”.

Q: What standard do the regular expressions in blocking rules use?

A: The regular expressions in blocking rules use Java’s regular expression syntax. Common expressions can refer to the SQL syntax manual. The complete manual can be found at https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

Workload Policy

SQL Block Rule is a configuration for circuit breaking during planning, but because the cost calculation during planning may be inaccurate (especially for very complex queries, making it difficult to estimate accurately), it can lead to rules not being effective or false positives. Workload Policy addresses this limitation by allowing real-time monitoring of certain metrics during query execution, and circuit breaking queries whose runtime state does not meet expectations. This prevents unexpected large queries from consuming excessive resources and affecting cluster stability. Common runtime monitoring metrics include:

- Query execution time
- Number of rows scanned per BE
- Number of bytes scanned per BE
- Memory usage per BE

Version Notes

Since Doris version 2.1, Workload Policy can be used to implement circuit breaking for large queries.

Version	since 2.1
select	yes
insert into select	yes

Version	since 2.1
insert into values	no
stream load	yes
routine load	yes
backup	no
compaction	no

Creating Workload Policy

Use the `CREATE WORKLOAD Policy` command to create a resource management policy.

In the example below, create a Policy named `test_cancel_Policy`, which will cancel queries running in the cluster for more than 1000ms. The current status is enabled. Creating a Workload Policy requires `admin_priv` privileges.

```
create workload policy test_cancel_Policy
Conditions(query_time > 1000)
Actions(cancel_query)
properties('enabled'='true');
```

When creating a Workload Policy, the following must be specified:

- **Condition** represents the policy trigger condition. Multiple Conditions can be linked using commas "," to represent an "AND" relationship. In the example above, `query_time > 1000` indicates that the Policy is triggered when the query time exceeds 1 second. Currently supported Conditions are:

Conditions	Description
username	The username carried by the query, only triggers the `set_session_variable` Action in FE.
be_scan_rows	The number of rows scanned by a SQL in a single BE process. If the SQL is executed concurrently on BE, it is the cumulative value of multiple concurrent executions.
be_scan_bytes	The number of bytes scanned by a SQL in a single BE process. If the SQL is executed concurrently on BE, it is the cumulative value of multiple concurrent executions, in bytes.
query_time	The runtime of a SQL in a single BE process, in milliseconds.
query_be_memory_bytes	The memory usage of a SQL in a single BE process. If the SQL is executed concurrently on BE, it is the cumulative value of multiple concurrent executions, in bytes.

- **Action** represents the action taken when the condition is triggered. Currently, a Policy can only define one Action (except for `set_session_variable`). In the example above, `cancel_query` indicates cancelling the query. Currently supported Actions are:

Actions	Description
cancel_query	Cancel the query.
set_session_variable	Triggers the `set session variable` statement. A single policy can have multiple `set_session_variable` options, currently only triggered in FE by the `username` Condition.

- **Properties** define the attributes of the current Policy, including whether it is enabled and its priority.

Properties	Description
enabled	Can be `true` or `false`, default is `true`, indicating the Policy is enabled. `false` indicates the Policy is disabled.
priority	An integer ranging from 0 to 100, default is 0, representing the Policy's priority. The higher the value, the higher the priority. This attribute mainly ensures that when a query matches multiple Policies, only the Policy with the highest priority is selected.
workload_group	Currently, a Policy can bind to one workload group, meaning this Policy only applies to queries from a specific Workload Group. The default is empty, meaning it applies to all queries.

Binding Workload Policy to Workload Group

By default, Workload Policies apply to all supported queries. If you want to specify that a Policy only targets a specific Workload Group, you need to bind the Workload Group through the `workload_group` option. The statement is as follows:

```
create workload policy test_cancel_big_query
Conditions(query_time > 1000)
Actions(cancel_query)
properties('workload_group'='normal')
```

Important Notes

- The Conditions and Actions of the same Policy must either both be FE or both be BE. For example, `set_session_variable` and `cancel_query` cannot be configured within the same Policy. Conditions `be_scan_rows` and `username` cannot be configured within the same Policy.
- Currently, Policies are executed by asynchronous threads at fixed time intervals, which introduces a certain latency in policy enforcement. For example, if a user configures a policy to cancel a query when the number of scanned rows exceeds 1,000,000, and the cluster resources are relatively idle at that time, the query may finish before the cancellation policy takes effect. The current interval is 500ms, meaning that queries with run times shorter than this interval may bypass the policy check.
- Currently supported load types include select/insert, select/stream load, broker load, and routine load.
- A single query may match multiple Policies, but only the Policy with the highest priority will take effect.
- Modifications to Actions and Conditions are currently not supported; they can only be modified by deleting and recreating them.

Workload Policy Demonstration

1. Session Variable Modification Test

Attempt to modify concurrency-related parameters in the session variables of the Admin account.

```
-- log on admin to check variables
show variables like '%parallel_fragment_exec_instance_num%';
+-----------------------------------+-------+---------------+---------+
| Variable_name                     | Value | Default_Value | Changed |
+-----------------------------------+-------+---------------+---------+
| parallel_fragment_exec_instance_num | 8     | 8             | 0       |
+-----------------------------------+-------+---------------+---------+
1 row in set (0.00 sec)
```

```
-- Create a Policy to modify the concurrency parameters of the admin account.
create workload Policy test_set_var_Policy
Conditions(username='admin')
Actions(set_session_variable 'parallel_fragment_exec_instance_num=1')

-- After some time, check the admin account's parameters again.
show variables like '%parallel_fragment_exec_instance_num%';
+-----------------------------------+-------+---------------+---------+
| Variable_name                     | Value | Default_Value | Changed |
+-----------------------------------+-------+---------------+---------+
| parallel_fragment_exec_instance_num | 1     | 8             | 1       |
+-----------------------------------+-------+---------------+---------+
1 row in set (0.01 sec)
```

2. Big Query Circuit Breaker Test

Test to circuit break queries that run longer than 3s. Below is an audit log of a successful run of ckbench's q29, showing that this SQL took 4.5s to complete.

```
mySQL [hits]>SELECT REGEXP_REPLACE(Referer, '^https?://(?:www\.)?([^/]+)/.*$', '\1') AS k, AVG(length(Referer)) AS l, COUNT(*) AS c,
MIN(Referer) FROM hits WHERE Referer <> '' GROUP BY k HAVING COUNT(*) > 100000 ORDER BY l DESC LIMIT 25;
+-------------------------------------------------------------------------+------------------+----------+-----------------------------
-------------------------------------------------------------------------------------------+
| k                                                                       | l                | c        | min(Referer)
|
+-------------------------------------------------------------------------+------------------+----------+-----------------------------
-------------------------------------------------------------------------------------------+
| 1                                                                       | 85.4611926713085 | 67259319 |
http://%26ad%3D1%25EA%25D0%26utm_source=web&cd=19590&input_onlist/би-2 место будущей кондицин                      |
| http:%2F%2Fwwww.regnancies/search&evL8gE&where=all&filmId=bEmYZc_WTDE |               69 |   207347 |
http:%2F%2Fwwww.regnancies/search&evL8gE&where=all&filmId=bEmYZc_WTDE                                             |
| http://новострашная                                                     |               31 |   740277 | http://новострашная
|
| http://loveche.html?ctid                                                |               24 |   144901 | http://loveche.html?ctid
|
| http://rukodeliveresult                                                 |               23 |   226135 | http://rukodeliveresult
|
| http://holodilnik.ru                                                    |               20 |   133893 | http://holodilnik.ru
|
| http://smeshariki.ru                                                    |               20 |   210736 | http://smeshariki.ru
|
| http:%2F%2Fviewtopic                                                    |               20 |   391115 | http:%2F%2Fviewtopic
|
| http:%2F%2Fwwww.ukr                                                     |               19 |   655178 | http:%2F%2Fwwww.ukr
|
| http:%2F%2FviewType                                                     |               19 |   148907 | http:%2F%2FviewType
|
| http://state=2008                                                       |               17 |   139630 | http://state=2008
|
+-------------------------------------------------------------------------+------------------+----------+-----------------------------
-------------------------------------------------------------------------------------------+
11 rows in set (4.50 sec)
```

Create a Policy that cancels a query if it runs longer than 3 seconds.

```
create workload Policy test_cancel_3s_query
Conditions(query_time > 3000)
Actions(cancel_query)
```

Upon re-executing the SQL, you can see that the SQL execution will directly report an error.

```
mySQL [hits]>SELECT REGEXP_REPLACE(Referer, '^https?://(?:www\.)?([^/]+)/.*$', '\1') AS k, AVG(length(Referer)) AS l, COUNT(*) AS c,
MIN(Referer) FROM hits WHERE Referer <> '' GROUP BY k HAVING COUNT(*) > 100000 ORDER BY l DESC LIMIT 25;
ERROR 1105 (HY000): errCode = 2, detailMessage = (127.0.0.1)[CANCELLED]query cancelled by workload Policy,id:12345
```

Kill Query

You can cancel currently executing operations or disconnect current connection sessions using the `KILL` command. This document introduces related operations and considerations.

Getting Query Identifiers

`KILL` requires a query identifier to cancel the corresponding query request. Query identifiers include: Query ID, Connection ID, and Trace ID.

You can obtain query identifiers through the following methods.

PROCESSLIST

Through the `processlist` system table, you can get all current session connections and query operations being executed in the connections. This includes the Query ID and Connection ID.

```
mysql> SHOW PROCESSLIST;
+------------------+------+------+--------------------+---------------------+----------+------+---------+------+-------+-----------
-----------------------+------------------+--------------+--------------+
| CurrentConnected | Id   | User | Host               | LoginTime           | Catalog  | Db   | Command | Time | State | QueryId
| Info             | FE            | CloudCluster |
+------------------+------+------+--------------------+---------------------+----------+------+---------+------+-------+-----------
-----------------------+------------------+--------------+--------------+
| No               |    2 | root | 172.20.32.136:54850 | 2025-05-11 10:41:52 | internal |      | Query   |    6 | OK    |
12ccf7f95c1c4d2c-b03fa9c652757c15 | select sleep(20) | 172.20.32.152 | NULL         |
| Yes              |    3 | root | 172.20.32.136:54862 | 2025-05-11 10:41:55 | internal |      | Query   |    0 | OK    |
b710ed990d4144ee-8b15bb53002b7710 | show processlist | 172.20.32.152 | NULL         |
| No               |    1 | root | 172.20.32.136:47964 | 2025-05-11 10:41:54 | internal |      | Sleep   |   11 | EOF   |
b60daa992bac4fe4-b29466aacce67d27 |                  | 172.20.32.153 | NULL         |
+------------------+------+------+--------------------+---------------------+----------+------+---------+------+-------+-----------
-----------------------+------------------+--------------+--------------+
```

- `CurrentConnected`: `Yes` indicates the connection corresponding to the current session.
- `Id`: The unique identifier of the connection, i.e., Connection ID.
- `QueryId`: The unique identifier of the Query. Displays the Query Id of the most recently executed or currently executing SQL command.

Note that by default, `SHOW PROCESSLIST` only displays all session connections on the FE node that the current session is connected to, and does not display session connections from other FE nodes.

If you want to display session connections from all FE nodes, you need to set the following session variable:

```
SET show_all_fe_connection=true;
```

Then execute the `SHOW PROCESSLIST` command again to display session connections from all FE nodes.

You can also view through the system table in `information_schema`:

```
SELECT * FROM information_schema.processlist;
```

By default, `processlist` displays session connections from all FE nodes without requiring additional settings.

TRACE ID

> This feature is supported since versions 2.1.11 and 3.0.7.

By default, the system automatically generates a Query ID for each query. Users need to first obtain the Query ID through the `processlist` system table before performing a KILL operation.

In addition, users can also customize a Trace ID and perform KILL operations using the Trace ID.

```
SET session_context = "trace_id:your_trace_id";
```

Where `your_trace_id` is the user-defined Trace ID. It can be any string but cannot contain the ; symbol.

Trace ID is a session-level parameter and only applies to the current session. Doris will map subsequent query requests in the current session to this Trace ID.

Kill Requests

The `KILL` statement supports canceling specified query operations as well as disconnecting specified session connections.

Regular users can cancel queries sent by their own user through the `KILL` operation. ADMIN users can cancel queries sent by themselves and any other users.

Kill Query

Syntax:

```
KILL QUERY "query_id" | "trace_id" | connection_id;
```

`KILL QUERY` is used to cancel a specified running query operation.

- `"query_id"`

  The Query ID obtained through the `processlist` system table. Must be wrapped in quotes. For example:

  `KILL QUERY "d36417cc05ff41ab-9d3afe49be251055";`

  This operation will attempt to find the Query ID on all FE nodes and cancel the corresponding query.

- `"trace_id"`

  The Trace ID customized through `session_context`. Must be wrapped in quotes. For example:

```
KILL QUERY "your_trace_id";
```

This operation will attempt to find the Trace ID on all FE nodes and cancel the corresponding query.

> This feature is supported since versions 2.1.11 and 3.0.7.

- `connection_id`

  The Connection ID obtained through the `processlist` system table. Must be an integer greater than 0 and cannot be wrapped in quotes. For example:

  ```
  KILL QUERY 55;
  ```

  This operation only applies to session connections on the currently connected FE, and will cancel the query currently being executed on the corresponding session connection.

Kill Connection

Killing a connection will disconnect the specified session connection and also cancel any query operations being executed on the connection.

Syntax:

```
KILL [CONNECTION] connection_id;
```

The `CONNECTION` keyword can be omitted.

- `connection_id`

  The Connection ID obtained through the `processlist` system table. Must be an integer greater than 0 and cannot be wrapped in quotes. For example:

  ```
  KILL CONNECTION 55;
  KILL 55;
  ```

  Connection IDs on different FEs may be the same, but this operation only affects the session connection on the currently connected FE.

Best Practices

1. Implementing query management through custom Trace ID

   Custom Trace IDs allow you to specify unique identifiers for queries in advance, making it easier for management system to implement the [Cancel Query] function. You can customize Trace IDs in the following ways:

   - Set `session_context` before each query

Users generate their own Trace ID. It is recommended to use UUID to ensure the uniqueness of the Trace ID.

```
SET session_context="trace_id:your_trace_id";
SELECT * FROM table ...;
```

- Add Trace ID in the query statement

```
SELECT /*+SET_VAR(session_context="trace_id:your_trace_id")*/ * FROM table ...;
```

Afterward, management system can cancel running operations at any time using the Trace ID.

Job Scheduler | Workload Management

Job Scheduler

Background

In the context of increasing demands for refined data management, scheduled scheduling plays an important role. It is typically applied in the following scenarios:

- Regular data updates, such as periodic data imports and ETL operations, reducing manual intervention and improving the efficiency and accuracy of data processing.
- Synchronizing external data sources with Catalog to ensure efficient and accurate integration of multi-source data into the target system, meeting complex business analysis needs.
- Regularly cleaning up expired/invalid data to free up storage space and prevent excessive expired/invalid data from impacting system performance.

In versions of Apache Doris prior to 2.1, it was typically necessary to rely on external scheduling systems, such as scheduling via business code or introducing third-party scheduling tools and distributed scheduling platforms to meet the above requirements. However, due to limitations of the external systems themselves, they may not be able to satisfy Doris's requirements for scheduling strategy and resource management flexibility. Additionally, if the external scheduling system fails, it not only increases business risks but also requires additional operational time and manpower to address.

Job Scheduler

To solve the above problems, Apache Doris introduced the Job Scheduler feature in version 2.1, achieving autonomous task scheduling capabilities with scheduling precision reaching the second level. The introduction of this feature not only ensures the integrity and consistency of data imports but also allows users to flexibly and conveniently adjust scheduling strategies. At the same time, by reducing dependence on external systems, it also lowers the risk of system failures and operational costs, providing community users with a more unified and reliable user experience.

Doris Job Scheduler is a task management system based on preset schedules, capable of triggering predefined operations at specific points in time or at specified time intervals, achieving automated task execution. The Job Scheduler has the following features: - **Efficient scheduling**: The Job Scheduler can arrange tasks and events within specified time intervals, ensuring the efficiency of data processing. It uses a time wheel algorithm to ensure events can be triggered precisely to the second level. - **Flexible scheduling**: The Job Scheduler provides multiple scheduling options, such as scheduling at minute, hour, day, or week intervals. It also supports one-time scheduling as well as recurring (periodic) event scheduling, and periodic scheduling can specify start and end times. - **Event pool and high-performance processing queue**: The Job Scheduler uses Disruptor to implement a high-performance producer-consumer model, maximizing the avoidance of task execution overload. - **Traceable scheduling records**: The Job Scheduler stores the latest Task execution records (configurable). Task execution records can be viewed through simple commands,

ensuring process traceability. - **High availability**: Leveraging Doris's own high-availability mechanisms, the Job Scheduler can easily achieve self-recovery and high availability.

Related Documentation: CREATE-JOB

Syntax Explanation

A valid Job statement must include the following:

- The keyword **CREATE JOB** must be followed by the job name, which uniquely identifies the event in the database.
- The **ON SCHEDULE** clause is used to specify the type, trigger time, and frequency of the Job.
    - **AT timestamp** is used for one-time events. It specifies that the JOB will execute only once at the given date and time, and **AT current_timestamp** specifies the current date and time. Since the JOB is created, it will run immediately and can also be used for asynchronous task creation.
    - **EVERY**: Used for periodic jobs, specifying the execution frequency of the job. A time interval (week, day, hour, minute) must be specified after the keyword.
        - **Interval**: Specifies the frequency of job execution. **1 DAY** means the job executes once a day, **1 HOUR** means once an hour, **1 MINUTE** means once a minute, and **1 WEEK** means once a week.
        - The **EVERY** clause includes an optional **STARTS** clause. After **STARTS** is a timestamp value, which defines the start time for repetition, and **CURRENT_TIMESTAMP** specifies the current date and time. Once the JOB is created, it runs immediately.
        - The **EVERY** clause includes an optional **ENDS** clause. After the **ENDS** keyword is a timestamp value, which defines the time when the JOB event stops running.
- The **DO** clause is used to specify the operation to be performed when the Job is triggered. Currently, only **INSERT** statements are supported.

```
CREATE
JOB
  job_name
  ON SCHEDULE schedule
  [COMMENT 'string']
  DO execute_sql;

schedule: {
    AT timestamp
    | EVERY interval
    [STARTS timestamp ]
    [ENDS timestamp ]
}
interval:
    quantity { WEEK |DAY | HOUR | MINUTE}
```

Examples Below:

```
CREATE JOB my_job ON SCHEDULE EVERY 1 MINUTE DO INSERT INTO db1.tbl1 SELECT * FROM db2.tbl2;
```

This statement creates a job named `my_job` that executes once every minute. The operation performed is importing data from `db2.tbl2` into `db1.tbl1`.

Usage Examples

Create a one-time Job: Execute once at `2025-01-01 00:00:00`, importing data from `db2.tbl2` into `db1.tbl1`.

```
CREATE JOB my_job ON SCHEDULE AT '2025-01-01 00:00:00' DO INSERT INTO db1.tbl1 SELECT * FROM db2.tbl2;
```

Create a periodic Job without a specified end time: Starting at `2025-01-01 00:00:00`, execute once a day, importing data from `db2.tbl2` into `db1.tbl1`.

```
CREATE JOB my_job ON SCHEDULE EVERY 1 DAY STARTS '2025-01-01 00:00:00' DO INSERT INTO db1.tbl1 SELECT * FROM db2.tbl2 WHERE
create_time >=  days_add(now(),-1);
```

Create a periodic Job with a specified end time: Starting at `2025-01-01 00:00:00`, execute once a day, importing data from `db2.tbl2` into `db1.tbl1`, and end at `2026-01-01 00:10:00`.

```
CREATE JOB my_job ON SCHEDULE EVERY 1 DAY STARTS '2025-01-01 00:00:00' ENDS '2026-01-01 00:10:00' DO INSERT INTO db1.tbl1 SELECT *
FROM db2.tbl2 WHERE create_time >=  days_add(now(),-1);
```

Implementing Asynchronous Execution with Job: Although Jobs in Doris are created as synchronous tasks, their execution process is asynchronous. This feature makes Jobs highly suitable for implementing asynchronous tasks, such as common `INSERT INTO SELECT` operations.

For example, if you need to import data from `db2.tbl2` into `db1.tbl1`, you only need to specify the JOB as a one-time task and set the start time to the current time.

```
CREATE JOB my_job ON SCHEDULE AT current_timestamp DO INSERT INTO db1.tbl1 SELECT * FROM db2.tbl2;
```

Data Automatic Synchronization Based on Catalog and Job Scheduler

Taking an e-commerce scenario as an example, users often need to extract business data from MySQL and synchronize this data into Doris for data analysis, thereby supporting precise marketing activities. The Job Scheduler can work in conjunction with the Multi Catalog data lake capability to efficiently complete periodic data synchronization across data sources.

```
CREATE TABLE IF NOT EXISTS user.activity (
    `user_id` INT NOT NULL,
    `date` DATE NOT NULL,
    `city` VARCHAR(20),
    `age` SMALLINT,
    `sex` TINYINT,
    `last_visit_date` DATETIME DEFAULT '1970-01-01 00:00:00',
    `cost` BIGINT DEFAULT '0',
```

```
    `max_dwell_time` INT DEFAULT '0',
    `min_dwell_time` INT DEFAULT '99999'
);
INSERT INTO user.activity VALUES
    (10000, '2017-10-01', 'BeiJing', 20, 0, '2017-10-01 06:00:00', 20, 10, 10),
    (10000, '2017-10-01', 'BeiJing', 20, 0, '2017-10-01 07:00:00', 15, 2, 2),
    (10001, '2017-10-01', 'BeiJing', 30, 1, '2017-10-01 17:05:00', 2, 22, 22),
    (10002, '2017-10-02', 'ShangHai', 20, 1, '2017-10-02 12:59:00', 200, 5, 5),
    (10003, '2017-10-02', 'GuangZhou', 32, 0, '2017-10-02 11:20:00', 30, 11, 11),
    (10004, '2017-10-01', 'ShenZhen', 35, 0, '2017-10-01 10:00:00', 100, 3, 3),
    (10004, '2017-10-03', 'ShenZhen', 35, 0, '2017-10-03 10:20:00', 11, 6, 6);
```

user_id	date	city	age	sex	last_visit_date	cost	max_dwell_time	min_dwell_time
10000	2017/10/1	BeiJing	20	0	2017/10/1 6:00	20	10	10
10000	2017/10/1	BeiJing	20	0	2017/10/1 7:00	15	2	2
10001	2017/10/1	BeiJing	30	1	2017/10/1 17:05	2	22	22
10002	2017/10/2	ShangHai	20	1	2017/10/2 12:59	200	5	5
10003	2017/10/2	GuangZhou	32	0	2017/10/2 11:20	30	11	11
10004	2017/10/1	ShenZhen	35	0	2017/10/1 10:00	100	3	3
10004	2017/10/3	ShenZhen	35	0	2017/10/3 10:20	11	6	6

Using the table above as an example, the user wants to query users who meet specific numerical conditions such as total spending amount, last visit time, gender, and city, and import the information of users who meet these conditions into Doris for subsequent targeted promotions.

1. First, create a Doris table

```
CREATE TABLE IF NOT EXISTS user_activity
  (
  `user_id` LARGEINT NOT NULL,
  `date` DATE NOT NULL,
  `city` VARCHAR(20),
  `age` SMALLINT,
  `sex` TINYINT,
  `last_visit_date` DATETIME REPLACE DEFAULT "1970-01-01 00:00:00",
  `cost` BIGINT SUM DEFAULT "0",
  `max_dwell_time` INT MAX DEFAULT "0",
  `min_dwell_time` INT MIN DEFAULT "99999"
  )
  AGGREGATE KEY(`user_id`, `date`, `city`, `age`, `sex`)
  DISTRIBUTED BY HASH(`user_id`) BUCKETS 1
  PROPERTIES (
```

```
  "replication_allocation" = "tag.location.default: 1"
  );
```

2. Secondly, create a Catalog corresponding to the MySQL database.

```
CREATE CATALOG activity PROPERTIES (
  "type"="jdbc",
  "user"="root",
  "password"="123456",
  "jdbc_url" = "jdbc:mysql://127.0.0.1:3306/user?useSSL=false",
  "driver_url" = "mysql-connector-java-5.1.49.jar",
  "driver_class" = "com.mysql.jdbc.Driver"
  );
```

3. Finally, import MySQL data into Doris. Use the Catalog + Insert Into method to import the full dataset. Since full imports may cause system service fluctuations, it is usually recommended to perform this operation during business off-peak hours.

- One-time scheduling: As shown in the code below, use a one-time task to trigger the full import task at a scheduled time, with the trigger set for 3:00 AM.

```
CREATE JOB one_time_load_job
  ON SCHEDULE
  AT '2024-8-10 03:00:00'
  DO
  INSERT INTO user_activity SELECT * FROM activity.user.activity
```

- Periodic scheduling: Users can also create a periodic scheduling task to update the latest data regularly.

```
CREATE JOB schedule_load
  ON SCHEDULE EVERY 1 DAY
  DO
  INSERT INTO user_activity SELECT * FROM activity.user.activity where last_visit_date >=  days_add(now(),-1)
```

Design and Implementation

Efficient scheduling often comes with significant resource consumption, and high-precision scheduling is even more demanding. The traditional approach involves using Java's built-in scheduling capabilities—scheduled tasks that access threads periodically—or using various scheduling utility libraries. However, these methods have significant issues in terms of precision and memory usage. To better guarantee performance while reducing resource consumption, we chose to combine the TimingWheel algorithm with Disruptor to achieve second-level task scheduling.

Specifically, we use Netty's `HashedWheelTimer` to implement the Timing Wheel algorithm. The Job Manager periodically (defaulting to every ten minutes) places future events into the timing wheel for scheduling. To ensure efficient task triggering and avoid excessive resource usage, we use Disruptor to build a single-producer, multiple-consumer model. The timing wheel only triggers events but does not execute tasks directly. For tasks that need to be triggered upon expiration, they are placed in the Dispatch thread, which is responsible for distributing the tasks to the appropriate execution thread pool. For tasks that need to be executed immediately, they are directly delivered to the corresponding task execution thread pool.

For one-time events, the event definition is deleted after the task is scheduled. For periodic events, system events in the timing wheel will regularly fetch the tasks for the next execution cycle. This helps avoid large numbers of tasks being concentrated in a single bucket, reducing unnecessary traversals and improving processing efficiency.

For transactional tasks, the Job Scheduler can ensure that the execution results of transactional tasks match expectations through strong associations with transactions and the transaction callback mechanism, thus ensuring data integrity and consistency.

Future Plans

Doris Job Scheduler is a powerful and flexible task scheduling tool, an essential feature in data processing. In addition to common use cases such as data lake analytics and internal ETL, Job Scheduler also plays a key role in the implementation of asynchronous materialized views. An asynchronous materialized view is a precomputed and stored result set, where the frequency of data updates is closely related to changes in the source tables. When the source table data is updated frequently, periodic refreshing of the materialized view is required to keep its data up-to-date. In version 2.1, we cleverly utilized the JOB scheduling feature to ensure the consistency between materialized views and source table data, significantly reducing the cost of manual intervention.

In the future, Doris Job Scheduler will also support the following features: - Support for viewing the distribution of tasks executed during different time periods via the UI. - Support for JOB workflow orchestration, i.e., DAG JOB. This means we can implement internal data warehouse task orchestration, and with the Catalog functionality, it will more efficiently complete data processing and analysis tasks. - Support for scheduling import tasks, UPDATE, and DELETE operations.

Disaster Recovery Management Overview

Doris provides disaster recovery capabilities, helping users effectively respond to data loss issues caused by hardware failures, software errors, or human mistakes through three main functions: cross-cluster data synchronization, backup and recovery, and recycle bin recovery, ensuring high availability and reliability of data.

1. Cross-Cluster Data Synchronization

Doris's cross-cluster data synchronization feature supports real-time data replication between different Doris clusters, ensuring that important data is distributed across multiple physically isolated clusters, achieving regional disaster recovery.

Key Features:

- **Real-time Synchronization**: Supports full and incremental synchronization. Full synchronization replicates all data in the initial phase; incremental synchronization continuously captures and synchronizes data changes, including data (addition, modification, deletion) and table structure changes (DDL).
- **Data Consistency**: Records data changes through a logging mechanism (such as Binlog), ensuring that the target cluster's data is completely consistent with the source cluster's data.
- **Regional Disaster Recovery**: Supports synchronization between clusters in different geographical locations, allowing other clusters to quickly take over business when one cluster fails.
- **Multi-Scenario Applications**: Suitable for disaster recovery backups, business separation (such as read-write separation), active-active clusters, and other scenarios.

Application Scenario Example:

A company has deployed two Doris clusters in different cities, with Cluster A as the primary cluster and Cluster B as the backup cluster. Through cross-cluster data synchronization, when Cluster A is interrupted due to a natural disaster, Cluster B can take over the business, minimizing downtime.

2. Backup and Recovery

Doris provides backup and recovery functions to regularly save data snapshots, preventing data loss due to unexpected events.

Key Features:

- **Backup**: Supports full backups of specified databases, tables, or partitions, saving complete data snapshots.
- **Recovery**: Supports recovery of databases, tables, or partitions from snapshots.

Application Scenario Example:

A company regularly backs up data and stores backup files in object storage services (such as Amazon S3). When an important table is accidentally deleted, the backup function quickly restores the lost data, ensuring normal business operations.

3. Recycle Bin Recovery

Doris provides a recycle bin feature, offering users a quick way to recover recently deleted data, reducing the impact of operational errors.

Key Features:

- **Temporary Deletion**: Tables or databases that are deleted are first moved to the recycle bin instead of being permanently deleted immediately.
- **Retention Period**: Deleted data is retained in the recycle bin for a configurable period, during which users can choose to recover it.
- **Quick Recovery**: Data can be easily retrieved from the recycle bin without needing a complete backup recovery.
- **Data Security**: If recovery is not needed, data in the recycle bin will be automatically cleared after the retention period.

Application Scenario Example:

A team accidentally deleted an important table during routine operations. Using the recycle bin feature, they quickly recovered the deleted data, avoiding a complex backup recovery process while ensuring business continuity.

Backup and Restore Overview

Introduction

Doris provides support for backup and restore operations. These features allow users to back up data from databases, tables, or partitions to remote storage systems and restore it when needed.

Requirements

- **Administrator Privileges**: Only users with **ADMIN** privileges can perform backup and restore operations.

Key Concepts

Snapshot: A snapshot is a time-point capture of data in a database, table, or partition. When creating a snapshot, a snapshot label must be specified, and a timestamp is generated upon completion, which can identify a snapshot through the Repository, snapshot label, and timestamp.

Repository: The remote storage location where backup files are stored. Supported remote storage includes S3, Azure, GCP, OSS, COS, MinIO, HDFS, and other S3-compatible object storage.

Backup Operation: The backup operation involves creating a snapshot of a database, table, or partition, uploading the snapshot file to a remote Repository, and storing metadata related to the backup.

Restore Operation: The restore operation involves retrieving a backup from the remote Repository and restoring it to the Doris cluster.

Key Features

1. **Backup Data**: Doris allows you to back up data from tables, partitions, or entire databases by creating snapshots. Data is backed up in file format and stored in HDFS, S3, or other S3-compatible remote storage systems.

2. **Restore Data**: You can restore backup data from the remote Repository to any Doris cluster. This includes full database restoration, full table restoration, and partition-level restoration, allowing for flexible data recovery.

3. **Snapshot Management**: Data is backed up in the form of snapshots. These snapshots are uploaded to remote storage systems and can be restored when needed. The restoration process involves downloading the snapshot file and mapping it to local metadata to make it effective.

4. **Data Migration**: In addition to backup and restore, this feature also supports data migration between different Doris clusters. You can back up data to a remote storage system and restore it to another Doris cluster, facilitating cluster migration scenarios.

5. **Replication Control**: When restoring data, you can specify the number of replicas for the restored data to ensure redundancy and fault tolerance.

Limitations

1. **Decoupling of Storage and Computing**: The storage-computing separation model does not support backup and restore.
2. **Asynchronous Materialized Views (MTMV) Not Supported**: Backup or restore of **asynchronous materialized views (MTMV)** is not supported. These views are not considered in backup and restore operations.
3. **Tables with Storage Policies Not Supported**: Tables that use **storage policies** **do not support** backup and restore operations.
4. **Incremental Backup**: Currently, Doris only supports full backups. Incremental backups (only storing data changed since the last backup) are not supported; you can back up specific partitions to achieve incremental backup.
5. **colocate_with Attribute**: During backup or restore operations, Doris does not retain the `colocate_with` attribute of the table. This may need to be reconfigured for colocated tables after restoration.
6. **Dynamic Partition Support**: After restoring a table, you need to manually enable this attribute using the `ALTER TABLE` command.
7. **Single Concurrency**: Only one backup or restore task can run simultaneously under a single database.

Backup

For concepts related to backup, please refer to Backup and Restore. This guide provides the steps to create a Repository and back up data.

Step 1. Create Repository

Use the appropriate statement to create a Repository based on your storage choice. For detailed usage, please refer to Create Repository. When backing up using the same path for the Repository across different clusters, ensure to use different labels to avoid conflicts that may cause data confusion.

Option 1: Create Repository on S3

To create a Repository on S3 storage, use the following SQL command:

```
CREATE REPOSITORY `s3_repo`
WITH S3
ON LOCATION "s3://bucket_name/s3_repo"
PROPERTIES
(
    "s3.endpoint" = "https://s3.us-east-1.amazonaws.com",
    "s3.region" = "us-east-1",
    "s3.access_key" = "ak",
    "s3.secret_key" = "sk"
);
```

- Replace `bucket_name` with your S3 bucket name.
- Provide the appropriate endpoint, access key, secret key, and region for S3 setup.

Option 2: Create Repository on Azure

Azure is supported since 3.0.4.

To create a Repository on Azure storage, use the following SQL command:

```
CREATE REPOSITORY `azure_repo`
WITH S3
ON LOCATION "s3://bucket_name/azure_repo"
PROPERTIES
(
    "s3.endpoint" = "selectdbcloudtestwestus3.blob.core.windows.net",
    "s3.region" = "dummy_region",
    "s3.access_key" = "ak",
    "s3.secret_key" = "sk",
    "provider" = "AZURE"
);
```

- Replace `bucket_name` with your Azure container name.
- Provide your Azure storage account and key for authentication.
- `s3.region` is a dummy but required field.

- The `provider` must be set to `AZURE` for Azure storage.

Option 3: Create Repository on GCP

To create a Repository on Google Cloud Platform (GCP) storage, use the following SQL command:

```
CREATE REPOSITORY `gcp_repo`
WITH S3
ON LOCATION "s3://bucket_name/backup/gcp_repo"
PROPERTIES
(
    "s3.endpoint" = "storage.googleapis.com",
    "s3.region" = "US-WEST2",
    "s3.access_key" = "ak",
    "s3.secret_key" = "sk"
);
```

- Replace `bucket_name` with your GCP bucket name.
- Provide your GCP endpoint, access key, and secret key.
- `s3.region` is a dummy but required field.

Option 4: Create Repository on OSS (Alibaba Cloud Object Storage Service)

To create a Repository on OSS, use the following SQL command:

```
CREATE REPOSITORY `oss_repo`
WITH S3
ON LOCATION "s3://bucket_name/oss_repo"
PROPERTIES
(
    "s3.endpoint" = "oss.aliyuncs.com",
    "s3.region" = "cn-hangzhou",
    "s3.access_key" = "ak",
    "s3.secret_key" = "sk"
);
```

- Replace `bucket_name` with your OSS bucket name.
- Provide your OSS endpoint, region, access key, and secret key.

Option 5: Create Repository on MinIO

To create a Repository on MinIO storage, use the following SQL command:

```
CREATE REPOSITORY `minio_repo`
WITH S3
ON LOCATION "s3://bucket_name/minio_repo"
PROPERTIES
(
    "s3.endpoint" = "yourminio.com",
    "s3.region" = "dummy-region",
    "s3.access_key" = "ak",
    "s3.secret_key" = "sk",
    "use_path_style" = "true"
);
```

- Replace `bucket_name` with your MinIO bucket name.
- Provide your MinIO endpoint, access key, and secret key.
- `s3.region` is a dummy but required field.
- If you do not enable Virtual Host-style, then `use_path_style` must be true.

Option 6: Create Repository on HDFS

To create a Repository on HDFS storage, use the following SQL command:

```
CREATE REPOSITORY `hdfs_repo`
WITH hdfs
ON LOCATION "hdfs://127.0.0.1:9000/prefix_path/hdfs_repo"
PROPERTIES
(
    "fs.defaultFS" = "hdfs://127.0.0.1:9000",
    "hadoop.username" = "doris-test"
)
```

- Replace `prefix_path` with the actual path.
- Provide your HDFS endpoint and username.

Step 2. Backup

Refer to the following statements to back up databases, tables, or partitions. For detailed usage, please refer to Backup.

It is recommended to use meaningful label names, such as those containing the databases and tables included in the backup.

Option 1: Backup Current Database

The following SQL statement backs up the current database to a Repository named `example_repo`, using the snapshot label `exampledb_20241225`.

```
BACKUP SNAPSHOT exampledb_20241225
TO example_repo;
```

Option 2: Backup Specified Database

The following SQL statement backs up a database named `destdb` to a Repository named `example_repo`, using the snapshot label `destdb_20241225`.

```
BACKUP SNAPSHOT destdb.`destdb_20241225`
TO example_repo;
```

Option 3: Backup Specified Tables

The following SQL statement backs up two tables to a Repository named `example_repo`, using the snapshot label `exampledb_tbl_tbl1_20241225`.

```
BACKUP SNAPSHOT exampledb_tbl_tbl1_20241225
TO example_repo
ON (example_tbl, example_tbl1);
```

Option 4: Backup Specified Partitions

The following SQL statement backs up a table named `example_tbl2` and two partitions named `p1` and `p2` to a Repository named `example_repo`, using the snapshot label `example_tbl_p1_p2_tbl1_20241225`.

```
BACKUP SNAPSHOT example_tbl_p1_p2_tbl1_20241225
TO example_repo
ON
(
      example_tbl PARTITION (p1,p2),
      example_tbl2
);
```

Option 5: Backup Current Database Excluding Certain Tables

The following SQL statement backs up the current database to a Repository named `example_repo`, using the snapshot label `exampledb_20241225`, excluding two tables named `example_tbl` and `example_tbl1`.

```
BACKUP SNAPSHOT exampledb_20241225
TO example_repo
EXCLUDE
(
      example_tbl,
      example_tbl1
);
```

Step 3. View Recent Backup Job Execution Status

The following SQL statement can be used to view the execution status of recent backup jobs.

```
mysql> show BACKUP\G;
*************************** 1. row ***************************
                  JobId: 17891847
           SnapshotName: exampledb_20241225
                 DbName: example_db
                  State: FINISHED
             BackupObjs: [example_db.example_tbl]
             CreateTime: 2022-04-08 15:52:29
   SnapshotFinishedTime: 2022-04-08 15:52:32
     UploadFinishedTime: 2022-04-08 15:52:38
           FinishedTime: 2022-04-08 15:52:44
       UnfinishedTasks:
                Progress:
             TaskErrMsg:
                 Status: [OK]
                Timeout: 86400
   1 row in set (0.01 sec)
```

Step 4. View Existing Backups in Repository

The following SQL statement can be used to view existing backups in a Repository named `example_repo`, where the Snapshot column is the snapshot label, and the Timestamp is the timestamp.

```
mysql> SHOW SNAPSHOT ON example_repo;
+-----------------+---------------------+--------+
| Snapshot        | Timestamp           | Status |
+-----------------+---------------------+--------+
| exampledb_20241225 | 2022-04-08-15-52-29 | OK     |
+-----------------+---------------------+--------+
1 row in set (0.15 sec)
```

Step 5. Cancel Backup (if necessary)

You can use `CANCEL BACKUP FROM db_name;` to cancel a backup task in a database. For more specific usage, refer to Cancel Backup.

Restore

Prerequisites

1. Ensure you have **administrator** privileges to perform the restore operation.
2. Ensure you have an existing **Repository** to store the backup. If not, follow the steps to create a Repository and perform a backup.
3. Ensure you have a valid **backup** snapshot available for restore.

1. Get the Backup Timestamp of the Snapshot

The following SQL statement can be used to view existing backups in the Repository named `example_repo`.

```
mysql> SHOW SNAPSHOT ON example_repo;
+--------------------+---------------------+--------+
| Snapshot           | Timestamp           | Status |
+--------------------+---------------------+--------+
| exampledb_20241225 | 2022-04-08-15-52-29 | OK     |
+--------------------+---------------------+--------+
1 row in set (0.15 sec)
```

2. Restore from Snapshot

Option 1: Restore Snapshot to Current Database

The following SQL statement restores the snapshot labeled `restore_label1` with the timestamp `2022-04-08-15-52-29` from the Repository named `example_repo` to the current database.

```
RESTORE SNAPSHOT `restore_label1`
FROM `example_repo`
PROPERTIES
(
    "backup_timestamp"="2022-04-08-15-52-29"
);
```

Option 2: Restore Snapshot to Specified Database

The following SQL statement restores the snapshot labeled `restore_label1` with the timestamp `2022-04-08-15-52-29` from the Repository named `example_repo` to a database named `destdb`.

```
RESTORE SNAPSHOT destdb.`restore_label1`
FROM `example_repo`
PROPERTIES
(
    "backup_timestamp"="2022-04-08-15-52-29"
);
```

Option 3: Restore a Single Table from Snapshot

Restore the table `backup_tbl` from the snapshot in `example_repo` to the current database, with the snapshot labeled `restore_label1` and timestamp `2022-04-08-15-52-29`.

```
RESTORE SNAPSHOT `restore_label1`
FROM `example_repo`
ON ( `backup_tbl` )
PROPERTIES
(
    "backup_timestamp"="2022-04-08-15-52-29"
);
```

Option 4: Restore Partitions and Tables from Snapshot

Restore partitions p1 and p2 of the table `backup_tbl`, as well as the table `backup_tbl2` to the current database `example_db1`, renaming it to `new_tbl`, from the backup snapshot `snapshot_2`, with the snapshot timestamp `"2018-05-04-17-11-01"`.

```
RESTORE SNAPSHOT `restore_label1`
FROM `example_repo`
ON
(
    `backup_tbl` PARTITION (`p1`, `p2`),
    `backup_tbl2` AS `new_tbl`
)
PROPERTIES
(
    "backup_timestamp"="2022-04-08-15-55-43"
);
```

3. Check the Execution Status of the Restore Job

```
mysql> SHOW RESTORE\G;
*************************** 1. row ***************************
               JobId: 17891851
               Label: snapshot_label1
           Timestamp: 2022-04-08-15-52-29
              DbName: default_cluster:example_db1
               State: FINISHED
           AllowLoad: false
     ReplicationNum: 3
        RestoreObjs: {
  "name": "snapshot_label1",
  "database": "example_db",
  "backup_time": 1649404349050,
  "content": "ALL",
  "olap_table_list": [
    {
      "name": "backup_tbl",
      "partition_names": [
        "p1",
        "p2"
      ]
    }
  ],
  "view_list": [],
```

```
  "odbc_table_list": [],
  "odbc_resource_list": []
}
          CreateTime: 2022-04-08 15:59:01
    MetaPreparedTime: 2022-04-08 15:59:02
SnapshotFinishedTime: 2022-04-08 15:59:05
DownloadFinishedTime: 2022-04-08 15:59:12
        FinishedTime: 2022-04-08 15:59:18
     UnfinishedTasks:
            Progress:
           TaskErrMsg:
              Status: [OK]
             Timeout: 86400
1 row in set (0.01 sec)
```

Overview | CCR

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

Overview

Overview

CCR (Cross Cluster Replication) is a cross-cluster data synchronization mechanism that synchronizes data changes from the source cluster to the target cluster at the database or table level. It is mainly used to enhance the data availability of online services, isolate read and write loads, and build a dual-site, three-center architecture. CCR currently does not support the separation of computing and storage modes.

Applicable Scenarios

CCR is suitable for the following common scenarios:

- **Disaster Recovery Backup**: Backing up enterprise data to another cluster and data center to ensure data recovery in case of business interruption or data loss. Industries such as finance, healthcare, and e-commerce typically require this high SLA disaster recovery backup.
- **Read-Write Separation**: By separating data query operations from write operations, the mutual impact between reads and writes is reduced, enhancing service stability. In scenarios with high concurrency or heavy write pressure, adopting read-write separation can effectively distribute the load and improve database performance and stability.
- **Data Centralization**: The headquarters of a group needs to manage and analyze data from branch offices located in different regions to avoid management chaos and decision-making errors caused by data inconsistency, thereby improving group management efficiency and decision quality.
- **Isolated Upgrades**: When upgrading system clusters, using CCR allows for validation and testing in the new cluster, avoiding rollback difficulties caused by version compatibility issues. Users can gradually upgrade each cluster while ensuring data consistency.
- **Cluster Migration**: When relocating a Doris cluster or replacing equipment, using CCR can synchronize data from the old cluster to the new cluster, ensuring data consistency during the migration process.

Job Categories

CCR supports two types of jobs:

- **Database-Level jobs**: Synchronize data for the entire database.

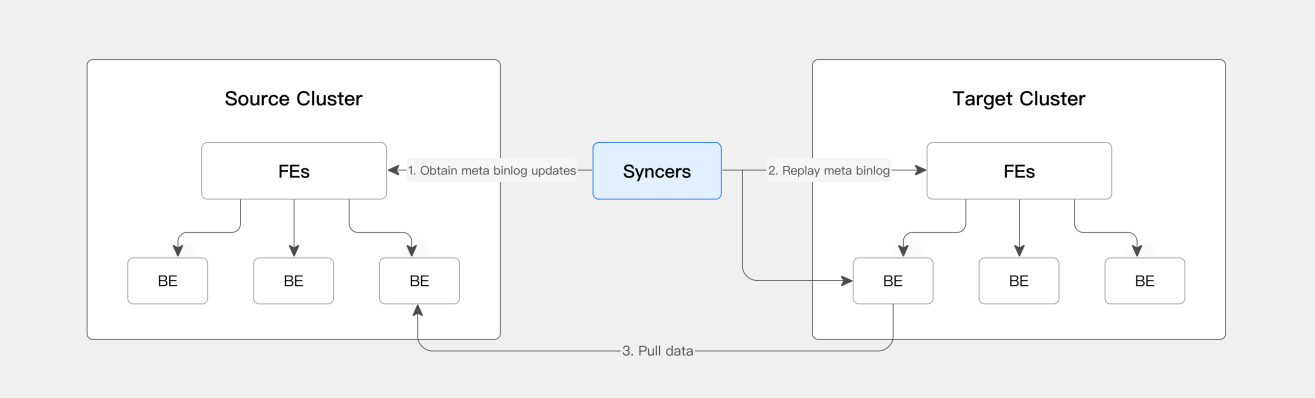
- **Table-Level jobs**: Only synchronize data for specified tables. Note that table-level synchronization does not support renaming or replacing tables. Each database in Doris can only run one snapshot job at a time, so full synchronization jobs for table-level synchronization need to be queued.

Principles and Architecture

Terminology

- **Source Cluster**: The cluster where the data source resides, usually the cluster where business data is written.
- **Target Cluster**: The target cluster for cross-cluster synchronization.
- **binlog**: The change log of the source cluster, which includes schema and data changes.
- **Syncer**: A lightweight process responsible for synchronizing data.
- **Upstream**: Refers to the upstream database in database-level jobs and the upstream table in table-level jobs.
- **Downstream**: Refers to the downstream database in database-level jobs and the downstream table in table-level jobs.

Architecture Description



CCR Architecture Description

CCR mainly relies on a lightweight process: `Syncer`. The `Syncer` is responsible for obtaining binlogs from the source cluster and applying metadata to the target cluster, notifying the target cluster to pull data from the source cluster, thus achieving full synchronization and incremental synchronization.

Principles

1. **Full Synchronization**:
    - CCR jobs will first perform full synchronization, copying the upstream data completely to the downstream.
2. **Incremental Synchronization**:
    - After full synchronization is complete, CCR jobs will continue with incremental synchronization to maintain data consistency between upstream and downstream.
3. **Restarting Full Synchronization**:
    - When encountering DDL operations that do not support incremental synchronization, CCR jobs will restart full synchronization. For specific DDL operations that do not support incremental

synchronization, please refer to Feature Details.
    - If the upstream binlog is interrupted due to expiration or other reasons, incremental synchronization will stop and restart full synchronization.
4. **Restarting Full Synchronization**:
    - During full synchronization, incremental synchronization will be paused.
    - After full synchronization is complete, the downstream data table will undergo atomic replacement to ensure data consistency.
    - After full synchronization is complete, incremental synchronization will resume.

Synchronization Methods

CCR supports four synchronization methods:

Synchronization Method	Principle	Trigger Timing
Full Sync	The upstream performs a full backup, and the downstream performs a restore. DB-level jobs trigger DB backups, and table-level jobs trigger table backups.	First synchronization or triggered by specific operations. For trigger conditions, please refer to Feature Details.
Partial Sync	The upstream performs table or partition-level backups, and the downstream performs table or partition-level restores.	Triggered by specific operations, for trigger conditions, please refer to Feature Details.
TXN	Incremental data synchronization, starting synchronization after the upstream commits.	Triggered by specific operations, for trigger conditions, please refer to Feature Details.
SQL	Replaying upstream SQL operations on the downstream.	Triggered by specific operations, for trigger conditions, please refer to Feature Details.

Download

requirement: glibc >= 2.28

Version	Arch	Tarball	SHA256
2.1	ARM64	ccr-syncer-2.1.10-rc08-arm64.tar.xz	060093e90150ee24f8a784066436a0a4a1116876ebd6f33d5a844e2dc67f10b0
2.1	X64	ccr-syncer-2.1.10-rc08-x64.tar.xz	656c0a46e3f0e12b9ff2fb76116ad8362e344a8d1ac31f1b26834aaaa1987a7b

Version	Arch	Tarball	SHA256
3.0	ARM64	ccr-syncer-3.0.6-rc07-arm64.tar.xz	dd5f154c68007732c3c3a9f808f16a7f287fd742bd35d0272ef596779f0eb8e6
3.0	X64	ccr-syncer-3.0.6-rc07-x64.tar.xz	31e514b4d55fb4f11204cd023369ef5988ffda3cb3728e974899623ea81dc1ad

Quick Start | CCR

> **Unavailable**
>
> Some package downloads in this document are currently unavailable. The related download commands may not work until the files are restored.

1. Open the binlog configuration for the source and target clusters

Configure the following information in the fe.conf and be.conf of both the source and target clusters:

```
enable_feature_binlog=true
```

2. Deploy Syncer

2.1. Download the latest package from the following link:

`https://assets-cdn.selectdb.com/downloads/releases/ccr-release/ccr-syncer-3.0.6-rc05-x64.tar.xz`

2.2. Start and stop Syncer

```
# Start
cd bin && sh start_syncer.sh --daemon
```

```
# Stop
sh stop_syncer.sh
```

Step 3. Open the Binlog for the synchronized database/table in the source cluster

```
-- If synchronizing the entire database, execute the following script to enable binlog for all tables in that database
./enable_db_binlog.sh --host $host --port $port --user $user --password $password --db $db

-- If synchronizing a single table, only enable the binlog for that table by executing:
ALTER TABLE your_table_name ENABLE BINLOG SET ("binlog.enable" = "true");
```

Step 4. Initiate a synchronization job in Syncer

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "ccr_test",
    "src": {
      "host": "localhost",
      "port": "9030",
      "thrift_port": "9020",
```

```
      "user": "root",
      "password": "",
      "database": "your_db_name",
      "table": "your_table_name"
    },
    "dest": {
      "host": "localhost",
      "port": "9030",
      "thrift_port": "9020",
      "user": "root",
      "password": "",
      "database": "your_db_name",
      "table": "your_table_name"
    }
}' http://127.0.0.1:9190/create_ccr
```

Explanation of the parameters for the synchronization job:

```
name: The name of the CCR synchronization job, must be unique
host, port: Correspond to the host and MySQL (JDBC) port of the cluster Master FE
user, password: The identity used by Syncer to start transactions and pull data
database, table:
If synchronizing at the database level, fill in your_db_name, and leave your_table_name empty
If synchronizing at the table level, fill in both your_db_name and your_table_name
The name used to initiate the synchronization job can only be used once
```

Operation Manual

Usage Requirements

Network Requirements

- Syncer needs to be able to communicate with both upstream and downstream FE and BE.
- The downstream BE must have direct access to the IP used by the Doris BE process (as seen in `show frontends/backends`).

Permission Requirements

When Syncer synchronizes, the user needs to provide accounts for both upstream and downstream, which must have the following permissions:

- Select_priv: Read-only permission for databases and tables.
- Load_priv: Write permission for databases and tables, including Load, Insert, Delete, etc.
- Alter_priv: Permission to modify databases and tables, including renaming databases/tables, adding/deleting/changing columns, adding/deleting partitions, etc.
- Create_priv: Permission to create databases, tables, and views.
- Drop_priv: Permission to delete databases, tables, and views.
- Admin permission (considered for removal later), used to check the enable binlog config.

Version Requirements

- Syncer Version >= Downstream Doris Version >= Upstream Doris Version. Therefore, upgrade Syncer first, then upgrade downstream Doris, and finally upgrade upstream Doris.
- The minimum version for Doris 2.0 is 2.0.15, and the minimum version for Doris 2.1 is 2.1.6.
- Starting from Syncer version 2.1.8 and 3.0.4, Syncer no longer supports Doris 2.0.

Configuration and Property Requirements

Property Requirements - `light_schema_change`: Syncer requires both upstream and downstream tables to set the `light_schema_table` property; otherwise, data synchronization errors may occur. Note: The latest version of Doris sets the `light_schema_change` property by default when creating tables. If using Doris version 1.1 or earlier or upgraded from it, the `light_schema_change` property must be set for existing OLAP tables before enabling Syncer synchronization.

Configuration Requirements - `restore_reset_index_id`: If the table to be synchronized has an inverted index, it must be configured as `false` on the target cluster. - `ignore_backup_tmp_partitions`: If the upstream creates temporary partitions, Doris will prohibit backup, causing Syncer synchronization to be interrupted; setting `ignore_backup_tmp_partitions=true` in FE can avoid this issue.

Enable binlog for all tables in the database

```
bash bin/enable_db_binlog.sh -h host -p port -u user -P password -d db
```

Start Syncer

Assuming the environment variable ${SYNCER_HOME} is set to the working directory of Syncer. You can start Syncer using `bin/start_syncer.sh`.

Option	Description	Command Example	Default Value
`--daemon`	Run Syncer in the background	`bin/start_syncer.sh --daemon`	`false`
`--db_type`	Syncer can use two types of databases to store metadata: `sqlite3` (local storage) and `mysql` (local or remote storage). When using `mysql` to store metadata, Syncer will create a database named `ccr` using `CREATE IF NOT EXISTS`, and the metadata table will be stored there.	`bin/start_syncer.sh --db_type mysql`	`sqlite3`
`--db_dir`	**Effective only when using** `sqlite3`; specifies the filename and path of the SQLite3 generated database file.	`bin/start_syncer.sh --db_dir /path/to/ccr.db`	`SYNCER_HOME/db/ccr.db`

`--db_host`
`--db_port`
`--db_user`
`--db_password`
	Effective only when using `mysql`; used to set the host, port, user, and password for MySQL.	`bin/start_syncer.sh --db_host 127.0.0.1 --db_port 3306 --db_user root --db_password "qwe123456"`	`db_host` and `db_port` default to example values; `db_user` and `db_password` default to empty.

`--log_dir`	Specify the log output path	`bin/start_syncer.sh --log_dir /path/to/ccr_syncer.log`	`SYNCER_HOME/log/ccr_syncer.log`
`--log_level`	Specify the log output level; the log format is as follows: `time level msg hooks`. The default value is `info` when running in the background; when running in the foreground, the default value is `trace`, and logs are saved to `log_dir` using `tee`.	`bin/start_syncer.sh --log_level info`	`info` (background) `trace` (foreground)
`--host` `--port`	Specify the `host` and `port` for Syncer. The `host` is used to distinguish instances of Syncer in the cluster and can be understood as the name of Syncer; the naming format for Syncer in the cluster is `host:port`.	`bin/start_syncer.sh --host 127.0.0.1 --port 9190`	`host` defaults to `127.0.0.1` `port` defaults to `9190`

Option	Description	Command Example	Default Value
`--pid_dir`	Specify the path to save the PID file. The PID file is the credential for the `stop_syncer.sh` script to stop Syncer, saving the corresponding Syncer's process number. For ease of cluster management, you can customize the path.	`bin/start_syncer.sh --pid_dir /path/to/pids`	`SYNCER_HOME/bin`

Stop Syncer

You can stop Syncer using `bin/stop_syncer.sh` in three ways:

Method/Option	Description	Command Example	Default Value
Method 1 Stop a single Syncer	Specify the `host` and `port` of the Syncer to stop; note that it must match the `host` used when starting.	`bash bin/stop_syncer.sh --host 127.0.0.1 --port 9190`	None
Method 2 Batch stop Syncers	Specify the PID file names to stop, separated by spaces and enclosed in " quotes.	`bash bin/stop_syncer.sh --files "127.0.0.1_9190.pid 127.0.0.1_9191.pid"`	None
Method 3 Stop all Syncers	By default, stops all Syncers corresponding to PID files in the `pid_dir` path.	`bash bin/stop_syncer.sh --pid_dir /path/to/pids`	None

Options for Method 3:

Option	Description	Command Example	Default Value
`--pid_dir`	Specify the directory where the PID files are located; all three stopping methods depend on this option to execute.	`bash bin/stop_syncer.sh --pid_dir /path/to/pids`	`SYNCER_HOME/bin`
`--host` `--port`	Stop the Syncer corresponding to `host:port` in the `pid_dir` path. If only `host` is specified, it degrades to **Method 3**; if both `host` and `port` are not empty, it will be effective as **Method 1**.	`bash bin/stop_syncer.sh --host 127.0.0.1 --port 9190`	`host`: 127.0.0.1 `port`: empty
`--files`	Stop the Syncers corresponding to the specified PID file names in the `pid_dir` path, separated by spaces and enclosed in " quotes.	`bash bin/stop_syncer.sh --files "127.0.0.1_9190.pid 127.0.0.1_9191.pid"`	None

Syncer Operation List

General Template for Requests

```
curl -X POST -H "Content-Type: application/json" -d {json_body} http://ccr_syncer_host:ccr_syncer_port/operator
```

json_body: Send the required information for the operation in JSON format.

operator: Corresponds to different operations of Syncer.

Thus, the interface returns are all in JSON; if successful, the `success` field will be true, and if there is an error, it will be false, with the error message in the `ErrMsgs` field.

```
{"success":true}

or

{"success":false,"error_msg":"job ccr_test not exist"}
```

Create Job

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "ccr_test",
    "src": {
    "host": "localhost",
    "port": "9030",
    "thrift_port": "9020",
    "user": "root",
    "password": "",
    "database": "demo",
    "table": "example_tbl"
    },
    "dest": {
    "host": "localhost",
    "port": "9030",
    "thrift_port": "9020",
    "user": "root",
    "password": "",
    "database": "ccrt",
    "table": "copy"
    }
}' http://127.0.0.1:9190/create_ccr
```

- name: The name of the CCR synchronization job, must be unique.
- host, port: Correspond to the host of the cluster master and the port of MySQL (jdbc).
- thrift_port: Corresponds to the rpc_port of FE.
- user, password: The identity under which Syncer opens transactions, pulls data, etc.
- database, table:
    - If it is a database-level synchronization, fill in dbName, leave tableName empty.
    - If it is table-level synchronization, fill in both dbName and tableName.

View Synchronization Progress

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "job_name"
}' http://ccr_syncer_host:ccr_syncer_port/get_lag
```

job_name is the name created during create_ccr.

Pause Job

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "job_name"
}' http://ccr_syncer_host:ccr_syncer_port/pause
```

Resume Job

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "job_name"
}' http://ccr_syncer_host:ccr_syncer_port/resume
```

Delete Job

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "job_name"
}' http://ccr_syncer_host:ccr_syncer_port/delete
```

Get Version

```
curl http://ccr_syncer_host:ccr_syncer_port/version

# > return
{"version": "2.0.1"}
```

View Job Status

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "job_name"
}' http://ccr_syncer_host:ccr_syncer_port/job_status

{
"success": true,
"status": {
    "name": "ccr_db_table_alias",
    "state": "running",
    "progress_state": "TableIncrementalSync"
}
}
```

End Synchronization

```
curl -X POST -H "Content-Type: application/json" -d '{
    "name": "job_name"
}' http://ccr_syncer_host:ccr_syncer_port/desync
```

Get Job List

```
curl http://ccr_syncer_host:ccr_syncer_port/list_jobs

{"success":true,"jobs":["ccr_db_table_alias"]}
```

Syncer High Availability

Syncer high availability relies on MySQL. If MySQL is used as backend storage, Syncer can discover other Syncers; if one crashes, others will take over its jobs.

Upgrade

1. Upgrade Syncer

Assuming the following environment variables are set: - ${SYNCER_HOME}: Syncer’s working directory. - ${SYNCER_PACKAGE_DIR}: Directory containing the new Syncer.

Upgrade every Syncer by following these steps.

1.1. Save start commands

Save the output of the following command to a file.

```
ps -elf | grep ccr_syncer
```

1.2. Stop the current Syncer

```
sh bin/stop_syncer.sh --pid_dir ${SYNCER_HOME}/bin
```

1.3. Backup the existing MetaService binaries

```
mv ${SYNCER_HOME}/bin bin_backup_$(date +%Y%m%d_%H%M%S)
```

1.4. Deploy the new package

```
cp ${SYNCER_PACKAGE_DIR}/bin ${SYNCER_HOME}/bin
```

1.5. Start the new Syncer

Start the new Syncer using the command saved in 1.1.

2. Upgrade downstream Doris (If Necessary)

Upgrade the upstream system by following the instructions in the Upgrade Doris guide.

3. Upgrade upstream Doris (If Necessary)

Upgrade the upstream system by following the instructions in the Upgrade Doris guide.

Usage Notes

> **The `is_being_synced` attribute should be fully controlled by Syncer to turn on or off under normal circumstances; users should not modify this attribute themselves.**

Important Notes

- During CCR synchronization, backup/restore jobs and binlogs are all in FE memory, so it is recommended to allocate at least 4GB or more heap memory for each CCR job in FE (both source and target clusters), and also modify the following configurations to reduce the memory consumption of unrelated jobs:
  - Modify the FE configuration `max_backup_restore_job_num_per_db`: Records the number of backup/restore jobs for each DB in memory. The default value is 10; setting it to 2 is sufficient.
  - Modify the source cluster db/table property to set the binlog retention limit:
    - `binlog.max_bytes`: Maximum memory occupied by binlog; it is recommended to retain at least 4GB (default is unlimited).
    - `binlog.ttl_seconds`: Binlog retention time; before version 2.0.5, the default was unlimited; after that, the default value is one day (86400). For example, to modify the binlog ttl seconds to retain for one hour: `ALTER TABLE table SET ("binlog.ttl_seconds"="3600")`.
- The correctness of CCR also relies on the transaction state of the target cluster, so it is necessary to ensure that transactions are not reclaimed too quickly during synchronization; the following configurations need to be increased:
  - `label_num_threshold`: Used to control the number of TXN Labels.
  - `stream_load_default_timeout_second`: Used to control the TXN timeout duration.
  - `label_keep_max_second`: Used to control the retention time after TXN ends.
  - `streaming_label_keep_max_second`: Same as above.
- If it is a database synchronization and the source cluster has a large number of tablets, the resulting CCR job may be very large, requiring modifications to several FE configurations:
  - `max_backup_tablets_per_job`: The upper limit of tablets involved in a single backup job; it needs to be adjusted based on the number of tablets (default value is 300,000; too many tablets may risk FE OOM, prioritize reducing the number of tablets).
  - `thrift_max_message_size`: The maximum single RPC packet size allowed by the FE thrift server; the default is 100MB. If the number of tablets is too large, causing the snapshot info size to exceed 100MB, this limit needs to be adjusted, with a maximum of 2GB.
    - The snapshot info size can be found in the ccr syncer logs, with the keywords: `snapshot response meta size: %d, job info size: %d`; the snapshot info size is approximately meta size + job info size.

    - `fe_thrift_max_pkg_bytes`: Same as above, an additional parameter that needs to be adjusted in version 2.0, with a default value of 20MB.
    - `restore_download_job_num_per_be`: The upper limit of download jobs sent to each BE; the default is 3, which is too small for restore jobs and needs to be adjusted to 0 (i.e., disable this limit); this configuration is no longer needed starting from versions 2.1.8 and 3.0.4.
    - `backup_upload_job_num_per_be`: The upper limit of upload jobs sent to each BE; the default is 3, which is too small for backup jobs and needs to be adjusted to 0 (i.e., disable this limit); this configuration is no longer needed starting from versions 2.1.8 and 3.0.4.
    - In addition to the above FE configurations, if the CCR job's db type is MySQL, some MySQL configurations also need to be adjusted:
        - The MySQL server will limit the size of the data packets returned/inserted in a single select/insert. Increase the following configurations to relax this limit, for example, adjust to the upper limit of 1GB:

```
[mysqld]
max_allowed_packet = 1024MB
```

        - The MySQL client will also have this limit; in ccr syncer versions 2.1.6/2.0.15 and earlier, the upper limit is 128MB; later versions can adjust this through the parameter `--mysql_max_allowed_packet` (in bytes), with a default value of 1024MB. > Note: In versions 2.1.8 and 3.0.4 and later, ccr syncer no longer stores snapshot info in the database, so the default data packet size is sufficient.
- Similarly, the BE side also needs to modify several configurations:
    - `thrift_max_message_size`: The maximum single RPC packet size allowed by the BE thrift server; the default is 100MB. If the number of tablets is too large, causing the agent job size to exceed 100MB, this limit needs to be adjusted, with a maximum of 2GB.
    - `be_thrift_max_pkg_bytes`: Same as above, only needs to be adjusted in version 2.0, with a default value of 20MB.
- Even if the above configurations are modified, if the number of tablets continues to rise, the resulting snapshot size may exceed 2GB, which is the threshold for Doris FE edit log and RPC message size, causing synchronization to fail. Starting from versions 2.1.8 and 3.0.4, Doris can further increase the number of tablets supported for backup and recovery by compressing snapshots. This can be enabled through the following parameters:
    - `restore_job_compressed_serialization`: Enable compression for restore jobs (affects metadata compatibility, default is off).
    - `backup_job_compressed_serialization`: Enable compression for backup jobs (affects metadata compatibility, default is off).
    - `enable_restore_snapshot_rpc_compression`: Enable compression for snapshot info, mainly affecting RPC (default is on). > Note: Since identifying whether backup/restore jobs are compressed requires additional code, and the code before versions 2.1.8 and 3.0.4 does not contain related code, once a backup/restore job is generated, it cannot revert to an earlier Doris version. There are two exceptions: backup/restore jobs that have already been canceled or finished will not be compressed, so waiting for the backup/restore job to complete or actively canceling the job before reverting can ensure safe rollback.
- CCR internally uses db/table names as labels for some internal jobs, so if the CCR job encounters labels exceeding the limit, the FE parameter `label_regex_length` can be adjusted to relax this limit (default value is 128).

- Since backup does not currently support backing up tables with cooldown tablets, encountering this will cause synchronization to terminate, so it is necessary to check whether any tables have the `storage_policy` property set before creating the CCR job. #### Performance-Related Parameters
- If the user's data volume is very large, the time required to complete backup and recovery may exceed one day (default value), so the following parameters need to be adjusted as needed:
    - `backup_job_default_timeout_ms`: Timeout duration for backup/restore jobs; both source and target clusters' FE need to configure this.
    - The upstream modifies the binlog retention time: `ALTER DATABASE $db SET PROPERTIES ("binlog.ttl_seconds" = "xxxx")`.
- Downstream BE download speed is slow:
    - `max_download_speed_kbps`: The download speed limit for a single download thread in a single downstream BE, default is 50MB/s.
    - `download_worker_count`: The number of threads executing download jobs in the downstream; it should be adjusted based on the customer's machine type, maximizing without affecting normal read/write operations; if this parameter is adjusted, there is no need to adjust `max_download_speed_kbps`.
        - For example, if the customer's machine network card provides a maximum bandwidth of 1GB, and the maximum allowed download thread utilizes 200MB of bandwidth, then without changing `max_download_speed_kbps`, `download_worker_count` should be configured to 4.
- Limit the download speed of binlogs from the downstream BE: BE-side configuration parameter: `shell download_binlog_rate_limit_kbs=1024 # Limit the speed of a single BE node pulling Binlog (including Local Snapshot) from the source cluster to 1 MB/s`. Detailed parameters and explanations:
    1. The `download_binlog_rate_limit_kbs` parameter is configured on the source cluster BE node, and by setting this parameter, the data pulling speed can be effectively limited.
    2. The `download_binlog_rate_limit_kbs` parameter is mainly used to set the speed of a single BE node; to calculate the overall speed of the cluster, the parameter value needs to be multiplied by the number of nodes in the cluster.

Feature Details

The Cross-Cluster Replication (CCR) feature in Doris is primarily used for efficiently synchronizing data between multiple clusters, thereby enhancing business continuity and disaster recovery capabilities. CCR supports various operations in Doris, ensuring data consistency across different clusters. Below are the details of the main Doris operations supported by CCR.

> **1. The - in Doris Version indicates Doris version 2.0 and above, all versions of CCR. It is recommended to use Doris version 2.0.15 or 2.1.6 or later.**
>
> 2. Version requirements for CCR Syncer and Doris: Syncer Version >= Downstream Doris Version >= Upstream Doris Version. Therefore, upgrade Syncer first, then the downstream Doris, and finally the upstream Doris.
> 3. CCR currently does not support the separation of storage and computation.

Database

Database Properties

Database-level jobs will synchronize the properties of the database during Full Sync.

Property	Supported	Doris Version	Sync Method	Description
replication_allocation	Supported	-	Full Sync	
data quota	Not Supported			
replica quota	Not Supported			

Modify Database Properties

CCR jobs do not synchronize operations that modify database properties.

Property	Supported	Can Upstream Operate	Can Downstream Operate	Description
replication_allocation	Not Supported	No	No	Operations on both sides will cause CCR jobs to be interrupted
data quota	Not Supported	Yes	Yes	
replica quota	Not Supported	Yes	Yes	

Rename Database

Renaming is not supported for upstream and downstream; if done, it may cause views to stop working.

Table

Table Properties

Property	Supported	Doris Version	Sync Method	Description
Table Model (duplicate, unique, aggregate)	Supported	-	SQL	
Partition and Bucketing	Supported	-	SQL	
replication_num	Supported	-	SQL	
replication_allocation (resource group)	Supported	-	SQL	Upstream must be consistent with downstream, BE tags must match; otherwise, CCR jobs will fail
colocate_with	Not Supported			
storage_policy	Not Supported			
dynamic_partition	Supported	-	SQL	
storage_medium	Supported	-	SQL	
auto_bucket	Supported	-	SQL	
group_commit series	Supported	-	SQL	
enable_unique_key_merge_on_write	Supported	-	SQL	
enable_single_replica_compaction	Supported	-	SQL	
disable_auto_compaction	Supported	-	SQL	
compaction_policy	Supported	-	SQL	
time_series_compaction series	Supported	-	SQL	
binlog series	Supported	-	SQL	

Property	Supported	Doris Version	Sync Method	Description
skip_write_index_on_load	Supported	-	SQL	
row_store series	Supported	-	SQL	
seq column	Supported	-	SQL	
enable_light_schema_change	Supported	-	SQL	
compression_type	Supported	-	SQL	
index	Supported	-	SQL	
bloom_filter_columns	Supported	-	SQL	
bloom_filter_fpp	Supported			
storage_cooldown_time	Not Supported			
generated column	Supported	-	SQL	
auto-increment id	Not Supported			Has issues

Basic Table Operations

Operation	Supported	Doris Version	Sync Method	Can Downstream Operate Independently	Description
create table	Supported	-	SQL/Partial Sync	Cannot operate on tables synchronized by CCR jobs.	Refer to the properties for creating tables; in most cases, use SQL for synchronization; for some operations, such as when users set certain session variables or when the create table statement includes inverted indexes, use partial sync
drop table	Supported	-	SQL/Full Sync	Same as above	Before 2.0.15/2.1.6: Full Sync, after: SQL
rename table	Table-level jobs do not support database-level jobs	2.1.8/3.0.4	SQL	Same as above	Table-level job renames will cause CCR jobs to stop
replace table	Supported	2.1.8/3.0.4	SQL/Full Sync	Same as above	Use SQL for database-level synchronization; table-level triggers full synchronization
truncate table	Supported	-	SQL	Same as above	
restore table	Not Supported			Same as above	

Modify Table Properties

The synchronization method is SQL.

Property	Supported	Doris Version	Can Upstream Operate	Can Downstream Operate	Description
colocate	Not Supported		Yes	No	Triggering full sync on downstream operations will cause data loss
distribution type	Not Supported		No	Same	
dynamic partition	Not Supported		Yes	Same	
replication_num	Not Supported		No	No	
replication_allocation	Not Supported		No		
storage policy	Not Supported		No	No	
enable_light_schema_change	Not Supported				CCR can only synchronize lightweight schema changes.
row_store	Supported	2.1.8/3.0.4			Through Partial Sync
bloom_filter_columns	Supported	2.1.8/3.0.4			Through Partial Sync
bloom_filter_fpp	Supported	2.1.8/3.0.4			Through Partial Sync
bucket num	Not Supported		Yes	No	Triggering full sync on downstream operations will cause data loss
isBeingSynced	Not Supported		No	No	
compaction series properties	Not Supported		Yes	No	Triggering full sync on downstream operations will cause data loss
skip_write_index_on_load	Not Supported		Yes	Same	
seq column	Supported	-	Yes	No	Triggering full sync on downstream operations will cause data loss
delete sign column	Supported	-	Yes	Same	

Property	Supported	Doris Version	Can Upstream Operate	Can Downstream Operate	Description
comment	Supported	2.1.8/3.0.4	Yes	No	Triggering full sync on downstream operations will cause data loss

Column Operations

Column operations on Base Index in the table.

Operation	Supported	Doris Version	Sync Method	Can Downstream Operate	Remarks
add key column	Supported	-	Partial Sync	No	
add value column	Supported	-	SQL	No	
drop key column	Supported	-	Partial Sync	Same	
drop value column	Supported	-	SQL	Same	
modify column	Supported	-	Partial Sync	Same	
order by	Supported	-	Partial Sync	Same	
rename	Supported	2.1.8/3.0.4	SQL	Same	
comment	Supported	2.1.8/3.0.4	SQL	Same	

> **Adding/dropping value columns requires setting the property `"light_schema_change" = "true"` when creating the table.**

Column operations on Rollup Index in the table.

Operation	Supported	Doris Version	Sync Method	Remarks
add key column	Supported	2.1.8/3.0.4	Partial Sync	
add value column	Supported	2.1.8/3.0.4	SQL	Requires enabling lightning schema change
drop column	Supported	2.1.8/3.0.4	Partial Sync	
modify column	Unknown	2.1.8/3.0.4	Partial Sync	Doris does not support directly modifying rollup column types
order by	Supported	2.1.8/3.0.4	Partial sync	

Rollup

Operation	Supported	Doris Version	Sync Method	Remarks
add rollup	Supported	2.1.8/3.0.4	Partial Sync	
drop rollup	Supported	2.1.8/3.0.4	SQL	
rename rollup	Supported	2.1.8/3.0.4	SQL	

Index

Inverted Index

Operation	Supported	Doris Version	Sync Method	Remarks
create index	Supported	2.1.8/3.0.4	Partial Sync	
drop index	Supported	2.1.8/3.0.4	SQL	
build index	Supported	2.1.8/3.0.4	SQL	

Bloom Filter

Operation	Supported	Doris Version	Sync Method	Remarks
add bloom filter	Supported	2.1.8/3.0.4	Partial Sync	
alter bloom filter	Supported	2.1.8/3.0.4	Partial Sync	This refers to modifying bloom_filter_columns
drop bloom filter	Supported	2.1.8/3.0.4	Partial Sync	

Data

Import

Import Method	Supported	Doris Version	Sync Method	Can Downstream Operate	Description
stream load	Supported (except for temporary partitions)	-	TXN	No, if downstream imports, subsequent triggers of full or Partial Sync will cause data loss	Upstream transactions are visible, i.e., data is visible when generating binlog, downstream starts syncing.
broker load	Supported (except for temporary partitions)	-	TXN	Same	Same
routine load	Supported (except for temporary partitions)	-	TXN	Same	Same

Import Method	Supported	Doris Version	Sync Method	Can Downstream Operate	Description
mysql load	Supported (except for temporary partitions)	-	TXN	Same	Same
group commit	Supported (except for temporary partitions)	2.1	TXN	Same	Same

Data Operations

Operation	Supported	Doris Version	Sync Method	Can Downstream Operate	Description
delete	Supported	-	TXN	No, if downstream operates, subsequent triggers of full or Partial Sync will cause data loss	Upstream transactions are visible, i.e., data is visible when generating binlog, downstream starts syncing.
update	Supported	-	TXN	Same	Same
insert	Supported	-	TXN	Same	Same
insert into overwrite	Supported (except for temporary partitions)	2.1.6	Partial Sync	Same	Same
insert into overwrite	Supported (except for temporary partitions)	2.0	full sync	Same	Same
Explicit transaction (3.0) begin commit	Not Supported				

Partition Operations

Operation	Supported	Doris Version	Sync Method	Can Downstream Operate Independently	Description
add partition	Supported	-	SQL	No, subsequent triggers of Full Sync or Partial Sync will cause downstream operations to be lost	Cooldown time property and its behavior are unknown
add temp partition	Not Supported			Same	Backup does not support temp partition; starting from Doris 2.1.8/3.0.4, you can modify upstream FE configuration: `ignore_backup_tmp_partitions` to bypass this issue

Operation	Supported	Doris Version	Sync Method	Can Downstream Operate Independently	Description
drop partition	Supported	-	SQL/Full Sync	Same	Before 2.0.15/2.1.6: Full Sync, after: SQL
replace partition	Supported	2.1.7/3.0.3	Partial Sync	Same	Partial Sync **only supports strict range and non-temp partition replace method**, otherwise it will trigger Full Sync.
modify partition	Not Supported			Same	Refers to modifying the property of the partition
rename partition	Supported	2.1.8/3.0.4	SQL	Same	

Views

Operation	Supported	Doris Version	Sync Method	Remarks
create view	Supported	-	SQL	Can work when upstream and downstream have the same name; if downstream already exists, it will be deleted before creation
alter view	Supported	2.1.8/3.0.4	SQL	
drop view	Supported	2.1.8/3.0.4	SQL	

Due to limitations in Doris implementation, column names/view names in views cannot be the same as database names.

Materialized Views

Synchronizing materialized views

Operation	Supported	Doris Version	Sync Method	Remarks
create materialized view	Supported	2.1.8/3.0.4	Partial Sync	Can work when upstream and downstream have the same name; if different names, downstream needs to manually rebuild the view.
drop materialized view	Supported	2.1.8/3.0.4	SQL	

Asynchronous materialized views

Operation	Supported
create async materialized view	Not Supported
alter async materialized view	Not Supported
drop async materialized view	Not Supported
refresh	Not Supported

Operation	Supported
pause	Not Supported
resume	Not Supported

Statistics

Not synchronized between upstream and downstream, operate independently.

Others

Operation	Supported
external table	Not Supported
recycle bin	Not Supported
catalog	Not Supported
workload group	Not Supported
job	Not Supported
function	Not Supported
policy	Not Supported
user	Not Supported
cancel alter job	Supported

Configuration Instructions

This document provides the configurations that need to be adjusted or paid attention to when using CCR.

FE Configuration

Configured in `fe.conf`, for example, `restore_reset_index_id = false`.

Name	Description	Default Value	Version
`restore_reset_index_id`	If inverted index or bitmap index is used in the synchronized table, it should be set to `false`.	false	Starting from 2.1.8 and 3.0.4.
`ignore_backup_tmp_partitions`	To avoid synchronization interruption caused by upstream creating temporary partition, it should be set to `true`.	false	Starting from 2.1.8 and 3.0.4.
`max_backup_restore_job_num_per_db`	Limit on the number of backup/restore jobs per DB in memory, it is recommended to set it to 2.	10	All versions.
`label_num_threshold`	Controls the number of TXN Labels to prevent transaction recovery from being too fast; too large may occupy more memory, too small may cause data duplication in exceptional situations, the default value is sufficient in most cases.	2000	Starting from 2.1.
`restore_job_compressed_serialization`	It is recommended to configure as true when the number of tablets exceeds 100,000. Before downgrading, turn off the configuration and ensure FE completes a checkpoint. When upgrading from 2.1 to 3.0, at least upgrade to 3.0.3.	false	Starting from 2.1.8 and 3.0.3.
`backup_job_compressed_serialization`	It is recommended to configure as true when the number of tablets exceeds 100,000. Before downgrading, turn off the configuration and ensure FE completes a checkpoint. When upgrading from 2.1 to 3.0, at least upgrade to 3.0.3.	false	Starting from 2.1.8 and 3.0.3.
`backup_job_default_timeout_ms`	Timeout for backup/restore jobs, both source and target cluster FE need to be configured.	None	Set according to requirements

Name	Description	Default Value	Version
enable_restore_snapshot_rpc_compression	Enable snapshot info compression to reduce RPC message size, it is recommended to set it to true.	true	Starting from 2.1.8 and 3.0.3.

BE

Configured in be.conf, for example, thrift_max_message_size = 2000000000.

Name	Description	Default Value	Version
thrift_max_message_size	BE thrift server single RPC packet limit, it is recommended to set to 2000000000 when the number of tablets involved in CCR jobs is large.	100MB	All versions
be_thrift_max_pkg_bytes	BE Thrift RPC message package size limit.	20MB	Specific to 2.0.
max_download_speed_kbps	Download speed limit for each download worker in downstream BE, default is 50MB/s per thread.	50MB/s	All versions
download_worker_count	Number of threads for download jobs, set according to network card, disk, and load.	1	All versions

Table Attributes

Set in Create Table or Alter Table.

Name	Description	Default Value	Version
binlog.max_bytes	Maximum memory usage for binlog, it is recommended to keep at least 4GB.	Unlimited	All versions
binlog.ttl_seconds	Retention time for binlog.	Unlimited before 2.0.5, starting from 2.0.5 it is 1 day (86400)	All versions

Performance

The performance data in this document is based on the default configuration. If you are facing high network latency or high throughput write scenarios, you can refer to the Operation Manual for optimization.

Test Dataset

- **Dataset**: TPC-H 1T

Test Cluster Configuration

Configuration Item	Upstream Configuration	Downstream Configuration
FE	2 cores 16 GB	2 cores 16 GB
BE	3 nodes, 16 cores 64 GB, 3*500 GB per node	3 nodes, 16 cores 64 GB, 3*500 GB per node

---

Incremental Synchronization Performance Test

Test Steps

1. Create the database and tables for TPC-H 1T in the upstream cluster.
2. Create a synchronization job for the TPC-H 1T database.
3. Wait for the TPC-H 1T data import to complete and record the completion time.
4. Wait for the downstream data synchronization to complete and record the completion time.

Test Conclusion

Incremental synchronization time difference: 33 seconds

Full Synchronization Performance Test

Test Steps

1. Create the database and table information for TPC-H 1T in the upstream cluster and complete the data import, recording the completion time.
2. Create a synchronization job for the TPC-H 1T database.
3. Wait for the downstream data synchronization to complete and record the completion time.

Test Conclusion

Full synchronization time difference: 6 minutes 1 second

Flink Synchronization Performance Test

Test Steps

1. Use Flink to import 100,000,000 records in the upstream.
2. Create a synchronization job for the library and table.
3. Observe the difference between the downstream synchronization completion time and the upstream import completion time at each stage (e.g., 1,000,000 records, 2,000,000 records, etc.).
4. Record the last import completion time in the upstream.
5. Record the downstream synchronization completion time.

Test Conclusion

The **lag time** at each stage is maintained within `5 seconds`.

Recover from Recycle Bin

Recover from Recycle Bin

To avoid disasters caused by accidental operations, Doris supports the recovery of accidentally deleted databases, tables, and partitions. After deleting a table or database, Doris does not immediately physically delete the data. When the user executes the `DROP DATABASE/TABLE/PARTITION` command without using `FORCE`, Doris moves the deleted database, table, or partition to the recycle bin. The `RECOVER` command can be used to restore all data of the deleted database, table, or partition from the recycle bin, making it visible again.

Note: If the deletion was performed using `DROP FORCE`, the data will be immediately deleted and cannot be recovered.

Query Recycle Bin

You can query the Recycle Bin with the following command:

```
SHOW CATALOG RECYCLE BIN [WHERE NAME [= "name" | LIKE "name_matcher"]];
```

For more detailed syntax and best practices, please refer to the SHOW-CATALOG-RECYCLE-BIN command manual, You can also type `help SHOW CATALOG RECYCLE BIN` on the MySQL client command line for more help.

Start Data Recovery

To recover deleted data, you can use the following commands:

1. *Recover the database* named `example_db`:

```
RECOVER DATABASE example_db;
```

2. *Recover the table* named `example_tbl`:

```
RECOVER TABLE example_db.example_tbl;
```

3. *Recover partition* named p1 in table example_tbl:

```
RECOVER PARTITION p1 FROM example_tbl;
```

For more detailed syntax and best practices for RECOVER, please refer to the RECOVER command manual, You can also type `HELP RECOVER` on the MySql client command line for more help.

FE Log Management

This document mainly introduces the log management of the Frontend (FE) process.

This document is applicable to Doris versions 2.1.4 and later.

Log Categories

When starting the FE process using `sh bin/start_fe.sh --daemon`, the following types of log files will be generated in the FE log directory:

- fe.log

  FE process running log. The main log file for FE. All levels of FE process logs (DEBUG, INFO, WARN, ERROR, etc.) will be printed to this log file.

- fe.warn.log

  FE process running log. Only prints logs at WARN level and above. The content in fe.warn.log is a subset of the fe.log log content. It is mainly used to quickly view warning or error level logs.

- fe.audit.log

  Audit log. Used to record all database operation records executed through this FE node. This includes SQL, DDL, DML statements, etc.

- fe.out

  Used to receive standard output stream and error data stream logs. For example, output from `echo` commands in start scripts, or other log information not captured by the log4j framework. Usually serves as a supplement to the running log. In rare cases, you may need to check the content of fe.out for more information.

- fe.gc.log

  GC log of the FE JVM. The behavior of this log is controlled by the JVM startup option `JAVA_OPTS` in fe.conf.

Log Configuration

Includes configuring the log storage path, retention time, retention count, size, etc.

The following configuration items are configured in the `fe.conf` file.

Configuration Item	Default Value	Options	Description
LOG_DIR	ENV(DORIS_HOME)/log		Storage path for all logs. By default, it is the log/ directory under the FE deployment path. Note that this is an environment variable, and the configuration name should be in uppercase.
sys_log_level	INFO	INFO, WARN, ERROR, FATAL	Log level of fe.log. Default is INFO. Not recommended to change, as INFO level contains many critical log information.
sys_log_roll_num	10		Controls the maximum number of files for fe.log and fe.warn.log in one day. Default is 10. When the number of log files exceeds this threshold due to log rolling or splitting, older log files will be deleted.
sys_log_verbose_modules			Can set specific Java package files to enable DEBUG level logging. See the “Enable DEBUG Log” section for details.
sys_log_enable_compress	false	true, false	Whether to enable compression for historical fe.log and fe.warn.log logs. Default is off. When enabled, historical audit logs will be archived using gzip compression.
log_rollover_strategy	age	age, size	Log retention strategy, default is age, which retains historical logs based on time. size retains historical logs based on log size.
sys_log_delete_age	7d	Supports formats like 7d, 10h, 60m, 120s	Only effective when log_rollover_strategy is age. Controls the number of days to retain fe.log and fe.warn.log files. Default is 7 days. Logs older than 7 days will be automatically deleted.
audit_log_delete_age	30d	Supports formats like 7d, 10h, 60m, 120s	Only effective when log_rollover_strategy is age. Controls the number of days to retain fe.audit.log files. Default is 30 days. Logs older than 30 days will be automatically deleted.

Configuration Item	Default Value	Options	Description
info_sys_accumulated_file_size	4		Only effective when log_rollover_strategy is size. Controls the cumulative size of fe.log files. Default is 4GB. When the cumulative log size exceeds this threshold, historical log files will be deleted.
warn_sys_accumulated_file_size	2		Only effective when log_rollover_strategy is size. Controls the cumulative size of fe.warn.log files. Default is 2GB. When the cumulative log size exceeds this threshold, historical log files will be deleted.
audit_sys_accumulated_file_size	4		Only effective when log_rollover_strategy is size. Controls the cumulative size of fe.audit.log files. Default is 4GB. When the cumulative log size exceeds this threshold, historical log files will be deleted.
log_roll_size_mb	1024		Controls the maximum size of individual files for fe.log, fe.warn.log, fe.audit.log. Default is 1024MB. When a single log file exceeds this threshold, a new file will be created automatically.
sys_log_roll_interval	DAY	DAY, HOUR	Controls the rolling interval of fe.log and fe.warn.log. Default is 1 day, generating a new log file every day.
audit_log_roll_num	90		Controls the maximum number of files for fe.audit.log. Default is 90. When the number of log files exceeds this threshold due to log rolling or splitting, older log files will be deleted.
audit_log_roll_interval	DAY	DAY, HOUR	Controls the rolling interval of fe.audit.log. Default is 1 day, generating a new log file every day.
audit_log_dir	ENV(DORIS_HOME)/log		Can specify a separate storage path for fe.audit.log. Default is the log/ directory under the FE deployment path.

Configuration Item	Default Value	Options	Description
audit_log_modules	{"slow_query", "query", "load", "stream_load"}		Module types in fe.audit.log. Default includes slow query, query, load, stream load. “Query” includes all DDL, DML, SQL operations. “Slow query” refers to operations that exceed the qe_slow_log_ms threshold. “Load” refers to Broker Load. “Stream load” refers to stream load operations.
qe_slow_log_ms	5000		When the execution time of DDL, DML, SQL statements exceeds this threshold, it will be separately recorded in the slow_query module of fe.audit.log. Default is 5000 ms.
audit_log_enable_compress	false	true, false	Whether to enable compression for historical fe.audit.log logs. Default is off. When enabled, historical audit logs will be archived using gzip compression.
sys_log_mode	NORMAL	NORMAL, BRIEF, ASYNC	FE log output mode, where NORMAL is the default output mode, log output is synchronous and includes location information. ASYNC is the default log output is asynchronous and includes location information. BRIEF mode is log output asynchronously but does not include location information. The performance of the three log output modes increases in sequence.

Starting from version 3.0.2, the default value of sys_log_mode configuration is changed to ASYNC.

> **sys_log_roll_num 控制的是一天的保留日志数量，而不是总数量，需要配合 sys_log_delete_age 共同确定总保留日志数量。**
>
> sys_log_roll_num controls the number of retained logs per day, not the total number. It needs to be combined with sys_log_delete_age to determine the total number of retained logs.

Enable DEBUG Log

The Debug level log of FE can be enabled by modifying the configuration file or through the interface or API during runtime.

- Enable through configuration file

  Add the configuration item `sys_log_verbose_modules` in fe.conf. For example:

```
# Only open DEBUG log for "org.apache.doris.catalog.Catalog"
sys_log_verbose_modules=org.apache.doris.catalog.Catalog

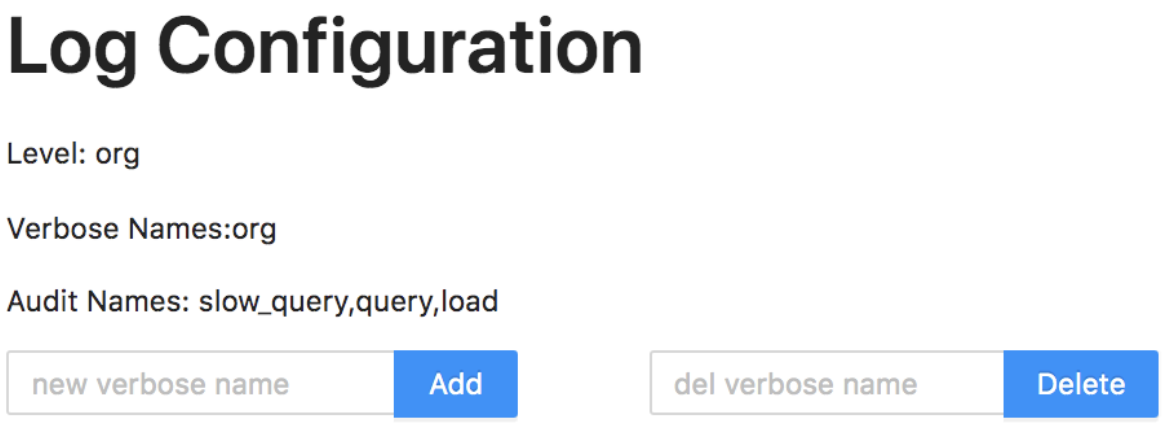
# Open DEBUG log for all classes in "org.apache.doris.catalog"
sys_log_verbose_modules=org.apache.doris.catalog

# Open DEBUG log for all classes in "org"
sys_log_verbose_modules=org
```

  Add the configuration item and restart the FE node to take effect.

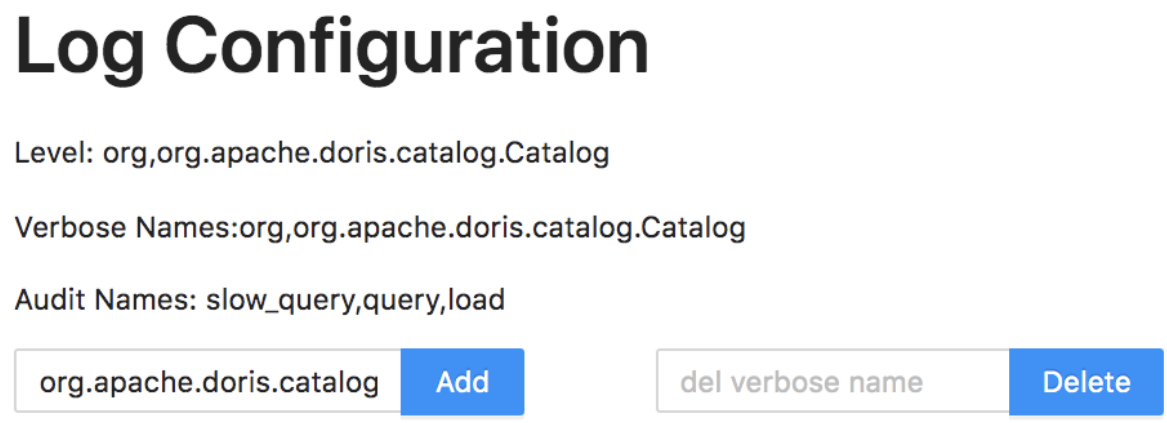
- Enable through FE UI interface

  You can modify the log level at runtime through the UI interface. No need to restart the FE node. Open the FE node's http port in the browser (default is 8030) and log in to the UI interface. Then click on the `Log` tab in the top navigation bar.



  Enable through FE UI interface

  In the Add input box, you can enter the package name or specific class name to open the corresponding Debug log. For example, entering `org.apache.doris.catalog.Catalog` will open the Debug log of the Catalog class:



  Enable through FE UI interface

You can also enter the package name or specific class name in the Delete input box to close the corresponding Debug log.

:::note The modification here will only affect the log level of the corresponding FE node. It will not affect the log level of other FE nodes. :::

- Modify through API

  You can also modify the log level at runtime through the following API. No need to restart the FE node.

  ```
  curl -X POST -uuser:passwd fe_host:http_port/rest/v1/log?add_verbose=org.apache.doris.catalog.Catalog
  ```

  Where the username and password are the root or admin users logged into Doris. The `add_verbose` parameter specifies the package name or class name for enabling Debug log. If successful, it will return:

  ```
  {
      "msg": "success",
      "code": 0,
      "data": {
          "LogConfiguration": {
              "VerboseNames": "org,org.apache.doris.catalog.Catalog",
              "AuditNames": "slow_query,query,load",
              "Level": "INFO"
          }
      },
      "count": 0
  }
  ```

  You can also close Debug log through the following API:

  ```
  curl -X POST -uuser:passwd fe_host:http_port/rest/v1/log?del_verbose=org.apache.doris.catalog.Catalog
  ```

  The `del_verbose` parameter specifies the package name or class name for closing Debug log.

Log Configuration for k8s

In some cases, the FE process is deployed through a container environment (such as k8s). All logs need to be output through standard output stream instead of files.

In this case, you can start the FE process in the foreground and output all logs to the standard output stream by using the command `sh bin/start_fe.sh --console`.

To distinguish different types of logs in the same standard output stream, a different prefix will be added before each log. For example:

```
RuntimeLogger 2024-06-24 00:05:21,522 INFO (main|1) [DorisFE.start():158] Doris FE starting...
RuntimeLogger 2024-06-24 00:05:21,530 INFO (main|1) [FrontendOptions.analyzePriorityCidrs():194] configured prior_cidrs value:
172.20.32.136/24
RuntimeLogger 2024-06-24 00:05:21,535 INFO (main|1) [FrontendOptions.initAddrUseIp():101] local address: /172.20.32.136.
RuntimeLogger 2024-06-24 00:05:21,740 INFO (main|1) [ConsistencyChecker.initWorkTime():106] consistency checker will work from 23:00
to 23:00
RuntimeLogger 2024-06-24 00:05:21,889 ERROR (main|1) [Util.report():128] SLF4J: Class path contains multiple SLF4J bindings.
```

The meanings of different prefixes are as follows:

- `StdoutLogger`: Logs in the standard output stream, corresponding to the content in `fe.out`.
- `StderrLogger`: Logs in the standard error stream, corresponding to the content in `fe.out`.
- `RuntimeLogger`: Logs in `fe.log`.
- `AuditLogger`: Logs in `fe.audit.log`.
- No prefix: Logs in `fe.gc.log`.

Additionally, there is an additional configuration parameter for the container environment:

Configuration Item	Default Value	Options	Description
`enable_file_logger`	true	true, false	Whether to enable file logging. Default is `true`. When starting the FE process with the `--console` command, logs will be output to both the standard output stream and the normal log file. When set to `false`, logs will only be output to the standard output stream and will not generate log files.

BE Log Management

This document mainly introduces the log management of the Backend (BE) process.

This document is applicable to Doris versions 2.1.4 and later.

Log Categories

When starting the BE process using `sh bin/start_be.sh --daemon`, the following types of log files will be generated in the BE log directory:

- be.INFO

  BE process running log. The main log file for BE. All levels of BE process running logs (DEBUG, INFO, WARN, ERROR, etc.) will be printed to this log file.

  Note that this file is a symbolic link pointing to the current latest BE running log file.

- be.WARNING

  BE process running log. However, only WARN level and above running logs will be printed. The content in be.WARNING is a subset of the be.INFO log content. It is mainly used for quickly viewing warning or error level logs.

  Note that this file is a symbolic link pointing to the current latest BE warning log file.

- be.out

  Used to receive standard output streams and error data streams. For example, output from `echo` commands in start scripts, or other log information not captured by the glog framework. Usually used as a supplement to running logs.

  Typically, in the event of a BE crash, you need to check this log to obtain the stack trace of the exception.

- jni.log

  Logs printed by Java programs when the BE process calls Java programs through JNI.

  TODO: In future versions, this part of the logs will be unified into the be.INFO log.

- be.gc.log

  BE JVM's GC log. The behavior of this log is controlled by the JVM startup option `JAVA_OPTS` in be.conf.

Log Configuration

Includes configuring log storage paths, retention time, retention count, size, etc.

The following configuration items are configured in the `be.conf` file.

Configuration Item	Default Value	Options	Description
LOG_DIR	ENV(DORIS_HOME)/log		Storage path for all logs. By default, it is the log/ directory under the BE deployment path. Note that this is an environment variable, and the configuration name needs to be in uppercase.
sys_log_level	INFO	INFO, WARNING, ERROR, FATAL	Log level for be.INFO. Default is INFO. Not recommended to change, as INFO level contains many critical log information.
sys_log_roll_num	10		Controls the maximum number of files for be.INFO and be.WARNING. Default is 10. When the number of log files exceeds this threshold due to log rolling or splitting, older log files will be deleted.
sys_log_verbose_modules			Can set specific code directories to enable DEBUG level logs. See the “Enable DEBUG Logs” section for details.
sys_log_verbose_level			See the “Enable DEBUG Logs” section for details.
sys_log_verbose_flags_v			See the “Enable DEBUG Logs” section for details.
sys_log_roll_mode	SIZE-MB-1024	TIME-DAY, TIME-HOUR, SIZE-MB-nnn	Rolling strategy for be.INFO and be.WARNING logs. Default is SIZE-MB-1024, meaning a new log file is generated after each log reaches 1024MB in size. Can also set to roll by day or hour.
log_buffer_level	Empty	Empty or -1	BE log output mode. By default, BE logs are asynchronously flushed to disk log files. If set to -1, log content will be flushed in real-time. Real-time flushing affects log performance but can retain the latest logs as much as possible. This allows viewing the last log information in the event of a BE crash.
disable_compaction_trace_log	true	true, false	Default is true, meaning tracing logs for compaction operations are disabled. If set to false, tracing logs related to Compaction operations will be printed for troubleshooting.

Configuration Item	Default Value	Options	Description
aws_log_level	0		Controls the log level for the AWS SDK. Default is 0, indicating AWS SDK logs are turned off. By default, AWS SDK logs are actively captured by glog and will be printed normally. In some cases, you may need to enable AWS SDK logs to view more uncaptured logs. Different numbers represent different log levels: Off = 0, Fatal = 1, Error = 2, Warn = 3, Info = 4, Debug = 5, Trace = 6.
s3_file_writer_log_interval_second	60		When performing S3 Upload operations, the progress of operations is printed every 60 seconds by default.
enable_debug_log_timeout_secs	0		When the value is greater than 0, some detailed execution logs of the pipeline execution engine will be printed. Mainly used for troubleshooting. By default, this is turned off.
sys_log_enable_custom_date_time_format	false		Whether to allow custom date format in logs (supported since version 2.1.7)
sys_log_custom_date_time_format	%Y-%m-%d %H:%M:%S		The default custom format for log date, only effective when sys_log_enable_custom_date_time_forma is true (supported since version 2.1.7)
sys_log_custom_date_time_ms_format	,{:03d}		The default time precision in the log date. This is only effective when sys_log_enable_custom_date_time_format is true (supported since version 2.1.7)

Enable DEBUG Log

Static Configuration

Set sys_log_verbose_modules and sys_log_verbose_level in be.conf:

```
sys_log_verbose_modules=plan_fragment_executor,olap_scan_node
sys_log_verbose_level=3
```

sys_log_verbose_modules Specifies the names of the files to be opened, which can be specified by the wildcard *. For example:

```
sys_log_verbose_modules=*
```

will turn on all BE verbose log.

`sys_log_verbose_level` Indicates the level of DEBUG. The higher the number, the more detailed the DEBUG log. The value ranges from 1 to 10.

In most situations, setting `sys_log_verbose_modules` and `sys_log_verbose_level` in `be.conf` is sufficient. Only in rare cases such as when debug logs do not appear as expected, you may also need to configure `sys_log_verbose_flags_v`, which is not limited by the module scope.

`sys_log_verbose_flags_v` is the `FLAGS_v` in glog, which controls the global verbosity level for `VLOG(n)` logging, where messages with `n <= FLAGS_v` will be printed, allowing fine-grained control over log output detail.

Dynamic Modification

Since 2.1, the DEBUG log of BE supports dynamic modification via the following RESTful API:

```
curl -X POST "http://<be_host>:<webport>/api/glog/adjust?module=<module_name>&level=<level_number>"
```

The dynamic adjustment method also supports wildcards, e.g. using `module=*&level=10` will turn on all BE vlogs, but wildcards are not attached to individual module names. e.g. adjusting the vlog level of `moduleA` to `10`, then using `module=*&level=-1` will **NOT** turn off the vlog of `moduleA`'s vlog.

Note: Dynamically adjusted configurations are not persisted and will expire after a BE reboot.

In addition, GLOG will create the corresponding log module if the module does not exist (no real effect) and will not return an error, regardless of the method.

Container Environment Log Configuration

In some cases, the FE process is deployed through container environments (such as k8s). All logs need to be output through standard output streams instead of files.

At this time, you can start the BE process in the foreground and output all logs to the standard output stream by using the command `sh bin/start_be.sh --console`.

To distinguish different types of logs in the same standard output stream, a different prefix will be added before each log. For example:

```
RuntimeLogger W20240624 00:36:46.325274 1460943 olap_server.cpp:710] Have not get FE Master heartbeat yet
RuntimeLogger I20240624 00:36:46.325999 1459644 olap_server.cpp:208] tablet checkpoint tasks producer thread started
RuntimeLogger I20240624 00:36:46.326066 1460954 olap_server.cpp:448] begin to produce tablet meta checkpoint tasks.
RuntimeLogger I20240624 00:36:46.326093 1459644 olap_server.cpp:213] tablet path check thread started
RuntimeLogger I20240624 00:36:46.326190 1459644 olap_server.cpp:219] cache clean thread started
RuntimeLogger I20240624 00:36:46.326336 1459644 olap_server.cpp:231] path gc threads started. number:1
RuntimeLogger I20240624 00:36:46.326643 1460958 olap_server.cpp:424] try to start path gc thread!
```

The meanings of different prefixes are as follows:

- `RuntimeLogger`: corresponds to the logs in `be.INFO`.

> Support for `jni.log` will be added in future versions.

In addition, there is an additional configuration parameter for container environments:

Configuration Item	Default Value	Options	Description
`enable_file_logger`	true	true, false	Whether to enable file logging. The default is `true`. When starting the BE process with the `--console` command, logs will be output to both the standard output stream and the normal log file. When set to `false`, logs will only be output to the standard output stream and will not generate log files.

Monitor Metrics

Monitor Metrics

Doris FE process and BE processes provide complete monitoring metrics. Monitoring metrics can be divided into two categories:

1. **Process monitoring**: mainly displays some monitoring values of the Doris process itself .
2. **Node monitoring**: mainly displays the monitoring of the node machine itself where the Doris process is located, such as CPU , memory, IO , network , etc.

You can obtain the current monitoring by accessing the http port of the FE or BE node . like :

```
curl http://fe_host:http_port/metrics
curl http://be_host:webserver_port/metrics
```

Monitoring metrics are generated in a Prometheus compatible format, for eg:

```
doris_fe_cache_added{type="partition"} 0
doris_fe_cache_added{type="sql"} 0
doris_fe_cache_hit{type="partition"} 0
doris_fe_cache_hit{type="sql"} 0
doris_fe_connection_total 2
```

Monitoring metrics in Json format can be fetched using `type` parameter in rest interface, for eg:

```
curl http://fe_host:http_port/metrics?type=json
curl http://be_host:webserver_port/metrics?type=json
```

Monitoring levels and best practices

The last column in the table marks the importance level of the monitoring items. P0 means the most important, the larger the value , the lower the importance.

The vast majority of monitoring metric types are Counter . That is the cumulative value. You can obtain effective information by collecting monitoring values at intervals (such as every 15 seconds) and calculating the slope per unit time .

The query error rate can be obtained by calculating the slope of `doris_fe_query_err`, presented as (error per second).

FE Monitoring Metrics

Process monitoring

name	Label	unit	Description	Implication	Grade
doris_fe_cache_added	{type="partition"}	Num	Cumulative value of the number of new Partition Cache		
	{type=" sql "}	Num	Cumulative value of the number of new SQL Cache		
doris_fe_cache_hit	{type="partition"}	Num	Count of partition cache hits		
	{type=" sql "}	Num	Count of SQL Cache hits		
doris_fe_connection_total		Num	Current number of FE MySQL port connections	Used to monitor the number of query connections. If the number of connections exceeds the limit, new connections will not be accessible .	P0
doris_fe_counter_hit_sql_block_rule		Num	Number of queries blocked by SQL BLOCK RULE		
doris_fe_edit_log_clean	{type="failed"}	Num	to clear historical metadata logs	It should not fail. If it fails, manual intervention is required.	P0
	{type="success"}	Num	The number of times the historical metadata log was successfully cleared		
doris_fe_edit_log	{type=" accumulated_bytes "}	byte	value of metadata log write amount	The write rate can be obtained by calculating the slope to observe whether there is a delay in metadata writing.	P0
	{type=" current_bytes "}	byte	Metadata log current value	Used to monitor editlog size. If the size exceeds the limit, manual intervention is required	P0
	{type="read"}	Num	Count of metadata log reads	Observe whether the metadata reading frequency is normal through the slope	P0
	{type="write"}	Num	Count of metadata log writes	Observe whether the metadata writing frequency is normal through the slope	P0
	{type="current"}	Num	Current number of metadata logs	Used to monitor editlog quantity. If the quantity exceeds the limit, manual intervention is required	P0
doris_fe_editlog_write_latency_ms		millisecond	metadata log write latency . For example, {quantile="0.75"} indicates the 75th percentile write latency .		

name	Label	unit	Description	Implication	Grade
doris_fe_image_clean	{type="failed"}	Num	to clean historical metadata image files	It should not fail. If it fails, manual intervention is required.	P0
	{type="success"}	Num	The number of times successful cleaning of historical metadata image files		
doris_fe_image_push	{type="failed"}	Num	The number of failures in pushing metadata image files to other FE nodes		
	{type="success"}	Num	metadata image files to other FE nodes		
doris_fe_image_write	{type="failed"}	Num	Number of failures to generate metadata image files	It should not fail. If it fails, manual intervention is required.	P0
	{type="success"}	Num	the metadata image file was successfully generated		
doris_fe_job		Num	Current count of different job types and different job statuses. For example, {job="load", type="INSERT", state="LOADING"} represents an import job of type INSERT and the number of jobs in the LOADING state .	observe the number of different types of jobs in the cluster as needed	P0
doris_fe_max_journal_id		Num	The maximum metadata log ID of the current FE node . If it is Master FE , it is the maximum ID currently written , if it is a non- Master FE , represents the maximum ID of the metadata log currently being played back	Used to observe whether the ID gap between multiple FEs is too large. If it is too large, it indicates a problem with metadata synchronization.	P0
doris_fe_max_tablet_compaction_score		Num	The largest compaction score value among all BE nodes .	This value can be used to observe the maximum compaction score of the current cluster to determine whether it is too high. If it is too high, query or write delays may occur.	P0
doris_fe_qps		Num/Sec	Current number of FE queries per second (only query requests are counted)	QPS	P0
doris_fe_query_err		Num	value of error query		
doris_fe_query_err_rate		Num/Sec	Error queries per second	Observe whether query errors occur in the cluster	P0
doris_fe_query_latency_ms		millisecond	Percentile statistics of query request latency. For example, {quantile="0.75"} indicates the query delay at the 75th percentile	Detailed observation of query latency in each quantile	P0

name	Label	unit	Description	Implication	Grade
doris_fe_query_latency_ms_db		millisecond	Percentile statistics of query request delay of each DB . For example, {quantile=“0.75”,db =“test”} indicates the query delay of the 75th percentile of DB test	Observe the query latency of each DB in detail	P0
doris_fe_query_olap_table		Num	the statistics of the number of requests for the internal table (OlapTable)		
doris_fe_query_total		Num	all query requests		
doris_fe_report_queue_size		Num	The queue length of various periodic reporting tasks of BE on the FE side	This value reflects the blocking degree of the reporting task on the Master FE node. The larger the value, the lower the processing capacity of the FE .	P0
doris_fe_request_total		Num	All operation requests received through the MySQL port (including queries and other statements)		
doris_fe_routine_load_error_rows		Num	Count the total number of error rows for all Routine Load jobs in the cluster		
doris_fe_routine_load_receive_bytes		byte	the amount of data received by all Routine Load jobs in the cluster		
doris_fe_routine_load_rows		Num	Count the number of data rows received by all Routine Load jobs in the cluster		
doris_fe_rps		Num	current number of FE requests per second (including queries and other types of statements)	Work with QPS to view the amount of requests processed by the cluster .	P0
doris_fe_scheduled_tablet_num		Num	tablets being scheduled by the Master FE node . Includes replicas being repaired and replicas being balanced	tablets being migrated . If there is a value for a long time, it means the cluster is unstable.	P0
doris_fe_tablet_max_compaction_score		Num	The compaction core reported by each BE node . For example, { backend=“172.21.0.1:9556”} represents the reported value of BE “172.21.0.1:9556”		
doris_fe_tablet_num		Num	current total number of tablets on each BE node . For example, {backend=“172.21.0.1:9556”} indicates the current number of tablets of the BE “172.21.0.1:9556”	You can check whether the tablet distribution is uniform and whether the absolute value is reasonable	P0

name	Label	unit	Description	Implication	Grade
doris_fe_tablet_status_count		Num	Statistics Master FE node The cumulative value of the number of tablets scheduled by the tablet scheduler .		
	{type=“added”}	Num	Statistics Master FE node The cumulative value of the number of tablets scheduled by the tablet scheduler . “added” indicates the number of tablets that have been scheduled		
	{type=” in_sched “}	Num	Same as above. Indicates the number of tablets that are scheduled repeatedly	If this value increases quickly, it means that a tablet has been in an unhealthy state for a long time , causing it to be scheduled repeatedly by the scheduler.	
	{type=” not_ready “}	Num	Same as above. Indicates the number of tablets that have not yet met the scheduling trigger conditions .	If this value increases quickly, it means that a large number of tablets are in an unhealthy state but cannot be scheduled.	
	{type=“total”}	Num	Same as above. Represents the cumulative number of tablets that have been checked (but not necessarily scheduled) .		
	{type=“unhealthy”}	Num	Same as above. Indicates the cumulative number of unhealthy tablets that have been checked .		
doris_fe_thread_pool		Num	Count the number of working threads and queuing status of various thread pools . “active_thread_num” Indicates the number of tasks being executed . “pool_size” Indicates the total number of threads in the thread pool . “task_in_queue” Indicates the number of tasks being queued		
	{name=“agent-task-pool”}	Num	Master FE is used to send Agent Task to BE ’s thread pool		
	{name=“connect-scheduler-check-timer”}	Num	pool for checking if MySQL idle connection has timed out		
	{name=“connect-scheduler-pool”}	Num	pool for receiving MySQL connection requests		
	{name=” mysql - nio - pool”}	Num	NIO MySQL Server thread pool for processing tasks		
	{name=“export-exporting-job-pool”}	Num	pool for export jobs in exporting state		

name	Label	unit	Description	Implication	Grade
	{name=“export-pending-job-pool”}	Num	pool for export jobs in pending state		
	{name=“heartbeat-mgr -pool”}	Num	Master FE is used to process the thread pool of each node’s heartbeat		
	{name=“loading-load-task-scheduler”}	Num	Master FE is used to schedule Broker In Load job, loading Task scheduling thread pool		
	{name=“pending-load-task-scheduler”}	Num	Master FE is used to schedule Broker Load job , pending Task scheduling thread pool		
	{name=“schema-change-pool”}	Num	pool used by Master FE to schedule schema change jobs		
	{name=“thrift-server-pool”}	Num	worker thread pool of ThriftServer on the FE side . correspond fe.conf middle rpc_port . Used to interact with BE .		
doris_fe_txn_counter		Num	value of the number of imported transactions in each status	You can observe the execution of imported transactions .	P0
	{type=“begin”}	Num	Number of transactions committed		
	{type=“failed”}	Num	Number of failed transactions		
	{type=“reject”}	Num	rejected transactions . (If the number of currently running transactions is greater than the threshold, new transactions will be rejected)		
	{type=” succes “}	Num	number of successful transactions		
doris_fe_txn_status		Num	Count the number of import transactions currently in various states. For example, {type=“committed”} indicates the number of transactions in the committed state.	You can observe the number of imported transactions in each state to determine whether there is accumulation.	P0
doris_fe_query_instance_num		Num	Specifies the fragment that the user is currently requesting Number of instances . For example, {user=” test_u “} represents the user test_u The number of instances currently being requested	This value can be used to observe whether the specified user takes up too many query resources .	P0

name	Label	unit	Description	Implication	Grade
doris_fe_query_instance_begin		Num	Specify the fragment where the user request starts Number of instances . For example, {user=" test_u "} represents the user test_u Number of instances to start requesting	This value can be used to observe whether a given user has submitted too many queries.	P0
doris_fe_query_rpc_total		Num	of RPCs sent to the specified BE . For example, { be="192.168.10.1"} indicates the number of RPCs sent to BE with IP address 192.168.10.1	This value can be used to observe whether too many RPCs are submitted to a certain BE .	
doris_fe_query_rpc_failed		Num	RPC failures sent to the specified BE . For example, { be="192.168.10.1"} indicates the number of RPC failures sent to BE with IP address 192.168.10.1	This value can be used to observe whether a certain BE has RPC problems.	
doris_fe_query_rpc_size		Num	Specify the RPC data size of BE . For example, { be="192.168.10.1"} indicates the number of RPC data bytes sent to BE with IP address 192.168.10.1	This value can be used to observe whether an excessively large RPC is submitted to a BE .	
doris_fe_txn_exec_latency_ms		millisecond	Percentile statistics of transaction execution time. For example, {quantile="0.75"} indicates the 75th percentile transaction execution time	Observe the execution time of each digit transaction in detail	P0
doris_fe_txn_publish_latency_ms		millisecond	Percentile statistics of transaction publish time. For example, {quantile="0.75"} indicates that the 75th percentile transaction publish time is	of the publishing time of each quantile transaction	P0
doris_fe_txn_num		Num	Specifies the number of transactions being performed by the DB . For example, { db ="test"} indicates the number of transactions currently being executed by DB test .	This value can be used to observe whether a certain DB has submitted a large number of transactions .	P0
doris_fe_publish_txn_num		Num	Specify the number of transactions being published by the DB . For example, { db ="test"} indicates the number of transactions currently being published by DB test .	This value can be used to observe the number of publish transactions of a certain DB .	P0
doris_fe_txn_replica_num		Num	Specifies the number of replicas opened by the transaction being executed by the DB . For example, { db ="test"} indicates the number of copies opened by the transaction currently being executed by DB test .	This value can be used to observe whether a certain DB has too many copies opened, which may affect the execution of other transactions.	P0

name	Label	unit	Description	Implication	Grade
doris_fe_thrift_rpc_total		Num	RPC requests received by each method of the FE thrift interface . For example, {method=“report”} indicates the number of RPC requests received by the report method.	This value can observe a certain thrift rpc method payload	
doris_fe_thrift_rpc_latency_ms		millisecond	The RPC requests received by each method of the FE thrift interface take time. For example, {method=“report”} indicates that the RPC request received by the report method takes time.	This value can observe a certain thrift rpc method payload	
doris_fe_external_schema_cache	{ catalog =“hive”}	Num	SpecifyExternal Catalog _ The number of corresponding schema caches		
doris_fe_hive_meta_cache	{ catalog =“hive”}	Num			
	{type=” partition_value “}	Num	Specify External Hive Metastore Catalog The number of corresponding partition value caches		
	{type=“partition”}	Num	Specify External Hive Metastore Catalog The number of corresponding partition caches		
	{type=“file”}	Num	Specify External Hive Metastore Catalog The number of corresponding file caches		

JVM metrics

name	Label	unit	Description	Impact	Grade
`jvm_heap_size_bytes`		byte	JVM memory metrics. The tags include max, used, committed , corresponding to the maximum value, used and requested memory respectively.	Observe JVM memory usage	P0
`jvm_non_heap_size_bytes`		byte	JVM off-heap memory statistics		
`<GarbageCollector>`			GC metrics .	GarbageCollector refers to a specific garbage collector	P0
	{type=“count”}	Num	Cumulative value of GC times		

name	Label	unit	Description	Impact	Grade
	{type="time"}	millisecond	Cumulative value of GC time consumption		
jvm_old_size_bytes		byte	JVM old generation memory statistics		P0
jvm_thread		Num	JVM thread count statistics	Observe whether the number of JVM threads is reasonable	P0
jvm_young_size_bytes		byte	JVM new generation memory statistics		P0

Machine metrics

name	Label	unit	Description	Impact	Grade
system_meminfo		byte	FE node machines. Collected from /proc/meminfo . include buffers , cached , memory_available , memory_free , memory_total		
system_snmp		FE node machines. Collected from /proc/net/ snmp .			
	{name=" tcp_in_errs "}	Num	tcp packet reception errors		
	{name=" tcp_in_segs "}	Num	tcp packets sent		
	{name=" tcp_out_segs "}	Num	tcp packets sent		
	{name=" tcp_retrans_segs "}	Num	Number of tcp packet retransmissions		

BE metrics metrics

Process metrics

name	Label	unit	Description	Impact	grade
doris_be_active_scan_context_count		Num	the number of scanners currently opened directly from the outside		

name	Label	unit	Description	Impact	grade
doris_be_add_batch_task_queue_size		Num	When recording import, the queue size of the thread pool that receives the batch	If it is greater than 0 , it means there is a backlog at the receiving end of the import task.	P0
agent_task_queue_size		Num	Display the length of each Agent Task processing queue, such as {type="CREATE_TABLE"} Indicates the length of the CREATE_TABLE task queue		
doris_be_brpc_endpoint_stub_count		Num	Created _ The number of brpc stubs used for interaction between BEs		
doris_be_brpc_function_endpoint_stub_count		Num	Created _ The number of brpc stubs used to interact with Remote RPC		
doris_be_cache_capacity			Record the capacity of the specified LRU Cache		
doris_be_cache_usage			Record the usage of the specified LRU Cache	Used to observe memory usage	P0
doris_be_cache_usage_ratio			Record the usage of the specified LRU Cache		
doris_be_cache_lookup_count			Record the number of times the specified LRU Cache is searched		
doris_be_cache_hit_count			Record the number of hits in the specified LRU Cache		
doris_be_cache_hit_ratio			Record the hit rate of the specified LRU Cache	Used to observe whether the cache is effective	P0
	{name=" DataPageCache "}	Num	DataPageCache Data Page for caching data	Data Cache , directly affects query efficiency	P0
	{name=" IndexPageCache "}	Num	IndexPageCache Index Page for caching data	Index Cache , directly affects query efficiency	P0
	{name=" LastestSuccessChannelCache "}	Num	LastestSuccessChannelCache Used to cache import receivers LoadChannel		
	{name=" SegmentCache "}	Num	SegmentCache Used to cache turned on Segment , such as index information		
doris_be_chunk_pool_local_core_alloc_count		Num	ChunkAllocator , the number of times memory is allocated from the memory queue of the bound core		

name	Label	unit	Description	Impact	grade
doris_be_chunk_pool_other_core_alloc_count		Num	ChunkAllocator , the number of times memory is allocated from the memory queue of other cores		
doris_be_chunk_pool_reserved_bytes		byte	ChunkAllocator The amount of memory reserved in		
doris_be_chunk_pool_system_alloc_cost_ns		nanosecond	SystemAllocator The cumulative value of time spent applying for memory	The slope can be used to observe the time taken for memory allocation.	P0
doris_be_chunk_pool_system_alloc_count		Num	SystemAllocator Number of times to apply for memory		
doris_be_chunk_pool_system_free_cost_ns		nanosecond	SystemAllocator Cumulative value of time taken to release memory	The slope can be used to observe the time it takes to release memory.	P0
doris_be_chunk_pool_system_free_count		Num	SystemAllocator The number of times memory is released		
doris_be_compaction_bytes_total		byte	value of the amount of data processed by compaction	What is recorded is the input in the compaction task rowset The disk size . It can be observed through the slope rate of compaction	P0
	{type=“base”}	byte	Accumulated data volume of Base Compaction		
	{type=“cumulative”}	byte	The cumulative data volume of Cumulative Compaction		
doris_be_compaction_deltas_total		Num	processed by compaction rowset The cumulative value of the number	What is recorded is the input in the compaction task rowset of number	
	{type=“base”}	Num	Processed by Base Compaction rowset Cumulative number		
	{type=“cumulative”}	Num	Processed by Cumulative Compaction rowset Cumulative number		
doris_be_disks_compaction_num		Num	compaction tasks being executed on the specified data directory . like {path=“/path1/”} means /path1 The number of tasks being executed on the directory	Used to observe whether the number of compaction tasks on each disk is reasonable.	P0
doris_be_disks_compaction_score		Num	Specifies the number of compaction tokens being executed on the data directory. like {path=“/path1/”} means /path1 Number of tokens being executed on the directory		

name	Label	unit	Description	Impact	grade
doris_be_compaction_used_permits		Num	The number of tokens used by the Compaction task	Used to reflect the resource consumption of Compaction	
doris_be_compaction_waitting_permits		Num	Compaction tokens awaiting		
doris_be_data_stream_receiver_count		Num	Number of data receiving terminals Receiver	FIXME : This indicator is missing for the vectorization engine	
doris_be_disks_avail_capacity		byte	Specify the remaining space on the disk where the specified data directory is located. like {path="/path1/"} express /path1 The remaining space on the disk where the directory is located		P0
doris_be_disks_local_used_capacity		byte	the specified data directory is located		
doris_be_disks_remote_used_capacity		byte	the specified data directory is located		
doris_be_disks_state		Boolean	Specifies the disk status of the data directory . 1 means normal. 0 means abnormal		
doris_be_disks_total_capacity		byte	capacity of the disk where the specified data directory is located	Cooperate doris_be_disks_avail_capacity Calculate disk usage	P0
doris_be_engine_requests_total		Num	value of various task execution statuses on BE		
	{status=" failed ",type ="xxx"}	Num	value of the number of failures for tasks of type xxx		
	{status=" total ",type ="xxx"}	Num	The cumulative value of the total number of tasks of type xxx .	Can monitor the number of failures of various tasks on demand	P0
	{status=" skip ",type =" report_all_tablets "}	Num	value of the number of times xxx type tasks have been skipped		
doris_be_fragment_endpoint_count		Num	same	FIXME: Same as doris_be_data_stream_receiver_count number. And the vectorization engine is missing	
doris_be_fragment_request_duration_us		microseconds	All fragment intance The cumulative execution time of	the execution time of instance through slope	P0
doris_be_fragment_requests_total		Num	The cumulative number of executed fragment instances		

name	Label	unit	Description	Impact	grade
doris_be_load_channel_count		Num	The number of load channels currently open	The larger the value , the more import tasks are currently being executed.	P0
doris_be_local_bytes_read_total		byte	Depend on LocalFileReader Number of bytes read		P0
doris_be_local_bytes_written_total		byte	Depend on LocalFileWriter Number of bytes written		P0
doris_be_local_file_reader_total		Num	opened LocalFileReader Cumulative count of		
doris_be_local_file_open_reading		Num	currently open LocalFileReader number		
doris_be_local_file_writer_total		Num	opened LocalFileWriter cumulative count .		
doris_be_mem_consumption		byte	Specifies the current memory overhead of the module . For example, {type="compaction"} represents the current total memory overhead of the compaction module .	Values taken from the same type MemTracker . FIXME	
doris_be_memory_allocated_bytes		byte	BE process physical memory size, taken from /proc/self/status/ VmRSS		P0
doris_be_memory_jemalloc		byte	Jemalloc stats, taken from je_mallctl .	Meaningreference : https://jemalloc.net/jemalloc.3.html	P0
doris_be_memory_pool_bytes_total		byte	all MemPool The size of memory currently occupied. Statistical value, does not represent actual memory usage .		
doris_be_memtable_flush_duration_us		microseconds	value of the time taken to write memtable to disk	Write latency can be observed via slope	P0
doris_be_memtable_flush_total		Num	number of memtable writes to disk	The slope can be used to calculate the frequency of writing to a file	P0
doris_be_meta_request_duration		microseconds	access RocksDB The cumulative time consumption of meta in	BE metadata read and write latency through slope	P0

name	Label	unit	Description	Impact	grade
	{type="read"}	microseconds	Reading time		
	{type="write"}	microseconds	Writing time		
doris_be_meta_request_total		Num	access RocksDB The cumulative number of meta in	BE metadata access frequency by slope	P0
	{type="read"}	Num	Read times		
	{type="write"}	Num	Number of writes		
doris_be_fragment_instance_count		Num	The number of fragment instances currently received	Observe whether instance accumulation occurs	P0
doris_be_process_fd_num_limit_hard		Num	BE process. pass /proc/ pid /limits collection		
doris_be_process_fd_num_limit_soft		Num	BE process. pass /proc/ pid /limits collection		
doris_be_process_fd_num_used		Num	The number of file handles used by the BE process. pass /proc/ pid /limits collection		
doris_be_process_thread_num		Num	BE process threads. pass /proc/ pid /task collection		P0
doris_be_query_cache_memory_total_byte		byte	Number of bytes occupied by Query Cache		
doris_be_query_cache_partition_total_count		Num	current number of Partition Cache caches		
doris_be_query_cache_sql_total_count		Num	Current number of SQL Cache caches		
doris_be_query_scan_bytes		byte	Read the cumulative value of the data amount. Here we only count reads Olap The amount of data in the table		
doris_be_query_scan_bytes_per_second		Bytes/second	according to doris_be_query_scan_bytes Calculated read rate	Observe query rate	P0

name	Label	unit	Description	Impact	grade
doris_be_query_scan_rows		Num	Read the cumulative value of the number of rows. Here we only count reads Olap The amount of data in the table. and is RawRowsRead (Some data rows may be skipped by the index and not actually read, but will still be recorded in this value)	Query rate by slope observation	P0
doris_be_result_block_queue_count		Num	The number of fragment instances in the current query result cache	This queue is only used when being read directly by an external system. For example, Spark on Doris queries data through external scan.	
doris_be_result_buffer_block_count		Num	The number of queries in the current query result cache	This value reflects how many query results are currently waiting for FE consumption in BE .	P0
doris_be_routine_load_task_count		Num	The number of routine load tasks currently being executed		
doris_be_rowset_count_generated_and_in_use		Num	New and in use since the last startup The number of rowset ids .		
doris_be_s3_bytes_read_total		Num	S3FileReader The cumulative number of opens		
doris_be_s3_file_open_reading		Num	currently open S3FileReader number		
doris_be_s3_bytes_read_total		byte	S3FileReader Read the cumulative value of bytes		
doris_be_scanner_thread_pool_queue_size		Num	used for OlapScanner The current queued number of thread pools	it is greater than zero , it means that Scanner starts to accumulate.	P0
doris_be_segment_read	{type=” segment_read_total “}	Num	Cumulative value of the number of segments read		
doris_be_segment_read	{type=” segment_row_total “}	Num	value of the number of rows in the segment read	This value also includes the number of rows filtered by the index. Equivalent to the number of segments read * the total number of rows in each segment	
doris_be_send_batch_thread_pool_queue_size		Num	The number of queues in the thread pool used to send data packets when importing	it is greater than 0 , it means there is accumulation	P0
doris_be_send_batch_thread_pool_thread_num		Num	The number of threads in the thread pool used to send packets when importing		

name	Label	unit	Description	Impact	grade
doris_be_small_file_cache_count		Num	currently cached by BE		
doris_be_streaming_load_current_processing		Num	Number of stream load tasks currently running	Contains only tasks sent by the curl command	
doris_be_streaming_load_duration_ms		millisecond	The cumulative value of the execution time of all stream load tasks		
doris_be_streaming_load_requests_total		Num	value of the number of stream load tasks	Observable task submission frequency via slope	P0
doris_be_stream_load_pipe_count		Num	current stream load data pipelines	Including stream load and routine load tasks	
doris_be_stream_load	{type=" load_rows "}	Num	number of rows finally imported by stream load	Including stream load and routine load tasks	P0
doris_be_stream_load	{type=" receive_bytes "}	byte	value of the number of bytes received by stream load	Including stream load data received from http , and routine load from kafka read data	P0
doris_be_tablet_base_max_compaction_score		Num	The current largest Base Compaction Score	This value changes in real time, and peak data may be lost. The higher the value, the more serious the compaction accumulation is .	P0
doris_be_tablet_cumulative_max_compaction_score		Num	Same as above. Current largest Cumulative Compaction Score		
doris_be_tablet_version_num_distribution		Num	The histogram of the number of tablet versions .	Distribution used to reflect the number of tablet versions	P0
doris_be_thrift_connections_total		Num	thrift connections created . like {name="heartbeat"} Indicates the cumulative number of connections to the heartbeat service	This value is the connection of thrift server with BE as the server.	
doris_be_thrift_current_connections		Num	current number of thrift connections. like {name="heartbeat"} Indicates the current number of connections to the heartbeat service .	Same as above	
doris_be_thrift_opened_clients		Num	thrift clients currently open . like {name="frontend"} Indicates the number of clients accessing the FE service		

name	Label	unit	Description	Impact	grade
doris_be_thrift_used_clients		Num	thrift clients currently in use . like {name="frontend"} Indicates the number of clients being used to access the FE service		
doris_be_timeout_canceled_fragment_count		Num	Cumulative value of the number of fragment instances canceled due to timeout	This value may be recorded repeatedly. For example, some fragment instances are canceled multiple times	P0
doris_be_stream_load_txn_request	{type="begin"}	Num	value of the number of transactions started by stream load	Including stream load and routine load tasks	
doris_be_stream_load_txn_request	{type="commit"}	Num	value of the number of transactions successfully executed by stream load	Same as above	
doris_be_stream_load_txn_request	{type="rollback"}		value of the number of transactions that failed to execute stream load	Same as above	
doris_be_unused_rowsets_count		Num	The number of currently abandoned rowsets	These rowsets will be deleted regularly under normal circumstances.	
doris_be_upload_fail_count		Num	cumulative value of rowset failed to be uploaded to remote storage		
doris_be_upload_rowset_count		Num	cumulative number of rowsets successfully uploaded to remote storage		
doris_be_upload_total_byte			byte	value of rowset data successfully uploaded to remote storage	
doris_be_load_bytes		byte	Cumulative quantity sent through tablet sink	Observable amount of imported data	P0
doris_be_load_rows		Num	Cumulative number of rows sent through tablet sink	Observable amount of imported data	P0
fragment_thread_pool_queue_size		Num	current query execution thread pool waiting queue	If it is greater than zero, the query thread has been exhausted and queries will pile up.	P0
doris_be_all_rowsets_num		Num	All currently rowset number of		P0
doris_be_all_segments_num		Num	The number of all current segments		P0
doris_be_heavy_work_max_threads		Num	brpc Number of heavy thread pool threads		p0
doris_be_light_work_max_threads		Num	brpc Number of light thread pool threads		p0

name	Label	unit	Description	Impact	grade
doris_be_heavy_work_pool_queue_size		Num	brpc The maximum length of the heavy thread pool queue will block the submission of work if it exceeds it .		p0
doris_be_light_work_pool_queue_size		Num	brpc The maximum length of the light thread pool queue . If it exceeds the maximum length, the submission of work will be blocked.		p0
doris_be_heavy_work_active_threads		Num	brpc Number of active threads in heavy thread pool		p0
doris_be_light_work_active_threads		Num	brpc Number of active threads in light thread pool		p0

Machine metrics

name	Label	unit	Description	Impact	Grade
doris_be_cpu		Num	CPU related metrics metrics, from /proc/stat collection. Each value of each logical core will be collected separately . like {device="cpu0",mode="nice"} Indicates the nice value of cpu0	CPU usage can be calculated	P0
doris_be_disk_bytes_read		byte	The cumulative value of disk reads. from /proc/ diskstats collection. The values of each disk will be collected separately . like {device=" vdd "} express vvd disk value		
doris_be_disk_bytes_written		byte	The cumulative value of disk writes. The collection method is the same as above		
doris_be_disk_io_time_ms		byte	The collection method is the same as above	IO Util can be calculated	P0
doris_be_disk_io_time_weighted		byte	The collection method is the same as above		
doris_be_disk_reads_completed		byte	The collection method is the same as above		
doris_be_disk_read_time_ms		byte	The collection method is the same as above		
doris_be_disk_writes_completed		byte	The collection method is the same as above		

name	Label	unit	Description	Impact	Grade
doris_be_disk_write_time_ms		byte	The collection method is the same as above		
doris_be_fd_num_limit		Num	System file handle limit upper limit. from /proc/sys/fs/file-nr collection		
doris_be_fd_num_used		Num	The number of file handles used by the system . from /proc/sys/fs/file-nr collection		
doris_be_file_created_total		Num	Cumulative number of local file creation times	all calls local_file_writer And finally close the file count	
doris_be_load_average		Num	Machine Load Avg Metric metrics. For example, {mode="15_minutes"} is 15 minutes Load Avg	Observe the overall machine load	P0
doris_be_max_disk_io_util_percent		percentage	value of the disk with the largest IO UTIL among all disks		P0
doris_be_max_network_receive_bytes_rate		Bytes/second	The maximum receive rate calculated among all network cards		P0
doris_be_max_network_send_bytes_rate		Bytes/second	The calculated maximum sending rate among all network cards		P0
doris_be_memory_pgpgin		byte	The amount of data written by the system from disk to memory page		
doris_be_memory_pgpgout		byte	The amount of data written to disk by system memory pages		
doris_be_memory_pswpin		byte	The number of times the system swapped from disk to memory	Normally, swap should be turned off, so this value should be 0	
doris_be_memory_pswpout		byte	The number of times the system swapped from memory to disk	Normally, swap should be turned off, so this value should be 0	
doris_be_network_receive_bytes		byte	each network card are accumulated. Collected from /proc/net/dev		
doris_be_network_receive_packets		Num	each network card is accumulated. Collected from /proc/net/dev		
doris_be_network_send_bytes		byte	each network card . Collected from /proc/net/dev		
doris_be_network_send_packets		Num	The total number of packets sent by each network card is accumulated. Collected from /proc/net/dev		
doris_be_proc	{mode=" ctxt_switch "}	Num	CPU context switches . Collected from /proc/stat	Observe whether there are abnormal context switches	P0

name	Label	unit	Description	Impact	Grade
doris_be_proc	{mode=“interrupt”}	Num	CPU interrupts . Collected from /proc/stat		
doris_be_proc	{mode=” procs_blocked “}	Num	The number of processes currently blocked in the system (such as waiting for IO). Collected from /proc/stat		
doris_be_proc	{mode=” procs_running “}	Num	The number of processes currently executing on the system . Collected from /proc/stat		
doris_be_snmp_tcp_in_errs		Num	tcp packet reception errors. Collected from /proc/net/ snmp	Observable network errors such as retransmission, packet loss, etc. Need and other snmp metrics used together with	P0
doris_be_snmp_tcp_in_segs		Num	tcp packets sent . Collected from /proc/net/ snmp		
doris_be_snmp_tcp_out_segs		Num	tcp packets sent. Collected from /proc/net/ snmp		
doris_be_snmp_tcp_retrans_segs		Num	TCP packet retransmissions . Collected from /proc/net/ snmp		

Monitoring and alarming

Monitoring and alarming

This document mainly introduces Doris's monitoring items and how to collect and display them. And how to configure alarm (TODO)

Dashboard template click download

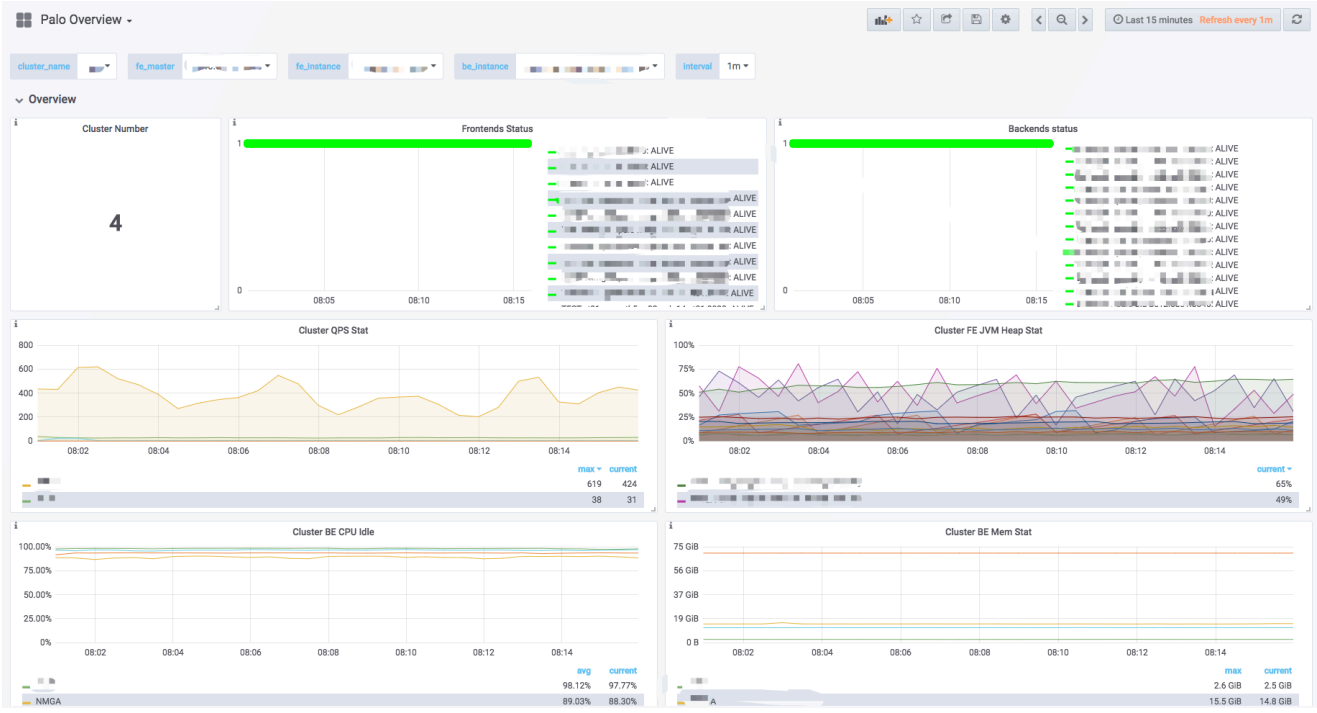
Doris Version	Dashboard Version
1.2.x	revision 5

Dashboard templates are updated from time to time. The way to update the template is shown in the last section.

Welcome to provide better dashboard.

Components

Doris uses Prometheus and Grafana to collect and display input monitoring items.



Components

1. Prometheus

   Prometheus is an open source system monitoring and alarm suite. It can collect monitored items by Pull or Push and store them in its own time series database. And through the rich multi-dimensional data query language, to meet the different data display needs of users.

2. Grafana

   Grafana is an open source data analysis and display platform. Support multiple mainstream temporal database sources including Prometheus. Through the corresponding database query statements, the display data is obtained from the data source. With flexible and configurable dashboard, these data can be quickly presented to users in the form of graphs.

> Note: This document only provides a way to collect and display Doris monitoring data using Prometheus and Grafana. In principle, these components are not developed or maintained. For more details on these components, please step through the corresponding official documents.

Monitoring data

Doris's monitoring data is exposed through the HTTP interface of Frontend and Backend. Monitoring data is presented in the form of key-value text. Each Key may also be distinguished by different Labels. When the user has built Doris, the monitoring data of the node can be accessed in the browser through the following interfaces:

- Frontend: `fe_host:fe_http_port/metrics`
- Backend: `be_host:be_web_server_port/metrics`
- Broker: Not available for now

Users will see the following monitoring item results (for example, FE partial monitoring items):

```
# HELP  jvm_heap_size_bytes jvm heap stat
# TYPE  jvm_heap_size_bytes gauge
jvm_heap_size_bytes{type="max"} 8476557312
jvm_heap_size_bytes{type="committed"} 1007550464
jvm_heap_size_bytes{type="used"} 156375280
# HELP  jvm_non_heap_size_bytes jvm non heap stat
# TYPE  jvm_non_heap_size_bytes gauge
jvm_non_heap_size_bytes{type="committed"} 194379776
jvm_non_heap_size_bytes{type="used"} 188201864
# HELP  jvm_young_size_bytes jvm young mem pool stat
# TYPE  jvm_young_size_bytes gauge
jvm_young_size_bytes{type="used"} 40652376
jvm_young_size_bytes{type="peak_used"} 277938176
jvm_young_size_bytes{type="max"} 907345920
# HELP  jvm_old_size_bytes jvm old mem pool stat
# TYPE  jvm_old_size_bytes gauge
jvm_old_size_bytes{type="used"} 114633448
jvm_old_size_bytes{type="peak_used"} 114633448
jvm_old_size_bytes{type="max"} 7455834112
# HELP  jvm_gc jvm gc stat
# TYPE  jvm_gc gauge
<GarbageCollector>{type="count"} 247
<GarbageCollector>{type="time"} 860
# HELP  jvm_thread jvm thread stat
# TYPE  jvm_thread gauge
jvm_thread{type="count"} 162
jvm_thread{type="peak_count"} 205
jvm_thread{type="new_count"} 0
jvm_thread{type="runnable_count"} 48
jvm_thread{type="blocked_count"} 1
jvm_thread{type="waiting_count"} 41
```

```
jvm_thread{type="timed_waiting_count"} 72
jvm_thread{type="terminated_count"} 0
...
```

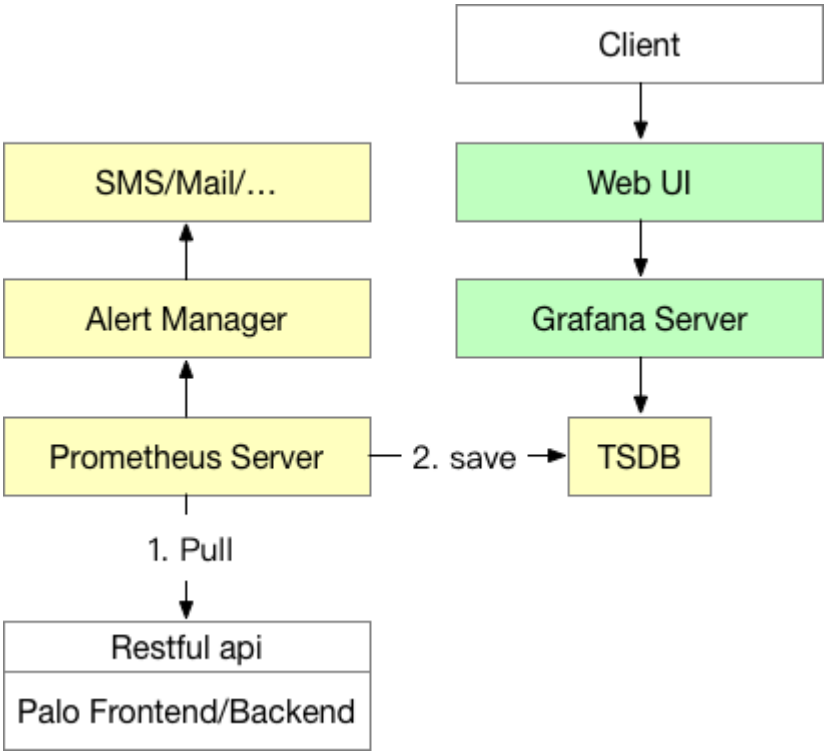
This is a monitoring data presented in Prometheus Format. We take one of these monitoring items as an example to illustrate:

```
# HELP  jvm_heap_size_bytes jvm heap stat
# TYPE  jvm_heap_size_bytes gauge
jvm_heap_size_bytes{type="max"} 8476557312
jvm_heap_size_bytes{type="committed"} 1007550464
jvm_heap_size_bytes{type="used"} 156375280
```

1. Behavior commentary line at the beginning of "#". HELP is the description of the monitored item; TYPE represents the data type of the monitored item, and Gauge is the scalar data in the example. There are also Counter, Histogram and other data types. Specifically, you can see Prometheus Official Document.
2. `jvm_heap_size_bytes` is the name of the monitored item (Key); `type= "max"` is a label named `type`, with a value of `max`. A monitoring item can have multiple Labels.
3. The final number, such as `8476557312`, is the monitored value.

Monitoring Architecture

The entire monitoring architecture is shown in the following figure:



Monitoring Architecture

1. The yellow part is Prometheus related components. Prometheus Server is the main process of Prometheus. At present, Prometheus accesses the monitoring interface of Doris node by Pull, and then stores the time series data in the time series database TSDB (TSDB is included in the Prometheus process, and need not be deployed separately). Prometheus also supports building Push Gateway to

allow monitored data to be pushed to Push Gateway by Push by monitoring system, and then data from Push Gateway by Prometheus Server through Pull.
2. Alert Manager is a Prometheus alarm component, which needs to be deployed separately (no solution is provided yet, but can be built by referring to official documents). Through Alert Manager, users can configure alarm strategy, receive mail, short messages and other alarms.
3. The green part is Grafana related components. Grafana Server is the main process of Grafana. After startup, users can configure Grafana through Web pages, including data source settings, user settings, Dashboard drawing, etc. This is also where end users view monitoring data.

Start building

Please start building the monitoring system after you have completed the deployment of Doris.

Prometheus

1. Download the latest version of Prometheus on the Prometheus Website. Here we take version 2.43.0-linux-amd64 as an example.

2. Unzip the downloaded tar file on the machine that is ready to run the monitoring service.

3. Open the configuration file prometheus.yml. Here we provide an example configuration and explain it (the configuration file is in YML format, pay attention to uniform indentation and spaces):

   Here we use the simplest way of static files to monitor configuration. Prometheus supports a variety of service discovery, which can dynamically sense the addition and deletion of nodes.

```
# my global config
global:
  scrape_interval:     15s # Global acquisition interval, default 1 m, set to 15s
  evaluation_interval: 15s # Global rule trigger interval, default 1 m, set 15s here

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'DORIS_CLUSTER' # Each Doris cluster, we call it a job. Job can be given a name here as the name of Doris
cluster in the monitoring system.
    metrics_path: '/metrics' # Here you specify the restful API to get the monitors. With host: port in the following
targets, Prometheus will eventually collect monitoring items through host: port/metrics_path.
    static_configs: # Here we begin to configure the target addresses of FE and BE, respectively. All FE and BE are written
into their respective groups.
      - targets: ['fe_host1:8030', 'fe_host2:8030', 'fe_host3:8030']
        labels:
          group: fe # Here configure the group of fe, which contains three Frontends

      - targets: ['be_host1:8040', 'be_host2:8040', 'be_host3:8040']
        labels:
          group: be # Here configure the group of be, which contains three Backends

  - job_name: 'DORIS_CLUSTER_2' # We can monitor multiple Doris clusters in a Prometheus, where we begin the configuration
```

```
of another Doris cluster. Configuration is the same as above, the following is outlined.
    metrics_path: '/metrics'
    static_configs:
      - targets: ['fe_host1:8030', 'fe_host2:8030', 'fe_host3:8030']
        labels:
          group: fe

      - targets: ['be_host1:8040', 'be_host2:8040', 'be_host3:8040']
        labels:
          group: be
```

4. start Prometheus

   Start Prometheus with the following command:

   `nohup ./prometheus --web.listen-address="0.0.0.0:8181" &`

   This command will run Prometheus in the background and specify its Web port as 8181. After startup, data is collected and stored in the data directory.

5. stop Promethues

   At present, there is no formal way to stop the process, kill - 9 directly. Of course, Prometheus can also be set as a service to start and stop in a service way.

6. access Prometheus

   Prometheus can be easily accessed through web pages. The page of Prometheus can be accessed by opening port 8181 through browser. Click on the navigation bar, `Status` -> `Targets`, and you can see all the monitoring host nodes of the grouped Jobs. Normally, all nodes should be `UP`, indicating that data acquisition is normal. Click on an `Endpoint` to see the current monitoring value. If the node state is not UP, you can first access Doris's metrics interface (see previous article) to check whether it is accessible, or query Prometheus related documents to try to resolve.

7. So far, a simple Prometheus has been built and configured. For more advanced usage, see Official Documents

Grafana

1. Download the latest version of Grafana on Grafana's official website. Here we take version 8.5.22.linux-amd64 as an example.

2. Unzip the downloaded tar file on the machine that is ready to run the monitoring service.

3. Open the configuration file conf/defaults.ini. Here we only list the configuration items that need to be changed, and the other configurations can be used by default.

```
# Path to where grafana can store temp files, sessions, and the sqlite3 db (if that is used)
data = data

# Directory where grafana can store logs
logs = data/log

# Protocol (http, https, socket)
protocal = http
```

```
# The ip address to bind to, empty will bind to all interfaces
http_addr =

# The http port to use
http_port = 8182
```

4. start Grafana

   Start Grafana with the following command

   `nohup ./bin/grafana-server &`

   This command runs Grafana in the background, and the access port is 8182 configured above.

5. stop Grafana

   At present, there is no formal way to stop the process, kill - 9 directly. Of course, you can also set Grafana as a service to start and stop as a service.

6. access Grafana

   Through the browser, open port 8182, you can start accessing the Grafana page. The default username password is admin.

7. Configure Grafana

   For the first landing, you need to set up the data source according to the prompt. Our data source here is Prometheus, which was configured in the previous step.

   The Setting page of the data source configuration is described as follows:

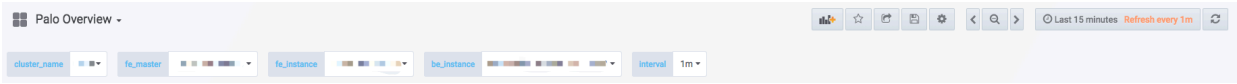
   1. Name: Name of the data source, customized, such as doris_monitor_data_source
   2. Type: Select Prometheus
   3. URL: Fill in the web address of Prometheus, such as http://host:8181
   4. Access: Here we choose the Server mode, which is to access Prometheus through the server where the Grafana process is located.
   5. The other options are available by default.
   6. Click `Save & Test` at the bottom. If `Data source is working`, it means that the data source is available.
   7. After confirming that the data source is available, click on the + number in the left navigation bar and start adding Dashboard. Here we have prepared Doris's dashboard template (at the beginning of this document). When the download is complete, click `New dashboard` -> `Import dashboard` -> `Upload.json File` above to import the downloaded JSON file.
   8. After importing, you can name Dashboard by default `Doris Overview`. At the same time, you need to select the data source, where you select the `doris_monitor_data_source` you created earlier.
   9. Click `Import` to complete the import. Later, you can see Doris's dashboard display.

8. So far, a simple Grafana has been built and configured. For more advanced usage, see Official Documents

Dashboard

Here we briefly introduce Doris Dashboard. The content of Dashboard may change with the upgrade of version. This document is not guaranteed to be the latest Dashboard description.

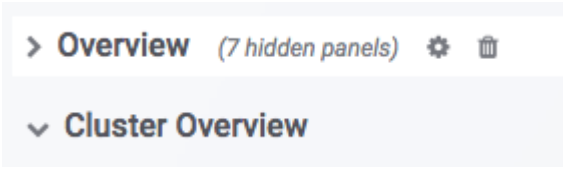
1. Top Bar



Top Bar

   - The upper left corner is the name of Dashboard.
   - The upper right corner shows the current monitoring time range. You can choose different time ranges by dropping down. You can also specify a regular refresh page interval.
   - Cluster name: Each job name in the Prometheus configuration file represents a Doris cluster. Select a different cluster, and the chart below shows the monitoring information for the corresponding cluster.
   - fe_master: The Master Frontend node corresponding to the cluster.
   - fe_instance: All Frontend nodes corresponding to the cluster. Select a different Frontend, and the chart below shows the monitoring information for the Frontend.
   - be_instance: All Backend nodes corresponding to the cluster. Select a different Backend, and the chart below shows the monitoring information for the Backend.
   - Interval: Some charts show rate-related monitoring items, where you can choose how much interval to sample and calculate the rate (Note: 15s interval may cause some charts to be unable to display).

2. Row.

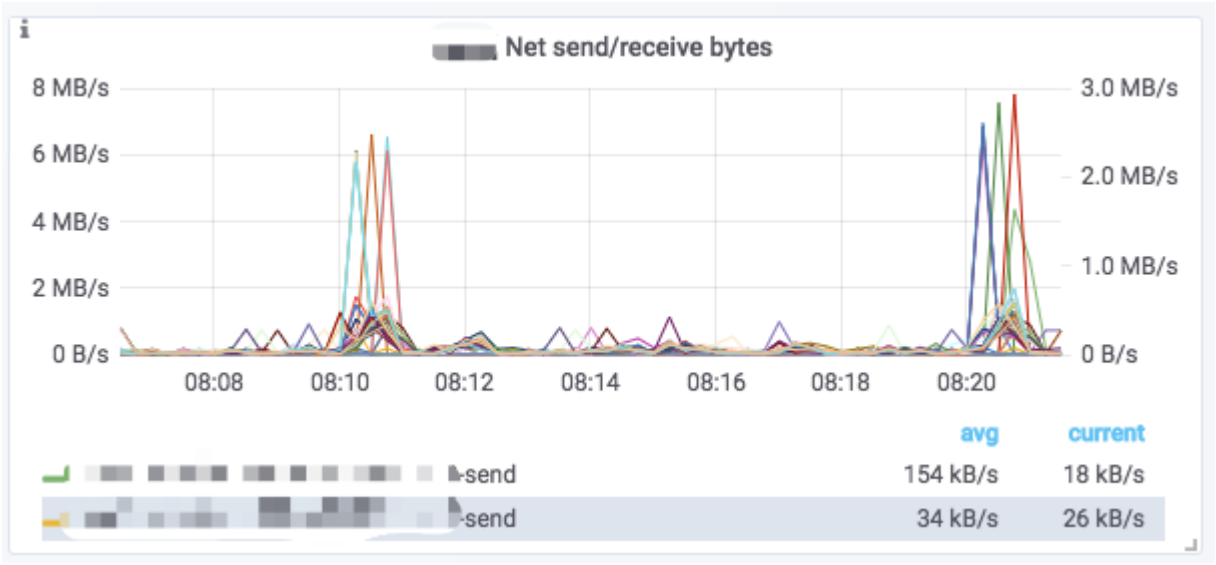


Row

In Grafana, the concept of Row is a set of graphs. As shown in the figure above, Overview and Cluster Overview are two different Rows. Row can be folded by clicking Row. Currently Dashboard has the following Rows (in continuous updates):

   1. Overview: A summary display of all Doris clusters.
   2. Cluster Overview: A summary display of selected clusters.
   3. Query Statistic: Query-related monitoring of selected clusters.
   4. FE JVM: Select Frontend's JVM monitoring.
   5. BE: A summary display of the backends of the selected cluster.
   6. BE Task: Display of Backends Task Information for Selected Clusters.

3. Charts



Charts

A typical icon is divided into the following parts:

1. Hover the I icon in the upper left corner of the mouse to see the description of the chart.
2. Click on the illustration below to view a monitoring item separately. Click again to display all.
3. Dragging in the chart can select the time range.
4. The selected cluster name is displayed in [] of the title.
5. Some values correspond to the Y-axis on the left and some to the right, which can be distinguished by the `-right` at the end of the legend.
6. Click on the name of the chart -> `Edit` to edit the chart.

Dashboard Update

1. Click on + in the left column of Grafana and `Dashboard`.
2. Click `New dashboard` in the upper left corner, and `Import dashboard` appears on the right.
3. Click `Upload .json File` to select the latest template file.
4. Selecting Data Sources
5. Click on `Import (Overwrite)` to complete the template update.

Disk Capacity Management

Disk Capacity Management

This document mainly introduces system parameters and processing strategies related to disk storage capacity.

If Doris' data disk capacity is not controlled, the process will hang because the disk is full. Therefore, we monitor the disk usage and remaining capacity, and control various operations in the Doris system by setting different warning levels, and try to avoid the situation where the disk is full.

Glossary

- Data Dir: Data directory, each data directory specified in the `storage_root_path` of the BE configuration file `be.conf`. Usually a data directory corresponds to a disk, so the following **disk** also refers to a data directory.

Basic Principles

BE will report disk usage to FE on a regular basis (every minute). FE records these statistical values and restricts various operation requests based on these statistical values.

Two thresholds, **High Watermark** and **Flood Stage**, are set in FE. Flood Stage is higher than High Watermark. When the disk usage is higher than High Watermark, Doris will restrict the execution of certain operations (such as replica balancing, etc.). If it is higher than Flood Stage, certain operations (such as load data) will be prohibited.

At the same time, a **Flood Stage** is also set on the BE. Taking into account that FE cannot fully detect the disk usage on BE in a timely manner, and cannot control certain BE operations (such as Compaction). Therefore, Flood Stage on the BE is used for the BE to actively refuse and stop certain operations to achieve the purpose of self-protection.

FE Parameter

High Watermark:

```
storage_high_watermark_usage_percent: default value is 85 (85%).
storage_min_left_capacity_bytes: default value is 2GB.
```

When disk capacity **more than** `storage_high_watermark_usage_percent`, **or** disk free capacity **less than** `storage_min_left_capacity_bytes`, the disk will no longer be used as the destination path for the following operations:

- Tablet Balance
- Colocation Relocation
- Decommission

Flood Stage:

```
storage_flood_stage_usage_percent: default value is 95 (95%).
storage_flood_stage_left_capacity_bytes: default value is 1GB.
```

When disk capacity **more than** `storage_flood_stage_usage_percent`, **or** disk free capacity **less than** `storage_flood_stage_left_capacity_bytes`, the disk will no longer be used as the destination path for the following operations:

- Tablet Balance
- Colocation Relocation
- Replica make up
- Restore
- Load/Insert

BE Parameter

Flood Stage:

```
storage_flood_stage_usage_percent: default value is 90 (90%).
storage_flood_stage_left_capacity_bytes: default value is 1GB.
```

When disk capacity **more than** `storage_flood_stage_usage_percent`, **and** disk free capacity **less than** `storage_flood_stage_left_capacity_bytes`, the following operations on this disk will be prohibited:

- Base/Cumulative Compaction
- Data load
- Clone Task (Usually occurs when the replica is repaired or balanced.)
- Push Task (Occurs during the Loading phase of Hadoop import, and the file is downloaded.)
- Alter Task (Schema Change or Rollup Task.)
- Download Task (The Downloading phase of the recovery operation.)

Disk Capacity Release

When the disk capacity is higher than High Watermark or even Flood Stage, many operations will be prohibited. At this time, you can try to reduce the disk usage and restore the system in the following ways.

- Delete table or partition

  By deleting tables or partitions, you can quickly reduce the disk space usage and restore the cluster.
  Note: Only the `DROP` operation can achieve the purpose of quickly reducing the disk space usage, the `DELETE` operation cannot.

  ```
  DROP TABLE tbl;
  ALTER TABLE tbl DROP PARTITION p1;
  ```

- BE expansion

After backend expansion, data tablets will be automatically balanced to BE nodes with lower disk usage. The expansion operation will make the cluster reach a balanced state in a few hours or days depending on the amount of data and the number of nodes.

- Modify replica of a table or partition

  You can reduce the number of replica of a table or partition. For example, the default 3 replica can be reduced to 2 replica. Although this method reduces the reliability of the data, it can quickly reduce the disk usage rate and restore the cluster to normal. This method is usually used in emergency recovery systems. Please restore the number of copies to 3 after reducing the disk usage rate by expanding or deleting data after recovery.
  Modifying the replica operation takes effect instantly, and the backends will automatically and asynchronously delete the redundant replica.

  ```
  ALTER TABLE tbl MODIFY PARTITION p1 SET("replication_num" = "2");
  ```

- Delete unnecessary files

  When the BE has crashed because the disk is full and cannot be started (this phenomenon may occur due to untimely detection of FE or BE), you need to delete some temporary files in the data directory to ensure that the BE process can start. Files in the following directories can be deleted directly:

  - log/: Log files in the log directory.
  - snapshot/: Snapshot files in the snapshot directory.
  - trash/ Trash files in the trash directory.

  This operation will affect Restore data from BE Recycle Bin.

  If the BE can still be started, you can use `ADMIN CLEAN TRASH ON(BackendHost:BackendHeartBeatPort);` to actively clean up temporary files. **all trash files** and expired snapshot files will be cleaned up, **This will affect the operation of restoring data from the trash bin**.

  If you do not manually execute `ADMIN CLEAN TRASH`, the system will still automatically execute the cleanup within a few minutes to tens of minutes.There are two situations as follows:

  - If the disk usage does not reach 90% of the **Flood Stage**, expired trash files and expired snapshot files will be cleaned up. At this time, some recent files will be retained without affecting the recovery of data.
  - If the disk usage has reached 90% of the **Flood Stage**, **all trash files** and expired snapshot files will be cleaned up, **This will affect the operation of restoring data from the trash bin**.

  The time interval for automatic execution can be changed by `max_garbage_sweep_interval` and `min_garbage_sweep_interval` in the configuration items.

  When the recovery fails due to lack of trash files, the following results may be returned:

  ```
  {"status": "Fail","msg": "can find tablet path in trash"}
  ```

- Delete data file (dangerous!!!)

When none of the above operations can free up capacity, you need to delete data files to free up space. The data file is in the `data/` directory of the specified data directory. To delete a tablet, you must first ensure that at least one replica of the tablet is normal, otherwise **deleting the only replica will result in data loss**.

Suppose we want to delete the tablet with id 12345:

- Find the directory corresponding to Tablet, usually under `data/shard_id/tablet_id/`. like:

  ```
  data/0/12345/
  ```

- Record the tablet id and schema hash. The schema hash is the name of the next-level directory of the previous step. The following is 352781111:

  ```
  data/0/12345/352781111
  ```

- Delete the data directory:

  ```
  rm -rf data/0/12345/
  ```

- Delete tablet metadata refer to Tablet metadata management tool

  ```
  ./lib/meta_tool --operation=delete_header --root_path=/path/to/root_path --tablet_id=12345 --schema_hash= 352781111
  ```

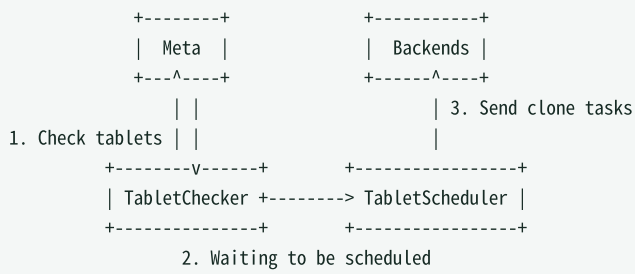
Data Replica Management

Beginning with version 0.9.0, Doris introduced an optimized replica management strategy and supported a richer replica status viewing tool. This document focuses on Doris data replica balancing, repair scheduling strategies, and replica management operations and maintenance methods. Help users to more easily master and manage the replica status in the cluster.

> Repairing and balancing copies of tables with Colocation attributes can be referred to HERE

Noun Interpretation

1. Tablet: The logical fragmentation of a Doris table, where a table has multiple fragments.
2. Replica: A sliced copy, defaulting to three copies of a slice.
3. Healthy Replica: A healthy copy that survives at Backend and has a complete version.
4. Tablet Checker (TC): A resident background thread that scans all Tablets regularly, checks the status of these Tablets, and decides whether to send them to Tablet Scheduler based on the results.
5. Tablet Scheduler (TS): A resident background thread that handles Tablets sent by Tablet Checker that need to be repaired. At the same time, cluster replica balancing will be carried out.
6. Tablet SchedCtx (TSC): is a tablet encapsulation. When TC chooses a tablet, it encapsulates it as a TSC and sends it to TS.
7. Storage Medium: Storage medium. Doris supports specifying different storage media for partition granularity, including SSD and HDD. The replica scheduling strategy is also scheduled for different storage media.



The figure above is a simplified workflow.

Duplicate status

Multiple copies of a Tablet may cause state inconsistencies due to certain circumstances. Doris will attempt to automatically fix the inconsistent copies of these states so that the cluster can recover from the wrong state as soon as possible.

The health status of a Replica is as follows:

1. BAD

   That is, the copy is damaged. Includes, but is not limited to, the irrecoverable damaged status of copies caused by disk failures, BUGs, etc.

2. VERSION_MISSING

   Version missing. Each batch of imports in Doris corresponds to a data version. A copy of the data consists of several consecutive versions. However, due to import errors, delays and other reasons, the data version of some copies may be incomplete.

3. HEALTHY

   Health copy. That is, a copy of the normal data, and the BE node where the copy is located is in a normal state (heartbeat is normal and not in the offline process).

The health status of a Tablet is determined by the status of all its copies. There are the following categories:

1. REPLICA_MISSING

   The copy is missing. That is, the number of surviving copies is less than the expected number of copies.

2. VERSION_INCOMPLETE

   The number of surviving copies is greater than or equal to the number of expected copies, but the number of healthy copies is less than the number of expected copies.

3. REPLICA_RELOCATING

   Have a full number of live copies of the replication num version, but the BE nodes where some copies are located are in unavailable state (such as decommission)

4. REPLICA_MISSING_IN_CLUSTER

   When using multi-cluster, the number of healthy replicas is greater than or equal to the expected number of replicas, but the number of replicas in the corresponding cluster is less than the expected number of replicas.

5. REDUNDANT

   Duplicate redundancy. Healthy replicas are in the corresponding cluster, but the number of replicas is larger than the expected number. Or there’s a spare copy of unavailable.

6. FORCE_REDUNDANT

   This is a special state. It only occurs when the number of existed replicas is greater than or equal to the number of available nodes, and the number of available nodes is greater than or equal to the number of expected replicas, and when the number of alive replicas is less than the number of expected replicas. In this case, you need to delete a copy first to ensure that there are available nodes for creating a new copy.

7. COLOCATE_MISMATCH

Fragmentation status of tables for Collocation attributes. Represents that the distribution of fragmented copies is inconsistent with the specified distribution of Colocation Group.

8. COLOCATE_REDUNDANT

   Fragmentation status of tables for Collocation attributes. Represents the fragmented copy redundancy of the Colocation table.

9. HEALTHY

   Healthy fragmentation, that is, conditions [1-5] are not satisfied.

Replica Repair

As a resident background process, Tablet Checker regularly checks the status of all fragments. For unhealthy fragmentation, it will be sent to Tablet Scheduler for scheduling and repair. The actual operation of repair is accomplished by clone task on BE. FE is only responsible for generating these clone tasks.

> Note 1: The main idea of replica repair is to make the number of fragmented replicas reach the desired value by creating or completing them first. Then delete the redundant copy.
>
> Note 2: A clone task is to complete the process of copying specified data from a specified remote end to a specified destination.

For different states, we adopt different repair methods:

1. REPLICA_MISSING/REPLICA_RELOCATING

   Select a low-load, available BE node as the destination. Choose a healthy copy as the source. Clone tasks copy a complete copy from the source to the destination. For replica completion, we will directly select an available BE node, regardless of the storage medium.

2. VERSION_INCOMPLETE

   Select a relatively complete copy as the destination. Choose a healthy copy as the source. The clone task attempts to copy the missing version from the source to the destination.

3. REPLICA_MISSING_IN_CLUSTER

   This state processing method is the same as REPLICA_MISSING.

4. REDUNDANT

   Usually, after repair, there will be redundant copies in fragmentation. We select a redundant copy to delete it. The selection of redundant copies follows the following priorities:

   1. The BE where the copy is located has been offline.
   2. The copy is damaged
   3. The copy is lost in BE or offline
   4. The replica is in the CLONE state (which is an intermediate state during clone task execution)
   5. The copy has version missing

6. The cluster where the copy is located is incorrect
7. The BE node where the replica is located has a high load

5. FORCE_REDUNDANT

   Unlike REDUNDANT, because at this point Tablet has a copy missing, because there are no additional available nodes for creating new copies. So at this point, a copy must be deleted to free up a available node for creating a new copy. The order of deleting copies is the same as REDUNDANT.

6. COLOCATE_MISMATCH

   Select one of the replica distribution BE nodes specified in Colocation Group as the destination node for replica completion.

7. COLOCATE_REDUNDANT

   Delete a copy on a BE node that is distributed by a copy specified in a non-Colocation Group.

   Doris does not deploy a copy of the same Tablet on a different BE of the same host when selecting a replica node. It ensures that even if all BEs on the same host are deactivated, all copies will not be lost.

Scheduling priority

Waiting for the scheduled fragments in Tablet Scheduler gives different priorities depending on the status. High priority fragments will be scheduled first. There are currently several priorities.

1. VERY_HIGH

   - REDUNDANT. For slices with duplicate redundancy, we give priority to them. Logically, duplicate redundancy is the least urgent, but because it is the fastest to handle and can quickly release resources (such as disk space, etc.), we give priority to it.
   - FORCE_REDUNDANT. Ditto.

2. HIGH

   - REPLICA_MISSING and most copies are missing (for example, 2 copies are missing in 3 copies)
   - VERSION_INCOMPLETE and most copies are missing
   - COLOCATE_MISMATCH We hope that the fragmentation related to the Collocation table can be repaired as soon as possible.
   - COLOCATE_REDUNDANT

3. NORMAL

   - REPLICA_MISSING, but most survive (for example, three copies lost one)
   - VERSION_INCOMPLETE, but most copies are complete
   - REPLICA_RELOCATING and relocate is required for most replicas (e.g. 3 replicas with 2 replicas)

4. LOW

   - REPLICA_MISSING_IN_CLUSTER
   - REPLICA_RELOCATING most copies stable

Manual priority

The system will automatically determine the scheduling priority. Sometimes, however, users want the fragmentation of some tables or partitions to be repaired faster. So we provide a command that the user can specify that a slice of a table or partition is repaired first:

```
ADMIN REPAIR TABLE tbl [PARTITION (p1, p2, ...)];
```

This command tells TC to give VERY HIGH priority to the problematic tables or partitions that need to be repaired first when scanning Tablets.

> Note: This command is only a hint, which does not guarantee that the repair will be successful, and the priority will change with the scheduling of TS. And when Master FE switches or restarts, this information will be lost.

Priority can be cancelled by the following commands:

```
ADMIN CANCEL REPAIR TABLE tbl [PARTITION (p1, p2, ...)];
```

Priority scheduling

Priority ensures that severely damaged fragments can be repaired first, and improves system availability. But if the high priority repair task fails all the time, the low priority task will never be scheduled. Therefore, we will dynamically adjust the priority of tasks according to the running status of tasks, so as to ensure that all tasks have the opportunity to be scheduled.

- If the scheduling fails for five consecutive times (e.g., no resources can be obtained, no suitable source or destination can be found, etc.), the priority will be lowered.
- If not scheduled for 30 minutes, priority will be raised.
- The priority of the same tablet task is adjusted at least five minutes apart.

At the same time, in order to ensure the weight of the initial priority, we stipulate that the initial priority is VERY HIGH, and the lowest is lowered to NORMAL. When the initial priority is LOW, it is raised to HIGH at most. The priority adjustment here also adjusts the priority set manually by the user.

Replicas Balance

Doris automatically balances replicas within the cluster. Currently supports two rebalance strategies, BeLoad and Partition. BeLoad rebalance will consider about the disk usage and replica count for each BE. Partition rebalance just aim at replica count for each partition, this helps to avoid hot spots. If you want high read/write performance, you may need this. Note that Partition rebalance do not consider about the disk usage, pay more attention to it when you are using Partition rebalance. The strategy selection config is not mutable at runtime.

BeLoad

The main idea of balancing is to create a replica of some fragments on low-load nodes, and then delete the replicas of these fragments on high-load nodes. At the same time, because of the existence of different storage media, there may or may not exist one or two storage media on different BE nodes in the same cluster. We

require that fragments of storage medium A be stored in storage medium A as far as possible after equalization. So we divide the BE nodes of the cluster according to the storage medium. Then load balancing scheduling is carried out for different BE node sets of storage media.

Similarly, replica balancing ensures that a copy of the same table will not be deployed on the BE of the same host.

BE Node Load

We use Cluster LoadStatistics (CLS) to represent the load balancing of each backend in a cluster. Tablet Scheduler triggers cluster equilibrium based on this statistic. We currently calculate a load Score for each BE as the BE load score by using **disk usage** and **number of copies**. The higher the score, the heavier the load on the BE.

Disk usage and number of copies have a weight factor, which is **capacityCoefficient** and **replicaNumCoefficient**, respectively. The sum of them is **constant to 1**. If a valid `backend_load_capacity_coeficient` parameter is configured (value range `0.0~1.0`), then `capacityCoefficient = backend_load_capacity_coeficient`. Otherwise, `capacityCoefficient` will dynamically adjust according to actual disk utilization. When the overall disk utilization of a BE is below 50%, the `capacityCoefficient` value is 0.5, and if the disk utilization is above 75% (configurable through the FE configuration item `capacity_used_percent_high_water`), the value is 1. If the utilization rate is between 50% and 75%, the weight coefficient increases smoothly. The formula is as follows:

```
capacityCoefficient = 2 * Disk Utilization - 0.5
```

The weight coefficient ensures that when disk utilization is too high, the backend load score will be higher to ensure that the BE load is reduced as soon as possible.

Tablet Scheduler updates CLS every 20 seconds.

Partition

The main idea of `partition rebalancing` is to decrease the skew of partitions. The skew of the partition is defined as the difference between the maximum replica count of the partition over all bes and the minimum replica count over all bes.

So we only consider about the replica count, do not consider replica size(disk usage). To fewer moves, we use TwoDimensionalGreedyAlgo which two dims are cluster & partition. It prefers a move that reduce the skew of the cluster when we want to rebalance a max skew partition.

Skew Info

The skew info is represented by `ClusterBalanceInfo`. `partitionInfoBySkew` is a multimap which key is the partition's skew, so we can get max skew partitions simply. `beByTotalReplicaCount` is a multimap which key is the total replica count of the backend.

`ClusterBalanceInfo` is in CLS, updated every 20 seconds.

When get more than one max skew partitions, we random select one partition to calculate the move.

Equilibrium strategy

Tablet Scheduler uses Load Balancer to select a certain number of healthy fragments as candidate fragments for balance in each round of scheduling. In the next scheduling, balanced scheduling will be attempted based on these candidate fragments.

Resource control

Both replica repair and balancing are accomplished by replica copies between BEs. If the same BE performs too many tasks at the same time, it will bring a lot of IO pressure. Therefore, Doris controls the number of tasks that can be performed on each node during scheduling. The smallest resource control unit is the disk (that is, a data path specified in be.conf). By default, we configure two slots per disk for replica repair. A clone task occupies one slot at the source and one slot at the destination. If the number of slots is zero, no more tasks will be assigned to this disk. The number of slots can be configured by FE's `schedule_slot_num_per_hdd_path` or `schedule_slot_num_per_ssd_path` parameter.

In addition, by default, we provide two separate slots per disk for balancing tasks. The purpose is to prevent high-load nodes from losing space by balancing because slots are occupied by repair tasks.

Tablet State View

Tablet state view mainly looks at the state of the tablet, as well as the state of the tablet repair and balancing tasks. Most of these states **exist only in** Master FE nodes. Therefore, the following commands need to be executed directly to Master FE.

Tablet state

1. Global state checking

   Through `SHOW PROC'/cluster_health/tablet_health';`commands can view the replica status of the entire cluster.

```
+-------+--------------------------------+-----------+------------+-------------------+----------------------+-------------
---------+--------------+----------------------------+------------------------+------------------+----------------------+-
--------------------+----------------------+------------------+-----------------------------+-----------------+-----------
--+------------+
| DbId  | DbName                         | TabletNum | HealthyNum | ReplicaMissingNum | VersionIncompleteNum |
ReplicaRelocatingNum | RedundantNum | ReplicaMissingInClusterNum | ReplicaMissingForTagNum | ForceRedundantNum |
ColocateMismatchNum | ColocateRedundantNum | NeedFurtherRepairNum | UnrecoverableNum | ReplicaCompactionTooSlowNum |
InconsistentNum | OversizeNum | CloningNum |
+-------+--------------------------------+-----------+------------+-------------------+----------------------+-------------
---------+--------------+----------------------------+------------------------+------------------+----------------------+-
--------------------+----------------------+------------------+-----------------------------+-----------------+-----------
--+------------+
| 10005 | default_cluster:doris_audit_db | 84        | 84         | 0                 | 0                    | 0
| 0            | 0                          | 0                      | 0                | 0                    | 0
| 0                    | 0               | 0                        | 0              | 0           | 0          |
| 13402 | default_cluster:ssb1           | 709       | 708        | 1                 | 0                    | 0
| 0            | 0                          | 0                      | 0                | 0                    | 0
| 0                    | 0               | 0                        | 0              | 0           | 0          |
| 10108 | default_cluster:tpch1          | 278       | 278        | 0                 | 0                    | 0
| 0            | 0                          | 0                      | 0                | 0                    | 0
| 0                    | 0               | 0                        | 0              | 0           | 0          |
| Total | 3                              | 1071      | 1070       | 1                 | 0                    | 0
```

```
| 0           | 0                       | 0                     | 0               | 0                 | 0
| 0                   | 0               | 0                       | 0             | 0           | 0          |
+-------+---------------------------------+-----------+------------+------------------+----------------------+-------------
---------+--------------+---------------------------+------------------------+------------------+--------------------+-
--------------------+----------------------+------------------+-----------------------------+-----------------+-----------
--+------------+
```

The `HealthyNum` column shows how many Tablets are in a healthy state in the corresponding database. `ReplicaCompactionTooSlowNum` column shows how many Tablets are in a too many versions state in the corresponding database, `InconsistentNum` column shows how many Tablets are in an inconsistent replica state in the corresponding database. The last `Total` line counts the entire cluster. Normally `TabletNum` and `HealthyNum` should be equal. If it's not equal, you can further see which Tablets are there. As shown in the figure above, one table in the ssb1 database is not healthy, you can use the following command to see which one is.

```
SHOW PROC '/cluster_health/tablet_health/13402';
```

Among them `13402` is the corresponding DbId.

```
+----------------------+-------------------------+-------------------------+-----------------+-------------------------
-------+-----------------------------+----------------------+-------------------------+--------------------------+--------
-----------------+----------------------+---------------------------------+---------------------+-----------------+
| ReplicaMissingTablets | VersionIncompleteTablets | ReplicaRelocatingTablets | RedundantTablets |
ReplicaMissingInClusterTablets | ReplicaMissingForTagTablets | ForceRedundantTablets | ColocateMismatchTablets |
ColocateRedundantTablets | NeedFurtherRepairTablets | UnrecoverableTablets | ReplicaCompactionTooSlowTablets |
InconsistentTablets | OversizeTablets |
+----------------------+-------------------------+-------------------------+-----------------+-------------------------
-------+-----------------------------+----------------------+-------------------------+--------------------------+--------
-----------------+----------------------+---------------------------------+---------------------+-----------------+
| 14679                |                         |                         |                 |
|                            |                      |                        |                       |
|                      |                               |                   |                 |
+----------------------+-------------------------+-------------------------+-----------------+-------------------------
-------+-----------------------------+----------------------+-------------------------+--------------------------+--------
-----------------+----------------------+---------------------------------+---------------------+-----------------+
```

The figure above shows the specific unhealthy Tablet ID (14679). Later we'll show you how to view the status of each copy of a specific Tablet.

2. Table (partition) level status checking

   Users can view the status of a copy of a specified table or partition through the following commands and filter the status through a WHERE statement. If you look at table tbl1, the state on partitions P1 and P2 is a copy of OK:

   ```
   SHOW REPLICA STATUS FROM tbl1 PARTITION (p1, p2) WHERE STATUS = "OK";
   ```

   ```
   +----------+-----------+-----------+---------+-------------------+--------------------+------------------+------------+----
   --------+-------+--------+--------+
   | TabletId | ReplicaId | BackendId | Version | LastFailedVersion | LastSuccessVersion | CommittedVersion | SchemaHash |
   VersionNum | IsBad | State  | Status |
   +----------+-----------+-----------+---------+-------------------+--------------------+------------------+------------+----
   --------+-------+--------+--------+
   | 29502429 | 29502432  | 10006     | 2       | -1                | 2                  | 1                | -1         | 2
   | false | NORMAL | OK     |
   | 29502429 | 36885996  | 10002     | 2       | -1                | -1                 | 1                | -1         | 2
   | false | NORMAL | OK     |
   ```

```
| 29502429 | 48100551  | 10007     | 2       | -1               | -1                | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502433 | 29502434  | 10001     | 2       | -1               | 2                 | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502433 | 44900737  | 10004     | 2       | -1               | -1                | 1               | -1         | 2
| false | NORMAL | OK     |
| 29502433 | 48369135  | 10006     | 2       | -1               | -1                | 1               | -1         | 2
| false | NORMAL | OK     |
+----------+-----------+-----------+---------+------------------+-------------------+-----------------+------------+----
--------+-------+--------+--------+
```

The status of all copies is shown here. Where `IsBad` is listed as `true`, the copy is damaged. The `Status` column displays other states. Specific status description, you can see help through `HELP SHOW REPLICA STATUS`.

`The SHOW REPLICA STATUS`command is mainly used to view the health status of copies. Users can also view additional information about copies of a specified table by using the following commands:

```
SHOW TABLETS FROM tbl1;
```

```
+----------+-----------+-----------+------------+---------+-------------+------------------+----------------------+------
------------+----------------------+---------------+----------+----------+--------+-------------------------+--------------
+----------------------+--------------+----------------------+----------------------+----------------------+
| TabletId | ReplicaId | BackendId | SchemaHash | Version | VersionHash | LstSuccessVersion | LstSuccessVersionHash |
LstFailedVersion | LstFailedVersionHash | LstFailedTime | DataSize | RowCount | State  | LstConsistencyCheckTime |
CheckVersion |     CheckVersionHash | VersionCount | PathHash             | MetaUrl              | CompactionStatus     |
+----------+-----------+-----------+------------+---------+-------------+------------------+----------------------+------
------------+----------------------+---------------+----------+----------+--------+-------------------------+--------------
+----------------------+--------------+----------------------+----------------------+----------------------+
| 29502429 | 29502432  | 10006     | 1421156361 | 2       | 0           | 2                | 0                    | -1
| 0                    | N/A           | 784      | 0        | NORMAL | N/A                    | -1           |     -1
| 2            | -5822326203532286804 | url                  | url                  |
| 29502429 | 36885996  | 10002     | 1421156361 | 2       | 0           | -1               | 0                    | -1
| 0                    | N/A           | 784      | 0        | NORMAL | N/A                    | -1           |     -1
| 2            | -1441285706148429853 | url                  | url                  |
| 29502429 | 48100551  | 10007     | 1421156361 | 2       | 0           | -1               | 0                    | -1
| 0                    | N/A           | 784      | 0        | NORMAL | N/A                    | -1           |     -1
| 2            | -4784691547051455525 | url                  | url                  |
+----------+-----------+-----------+------------+---------+-------------+------------------+----------------------+------
------------+----------------------+---------------+----------+----------+--------+-------------------------+--------------
+----------------------+--------------+----------------------+----------------------+----------------------+
```

The figure above shows some additional information, including copy size, number of rows, number of versions, where the data path is located.

> Note: The contents of the `State` column shown here do not represent the health status of the replica, but the status of the replica under certain tasks, such as CLONE, SCHEMA CHANGE, ROLLUP, etc.

In addition, users can check the distribution of replicas in a specified table or partition by following commands.

```
SHOW REPLICA DISTRIBUTION FROM tbl1;
```

```
+-----------+------------+-------+---------+
| BackendId | ReplicaNum | Graph | Percent |
+-----------+------------+-------+---------+
| 10000     | 7          |       | 7.29 %  |
| 10001     | 9          |       | 9.38 %  |
| 10002     | 7          |       | 7.29 %  |
| 10003     | 7          |       | 7.29 %  |
| 10004     | 9          |       | 9.38 %  |
| 10005     | 11         | >     | 11.46 % |
| 10006     | 18         | >     | 18.75 % |
| 10007     | 15         | >     | 15.62 % |
| 10008     | 13         | >     | 13.54 % |
+-----------+------------+-------+---------+
```

Here we show the number and percentage of replicas of table tbl1 on each BE node, as well as a simple graphical display.

3. Tablet level status checking

   When we want to locate a specific Tablet, we can use the following command to view the status of a specific Tablet. For example, check the tablet with ID 2950253:

   ```
   SHOW TABLET 29502553;
   ```

```
+------------------------+-----------+---------------+-----------+----------+----------+-------------+----------+--------+-
-------------------------------------------------------------------------------+
| DbName                 | TableName | PartitionName | IndexName | DbId     | TableId  | PartitionId | IndexId  | IsSync |
DetailCmd                                                                      |
+------------------------+-----------+---------------+-----------+----------+----------+-------------+----------+--------+-
-------------------------------------------------------------------------------+
| default_cluster:test   | test      | test          | test      | 29502391 | 29502428 | 29502427    | 29502428 | true   |
SHOW PROC '/dbs/29502391/29502428/partitions/29502427/29502428/29502553'; |
+------------------------+-----------+---------------+-----------+----------+----------+-------------+----------+--------+-
-------------------------------------------------------------------------------+
```

   The figure above shows the database, tables, partitions, roll-up tables and other information corresponding to this tablet. The user can copy the command in the `DetailCmd` command to continue executing:

   ```
   Show Proc'/DBS/29502391/29502428/Partitions/29502427/29502428/29502553;
   ```

```
+-----------+-----------+---------+-------------+-------------------+-----------------------+------------------+-----------
-----------+---------------+------------+----------+----------+--------+-------------+----------------------+
| ReplicaId | BackendId | Version | VersionHash | LstSuccessVersion | LstSuccessVersionHash | LstFailedVersion |
LstFailedVersionHash | LstFailedTime | SchemaHash | DataSize | RowCount | State  | IsBad | VersionCount | PathHash
|
+-----------+-----------+---------+-------------+-------------------+-----------------------+------------------+-----------
-----------+---------------+------------+----------+----------+--------+-------------+----------------------+
| 43734060  | 10004     | 2       | 0           | -1                | 0                     | -1               | 0
| N/A           | -1         | 784      | 0        | NORMAL | false | 2            | -8566523878520798656 |
| 29502555  | 10002     | 2       | 0           | 2                 | 0                     | -1               | 0
| N/A           | -1         | 784      | 0        | NORMAL | false | 2            | 1885826196444191611  |
| 39279319  | 10007     | 2       | 0           | -1                | 0                     | -1               | 0
| N/A           | -1         | 784      | 0        | NORMAL | false | 2            | 1656508631294397870  |
+-----------+-----------+---------+-------------+-------------------+-----------------------+------------------+-----------
-----------+---------------+------------+----------+----------+--------+-------------+----------------------+
```

The figure above shows all replicas of the corresponding Tablet. The content shown here is the same as `SHOW TABLETS FROM tbl1;`. But here you can clearly see the status of all copies of a specific Tablet.

Duplicate Scheduling Task

1. View tasks waiting to be scheduled

```
SHOW PROC '/cluster_balance/pending_tablets';
```

```
+----------+--------+-----------------+---------+----------+----------+-------+---------+--------+----------+---------+----
----------------+---------------------+---------------------+----------+------+-------------+---------------+-------------
--------+------------+---------------------+--------+---------------------+------------------------------+
| TabletId | Type   | Status          | State   | OrigPrio | DynmPrio | SrcBe | SrcPath | DestBe | DestPath | Timeout |
Create              | LstSched            | LstVisit            | Finished | Rate | FailedSched | FailedRunning |
LstAdjPrio          | VisibleVer | VisibleVerHash      | CmtVer | CmtVerHash          | ErrMsg                       |
+----------+--------+-----------------+---------+----------+----------+-------+---------+--------+----------+---------+----
----------------+---------------------+---------------------+----------+------+-------------+---------------+-------------
--------+------------+---------------------+--------+---------------------+------------------------------+
| 4203036  | REPAIR | REPLICA_MISSING | PENDING | HIGH     | LOW      | -1    | -1      | -1     | -1       | 0       |
2019-02-21 15:00:20 | 2019-02-24 11:18:41 | 2019-02-24 11:18:41 | N/A      | N/A  | 2           | 0             | 2019-02-
21 15:00:43 | 1          | 0                   | 2      | 0                   | unable to find source replica |
+----------+--------+-----------------+---------+----------+----------+-------+---------+--------+----------+---------+----
----------------+---------------------+---------------------+----------+------+-------------+---------------+-------------
--------+------------+---------------------+--------+---------------------+------------------------------+
```

The specific meanings of each column are as follows:

- TabletId: The ID of the Tablet waiting to be scheduled. A scheduling task is for only one Tablet
- Type: Task type, which can be REPAIR (repair) or BALANCE (balance)
- Status: The current status of the Tablet, such as REPLICA_MISSING (copy missing)
- State: The status of the scheduling task may be PENDING/RUNNING/FINISHED/CANCELLED/TIMEOUT/UNEXPECTED
- OrigPrio: Initial Priority
- DynmPrio: Current dynamically adjusted priority
- SrcBe: ID of the BE node at the source end
- SrcPath: hash value of the path of the BE node at the source end
- DestBe: ID of destination BE node
- DestPath: hash value of the path of the destination BE node
- Timeout: When the task is scheduled successfully, the timeout time of the task is displayed here in units of seconds.
- Create: The time when the task was created
- LstSched: The last time a task was scheduled
- LstVisit: The last time a task was accessed. Here “accessed” refers to the processing time points associated with the task, including scheduling, task execution reporting, and so on.
- Finished: Task End Time
- Rate: Clone Task Data Copy Rate
- Failed Sched: Number of Task Scheduling Failures
- Failed Running: Number of task execution failures
- LstAdjPrio: Time of last priority adjustment
- CmtVer/CmtVerHash/VisibleVer/VisibleVerHash: version information for clone tasks
- ErrMsg: Error messages that occur when tasks are scheduled and run

2. View running tasks

   ```
   SHOW PROC '/cluster_balance/running_tablets';
   ```

   The columns in the result have the same meaning as `pending_tablets`.

3. View completed tasks

   ```
   SHOW PROC '/cluster_balance/history_tablets';
   ```

   By default, we reserve only the last 1,000 completed tasks. The columns in the result have the same meaning as `pending_tablets`. If `State` is listed as `FINISHED`, the task is normally completed. For others, you can see the specific reason based on the error information in the `ErrMsg` column.

Viewing Cluster Load and Scheduling Resources

1. Cluster load

   You can view the current load of the cluster by following commands:

   ```
   SHOW PROC '/cluster_balance/cluster_load_stat/location_default';
   ```

   First of all, we can see the division of different storage media:

   ```
   +---------------+
   | StorageMedium |
   +---------------+
   | HDD           |
   | SSD           |
   +---------------+
   ```

   Click on a storage medium to see the equilibrium state of the BE node that contains the storage medium:

   ```
   SHOW PROC '/cluster_balance/cluster_load_stat/location_default/HDD';
   ```

   ```
   +----------+-----------------+-----------+---------------+----------------+-------------+------------+----------+----------
   -+--------------------+-------+
   | BeId     | Cluster         | Available | UsedCapacity  | Capacity       | UsedPercent | ReplicaNum | CapCoeff | ReplCoeff
   | Score              | Class |
   +----------+-----------------+-----------+---------------+----------------+-------------+------------+----------+----------
   -+--------------------+-------+
   | 10003    | default_cluster | true      | 3477875259079 | 19377459077121 | 17.948      | 493477     | 0.5      | 0.5
   | 0.9284678149967587 | MID   |
   | 10002    | default_cluster | true      | 3607326225443 | 19377459077121 | 18.616      | 496928     | 0.5      | 0.5
   | 0.948660871419998  | MID   |
   | 10005    | default_cluster | true      | 3523518578241 | 19377459077121 | 18.184      | 545331     | 0.5      | 0.5
   | 0.9843539990641831 | MID   |
   | 10001    | default_cluster | true      | 3535547090016 | 19377459077121 | 18.246      | 558067     | 0.5      | 0.5
   | 0.9981869446537612 | MID   |
   | 10006    | default_cluster | true      | 3636050364835 | 19377459077121 | 18.764      | 547543     | 0.5      | 0.5
   | 1.0011489897614072 | MID   |
   | 10004    | default_cluster | true      | 3506558163744 | 15501967261697 | 22.620      | 468957     | 0.5      | 0.5
   | 1.0228319835582569 | MID   |
   | 10007    | default_cluster | true      | 4036460478905 | 19377459077121 | 20.831      | 551645     | 0.5      | 0.5
   | 1.057279369420761  | MID   |
   | 10000    | default_cluster | true      | 4369719923760 | 19377459077121 | 22.551      | 547175     | 0.5      | 0.5
   ```

```
| 1.0964036415787461 | MID   |
+----------+----------------+-----------+--------------+---------------+-------------+------------+----------+----------
-+--------------------+-------+
```

Some of these columns have the following meanings:

- Available: True means that BE heartbeat is normal and not offline.
- UsedCapacity: Bytes, the size of disk space used on BE
- Capacity: Bytes, the total disk space size on BE
- UsedPercent: Percentage, disk space utilization on BE
- ReplicaNum: Number of copies on BE
- CapCoeff/ReplCoeff: Weight Coefficient of Disk Space and Copy Number
- Score: Load score. The higher the score, the heavier the load.
- Class: Classified by load, LOW/MID/HIGH. Balanced scheduling moves copies from high-load nodes to low-load nodes

Users can further view the utilization of each path on a BE, such as the BE with ID 10001:

```
SHOW PROC '/cluster_balance/cluster_load_stat/location_default/HDD/10001';
```

```
+------------------+------------------+---------------+---------------+---------+--------+----------------------+
| RootPath         | DataUsedCapacity | AvailCapacity | TotalCapacity | UsedPct | State  | PathHash             |
+------------------+------------------+---------------+---------------+---------+--------+----------------------+
| /home/disk4/palo | 498.757 GB       | 3.033 TB      | 3.525 TB      | 13.94 % | ONLINE | 4883406271918338267  |
| /home/disk3/palo | 704.200 GB       | 2.832 TB      | 3.525 TB      | 19.65 % | ONLINE | -5467083960906519443 |
| /home/disk1/palo | 512.833 GB       | 3.007 TB      | 3.525 TB      | 14.69 % | ONLINE | -7733211489989964053 |
| /home/disk2/palo | 881.955 GB       | 2.656 TB      | 3.525 TB      | 24.65 % | ONLINE | 4870995507205544622  |
| /home/disk5/palo | 694.992 GB       | 2.842 TB      | 3.525 TB      | 19.36 % | ONLINE | 1916696897889786739  |
+------------------+------------------+---------------+---------------+---------+--------+----------------------+
```

The disk usage of each data path on the specified BE is shown here.

2. Scheduling resources

Users can view the current slot usage of each node through the following commands:

```
SHOW PROC '/cluster_balance/working_slots';
```

```
+----------+----------------------+------------+------------+-------------+----------------------+
| BeId     | PathHash             | AvailSlots | TotalSlots | BalanceSlot | AvgRate              |
+----------+----------------------+------------+------------+-------------+----------------------+
| 10000    | 8110346074333016794  | 2          | 2          | 2           | 2.459007474009069E7  |
| 10000    | -5617618290584731137 | 2          | 2          | 2           | 2.4730105014001578E7 |
| 10001    | 4883406271918338267  | 2          | 2          | 2           | 1.6711402709780257E7 |
| 10001    | -5467083960906519443 | 2          | 2          | 2           | 2.7540126380326536E7 |
| 10002    | 9137404661108133814  | 2          | 2          | 2           | 2.417217089806745E7  |
| 10002    | 1885826196444191611  | 2          | 2          | 2           | 1.6327378456676323E7 |
+----------+----------------------+------------+------------+-------------+----------------------+
```

In this paper, data path is used as granularity to show the current use of slot. Among them, `AvgRate'is the copy rate of clone task in bytes/seconds on the path of historical statistics.

3. Priority repair view

The following command allows you to view the priority repaired tables or partitions set by the `ADMIN REPAIR TABLE'command.

```
SHOW PROC '/cluster_balance/priority_repair';
```

Among them, `Remaining TimeMs'indicates that these priority fixes will be automatically removed from the priority fix queue after this time. In order to prevent resources from being occupied due to the failure of priority repair.

Scheduler Statistical Status View

We have collected some statistics of Tablet Checker and Tablet Scheduler during their operation, which can be viewed through the following commands:

```
SHOW PROC '/cluster_balance/sched_stat';
```

```
+---------------------------------------------------+-------------+
| Item                                              | Value       |
+---------------------------------------------------+-------------+
| num of tablet check round                         | 12041       |
| cost of tablet check(ms)                          | 7162342     |
| num of tablet checked in tablet checker           | 18793506362 |
| num of unhealthy tablet checked in tablet checker | 7043900     |
| num of tablet being added to tablet scheduler     | 1153        |
| num of tablet schedule round                      | 49538       |
| cost of tablet schedule(ms)                       | 49822       |
| num of tablet being scheduled                     | 4356200     |
| num of tablet being scheduled succeeded           | 320         |
| num of tablet being scheduled failed              | 4355594     |
| num of tablet being scheduled discard             | 286         |
| num of tablet priority upgraded                   | 0           |
| num of tablet priority downgraded                 | 1096        |
| num of clone task                                 | 230         |
| num of clone task succeeded                       | 228         |
| num of clone task failed                          | 2           |
| num of clone task timeout                         | 2           |
| num of replica missing error                      | 4354857     |
| num of replica version missing error              | 967         |
| num of replica relocating                         | 0           |
| num of replica redundant error                    | 90          |
| num of replica missing in cluster error           | 0           |
| num of balance scheduled                          | 0           |
+---------------------------------------------------+-------------+
```

The meanings of each line are as follows:

- num of tablet check round: Number of Tablet Checker inspections
- cost of tablet check(ms): Total time consumed by Tablet Checker inspections (milliseconds)
- num of tablet checked in tablet checker: Number of tablets checked by the Tablet Checker
- num of unhealthy tablet checked in tablet checker: Number of unhealthy tablets checked by the Tablet Checker
- num of tablet being added to tablet scheduler: Number of tablets submitted to the Tablet Scheduler

- num of tablet schedule round: Number of Tablet Scheduler runs
- cost of tablet schedule(ms): Total time consumed by Tablet Scheduler runs (milliseconds)
- num of tablet being scheduled: Total number of tablets scheduled
- num of tablet being scheduled succeeded: Total number of tablets successfully scheduled
- num of tablet being scheduled failed: Total number of tablets that failed scheduling
- num of tablet being scheduled discard: Total number of tablets discarded due to scheduling failures
- num of tablet priority upgraded: Number of tablet priority upgrades
- num of tablet priority downgraded: Number of tablet priority downgrades
- num of clone task: Number of clone tasks generated
- num of clone task succeeded: Number of successful clone tasks
- num of clone task failed: Number of failed clone tasks
- num of clone task timeout: Number of clone tasks that timed out
- num of replica missing error: Number of tablets whose status is checked as missing replicas
- num of replica version missing error: Number of tablets checked with missing version status (this statistic includes num of replica relocating and num of replica missing in cluster error)
- num of replica relocation: Number of replica relocations
- num of replica redundant error: Number of tablets whose checked status is replica redundant
- num of replica missing in cluster error: Number of tablets checked with a status indicating they are missing from the corresponding cluster
- num of balance scheduled: Number of balanced scheduling attempts

> **Note**
>
> The above states are only historical accumulative values. We also print these statistics regularly in the FE logs, where the values in parentheses represent the number of changes in each statistical value since the last printing dependence of the statistical information.

Relevant configuration instructions

Adjustable parameters

The following adjustable parameters are all configurable parameters in fe.conf.

- use_new_tablet_scheduler

    - Description: Whether to enable the new replica scheduling mode. The new replica scheduling method is the replica scheduling method introduced in this document. If turned on, `disable_colocate_join` must be `true`. Because the new scheduling strategy does not support data fragmentation scheduling of co-location tables for the time being.
    - Default value:true
    - Importance: High

- tablet_repair_delay_factor_second

    - Note: For different scheduling priorities, we will delay different time to start repairing. In order to prevent a large number of unnecessary replica repair tasks from occurring in the process of routine restart and upgrade. This parameter is a reference coefficient. For HIGH priority, the delay is the reference coefficient * 1; for NORMAL priority, the delay is the reference coefficient * 2; for LOW priority, the delay is the reference coefficient * 3. That is, the lower the priority, the longer the delay waiting time. If the user wants to repair the copy as soon as possible, this parameter can be reduced appropriately.
    - Default value: 60 seconds
    - Importance: High

- schedule_slot_num_per_path

    - Note: The default number of slots allocated to each disk for replica repair. This number represents the number of replica repair tasks that a disk can run simultaneously. If you want to repair the copy faster, you can adjust this parameter appropriately. The higher the single value, the greater the impact on IO.
    - Default value: 2
    - Importance: High

- balance_load_score_threshold

    - Description: Threshold of Cluster Equilibrium. The default is 0.1, or 10%. When the load core of a BE node is not higher than or less than 10% of the average load core, we think that the node is balanced. If you want to make the cluster load more even, you can adjust this parameter appropriately.
    - Default value: 0.1
    - Importance:

- storage_high_watermark_usage_percent 和 storage_min_left_capacity_bytes

    - Description: These two parameters represent the upper limit of the maximum space utilization of a disk and the lower limit of the minimum space remaining, respectively. When the space utilization of a disk is greater than the upper limit or the remaining space is less than the lower limit, the disk will no longer be used as the destination address for balanced scheduling.
    - Default values: 0.85 and 2097152000 (2GB)
    - Importance:

- disable_balance

    - Description: Control whether to turn off the balancing function. When replicas are in equilibrium, some functions, such as ALTER TABLE, will be banned. Equilibrium can last for a long time.

Therefore, if the user wants to do the prohibited operation as soon as possible. This parameter can be set to true to turn off balanced scheduling.
    - Default value: false
    - Importance:

The following adjustable parameters are all configurable parameters in be.conf.

- clone_worker_count

    - Description: Affects the speed of copy equalization. In the case of low disk pressure, you can speed up replica balancing by adjusting this parameter.
    - Default: 3
    - Importance: Medium #### Unadjustable parameters

The following parameters do not support modification for the time being, just for illustration.

- Tablet Checker scheduling interval

  Tablet Checker schedules checks every 20 seconds.

- Tablet Scheduler scheduling interval

  Tablet Scheduler schedules every five seconds

- Number of Tablet Scheduler Schedules per Batch

  Tablet Scheduler schedules up to 50 tablets at a time.

- Tablet Scheduler Maximum Waiting Schedule and Number of Tasks in Operation

  The maximum number of waiting tasks and running tasks is 2000. When over 2000, Tablet Checker will no longer generate new scheduling tasks to Tablet Scheduler.

- Tablet Scheduler Maximum Balanced Task Number

  The maximum number of balanced tasks is 500. When more than 500, there will be no new balancing tasks.

- Number of slots per disk for balancing tasks

  The number of slots per disk for balancing tasks is 2. This slot is independent of the slot used for replica repair.

- Update interval of cluster equilibrium

  Tablet Scheduler recalculates the load score of the cluster every 20 seconds.

- Minimum and Maximum Timeout for Clone Tasks

  A clone task timeout time range is 3 minutes to 2 hours. The specific timeout is calculated by the size of the tablet. The formula is (tablet size)/ (5MB/s). When a clone task fails three times, the task terminates.

- Dynamic Priority Adjustment Strategy

The minimum priority adjustment interval is 5 minutes. When a tablet schedule fails five times, priority is lowered. When a tablet is not scheduled for 30 minutes, priority is raised.

Relevant issues

- In some cases, the default replica repair and balancing strategy may cause the network to be full (mostly in the case of gigabit network cards and a large number of disks per BE). At this point, some parameters need to be adjusted to reduce the number of simultaneous balancing and repair tasks.
- Current balancing strategies for copies of Colocate Table do not guarantee that copies of the same Tablet will not be distributed on the BE of the same host. However, the repair strategy of the copy of Colocate Table detects this distribution error and corrects it. However, it may occur that after correction, the balancing strategy regards the replicas as unbalanced and rebalances them. As a result, the Colocate Group cannot achieve stability because of the continuous alternation between the two states. In view of this situation, we suggest that when using Colocate attribute, we try to ensure that the cluster is isomorphic, so as to reduce the probability that replicas are distributed on the same host.

Best Practices

Control and manage the progress of replica repair and balancing of clusters

In most cases, Doris can automatically perform replica repair and cluster balancing by default parameter configuration. However, in some cases, we need to manually intervene to adjust the parameters to achieve some special purposes. Such as prioritizing the repair of a table or partition, disabling cluster balancing to reduce cluster load, prioritizing the repair of non-colocation table data, and so on.

This section describes how to control and manage the progress of replica repair and balancing of the cluster by modifying the parameters.

1. Deleting Corrupt Replicas

   In some cases, Doris may not be able to automatically detect some corrupt replicas, resulting in frequent query or import errors on the corrupt replicas. In this case, we need to delete the corrupted copies manually. This method can be used to: delete a copy with a high version number resulting in a -235 error, delete a corrupted copy of a file, etc.

   First, find the tablet id of the corresponding copy, let's say 10001, and use `show tablet 10001;` and execute the `show proc` statement to see the details of each copy of the corresponding tablet.

   Assuming that the backend id of the copy to be deleted is 20001, the following statement is executed to mark the copy as `bad`.

   ```
   ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10001", "backend_id" = "20001", "status" = "bad");
   ```

   At this point, the `show proc` statement again shows that the `IsBad` column of the corresponding copy has a value of `true`.

   The replica marked as `bad` will no longer participate in imports and queries. The replica repair logic will automatically replenish a new replica at the same time. 2.

2. prioritize repairing a table or partition

   `help admin repair table`; View help. This command attempts to repair the tablet of the specified table or partition as a priority.

3. Stop the balancing task

   The balancing task will take up some network bandwidth and IO resources. If you wish to stop the generation of new balancing tasks, you can do so with the following command.

```
ADMIN SET FRONTEND CONFIG ("disable_balance" = "true");
```

4. Stop all replica scheduling tasks

   Copy scheduling tasks include balancing and repair tasks. These tasks take up some network bandwidth and IO resources. All replica scheduling tasks (excluding those already running, including colocation tables and common tables) can be stopped with the following command.

```
ADMIN SET FRONTEND CONFIG ("disable_tablet_scheduler" = "true");
```

5. Stop the copy scheduling task for all colocation tables.

   The colocation table copy scheduling is run separately and independently from the regular table. In some cases, users may wish to stop the balancing and repair of colocation tables first and use the cluster resources for normal table repair with the following command.

```
ADMIN SET FRONTEND CONFIG ("disable_colocate_balance" = "true");
```

6. Repair replicas using a more conservative strategy

   Doris automatically repairs replicas when it detects missing replicas, BE downtime, etc. However, in order to reduce some errors caused by jitter (e.g., BE being down briefly), Doris delays triggering these tasks.

   - The `tablet_repair_delay_factor_second` parameter. Default 60 seconds. Depending on the priority of the repair task, it will delay triggering the repair task for 60 seconds, 120 seconds, or 180 seconds. This time can be extended so that longer exceptions can be tolerated to avoid triggering unnecessary repair tasks by using the following command.

```
ADMIN SET FRONTEND CONFIG ("tablet_repair_delay_factor_second" = "120");
```

7. use a more conservative strategy to trigger redistribution of colocation groups

   Redistribution of colocation groups may be accompanied by a large number of tablet migrations. `colocate_group_relocate_delay_second` is used to control the redistribution trigger delay. The default is 1800 seconds. If a BE node is likely to be offline for a long time, you can try to increase this parameter to avoid unnecessary redistribution by.

```
ADMIN SET FRONTEND CONFIG ("colocate_group_relocate_delay_second" = "3600");
```

8. Faster Replica Balancing

   Doris' replica balancing logic adds a normal replica first and then deletes the old one for the purpose of replica migration. When deleting the old replica, Doris waits for the completion of the import task that has already started on this replica to avoid the balancing task from affecting the import task. However, this will slow down the execution speed of the balancing logic. In this case, you can make Doris ignore this wait and delete the old replica directly by modifying the following parameters.

   ```
   ADMIN SET FRONTEND CONFIG ("enable_force_drop_redundant_replica" = "true");
   ```

   This operation may cause some import tasks to fail during balancing (requiring a retry), but it will speed up balancing significantly.

Overall, when we need to bring the cluster back to a normal state quickly, consider handling it along the following lines.

1. find the tablet that is causing the highly optimal task to report an error and set the problematic copy to bad.
2. repair some tables with the `admin repair` statement.
3. Stop the replica balancing logic to avoid taking up cluster resources, and then turn it on again after the cluster is restored.
4. Use a more conservative strategy to trigger repair tasks to deal with the avalanche effect caused by frequent BE downtime.
5. Turn off scheduling tasks for colocation tables on-demand and focus cluster resources on repairing other high-optimality data.

Automated Service Startup

Automatic service start

This document describes how to configure the automatic pull-up of the Doris cluster to ensure that services are not pulled up in time after service breaks down due to special circumstances in the production environment.

The automatic pull-up service of FE and BE must be configured after the Doris cluster is completely set up.

Systemd Configures the Doris service

For details about systemd usage and parameter parsing, see here

sudo permission control

sudo permissions are required to control the doris service using systemd. To ensure the minimum granularity of sudo permission assignment, you can assign the systemd control permission of doris-fe and doris-be services to specified non-root users. Configure the systemctl management permission for doris-fe and doris-be in visudo.

```
Cmnd_Alias DORISCTL=/usr/bin/systemctl start doris-fe,/usr/bin/systemctl stop doris-fe,/usr/bin/systemctl start doris-
be,/usr/bin/systemctl stop doris-be

## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
doris   ALL=(ALL)       NOPASSWD:DORISCTL
```

Configuration procedure

1. You should config the “JAVA_HOME” variable in the config file, both fe.conf and be.conf, or you can’t use the command “systemctl start” to start doris

   ```
   echo "JAVA_HOME=your_java_home" >> /home/doris/fe/conf/fe.conf
   echo "JAVA_HOME=your_java_home" >> /home/doris/be/conf/be.conf
   ```

2. Download the doris-fe.service file: doris-fe.service

3. The details of doris-fe.service are as follows:

   ```
   # Licensed to the Apache Software Foundation (ASF) under one
   # or more contributor license agreements.  See the NOTICE file
   # distributed with this work for additional information
   # regarding copyright ownership.  The ASF licenses this file
   # to you under the Apache License, Version 2.0 (the
   # "License"); you may not use this file except in compliance
   # with the License.  You may obtain a copy of the License at
   #
   #   http://www.apache.org/licenses/LICENSE-2.0
   #
   ```

```
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.

[Unit]
Description=Doris FE
After=network-online.target
Wants=network-online.target

[Service]
Type=forking
User=root
Group=root
LimitCORE=infinity
LimitNOFILE=200000
Restart=on-failure
RestartSec=30
StartLimitInterval=120
StartLimitBurst=3
KillMode=none
ExecStart=/home/doris/fe/bin/start_fe.sh --daemon
ExecStop=/home/doris/fe/bin/stop_fe.sh

[Install]
WantedBy=multi-user.target
```

Matters needing attention

- ExecStart and ExecStop are configured based on actual fe paths

4. Download the doris-be.service file : doris-be.service

5. The details of doris-be.service are as follows: ``` # Licensed to the Apache Software Foundation (ASF) under one # or more contributor license agreements. See the NOTICE file # distributed with this work for additional information # regarding copyright ownership. The ASF licenses this file # to you under the Apache License, Version 2.0 (the # “License”); you may not use this file except in compliance # with the License. You may obtain a copy of the License at # # http://www.apache.org/licenses/LICENSE-2.0 # # Unless required by applicable law or agreed to in writing, # software distributed under the License is distributed on an # “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY # KIND, either express or implied. See the License for the # specific language governing permissions and limitations # under the License.

   [Unit] Description=Doris BE After=network-online.target Wants=network-online.target

   [Service] Type=forking User=root Group=root LimitCORE=infinity LimitNOFILE=200000 Restart=on-failure RestartSec=30 StartLimitInterval=120 StartLimitBurst=3 KillMode=none ExecStart=/home/doris/be/bin/start_be.sh –daemon ExecStop=/home/doris/be/bin/stop_be.sh

   [Install] WantedBy=multi-user.target ```

Matters needing attention

- ExecStart and ExecStop are configured based on actual be paths

6. Service configuration

   Place doris-fe.service and doris-be.service in the /usr/lib/systemd/system directory

7. Set self-start

   After you add or modify the configuration file, you need to reload it

   ```
   systemctl daemon-reload
   ```

   Set the start, the essence is in the/etc/systemd/system/multi - user. Target. Wants/add service file link

   ```
   systemctl enable doris-fe
   systemctl enable doris-be
   ```

8. Service initiation

   ```
   systemctl start doris-fe
   systemctl start doris-be
   ```

Supervisor configures the Doris service

Supervisor Specific use and parameter analysis can be referred to here

Supervisor configuration automatically pulls up the supervisor configuration. You can install the supervisor directly using the yum command or manually using pip. The pip manual installation process is complicated, and only the yum deployment mode is displayed.Manual deployment refer to [here] (http://supervisord.org/installing.html) for installation deployment.

Configuration procedure

1. yum Install supervisor

   ```
   yum install epel-release
   yum install -y supervisor
   ```

2. Start the service and view the status

   ```
   systemctl enable supervisord # bootstrap
   systemctl start supervisord # Start the supervisord service
   systemctl status supervisord # Check the supervisord service status
   ps -ef|grep supervisord # Check whether the supervisord process exists
   ```

3. Configure BE process management

   ```
   Modify the start_be.sh script remove the last symbol &

   vim /path/doris/be/bin/start_be.sh
   Take this code : nohup $LIMIT ${DORIS_HOME}/lib/palo_be "$@" >> $LOG_DIR/be.out 2>&1 </dev/null &
   Be changed to : nohup $LIMIT ${DORIS_HOME}/lib/palo_be "$@" >> $LOG_DIR/be.out 2>&1 </dev/null
   ```

Create a supervisor process management configuration file for the BE

```
vim /etc/supervisord.d/doris-be.ini

[program:doris_be]
process_name=%(program_name)s
directory=/path/doris/be/be
command=sh /path/doris/be/bin/start_be.sh
autostart=true
autorestart=true
user=root
numprocs=1
startretries=3
stopasgroup=true
killasgroup=true
startsecs=5
#redirect_stderr = true
#stdout_logfile_maxbytes = 20MB
#stdout_logfile_backups = 10
#stdout_logfile=/var/log/supervisor-palo_be.log
```

4. Configure FE process management

```
Modify the start_fe.sh script remove the last symbol &

vim /path/doris/fe/bin/start_fe.sh
Take this code : nohup $LIMIT $JAVA $final_java_opt org.apache.doris.PaloFe ${HELPER} "$@" >> $LOG_DIR/fe.out 2>&1
</dev/null &
Be changed to : nohup $LIMIT $JAVA $final_java_opt org.apache.doris.PaloFe ${HELPER} "$@" >> $LOG_DIR/fe.out 2>&1
</dev/null
```

Create a supervisor process management configuration file for FE

```
vim /etc/supervisord.d/doris-fe.ini

[program:PaloFe]
environment = JAVA_HOME="/usr/local/java"
process_name=PaloFe
directory=/path/doris/fe
command=sh /path/doris/fe/bin/start_fe.sh
autostart=true
autorestart=true
user=root
numprocs=1
startretries=3
stopasgroup=true
killasgroup=true
startsecs=10
#redirect_stderr=true
#stdout_logfile_maxbytes=20MB
#stdout_logfile_backups=10
#stdout_logfile=/var/log/supervisor-PaloFe.log
```

5. Configure Broker process management

```
Modify the start_broker.sh script remove the last symbol &

vim /path/apache_hdfs_broker/bin/start_broker.sh
Take this code : nohup $LIMIT $JAVA $JAVA_OPTS org.apache.doris.broker.hdfs.BrokerBootstrap "$@" >>
```

```
$BROKER_LOG_DIR/apache_hdfs_broker.out 2>&1 </dev/null &
Be changed to : nohup $LIMIT $JAVA $JAVA_OPTS org.apache.doris.broker.hdfs.BrokerBootstrap "$@" >>
$BROKER_LOG_DIR/apache_hdfs_broker.out 2>&1 </dev/null
```

Create the supervisor process management profile for the Broker

```
vim /etc/supervisord.d/doris-broker.ini

[program:BrokerBootstrap]
environment = JAVA_HOME="/usr/local/java"
process_name=%(program_name)s
directory=/path/apache_hdfs_broker
command=sh /path/apache_hdfs_broker/bin/start_broker.sh
autostart=true
autorestart=true
user=root
numprocs=1
startretries=3
stopasgroup=true
killasgroup=true
startsecs=5
#redirect_stderr=true
#stdout_logfile_maxbytes=20MB
#stdout_logfile_backups=10
#stdout_logfile=/var/log/supervisor-BrokerBootstrap.log
```

6. First determine whether the Doris service is stopped, then use supervisor to automatically pull up Doris, and then determine whether the process starts normally

```
supervisorctl reload # Reload all the Supervisor configuration files
supervisorctl status # Check the supervisor status and verify that the Doris service process starts normally

其他命令 :
supervisorctl start all # supervisorctl start It is capable of opening processes
supervisorctl stop doris-be # The process is supervisorctl stop
```

Matters needing attention:

- If the supervisor installed using yum starts, an error occurs: pkg_resources.DistributionNotFound: The ‘supervisor==3.4.0’ distribution was not found

```
supervisor installed directly using the yum command only supports python2,Therefore, the file contents in /usr/bin/supervisorctl and /usr/bin/supervisorctl should be changed at the beginning Change #! /usr/bin/python to #! /usr/bin/python2, python2 must be installed
```

- If the supervisor is configured to automatically pull up the Doris process, if the BE node breaks down due to abnormal factors on Doris, the error stack information that should be output to be.out will be intercepted by the supervisor. We need to look it up in supervisor’s log for further analysis.

Config Dir

Config Dir

The configuration file directory for FE and BE is `conf/`. In addition to storing the default fe.conf, be.conf and other files, this directory is also used for the common configuration file storage directory.

Users can store some configuration files in it, and the system will automatically read them.

> **Tips**
>
> This features is supported since the Apache Doris 1.2 version

hdfs-site.xml and hive-site.xml

In some functions of Doris, you need to access data on HDFS, or access Hive metastore.

We can manually fill in various HDFS/Hive parameters in the corresponding statement of the function.

But these parameters are very many, if all are filled in manually, it is very troublesome.

Therefore, users can place the HDFS or Hive configuration file hdfs-site.xml/hive-site.xml directly in the `conf/` directory. Doris will automatically read these configuration files.

The configuration that the user fills in the command will overwrite the configuration items in the configuration file.

In this way, users only need to fill in a small amount of configuration to complete the access to HDFS/Hive.

FE Configuration | Config

FE Configuration

This document mainly introduces the relevant configuration items of FE.

The FE configuration file `fe.conf` is usually stored in the `conf/` directory of the FE deployment path. In version 0.14, another configuration file `fe_custom.conf` will be introduced. The configuration file is used to record the configuration items that are dynamically configured and persisted by the user during operation.

After the FE process is started, it will read the configuration items in `fe.conf` first, and then read the configuration items in `fe_custom.conf`. The configuration items in `fe_custom.conf` will overwrite the same configuration items in `fe.conf`.

The location of the `fe_custom.conf` file can be configured in `fe.conf` through the `custom_config_dir` configuration item.

View configuration items

There are two ways to view the configuration items of FE:

1. FE web page

   Open the FE web page `http://fe_host:fe_http_port/variable` in the browser. You can see the currently effective FE configuration items in `Configure Info`.

2. View by command

   After the FE is started, you can view the configuration items of the FE in the MySQL client with the following command,Concrete language law reference SHOW-CONFIG:

   ```
   SHOW FRONTEND CONFIG;
   ```

   The meanings of the columns in the results are as follows:

   - Key: the name of the configuration item.
   - Value: The value of the current configuration item.
   - Type: The configuration item value type, such as integer or string.
   - IsMutable: whether it can be dynamically configured. If true, the configuration item can be dynamically configured at runtime. If false, it means that the configuration item can only be configured in `fe.conf` and takes effect after restarting FE.
   - MasterOnly: Whether it is a unique configuration item of Master FE node. If it is true, it means that the configuration item is meaningful only at the Master FE node, and is meaningless to other types of FE nodes. If false, it means that the configuration item is meaningful in all types of FE nodes.
   - Comment: The description of the configuration item.

Set configuration items

There are two ways to configure FE configuration items:

1. Static configuration

   Add and set configuration items in the `conf/fe.conf` file. The configuration items in `fe.conf` will be read when the FE process starts. Configuration items not in `fe.conf` will use default values.

2. Dynamic configuration via MySQL protocol

   After the FE starts, you can set the configuration items dynamically through the following commands. This command requires administrator privilege.

   ```
   ADMIN SET FRONTEND CONFIG (" fe_config_name "=" fe_config_value ");
   ```

   Not all configuration items support dynamic configuration. You can check whether the dynamic configuration is supported by the `IsMutable` column in the`SHOW FRONTEND CONFIG;`command result.

   If the configuration item of `MasterOnly` is modified, the command will be directly forwarded to the Master FE and only the corresponding configuration item in the Master FE will be modified.

   Configuration items modified in this way will become invalid after the FE process restarts.

   For more help on this command, you can view it through the `HELP ADMIN SET CONFIG;` command.

3. Dynamic configuration via HTTP protocol

   For details, please refer to Set Config Action

   This method can also persist the modified configuration items. The configuration items will be persisted in the `fe_custom.conf` file and will still take effect after FE is restarted.

Examples

1. Modify `async_pending_load_task_pool_size`

   Through `SHOW FRONTEND CONFIG;` you can see that this configuration item cannot be dynamically configured (`IsMutable` is false). You need to add in `fe.conf`:

   ```
   async_pending_load_task_pool_size = 20
   ```

   Then restart the FE process to take effect the configuration.

2. Modify `dynamic_partition_enable`

   Through `SHOW FRONTEND CONFIG;` you can see that the configuration item can be dynamically configured (`IsMutable` is true). And it is the unique configuration of Master FE. Then first we can connect to any FE and execute the following command to modify the configuration:

   ```
   ADMIN SET FRONTEND CONFIG ("dynamic_partition_enable" = "true"); `
   ```

   Afterwards, you can view the modified value with the following command:

```
set forward_to_master = true;
SHOW FRONTEND CONFIG;
```

After modification in the above manner, if the Master FE restarts or a Master election is performed, the configuration will be invalid. You can add the configuration item directly in `fe.conf` and restart the FE to make the configuration item permanent.

3. Modify `max_distribution_pruner_recursion_depth`

   Through `SHOW FRONTEND CONFIG;` you can see that the configuration item can be dynamically configured (`IsMutable` is true). It is not unique to Master FE.

   Similarly, we can modify the configuration by dynamically modifying the configuration command. Because this configuration is not unique to the Master FE, user need to connect to different FEs separately to modify the configuration dynamically, so that all FEs use the modified configuration values.

Configurations

Metadata And Cluster

`meta_dir`

Default：DORIS_HOME_DIR + “/doris-meta”

Type: string Description: Doris meta data will be saved here.The storage of this dir is highly recommended as to be:

- High write performance (SSD)
- Safe (RAID）

`catalog_try_lock_timeout_ms`

Default：5000 （ms）

IsMutable：true

The tryLock timeout configuration of catalog lock. Normally it does not need to change, unless you need to test something.

`enable_bdbje_debug_mode`

Default：false

If set to true, FE will be started in BDBJE debug mode

`max_bdbje_clock_delta_ms`

Default：5000 （5s）

Set the maximum acceptable clock skew between non-master FE to Master FE host. This value is checked whenever a non-master FE establishes a connection to master FE via BDBJE. The connection is abandoned if the clock skew is larger than this value.

`metadata_failure_recovery`

Default：false

If true, FE will reset bdbje replication group(that is, to remove all electable nodes info) and is supposed to start as Master. If all the electable nodes can not start, we can copy the meta data to another node and set this config to true to try to restart the FE..

`txn_rollback_limit`

Default：100

the max txn number which bdbje can rollback when trying to rejoin the group

`grpc_threadmgr_threads_nums`

Default: 4096

Num of thread to handle grpc events in grpc_threadmgr.

`bdbje_replica_ack_timeout_second`

Default：10 (s)

The replica ack timeout when writing to bdbje , When writing some relatively large logs, the ack time may time out, resulting in log writing failure. At this time, you can increase this value appropriately.

`bdbje_lock_timeout_second`

Default：5

The lock timeout of bdbje operation, If there are many LockTimeoutException in FE WARN log, you can try to increase this value

`bdbje_heartbeat_timeout_second`

Default：30

The heartbeat timeout of bdbje between master and follower. the default is 30 seconds, which is same as default value in bdbje. If the network is experiencing transient problems, of some unexpected long java GC annoying you, you can try to increase this value to decrease the chances of false timeouts

`replica_ack_policy`

Default：SIMPLE_MAJORITY

OPTION: ALL, NONE, SIMPLE_MAJORITY

Replica ack policy of bdbje. more info, see:
http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.ReplicaAckPolicy.html

`replica_sync_policy`

Default：SYNC

选项：SYNC, NO_SYNC, WRITE_NO_SYNC

Follower FE sync policy of bdbje.

`master_sync_policy`

Default：SYNC

选项：SYNC, NO_SYNC, WRITE_NO_SYNC

Master FE sync policy of bdbje. If you only deploy one Follower FE, set this to 'SYNC'. If you deploy more than 3 Follower FE, you can set this and the following 'replica_sync_policy' to WRITE_NO_SYNC. more info, see:
http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.SyncPolicy.html

`bdbje_reserved_disk_bytes`

The desired upper limit on the number of bytes of reserved space to retain in a replicated JE Environment.

Default: 1073741824

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: false

`ignore_meta_check`

Default：false

IsMutable：true

If true, non-master FE will ignore the meta data delay gap between Master FE and its self, even if the metadata delay gap exceeds *meta_delay_toleration_second*. Non-master FE will still offer read service. This is helpful when you try to stop the Master FE for a relatively long time for some reason, but still wish the non-master FE can offer read service.

`meta_delay_toleration_second`

Default: 300 (5 min)

Non-master FE will stop offering service if meta data delay gap exceeds *meta_delay_toleration_second*

`edit_log_port`

Default：9010

bdbje port

`edit_log_type`

Default：BDB

Edit log type. BDB: write log to bdbje LOCAL: deprecated..

`edit_log_roll_num`

Default：50000

IsMutable：true

MasterOnly：true

Master FE will save image every *edit_log_roll_num* meta journals.

`force_do_metadata_checkpoint`

Default：false

IsMutable：true

MasterOnly：true

If set to true, the checkpoint thread will make the checkpoint regardless of the jvm memory used percent

`metadata_checkpoint_memory_threshold`

Default：60 （60%）

IsMutable：true

MasterOnly：true

If the jvm memory used percent(heap or old mem pool) exceed this threshold, checkpoint thread will not work to avoid OOM.

`max_same_name_catalog_trash_num`

It is used to set the maximum number of meta information with the same name in the catalog recycle bin. When the maximum value is exceeded, the earliest deleted meta trash will be completely deleted and cannot be recovered. 0 means not to keep objects of the same name. < 0 means no limit.

Note: The judgment of metadata with the same name will be limited to a certain range. For example, the judgment of the database with the same name will be limited to the same cluster, the judgment of the table with the same name will be limited to the same database (with the same database id), the judgment of the partition with the same name will be limited to the same database (with the same database id) and the same table (with the same table) same table id).

Default: 3

Is it possible to dynamically configure: true

Is it a configuration item unique to the Master FE node: true

`cluster_id`

Default：-1

node(FE or BE) will be considered belonging to the same Palo cluster if they have same cluster id. Cluster id is usually a random integer generated when master FE start at first time. You can also specify one.

`heartbeat_mgr_blocking_queue_size`

Default：1024

MasterOnly：true

blocking queue size to store heartbeat task in heartbeat_mgr.

`heartbeat_mgr_threads_num`

Default：8

MasterOnly：true

num of thread to handle heartbeat events in heartbeat_mgr.

`disable_cluster_feature`

Default：true

IsMutable：true

The multi cluster feature will be deprecated in version 0.12 , set this config to true will disable all operations related to cluster feature, include:

1. create/drop cluster
2. add free backend/add backend to cluster/decommission cluster balance
3. change the backends num of cluster
4. link/migration db

`enable_fqdn_mode`

This configuration is mainly used in the k8s cluster environment. When enable_fqdn_mode is true, the name of the pod where the be is located will remain unchanged after reconstruction, while the ip can be changed.

Default: false

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: true

`enable_token_check`

Default: true

For forward compatibility, will be removed later. check token when download image file.

`enable_multi_tags`

Default: false

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: true

Whether to enable the multi-tags function of a single BE

`initial_root_password`

Set root user initial 2-staged SHA-1 encrypted password, default as '', means no root password. Subsequent `set password` operations for root user will overwrite the initial root password.

Example: If you want to configure a plaintext password `root@123`. You can execute Doris SQL `select password('root@123')` to generate encrypted password `*A00C34073A26B40AB4307650BFB9309D6BFA6999`.

Default: empty string

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: true

Service

`query_port`

Default：9030

FE MySQL server port

`arrow_flight_sql_port`

Default：-1

Arrow Flight SQL server port

`frontend_address`

Status: Deprecated, not recommended use. This parameter may be deleted later

Type: string

Description: Explicitly set the IP address of FE instead of using *InetAddress.getByName* to get the IP address. Usually in *InetAddress.getByName* When the expected results cannot be obtained. Only IP address is supported, not hostname.

Default value: 0.0.0.0

`priority_networks`

Default：none

Declare a selection strategy for those servers have many ips. Note that there should at most one ip match this list. this is a list in semicolon-delimited format, in CIDR notation, e.g. 10.10.10.0/24 , If no ip match this rule, will choose one randomly.

`http_port`

Default：8030

HTTP bind port. All FE http ports must be same currently.

`https_port`

Default：8050

HTTPS bind port. All FE https ports must be same currently.

`enable_https`

Default：false

Https enable flag. If the value is false, http is supported. Otherwise, both http and https are supported, and http requests are automatically redirected to https. If enable_https is true, you need to configure ssl certificate information in fe.conf.

`enable_ssl`

Default：true

If set to true, doris will establish an encrypted channel based on the SSL protocol with mysql.

`qe_max_connection`

Default：1024

Maximal number of connections per FE.

`check_java_version`

Default：true

Doris will check whether the compiled and run Java versions are compatible, if not, it will throw a Java version mismatch exception message and terminate the startup

`rpc_port`

Default：9020

FE Thrift Server port

`thrift_server_type`

This configuration represents the service model used by The Thrift Service of FE, is of type String and is case-insensitive.

If this parameter is ‘SIMPLE’, then the ‘TSimpleServer’ model is used, which is generally not suitable for production and is limited to test use.

If the parameter is ‘THREADED’, then the ‘TThreadedSelectorServer’ model is used, which is a non-blocking I/O model, namely the master-slave Reactor model, which can timely respond to a large number of concurrent connection requests and performs well in most scenarios.

If this parameter is `THREAD_POOL`, then the `TThreadPoolServer` model is used, the model for blocking I/O model, use the thread pool to handle user connections, the number of simultaneous connections are limited by the number of thread pool, if we can estimate the number of concurrent requests in advance, and tolerant enough thread resources cost, this model will have a better performance, the service model is used by default

`thrift_server_max_worker_threads`

Default：4096

The thrift server max worker threads

`thrift_backlog_num`

Default：1024

The backlog_num for thrift server , When you enlarge this backlog_num, you should ensure it's value larger than the linux /proc/sys/net/core/somaxconn config

`thrift_client_timeout_ms`

Default：0

The connection timeout and socket timeout config for thrift server.

The value for thrift_client_timeout_ms is set to be zero to prevent read timeout.

`thrift_max_message_size`

> **Tips**
>
> This configuration is supported since the Apache Doris 1.2.4 version

Default: 100MB

The maximum size of a (received) message of the thrift server, in bytes. If the size of the message sent by the client exceeds this limit, the Thrift server will reject the request and close the connection. As a result, the client will encounter the error: "connection has been closed by peer." In this case, you can try increasing this parameter.

`use_compact_thrift_rpc`

Default: true

Whether to use compressed format to send query plan structure. After it is turned on, the size of the query plan structure can be reduced by about 50%, thereby avoiding some "send fragment timeout" errors. However, in some high-concurrency small query scenarios, the concurrency may be reduced by about 10%.

`grpc_max_message_size_bytes`

Default：1G

Used to set the initial flow window size of the GRPC client channel, and also used to max message size. When the result set is large, you may need to increase this value.

`max_mysql_service_task_threads_num`

Default：4096

The number of threads responsible for Task events.

`mysql_service_io_threads_num`

Default：4

When FE starts the MySQL server based on NIO model, the number of threads responsible for IO events.

`mysql_nio_backlog_num`

Default：1024

The backlog_num for mysql nio server, When you enlarge this backlog_num, you should enlarge the value in the linux /proc/sys/net/core/somaxconn file at the same time

`broker_timeout_ms`

Default：10000 （10s）

Default broker RPC timeout

`backend_rpc_timeout_ms`

Timeout millisecond for Fe sending rpc request to BE

Default: 60000

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: true

`drop_backend_after_decommission`

Default：false

IsMutable：true

MasterOnly：true

1. This configuration is used to control whether the system drops the BE after successfully decommissioning the BE. If true, the BE node will be deleted after the BE is successfully offline. If false,

after the BE successfully goes offline, the BE will remain in the DECOMMISSION state, but will not be dropped.

This configuration can play a role in certain scenarios. Assume that the initial state of a Doris cluster is one disk per BE node. After running for a period of time, the system has been vertically expanded, that is, each BE node adds 2 new disks. Because Doris currently does not support data balancing among the disks within the BE, the data volume of the initial disk may always be much higher than the data volume of the newly added disk. At this time, we can perform manual inter-disk balancing by the following operations:

1. Set the configuration item to false.
2. Perform a decommission operation on a certain BE node. This operation will migrate all data on the BE to other nodes.
3. After the decommission operation is completed, the BE will not be dropped. At this time, cancel the decommission status of the BE. Then the data will start to balance from other BE nodes back to this node. At this time, the data will be evenly distributed to all disks of the BE.
4. Perform steps 2 and 3 for all BE nodes in sequence, and finally achieve the purpose of disk balancing for all nodes

`max_backend_down_time_second`

Default: 3600 (1 hour)

IsMutable：true

MasterOnly：true

If a backend is down for *max_backend_down_time_second*, a BACKEND_DOWN event will be triggered.

`disable_backend_black_list`

Used to disable the BE blacklist function. After this function is disabled, if the query request to the BE fails, the BE will not be added to the blacklist. This parameter is suitable for regression testing environments to reduce occasional bugs that cause a large number of regression tests to fail.

Default: false

Is it possible to configure dynamically: true

Is it a configuration item unique to the Master FE node: false

`max_backend_heartbeat_failure_tolerance_count`

The maximum tolerable number of BE node heartbeat failures. If the number of consecutive heartbeat failures exceeds this value, the BE state will be set to dead. This parameter is suitable for regression test environments to reduce occasional heartbeat failures that cause a large number of regression test failures.

Default: 1

Is it possible to configure dynamically: true

Whether it is a configuration item unique to the Master FE node: true

`abort_txn_after_lost_heartbeat_time_second`

Abort transaction time after lost heartbeat. The default value is 300, which means transactions of be will be aborted after lost heartbeat 300s.

Default: 300(s)

Is it possible to configure dynamically: true

Whether it is a configuration item unique to the Master FE node: true

`enable_access_file_without_broker`

Default：false

IsMutable：true

MasterOnly：true

This config is used to try skip broker when access bos or other cloud storage via broker

`agent_task_resend_wait_time_ms`

Default：5000

IsMutable：true

MasterOnly：true

This configuration will decide whether to resend agent task when create_time for agent_task is set, only when current_time - create_time > agent_task_resend_wait_time_ms can ReportHandler do resend agent task.

This configuration is currently mainly used to solve the problem of repeated sending of `PUBLISH_VERSION` agent tasks. The current default value of this configuration is 5000, which is an experimental value.

Because there is a certain time delay between submitting agent tasks to AgentTaskQueue and submitting to be, Increasing the value of this configuration can effectively solve the problem of repeated sending of agent tasks,

But at the same time, it will cause the submission of failed or failed execution of the agent task to be executed again for an extended period of time

`max_agent_task_threads_num`

Default：4096

MasterOnly：true

max num of thread to handle agent task in agent task thread-pool.

`remote_fragment_exec_timeout_ms`

Default：30000 （ms）

IsMutable：true

The timeout of executing async remote fragment. In normal case, the async remote fragment will be executed in a short time. If system are under high load condition, try to set this timeout longer.

`auth_token`

Default：empty

Cluster token used for internal authentication.

`enable_http_server_v2`

Default: The default is true after the official 0.14.0 version is released, and the default is false before

HTTP Server V2 is implemented by SpringBoot. It uses an architecture that separates the front and back ends. Only when HTTPv2 is enabled can users use the new front-end UI interface.

`http_api_extra_base_path`

In some deployment environments, user need to specify an additional base path as the unified prefix of the HTTP API. This parameter is used by the user to specify additional prefixes. After setting, user can get the parameter value through the `GET /api/basepath` interface. And the new UI will also try to get this base path first to assemble the URL. Only valid when `enable_http_server_v2` is true.

The default is empty, that is, not set

`jetty_server_acceptors`

Default：2

`jetty_server_selectors`

Default：4

`jetty_server_workers`

Default：0

With the above three parameters, Jetty's thread architecture model is very simple, divided into acceptors, selectors and workers three thread pools. Acceptors are responsible for accepting new connections, and then hand them over to selectors to process the unpacking of the HTTP message protocol, and finally workers process the request. The first two thread pools adopt a non-blocking model, and one thread can handle the read and write of many sockets, so the number of thread pools is small.

For most projects, only 1-2 acceptors threads are required, and 2 to 4 selectors threads are sufficient. Workers are obstructive business logic, often have more database operations, and require a large number of threads. The specific number depends on the proportion of QPS and IO events of the application. The higher the QPS, the more threads are required, the higher the proportion of IO, the more threads waiting, and the more total threads required.

Worker thread pool is not set by default, set according to your needs

`jetty_server_max_http_post_size`

Default: `100 * 1024 * 1024` (100MB)

This is the maximum number of bytes of the file uploaded by the put or post method, the default value: 100MB

`jetty_server_max_http_header_size`

Default: 1048576 (1M)

http header size configuration parameter, the default value is 1M.

`http_sql_submitter_max_worker_threads`

Default: 2

The max number work threads of http sql submitter

`http_load_submitter_max_worker_threads`

Default: 2

The max number work threads of http upload submitter

Query Engine

`default_max_query_instances`

The default value when user property max_query_instances is equal or less than 0. This config is used to limit the max number of instances for a user. This parameter is less than or equal to 0 means unlimited.

The default value is -1

`max_query_retry_time`

Default: 3

IsMutable: true

The number of query retries. A query may retry if we encounter RPC exception and no result has been sent to user. You may reduce this number to avoid Avalanche disaster

`max_dynamic_partition_num`

Default：500

IsMutable：true

MasterOnly：true

Used to limit the maximum number of partitions that can be created when creating a dynamic partition table, to avoid creating too many partitions at one time. The number is determined by “start” and “end” in the dynamic partition parameters.

`dynamic_partition_enable`

Default：true

IsMutable：true

MasterOnly：true

Whether to enable dynamic partition scheduler, enabled by default

`dynamic_partition_check_interval_seconds`

Default：600 （s）

IsMutable：true

MasterOnly：true

Decide how often to check dynamic partition

`max_multi_partition_num`

> **Tips**
>
> This configuration is supported since the Apache Doris 1.2 version

Default：4096

IsMutable：true

MasterOnly：true

Use this parameter to set the partition name prefix for multi partition,Only multi partition takes effect, not dynamic partitions. The default prefix is “p_”.

`multi_partition_name_prefix`

Default: p_

IsMutable: true

MasterOnly: true

Use this parameter to set the partition name prefix for multi partition, Only multi partition takes effect, not dynamic partitions.The default prefix is “p_”.

`partition_in_memory_update_interval_secs`

Default: 300 (s)

IsMutable: true

MasterOnly: true

Time to update global partition information in memory

`enable_concurrent_update`

Default: false

IsMutable: false

MasterOnly: true

Whether to enable concurrent update

`lower_case_table_names`

Default: 0

IsMutable: false

MasterOnly: true

This configuration can only be configured during cluster initialization and cannot be modified during cluster restart and upgrade after initialization is complete.

0: table names are stored as specified and comparisons are case sensitive. 1: table names are stored in lowercase and comparisons are not case sensitive. 2: table names are stored as given but compared in lowercase.

`table_name_length_limit`

Default: 64

IsMutable: true

MasterOnly：true

Used to control the maximum table name length

`cache_enable_sql_mode`

Default：true

IsMutable：true

MasterOnly：false

If this switch is turned on, the SQL query result set will be cached. If the interval between the last visit version time in all partitions of all tables in the query is greater than cache_last_version_interval_second, and the result set is less than cache_result_max_row_count, and the data size is less than cache_result_max_data_size, the result set will be cached, and the next same SQL will hit the cache

If set to true, fe will enable sql result caching. This option is suitable for offline data update scenarios

	case1	case2	case3	case4
enable_sql_cache	false	true	true	false
enable_partition_cache	false	false	true	true

`cache_enable_partition_mode`

Default：true

IsMutable：true

MasterOnly：false

If set to true, fe will get data from be cache, This option is suitable for real-time updating of partial partitions.

`cache_result_max_row_count`

Default：3000

IsMutable：true

MasterOnly：false

In order to avoid occupying too much memory, the maximum number of rows that can be cached is 3000 by default. If this threshold is exceeded, the cache cannot be set

`cache_result_max_data_size`

Default: 31457280

IsMutable: true

MasterOnly: false

In order to avoid occupying too much memory, the maximum data size of rows that can be cached is 10MB by default. If this threshold is exceeded, the cache cannot be set

`cache_last_version_interval_second`

Default：30

IsMutable：true

MasterOnly：false

The time interval of the latest partitioned version of the table refers to the time interval between the data update and the current version. It is generally set to 900 seconds, which distinguishes offline and real-time import

`enable_batch_delete_by_default`

Default：false

IsMutable：true

MasterOnly：true

Whether to add a delete sign column when create unique table

`max_allowed_in_element_num_of_delete`

Default：1024

IsMutable：true

MasterOnly：true

This configuration is used to limit element num of InPredicate in delete statement.

`max_running_rollup_job_num_per_table`

Default：1

IsMutable：true

MasterOnly：true

Control the concurrency limit of Rollup jobs

`max_distribution_pruner_recursion_depth`

Default：100

IsMutable：true

MasterOnly：false

This will limit the max recursion depth of hash distribution pruner. eg: where a in (5 elements) and b in (4 elements) and c in (3 elements) and d in (2 elements). a/b/c/d are distribution columns, so the recursion depth will be 5 *4* 3 * 2 = 120, larger than 100, So that distribution pruner will no work and just return all buckets. Increase the depth can support distribution pruning for more elements, but may cost more CPU.

`enable_local_replica_selection`

Default：false

IsMutable：true

If set to true, Planner will try to select replica of tablet on same host as this Frontend. This may reduce network transmission in following case:

- N hosts with N Backends and N Frontends deployed.
- The data has N replicas.
- High concurrency queries are evenly sent to all Frontends evenly

In this case, all Frontends can only use local replicas to do the query. If you want to allow fallback to non-local replicas when no local replicas available, set enable_local_replica_selection_fallback to true.

`enable_local_replica_selection_fallback`

Default：false

IsMutable：true

Used with enable_local_replica_selection. If the local replicas is not available, fallback to the non-local replicas.

`expr_depth_limit`

Default：3000

IsMutable：true

Limit on the depth of an expr tree. Exceed this limit may cause long analysis time while holding db read lock. Do not set this if you know what you are doing

`expr_children_limit`

Default：10000

IsMutable：true

Limit on the number of expr children of an expr tree. Exceed this limit may cause long analysis time while holding database read lock.

`be_exec_version`

Used to define the serialization format for passing blocks between fragments.

Sometimes some of our code changes will change the data format of the block. In order to make the BE compatible with each other during the rolling upgrade process, we need to issue a data version from the FE to decide what format to send the data in.

Specifically, for example, there are 2 BEs in the cluster, one of which can support the latest v_1 after being upgraded, while the other only supports v_0. At this time, since the FE has not been upgraded yet, $v_0 is issued uniformly. $, BE interact in the old data format. After all BEs are upgraded, we will upgrade FE. At this time, the new FE will issue v_1, and the cluster will be uniformly switched to the new data format.

The default value is `max_be_exec_version`. If there are special needs, we can manually set the format version to lower, but it should not be lower than `min_be_exec_version`.

Note that we should always keep the value of this variable between `BeExecVersionManager::min_be_exec_version` and `BeExecVersionManager::max_be_exec_version` for all BEs. (That is to say, if a cluster that has completed the update needs to be downgraded, it should ensure the order of downgrading FE and then downgrading BE, or manually lower the variable in the settings and downgrade BE)

`max_be_exec_version`

The latest data version currently supported, cannot be modified, and should be consistent with the `BeExecVersionManager::max_be_exec_version` in the BE of the matching version.

`min_be_exec_version`

The oldest data version currently supported, which cannot be modified, should be consistent with the `BeExecVersionManager::min_be_exec_version` in the BE of the matching version.

`max_query_profile_num`

The max number of query profile.

Default: 100

Is it possible to dynamically configure: true

Is it a configuration item unique to the Master FE node: false

`publish_version_interval_ms`

Default：10 （ms）

minimal intervals between two publish version action

`publish_version_timeout_second`

Default：30 （s）

IsMutable：true

MasterOnly：true

Maximal waiting time for all publish version tasks of one transaction to be finished

`query_colocate_join_memory_limit_penalty_factor`

Default：1

IsMutable：true

Colocote join PlanFragment instance 的 memory_limit = exec_mem_limit / min (query_colocate_join_memory_limit_penalty_factor, instance_num)

`rewrite_count_distinct_to_bitmap_hll`

Default: true

This variable is a session variable, and the session level takes effect.

- Type: boolean
- Description: **Only for the table of the AGG model**, when the variable is true, when the user query contains aggregate functions such as count(distinct c1), if the type of the c1 column itself is bitmap, count distinct will be rewritten It is bitmap_union_count(c1). When the type of the c1 column itself is hll, count distinct will be rewritten as hll_union_agg(c1) If the variable is false, no overwriting occurs..

Load And Export

`enable_pipeline_load`

Default value: true

IsMutable: true

Whether or not to enable the Pipeline engine to perform import tasks such as Streamload.

`enable_vectorized_load`

Default: true

Whether to enable vectorized load

`enable_new_load_scan_node`

Default: true

Whether to enable file scan node

`default_max_filter_ratio`

Default：0

IsMutable：true

MasterOnly：true

Maximum percentage of data that can be filtered (due to reasons such as data is irregularly) , The default value is 0.

`max_running_txn_num_per_db`

Default：1000

IsMutable：true

MasterOnly：true

This configuration is mainly used to control the number of concurrent load jobs of the same database.

When there are too many load jobs running in the cluster, the newly submitted load jobs may report errors:

```
current running txns on db xxx is xx, larger than limit xx
```

When this error is encountered, it means that the load jobs currently running in the cluster exceeds the configuration value. At this time, it is recommended to wait on the business side and retry the load jobs.

If you use the Connector, the value of this parameter can be adjusted appropriately, and there is no problem with thousands

`using_old_load_usage_pattern`

Default：false

IsMutable：true

MasterOnly：true

If set to true, the insert stmt with processing error will still return a label to user. And user can use this label to check the load job's status. The default value is false, which means if insert operation encounter errors, exception will be thrown to user client directly without load label.

`disable_load_job`

Default：false

IsMutable：true

MasterOnly：true

if this is set to true

- all pending load job will failed when call begin txn api
- all prepare load job will failed when call commit txn api
- all committed load job will waiting to be published

`commit_timeout_second`

Default：30

IsMutable：true

MasterOnly：true

Maximal waiting time for all data inserted before one transaction to be committed This is the timeout second for the command “commit”

`max_unfinished_load_job`

Default：1000

IsMutable：true

MasterOnly：true

Max number of load jobs, include PENDING, ETL, LOADING, QUORUM_FINISHED. If exceed this number, load job is not allowed to be submitted

`db_used_data_quota_update_interval_secs`

Default：300 (s)

IsMutable：true

MasterOnly：true

One master daemon thread will update database used data quota for db txn manager every `db_used_data_quota_update_interval_secs`

For better data load performance, in the check of whether the amount of data used by the database before data load exceeds the quota, we do not calculate the amount of data already used by the database in real time, but obtain the periodically updated value of the daemon thread.

This configuration is used to set the time interval for updating the value of the amount of data used by the database

`disable_show_stream_load`

Default：false

IsMutable：true

MasterOnly：true

Whether to disable show stream load and clear stream load records in memory.

`max_stream_load_record_size`

Default：5000

IsMutable：true

MasterOnly：true

Default max number of recent stream load record that can be stored in memory.

`fetch_stream_load_record_interval_second`

Default：120

IsMutable：true

MasterOnly：true

fetch stream load record interval.

`max_bytes_per_broker_scanner`

Default：`500 * 1024 * 1024 * 1024L` （500G）

IsMutable：true

MasterOnly：true

Max bytes a broker scanner can process in one broker load job. Commonly, each Backends has one broker scanner.

`default_load_parallelism`

Default: 8

IsMutable：true

MasterOnly：true

Default parallelism of the broker load execution plan on a single node. If the user to set the parallelism when the broker load is submitted, this parameter will be ignored. This parameter will determine the concurrency of import tasks together with multiple configurations such as `max broker concurrency`, `min bytes per broker scanner`.

`max_broker_concurrency`

Default: 10

IsMutable: true

MasterOnly: true

Maximal concurrency of broker scanners.

`min_bytes_per_broker_scanner`

Default: 67108864L (64M)

IsMutable: true

MasterOnly: true

Minimum bytes that a single broker scanner will read.

`period_of_auto_resume_min`

Default: 5 (s)

IsMutable: true

MasterOnly: true

Automatically restore the cycle of Routine load

`max_tolerable_backend_down_num`

Default: 0

IsMutable: true

MasterOnly: true

As long as one BE is down, Routine Load cannot be automatically restored

`max_routine_load_task_num_per_be`

Default: 1024

IsMutable: true

MasterOnly: true

the max concurrent routine load task num per BE. This is to limit the num of routine load tasks sending to a BE, and it should also less than BE config ‘max_routine_load_thread_pool_size’(default 1024), which is the routine load task thread pool size on BE. In major versions of 2.1, the default value is 1024 starting with version 2.1.4. And before version 2.1.4 the default value was 5.

`max_routine_load_task_concurrent_num`

Default：256

IsMutable：true

MasterOnly：true

the max concurrent routine load task num of a single routine load job. In major versions of 2.1, the default value is 256 starting with version 2.1.4. And before version 2.1.4 the default value was 5.

`max_routine_load_job_num`

Default：100

the max routine load job num, including NEED_SCHEDULED, RUNNING, PAUSE

`desired_max_waiting_jobs`

Default：100

IsMutable：true

MasterOnly：true

Default number of waiting jobs for routine load and version 2 of load , This is a desired number. In some situation, such as switch the master, the current number is maybe more than desired_max_waiting_jobs.

`disable_hadoop_load`

Default：false

IsMutable：true

MasterOnly：true

Load using hadoop cluster will be deprecated in future. Set to true to disable this kind of load.

`enable_spark_load`

Default：false

IsMutable：true

MasterOnly：true

Whether to enable spark load temporarily, it is not enabled by default

Note: This parameter has been deleted in version 1.2, spark_load is enabled by default

`spark_load_checker_interval_second`

Default: 60

Spark load scheduler run interval, default 60 seconds

`async_loading_load_task_pool_size`

Default: 10

IsMutable: false

MasterOnly: true

The loading_load task executor pool size. This pool size limits the max running loading_load tasks.

Currently, it only limits the loading_load task of broker load

`async_pending_load_task_pool_size`

Default: 10

IsMutable: false

MasterOnly: true

The pending_load task executor pool size. This pool size limits the max running pending_load tasks.

Currently, it only limits the pending_load task of broker load and spark load.

It should be less than 'max_running_txn_num_per_db'

`async_load_task_pool_size`

Default: 10

IsMutable: false

MasterOnly: true

This configuration is just for compatible with old version, this config has been replaced by async_loading_load_task_pool_size, it will be removed in the future.

`enable_single_replica_load`

Default: false

IsMutable：true

MasterOnly：true

Whether to enable to write single replica for stream load and broker load.

`min_load_timeout_second`

Default：1 （1s）

IsMutable：true

MasterOnly：true

Min stream load timeout applicable to all type of load

`max_stream_load_timeout_second`

Default: 259200 (3 day)

IsMutable：true

MasterOnly：true

This configuration is specifically used to limit timeout setting for stream load. It is to prevent that failed stream load transactions cannot be canceled within a short time because of the user’s large timeout setting

`max_load_timeout_second`

Default: 259200 (3 day)

IsMutable：true

MasterOnly：true

Max load timeout applicable to all type of load except for stream load

`stream_load_default_timeout_second`

Default: 86400 * 3 (3 day)

IsMutable：true

MasterOnly：true

Default stream load and streaming mini load timeout

`stream_load_default_precommit_timeout_second`

Default：3600（s）

IsMutable：true

MasterOnly：true

Default stream load pre-submission timeout

`stream_load_default_memtable_on_sink_node`

Default：false

IsMutable：true

MasterOnly：true

Enable memtable on sink node for stream load by default. When HTTP header `memtable_on_sink_node` is not set.

`insert_load_default_timeout_second`

Default: 3600 (1 hour)

IsMutable：true

MasterOnly：true

Default insert load timeout

`mini_load_default_timeout_second`

Default: 3600 (1 hour)

IsMutable：true

MasterOnly：true

Default non-streaming mini load timeout

`broker_load_default_timeout_second`

Default: 14400 (4 hour)

IsMutable：true

MasterOnly：true

Default broker load timeout

`spark_load_default_timeout_second`

Default: 86400 (1 day)

IsMutable：true

MasterOnly：true

Default spark load timeout

`hadoop_load_default_timeout_second`

Default: 86400 * 3 (3 day)

IsMutable：true

MasterOnly：true

Default hadoop load timeout

`load_running_job_num_limit`

Default：0

IsMutable：true

MasterOnly：true

The number of loading tasks is limited, the default is 0, no limit

`load_input_size_limit_gb`

Default：0

IsMutable：true

MasterOnly：true

The size of the data entered by the Load job, the default is 0, unlimited

`load_etl_thread_num_normal_priority`

Default：10

Concurrency of NORMAL priority etl load jobs. Do not change this if you know what you are doing.

`load_etl_thread_num_high_priority`

Default：3

Concurrency of HIGH priority etl load jobs. Do not change this if you know what you are doing

`load_pending_thread_num_normal_priority`

Default：10

Concurrency of NORMAL priority pending load jobs. Do not change this if you know what you are doing.

`load_pending_thread_num_high_priority`

Default: 3

Concurrency of HIGH priority pending load jobs. Load job priority is defined as HIGH or NORMAL. All mini batch load jobs are HIGH priority, other types of load jobs are NORMAL priority. Priority is set to avoid that a slow load job occupies a thread for a long time. This is just a internal optimized scheduling policy. Currently, you can not specified the job priority manually, and do not change this if you know what you are doing.

`load_checker_interval_second`

Default: 5 (s)

The load scheduler running interval. A load job will transfer its state from PENDING to LOADING to FINISHED. The load scheduler will transfer load job from PENDING to LOADING while the txn callback will transfer load job from LOADING to FINISHED. So a load job will cost at most one interval to finish when the concurrency has not reached the upper limit.

`label_keep_max_second`

Default: `3 * 24 * 3600` (3 day)

IsMutable: true

MasterOnly: true

labels of finished or cancelled load jobs will be removed after `label_keep_max_second` ,

1. The removed labels can be reused.
2. Set a short time will lower the FE memory usage. (Because all load jobs' info is kept in memory before being removed)

In the case of high concurrent writes, if there is a large backlog of jobs and call frontend service failed, check the log. If the metadata write takes too long to lock, you can adjust this value to 12 hours, or 6 hours less

`streaming_label_keep_max_second`

Default: 43200 (12 hour)

IsMutable: true

MasterOnly: true

For some high-frequency load work, such as: INSERT, STREAMING LOAD, ROUTINE_LOAD_TASK. If it expires, delete the completed job or task.

`label_clean_interval_second`

Default: 1 * 3600 (1 hour)

Load label cleaner will run every *label_clean_interval_second* to clean the outdated jobs.

`label_regex_length`

Default Value: 128 (characters)

The maximum character length of the load label, the default is 128 characters.

`transaction_clean_interval_second`

Default：30

the transaction will be cleaned after transaction_clean_interval_second seconds if the transaction is visible or aborted we should make this interval as short as possible and each clean cycle as soon as possible

`sync_commit_interval_second`

The maximum time interval for committing transactions. If there is still data in the channel that has not been submitted after this time, the consumer will notify the channel to submit the transaction.

Default: 10 (seconds)

Is it possible to configure dynamically: true

Whether it is a configuration item unique to the Master FE node: true

`sync_checker_interval_second`

Data synchronization job running status check.

Default: 10 (s)

`max_sync_task_threads_num`

The maximum number of threads in the data synchronization job thread pool.

默认值：10

`min_sync_commit_size`

The minimum number of events that must be satisfied to commit a transaction. If the number of events received by Fe is less than it, it will continue to wait for the next batch of data until the time exceeds `sync_commit_interval_second`. The default value is 10000 events. If you want to modify this configuration, please make sure that this value is smaller than the `canal.instance.memory.buffer.size` configuration on the canal side

(default 16384), otherwise Fe will try to get the queue length longer than the store before ack More events cause the store queue to block until it times out.

Default: 10000

Is it possible to configure dynamically: true

Whether it is a configuration item unique to the Master FE node: true

`min_bytes_sync_commit`

The minimum data size required to commit a transaction. If the data size received by Fe is smaller than it, it will continue to wait for the next batch of data until the time exceeds `sync_commit_interval_second`. The default value is 15MB, if you want to modify this configuration, please make sure this value is less than the product of `canal.instance.memory.buffer.size` and `canal.instance.memory.buffer.memunit` on the canal side (default 16MB), otherwise Before the ack, Fe will try to obtain data that is larger than the store space, causing the store queue to block until it times out.

Default: `15*1024*1024` (15M)

Is it possible to configure dynamically: true

Whether it is a configuration item unique to the Master FE node: true

`max_bytes_sync_commit`

The maximum number of threads in the data synchronization job thread pool. There is only one thread pool in the entire FE, which is used to process all data synchronization tasks in the FE that send data to the BE. The implementation of the thread pool is in the `SyncTaskPool` class.

Default: 10

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: false

`enable_outfile_to_local`

Default：false

Whether to allow the outfile function to export the results to the local disk.

`export_tablet_num_per_task`

Default：5

IsMutable：true

MasterOnly：true

Number of tablets per export query plan

`export_task_default_timeout_second`

Default: 2 * 3600 (2 hour)

IsMutable：true

MasterOnly：true

Default timeout of export jobs.

`export_running_job_num_limit`

Default：5

IsMutable：true

MasterOnly：true

Limitation of the concurrency of running export jobs. Default is 5. 0 is unlimited

`export_checker_interval_second`

Default：5

Export checker’s running interval.

Log

`log_roll_size_mb`

Default：1024 （1G）

The max size of one sys log and audit log

`sys_log_dir`

Default: DorisFE.DORIS_HOME_DIR + “/log”

This specifies FE log dir. FE will produces 2 log files:

fe.log: all logs of FE process. fe.warn.log all WARNING and ERROR log of FE process.

`sys_log_level`

Default：INFO

log level: INFO, WARN, ERROR, FATAL

`sys_log_roll_num`

Default：10

Maximal FE log files to be kept within an sys_log_roll_interval. default is 10, which means there will be at most 10 log files in a day

`sys_log_verbose_modules`

Default：{}

Verbose modules. VERBOSE level is implemented by log4j DEBUG level.

eg： sys_log_verbose_modules = org.apache.doris.catalog This will only print debug log of files in package org.apache.doris.catalog and all its sub packages.

`sys_log_roll_interval`

Default：DAY

sys_log_roll_interval:

- DAY: log suffix is yyyyMMdd
- HOUR: log suffix is yyyyMMddHH

`sys_log_delete_age`

Default：7d

default is 7 days, if log’s last modify time is 7 days ago, it will be deleted.

support format:

- 7d 7 day
- 10h 10 hours
- 60m 60 min
- 120s 120 seconds

`sys_log_roll_mode`

Default：SIZE-MB-1024

The size of the log split, split a log file every 1 G

`sys_log_enable_compress`

Default: false

If true, will compress fe.log & fe.warn.log by gzip

`audit_log_dir`

Default：DORIS_HOME_DIR + “/log”

audit_log_dir： This specifies FE audit log dir.. Audit log fe.audit.log contains all requests with related infos such as user, host, cost, status, etc

`audit_log_roll_num`

Default：90

Maximal FE audit log files to be kept within an audit_log_roll_interval.

`audit_log_modules`

Default：{“slow_query”, “query”, “load”, “stream_load”}

Slow query contains all queries which cost exceed *qe_slow_log_ms*

`qe_slow_log_ms`

Default: 5000 (5 seconds)

If the response time of a query exceed this threshold, it will be recorded in audit log as slow_query.

`audit_log_roll_interval`

Default：DAY

DAY: logsuffix is : yyyyMMdd HOUR: logsuffix is : yyyyMMddHH

`audit_log_delete_age`

Default：30d

default is 30 days, if log’s last modify time is 30 days ago, it will be deleted.

support format: * 7d 7 day * 10h 10 hours * 60m 60 min * 120s 120 seconds

`audit_log_enable_compress`

Default: false

If true, will compress fe.audit.log by gzip

`nereids_trace_log_dir`

Default: DorisFE.DORIS_HOME_DIR + “/log/nereids_trace”

Used to specify the directory of the nereids trace log

Storage

`min_replication_num_per_tablet`

Default: 1

Used to set minimal number of replication per tablet.

`max_replication_num_per_tablet`

Default: 32767

Used to set maximal number of replication per tablet.

`default_db_data_quota_bytes`

Default：8192PB

IsMutable：true

MasterOnly：true

Used to set the default database data quota size. To set the quota size of a single database, you can use:

```
Set the database data quota, the unit is:B/K/KB/M/MB/G/GB/T/TB/P/PB
ALTER DATABASE db_name SET DATA QUOTA quota;
View configuration
show data  (Detail: HELP SHOW DATA)
```

`default_db_replica_quota_size`

Default: 1073741824

IsMutable：true

MasterOnly：true

Used to set the default database replica quota. To set the quota size of a single database, you can use:

```
Set the database replica quota
ALTER DATABASE db_name SET REPLICA QUOTA quota;
View configuration
show data  (Detail: HELP SHOW DATA)
```

`recover_with_empty_tablet`

Default：false

IsMutable：true

MasterOnly：true

In some very special circumstances, such as code bugs, or human misoperation, etc., all replicas of some tablets may be lost. In this case, the data has been substantially lost. However, in some scenarios, the business still hopes to ensure that the query will not report errors even if there is data loss, and reduce the perception of the user layer. At this point, we can use the blank Tablet to fill the missing replica to ensure that the query can be executed normally.

Set to true so that Doris will automatically use blank replicas to fill tablets which all replicas have been damaged or missing

`min_clone_task_timeout_sec And max_clone_task_timeout_sec`

Default: Minimum 3 minutes, maximum two hours

IsMutable：true

MasterOnly：true

Can cooperate with `mix_clone_task_timeout_sec` to control the maximum and minimum timeout of a clone task. Under normal circumstances, the timeout of a clone task is estimated by the amount of data and the minimum transfer rate (5MB/s). In some special cases, these two configurations can be used to set the upper and lower bounds of the clone task timeout to ensure that the clone task can be completed successfully.

`disable_storage_medium_check`

Default：false

IsMutable：true

MasterOnly：true

If disable_storage_medium_check is true, ReportHandler would not check tablet’s storage medium and disable storage cool down function, the default value is false. You can set the value true when you don’t care what the storage medium of the tablet is.

`decommission_tablet_check_threshold`

Default：5000

IsMutable：true

MasterOnly：true

This configuration is used to control whether the Master FE need to check the status of tablets on decommissioned BE. If the size of tablets on decommissioned BE is lower than this threshold, FE will start a periodic check, if all tablets on decommissioned BE have been recycled, FE will drop this BE immediately.

For performance consideration, please don’t set a very high value for this configuration.

`partition_rebalance_max_moves_num_per_selection`

Default：10

IsMutable：true

MasterOnly：true

Valid only if use PartitionRebalancer,

`partition_rebalance_move_expire_after_access`

Default：600 (s)

IsMutable：true

MasterOnly：true

Valid only if use PartitionRebalancer. If this changed, cached moves will be cleared

`tablet_rebalancer_type`

Default：BeLoad

MasterOnly：true

Rebalancer type(ignore case): BeLoad, Partition. If type parse failed, use BeLoad as default

`max_balancing_tablets`

Default：100

IsMutable：true

MasterOnly：true

if the number of balancing tablets in TabletScheduler exceed max_balancing_tablets, no more balance check

`max_scheduling_tablets`

Default：2000

IsMutable：true

MasterOnly：true

if the number of scheduled tablets in TabletScheduler exceed max_scheduling_tablets skip checking.

`disable_balance`

Default：false

IsMutable：true

MasterOnly：true

if set to true, TabletScheduler will not do balance.

`disable_disk_balance`

Default：true

IsMutable：true

MasterOnly：true

if set to true, TabletScheduler will not do disk balance.

`balance_load_score_threshold`

Default: 0.1 (10%)

IsMutable：true

MasterOnly：true

the threshold of cluster balance score, if a backend's load score is 10% lower than average score, this backend will be marked as LOW load, if load score is 10% higher than average score, HIGH load will be marked

`capacity_used_percent_high_water`

Default: 0.75 (75%)

IsMutable：true

MasterOnly：true

The high water of disk capacity used percent. This is used for calculating load score of a backend

`clone_distribution_balance_threshold`

Default: 0.2

IsMutable：true

MasterOnly：true

Balance threshold of num of replicas in Backends.

`clone_capacity_balance_threshold`

Default: 0.2

IsMutable：true

MasterOnly：true

- Balance threshold of data size in BE.

  The balance algorithm is:

  1. Calculate the average used capacity(AUC) of the entire cluster. (total data size / total backends num)
  2. The high water level is (AUC * (1 + clone_capacity_balance_threshold))
  3. The low water level is (AUC * (1 - clone_capacity_balance_threshold))
  4. The Clone checker will try to move replica from high water level BE to low water level BE.

`disable_colocate_balance`

Default：false

IsMutable：true

MasterOnly：true

This configs can set to true to disable the automatic colocate tables's relocate and balance. If 'disable_colocate_balance' is set to true, ColocateTableBalancer will not relocate and balance colocate tables.

Attention:

1. Under normal circumstances, there is no need to turn off balance at all.
2. Because once the balance is turned off, the unstable colocate table may not be restored
3. Eventually the colocate plan cannot be used when querying.

`balance_slot_num_per_path`

Default: 1

IsMutable：true

MasterOnly：true

Default number of slots per path during balance.

`disable_tablet_scheduler`

Default:false

IsMutable：true

MasterOnly：true

If set to true, the tablet scheduler will not work, so that all tablet repair/balance task will not work.

`enable_force_drop_redundant_replica`

Default: false

Dynamically configured: true

Only for Master FE: true

If set to true, the system will immediately drop redundant replicas in the tablet scheduling logic. This may cause some load jobs that are writing to the corresponding replica to fail, but it will speed up the balance and repair speed of the tablet. When there are a large number of replicas waiting to be balanced or repaired in the cluster, you can try to set this config to speed up the balance and repair of replicas at the expense of partial load success rate.

`colocate_group_relocate_delay_second`

Default: 1800

Dynamically configured: true

Only for Master FE: true

The relocation of a colocation group may involve a large number of tablets moving within the cluster. Therefore, we should use a more conservative strategy to avoid relocation of colocation groups as much as possible. Relocation usually occurs after a BE node goes offline or goes down. This parameter is used to delay the determination of BE node unavailability. The default is 30 minutes, i.e., if a BE node recovers within 30 minutes, relocation of the colocation group will not be triggered.

`allow_replica_on_same_host`

Default: false

Dynamically configured: false

Only for Master FE: false

Whether to allow multiple replicas of the same tablet to be distributed on the same host. This parameter is mainly used for local testing, to facilitate building multiple BEs to test certain multi-replica situations. Do not use it for non-test environments.

`repair_slow_replica`

Default: false

IsMutable：true

MasterOnly: true

If set to true, the replica with slower compaction will be automatically detected and migrated to other machines. The detection condition is that the version count of the fastest replica exceeds the value of

min_version_count_indicate_replica_compaction_too_slow, and the ratio of the version count difference from the fastest replica exceeds the value of valid_version_count_delta_ratio_between_replicas

min_version_count_indicate_replica_compaction_too_slow

Default: 200

Dynamically configured: true

Only for Master FE: false

The version count threshold used to judge whether replica compaction is too slow

skip_compaction_slower_replica

Default: true

Dynamically configured: true

Only for Master FE: false

If set to true, the compaction slower replica will be skipped when select get queryable replicas

valid_version_count_delta_ratio_between_replicas

Default: 0.5

Dynamically configured: true

Only for Master FE: true

The valid ratio threshold of the difference between the version count of the slowest replica and the fastest replica. If repair_slow_replica is set to true, it is used to determine whether to repair the slowest replica

min_bytes_indicate_replica_too_large

Default: 2 * 1024 * 1024 * 1024 (2G)

Dynamically configured: true

Only for Master FE: true

The data size threshold used to judge whether replica is too large

schedule_slot_num_per_hdd_path

Default：4

the default slot number per path in tablet scheduler for hdd , remove this config and dynamically adjust it by clone task statistic

`schedule_slot_num_per_ssd_path`

Default：8

the default slot number per path in tablet scheduler for ssd , remove this config and dynamically adjust it by clone task statistic

`tablet_repair_delay_factor_second`

Default：60 （s）

IsMutable：true

MasterOnly：true

the factor of delay time before deciding to repair tablet.

- if priority is VERY_HIGH, repair it immediately.
- HIGH, delay tablet_repair_delay_factor_second * 1;
- NORMAL: delay tablet_repair_delay_factor_second * 2;
- LOW: delay tablet_repair_delay_factor_second * 3;

`tablet_stat_update_interval_second`

Default：300（5min）

update interval of tablet stat, All frontends will get tablet stat from all backends at each interval

`storage_flood_stage_usage_percent`

Default：95 （95%）

IsMutable：true

MasterOnly：true

`storage_flood_stage_left_capacity_bytes`

Default：`1 * 1024 * 1024 * 1024` (1GB)

IsMutable：true

MasterOnly：true

If capacity of disk reach the ‘storage_flood_stage_usage_percent’ and ‘storage_flood_stage_left_capacity_bytes’, the following operation will be rejected:

1. load job
2. restore job

`storage_high_watermark_usage_percent`

Default：85 (85%)

IsMutable：true

MasterOnly：true

`storage_min_left_capacity_bytes`

Default： `2 * 1024 * 1024 * 1024` (2GB)

IsMutable：true

MasterOnly：true

‘storage_high_watermark_usage_percent’ limit the max capacity usage percent of a Backend storage path. ‘storage_min_left_capacity_bytes’ limit the minimum left capacity of a Backend storage path. If both limitations are reached, this storage path can not be chose as tablet balance destination. But for tablet recovery, we may exceed these limit for keeping data integrity as much as possible.

`catalog_trash_expire_second`

Default: 86400L (1 day)

IsMutable：true

MasterOnly：true

After dropping database(table/partition), you can recover it by using RECOVER stmt. And this specifies the maximal data retention time. After time, the data will be deleted permanently.

`storage_cooldown_second`

> **Tips**
>
> This feature is deprecated since the Apache Doris 2.0 version

Default：`30 * 24 * 3600L` （30 day）

When create a table(or partition), you can specify its storage medium(HDD or SSD). If set to SSD, this specifies the default duration that tablets will stay on SSD. After that, tablets will be moved to HDD automatically. You can set storage cooldown time in CREATE TABLE stmt.

`default_storage_medium`

Default：HDD

When create a table(or partition), you can specify its storage medium(HDD or SSD). If not set, this specifies the default medium when create.

`enable_storage_policy`

- Whether to enable the Storage Policy feature. This config allows users to separate hot and cold data. Default: false

Is it possible to dynamically configure: true

Is it a configuration item unique to the Master FE node: true

`check_consistency_default_timeout_second`

Default: 600 (10 minutes)

IsMutable：true

MasterOnly：true

Default timeout of a single consistency check task. Set long enough to fit your tablet size

`consistency_check_start_time`

Default：23

IsMutable：true

MasterOnly：true

Consistency check start time

Consistency checker will run from *consistency_check_start_time* to *consistency_check_end_time*.

If the two times are the same, no consistency check will be triggered.

`consistency_check_end_time`

Default：23

IsMutable：true

MasterOnly：true

Consistency check end time

Consistency checker will run from *consistency_check_start_time* to *consistency_check_end_time*.

If the two times are the same, no consistency check will be triggered.

`replica_delay_recovery_second`

Default：0

IsMutable：true

MasterOnly：true

the minimal delay seconds between a replica is failed and fe try to recovery it using clone.

`tablet_create_timeout_second`

Default：1（s）

IsMutable：true

MasterOnly：true

Maximal waiting time for creating a single replica.

eg. if you create a table with #m tablets and #n replicas for each tablet, the create table request will run at most (m *n* tablet_create_timeout_second) before timeout.

`tablet_delete_timeout_second`

Default：2

IsMutable：true

MasterOnly：true

Same meaning as *tablet_create_timeout_second*, but used when delete a tablet.

`delete_job_max_timeout_second`

Default: 300(s)

Mutable: true

Master only: true

Maximal timeout for delete job, in seconds.

`alter_table_timeout_second`

Default: 86400 * 30 (1 month)

IsMutable：true

MasterOnly：true

Maximal timeout of ALTER TABLE request. Set long enough to fit your table data size.

`max_replica_count_when_schema_change`

The maximum number of replicas allowed when OlapTable is doing schema changes. Too many replicas will lead to FE OOM.

Default: 100000

Is it possible to configure dynamically: true

Whether it is a configuration item unique to the Master FE node: true

`history_job_keep_max_second`

Default：`7 * 24 * 3600` （7 day）

IsMutable：true

MasterOnly：true

The max keep time of some kind of jobs. like schema change job and rollup job.

`max_create_table_timeout_second`

Default：60 （s）

IsMutable：true

MasterOnly：true

In order not to wait too long for create table(index), set a max timeout.

External Table

`file_scan_node_split_num`

Default：128

IsMutable：true

MasterOnly：false

multi catalog concurrent file scanning threads

`file_scan_node_split_size`

Default：`256 * 1024 * 1024`

IsMutable：true

MasterOnly：false

multi catalog concurrent file scan size

`enable_odbc_mysql_broker_table`

Default：false

IsMutable：true

MasterOnly：false

Starting from version 2.1, we no longer support create ODBC, JDBC and broker external table. For odbc and mysql external table, use JDBC table or JDBC catalog instead. For broker table, use table valued function instead.

`max_hive_partition_cache_num`

The maximum number of caches for the hive partition.

Default: 100000

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: false

`hive_metastore_client_timeout_second`

The default connection timeout for hive metastore.

Default: 10

Is it possible to dynamically configure: true

Is it a configuration item unique to the Master FE node: true

`max_external_cache_loader_thread_pool_size`

Maximum thread pool size for loading external meta cache.

Default: 10

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: false

`max_external_file_cache_num`

Maximum number of file cache to use for external tables.

Default: 100000

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: false

`max_external_schema_cache_num`

Maximum number of schema cache to use for external external tables.

Default: 10000

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: false

`external_cache_expire_time_minutes_after_access`

Set how long the data in the cache expires after the last access. The unit is minutes. Applies to External Schema Cache as well as Hive Partition Cache.

Default: 1440

Is it possible to dynamically configure: false

Is it a configuration item unique to the Master FE node: false

`es_state_sync_interval_second`

Default：10

fe will call es api to get es index shard info every es_state_sync_interval_secs

External Resources

`dpp_hadoop_client_path`

Default：/lib/hadoop-client/hadoop/bin/hadoop

`dpp_bytes_per_reduce`

Default：`100 * 1024 * 1024L` (100M)

`dpp_default_cluster`

Default：palo-dpp

`dpp_default_config_str`

Default：{ hadoop_configs :
‘mapred.job.priority=NORMAL;mapred.job.map.capacity=50;mapred.job.reduce.capacity=50;mapred.hce.repla

ce.streaming=false;abaci.long.stored.job=true;dce.shuffle.enable=false;dfs.client.authserver.force_stop=true;dfs.client.auth.method=0' }

`dpp_config_str`

Default：{ palo-dpp : { hadoop_palo_path : '/dir', hadoop_configs : 'fs.default.name=hdfs://host:port;mapred.job.tracker=host:port;hadoop.job.ugi=user,password' } }

`yarn_config_dir`

Default: DorisFE.DORIS_HOME_DIR + "/lib/yarn-config"

Default yarn config file directory , Each time before running the yarn command, we need to check that the config file exists under this path, and if not, create them.

`yarn_client_path`

Default：DORIS_HOME_DIR + "/lib/yarn-client/hadoop/bin/yarn"

Default yarn client path

`spark_launcher_log_dir`

Default： sys_log_dir + "/spark_launcher_log"

The specified spark launcher log dir

`spark_resource_path`

Default：none

Default spark dependencies path

`spark_home_default_dir`

Default：DORIS_HOME_DIR + "/lib/spark2x"

Default spark home dir

`spark_dpp_version`

Default: 1.0.0

Default spark dpp version

Else

`tmp_dir`

Default: DorisFE.DORIS_HOME_DIR + “/temp_dir”

temp dir is used to save intermediate results of some process, such as backup and restore process. file in this dir will be cleaned after these process is finished.

`custom_config_dir`

Default: DorisFE.DORIS_HOME_DIR + “/conf”

Custom configuration file directory

Configure the location of the `fe_custom.conf` file. The default is in the `conf/` directory.

In some deployment environments, the `conf/` directory may be overwritten due to system upgrades. This will cause the user modified configuration items to be overwritten. At this time, we can store `fe_custom.conf` in another specified directory to prevent the configuration file from being overwritten.

`plugin_dir`

Default：DORIS_HOME + “/plugins

plugin install directory

`plugin_enable`

Default:true

IsMutable：true

MasterOnly：true

Whether the plug-in is enabled, enabled by default

`small_file_dir`

Default：DORIS_HOME_DIR/small_files

Save small files

`max_small_file_size_bytes`

Default：1M

IsMutable：true

MasterOnly：true

The max size of a single file store in SmallFileMgr

`max_small_file_number`

Default：100

IsMutable：true

MasterOnly：true

The max number of files store in SmallFileMgr

`enable_metric_calculator`

Default：true

If set to true, metric collector will be run as a daemon timer to collect metrics at fix interval

`report_queue_size`

Default： 100

IsMutable：true

MasterOnly：true

This threshold is to avoid piling up too many report task in FE, which may cause OOM exception. In some large Doris cluster, eg: 100 Backends with ten million replicas, a tablet report may cost several seconds after some modification of metadata(drop partition, etc..). And one Backend will report tablets info every 1 min, so unlimited receiving reports is unacceptable. we will optimize the processing speed of tablet report in future, but now, just discard the report if queue size exceeding limit. Some online time cost: 1. disk report: 0-1 msta 2. sk report: 0-1 ms 3. tablet report 4. 10000 replicas: 200ms

`backup_job_default_timeout_ms`

Default: 86400 * 1000 (1 day)

IsMutable：true

MasterOnly：true

default timeout of backup job

`backup_upload_task_num_per_be`

Default：3

IsMutable：true

MasterOnly：true

The max number of upload tasks assigned to each be during the backup process, the default value is 3.

`restore_download_task_num_per_be`

Default：3

IsMutable：true

MasterOnly：true

The max number of download tasks assigned to each be during the restore process, the default value is 3.

`max_backup_restore_job_num_per_db`

Default: 10

This configuration is mainly used to control the number of backup/restore tasks recorded in each database.

`max_backup_tablets_per_job`

Default: 300000

IsMutable：true

MasterOnly：true

Control the max num of tablets per backup job involved, to avoid FE OOM caused by saving too much metadata.

> **TIPS**
>
> This configuration is supported since the Apache Doris 2.1.6 version

`enable_quantile_state_type`

Default：false

IsMutable：true

MasterOnly：true

Whether to enable the quantile_state data type

`enable_date_conversion`

Default：true

IsMutable：true

MasterOnly：true

FE will convert date/datetime to datev2/datetimev2(0) automatically.

`enable_decimal_conversion`

Default：true

IsMutable：true

MasterOnly：true

FE will convert DecimalV2 to DecimalV3 automatically.

`proxy_auth_magic_prefix`

Default：x@8

`proxy_auth_enable`

Default：false

`enable_func_pushdown`

Default：true

IsMutable：true

MasterOnly：false

Whether to push the filter conditions with functions down to MYSQL, when execute query of ODBC, JDBC external tables

`jdbc_drivers_dir`

Default: `${DORIS_HOME}/jdbc_drivers`;

IsMutable：false

MasterOnly：false

The default dir to put jdbc drivers.

`max_error_tablet_of_broker_load`

Default: 3;

IsMutable：true

MasterOnly：true

Maximum number of error tablet showed in broker load.

`default_db_max_running_txn_num`

Default：-1

IsMutable：true

MasterOnly：true

Used to set the default database transaction quota size.

The default value setting to -1 means using `max_running_txn_num_per_db` instead of `default_db_max_running_txn_num`.

To set the quota size of a single database, you can use:

```
Set the database transaction quota
ALTER DATABASE db_name SET TRANSACTION QUOTA quota;
View configuration
show data  (Detail: HELP SHOW DATA)
```

`prefer_compute_node_for_external_table`

Default：false

IsMutable：true

MasterOnly：false

If set to true, query on external table will prefer to assign to compute node. And the max number of compute node is controlled by `min_backend_num_for_external_table`. If set to false, query on external table will assign to any node.

`min_backend_num_for_external_table`

Default：3

IsMutable：true

MasterOnly：false

Only take effect when `prefer_compute_node_for_external_table` is true. If the compute node number is less than this value, query on external table will try to get some mix node to assign, to let the total number of node reach this value. If the compute node number is larger than this value, query on external table will assign to compute node only.

`infodb_support_ext_catalog`

> **Tips**
>
> This configuration is supported since the Apache Doris 1.2.4 version

Default: false

IsMutable: true

MasterOnly: false

If false, when select from tables in information_schema database, the result will not contain the information of the table in external catalog. This is to avoid query time when external catalog is not reachable.

`enable_query_hit_stats`

Default: false

IsMutable: true

MasterOnly: false

Controls whether to enable query hit statistics. The default is false.

`div_precision_increment`

Default: 4

This variable indicates the number of digits by which to increase the scale of the result of division operations performed with the / operator.

`enable_convert_light_weight_schema_change`

Default：true

Temporary configuration option. After it is enabled, a background thread will be started to automatically modify all olap tables to light schema change. The modification results can be viewed through the command `show convert_light_schema_change [from db]`, and the conversion results of all non-light schema change tables will be displayed.

`disable_local_deploy_manager_drop_node`

Default：true

Forbid LocalDeployManager drop nodes to prevent errors in the cluster.info file from causing nodes to be dropped.

`mysqldb_replace_name`

Default: mysql

To ensure compatibility with the MySQL ecosystem, Doris includes a built-in database called mysql. If this database conflicts with a user's own database, please modify this field to replace the name of the Doris built-in MySQL database with a different name.

`max_auto_partition_num`

Default value: 2000

For auto-partitioned tables to prevent users from accidentally creating a large number of partitions, the number of partitions allowed per OLAP table is `max_auto_partition_num`. Default 2000.

BE Configuration

BE Configuration

This document mainly introduces the relevant configuration items of BE.

The BE configuration file `be.conf` is usually stored in the `conf/` directory of the BE deployment path. In version 0.14, another configuration file `be_custom.conf` will be introduced. The configuration file is used to record the configuration items that are dynamically configured and persisted by the user during operation.

After the BE process is started, it will read the configuration items in `be.conf` first, and then read the configuration items in `be_custom.conf`. The configuration items in `be_custom.conf` will overwrite the same configuration items in `be.conf`.

The location of the `be_custom.conf` file can be configured in `be.conf` through the `custom_config_dir` configuration item.

View configuration items

Users can view the current configuration items by visiting BE's web page:

`http://be_host:be_webserver_port/varz`

Set configuration items

There are two ways to configure BE configuration items:

1. Static configuration

   Add and set configuration items in the `conf/be.conf` file. The configuration items in `be.conf` will be read when BE starts. Configuration items not in `be.conf` will use default values.

2. Dynamic configuration

   After BE starts, the configuration items can be dynamically set with the following commands.

   ```
   curl -X POST http://{be_ip}:{be_http_port}/api/update_config?{key}={value}
   ```

   In version 0.13 and before, the configuration items modified in this way will become invalid after the BE process restarts. In 0.14 and later versions, the modified configuration can be persisted through the following command. The modified configuration items are stored in the `be_custom.conf` file.

   ```
   curl -X POST http://{be_ip}:{be_http_port}/api/update_config?{key}={value}\&persist=true
   ```

Examples

1. Modify `max_base_compaction_threads` statically

   By adding in the `be.conf` file:

   ```
   max_base_compaction_threads=5
   ```

   Then restart the BE process to take effect the configuration.

2. Modify `streaming_load_max_mb` dynamically

   After BE starts, the configuration item `streaming_load_max_mb` is dynamically set by the following command:

   ```
   curl -X POST http://{be_ip}:{be_http_port}/api/update_config?streaming_load_max_mb=1024
   ```

   The return value is as follows, indicating that the setting is successful.

   ```
   {
       "status": "OK",
       "msg": ""
   }
   ```

   The configuration will become invalid after the BE restarts. If you want to persist the modified results, use the following command:

   ```
   curl -X POST http://{be_ip}:{be_http_port}/api/update_config?streaming_load_max_mb=1024\&persist=true
   ```

Configurations

Services

`be_port`

- Type: int32
- Description: The port of the thrift server on BE which used to receive requests from FE
- Default value: 9060

`heartbeat_service_port`

- Type: int32
- Description: Heartbeat service port (thrift) on BE, used to receive heartbeat from FE
- Default value: 9050

`webserver_port`

- Type: int32

- Description: Service port of http server on BE
- Default value: 8040

`brpc_port`

- Type: int32
- Description: The port of BRPC on BE, used for communication between BEs
- Default value: 8060

`arrow_flight_sql_port`

- Type: int32
- Description: The port of Arrow Flight SQL server on BE, used for communication between Arrow Flight Client and BE
- Default value: -1

`enable_https`

- Type: bool
- Description: Whether https is supported. If so, configure `ssl_certificate_path` and `ssl_private_key_path` in be.conf.
- Default value: false

`priority_networks`

- Description: Declare a selection strategy for those servers with many IPs. Note that at most one ip should match this list. This is a semicolon-separated list in CIDR notation, such as 10.10.10.0/24. If there is no IP matching this rule, one will be randomly selected
- Default value: blank

`storage_root_path`

- Type: string
- Description: data root path, separate by ';'.you can specify the storage medium of each root path, HDD or SSD. you can add capacity limit at the end of each root path, separate by ',' .If the user does not use a mix of SSD and HDD disks, they do not need to configure the configuration methods in Example 1 and Example 2 below, but only need to specify the storage directory; they also do not need to modify the default storage media configuration of FE.

  eg.1: `storage_root_path=/home/disk1/doris.HDD;/home/disk2/doris.SSD;/home/disk2/doris`

  - 1./home/disk1/doris.HDD, indicates that the storage medium is HDD;
  - 2./home/disk2/doris.SSD, indicates that the storage medium is SSD;
  - 3./home/disk2/doris, indicates that the storage medium is HDD by default

  eg.2: `storage_root_path=/home/disk1/doris,medium:hdd;/home/disk2/doris,medium:ssd`

  - 1./home/disk1/doris,medium:hdd, indicates that the storage medium is HDD;

    - 2./home/disk2/doris,medium:ssd, indicates that the storage medium is SSD;

- Default value: ${DORIS_HOME}/storage

`heartbeat_service_thread_count`

- Type: int32
- Description: The number of threads that execute the heartbeat service on BE. the default is 1, it is not recommended to modify
- Default value: 1

`ignore_broken_disk`

- Type: bool

- Description: When BE starts, check `storage_root_path` All paths under configuration.

    - `ignore_broken_disk=true`

    If the path does not exist or the file (bad disk) cannot be read or written under the path, the path will be ignored. If there are other available paths, the startup will not be interrupted.

    - `ignore_broken_disk=false`

    If the path does not exist or the file (bad disk) cannot be read or written under the path, the system will abort the startup failure and exit.

- Default value: false

`mem_limit`

- Type: string
- Description: Limit the percentage of the server's maximum memory used by the BE process. It is used to prevent BE memory from occupying too the machine's memory. This parameter must be greater than 0. When the percentage is greater than 100%, the value will default to 100%.
- Default value: 90%

`cluster_id`

- Type: int32
- Description: Configure the cluster id to which the BE belongs.
    - This value is usually delivered by the FE to the BE by the heartbeat, no need to configure. When it is confirmed that a BE belongs to a certain Doris cluster, it can be configured. The cluster_id file under the data directory needs to be modified to make sure same as this parament.
- Default value: -1

`custom_config_dir`

- Description: Configure the location of the `be_custom.conf` file. The default is in the `conf/` directory.

    - In some deployment environments, the `conf/` directory may be overwritten due to system upgrades. This will cause the user modified configuration items to be overwritten. At this time, we can store `be_custom.conf` in another specified directory to prevent the configuration file from being overwritten.
- Default value: blank

`trash_file_expire_time_sec`

- Description: The interval for cleaning the recycle bin is 24 hours. When the disk space is insufficient, the file retention period under trash may not comply with this parameter
- Default value: 86400

`es_http_timeout_ms`

- Description: The timeout period for connecting to ES via http.
- Default value: 5000 (ms)

`es_scroll_keepalive`

- Description: es scroll keep-alive hold time
- Default value: 5 (m)

`external_table_connect_timeout_sec`

- Type: int32
- Description: The timeout when establishing connection with external table such as ODBC table.
- Default value: 5 seconds

`status_report_interval`

- Description: Interval between profile reports
- Default value: 5 seconds

`brpc_max_body_size`

- Description: This configuration is mainly used to modify the parameter `max_body_size` of brpc.
    - Sometimes the query fails and an error message of `body_size is too large` will appear in the BE log. This may happen when the SQL mode is “multi distinct + no group by + more than 1T of data”.This error indicates that the packet size of brpc exceeds the configured value. At this time, you can avoid this error by increasing the configuration.

`brpc_socket_max_unwritten_bytes`

- Description: This configuration is mainly used to modify the parameter `socket_max_unwritten_bytes` of brpc.
    - Sometimes the query fails and an error message of `The server is overcrowded` will appear in the BE log. This means there are too many messages to buffer at the sender side, which may happen

when the SQL needs to send large bitmap value. You can avoid this error by increasing the configuration.

`transfer_large_data_by_brpc`

- Type: bool
- Description: This configuration is used to control whether to serialize the protoBuf request and embed the Tuple/Block data into the controller attachment and send it through http brpc when the length of the Tuple/Block data is greater than 1.8G. To avoid errors when the length of the protoBuf request exceeds 2G: Bad request, error_text=[E1003]Fail to compress request. In the past version, after putting Tuple/Block data in the attachment, it was sent through the default baidu_std brpc, but when the attachment exceeds 2G, it will be truncated. There is no 2G limit for sending through http brpc.
- Default value: true

`brpc_num_threads`

- Description: This configuration is mainly used to modify the number of bthreads for brpc. The default value is set to 256.
    - User can set this configuration to a larger value to get better QPS performance. For more information, please refer to `https://github.com/apache/brpc/blob/master/docs/cn/benchmark.md`
- Default value: 256

`thrift_rpc_timeout_ms`

- Description: thrift default timeout time
- Default value: 60000

`thrift_client_retry_interval_ms`

- Type: int64
- Description: Used to set retry interval for thrift client in be to avoid avalanche disaster in fe thrift server, the unit is ms.
- Default value: 1000

`thrift_connect_timeout_seconds`

- Description: The default thrift client connection timeout time
- Default value: 3 (s)

`thrift_server_type_of_fe`

- Type: string
- Description:This configuration indicates the service model used by FE's Thrift service. The type is string and is case-insensitive. This parameter needs to be consistent with the setting of fe's thrift_server_type parameter. Currently there are two values for this parameter, `THREADED` and `THREAD_POOL`.
    - If the parameter is `THREADED`, the model is a non-blocking I/O model.

    - If the parameter is `THREAD_POOL`, the model is a blocking I/O model.

`thrift_max_message_size`

:::tip Tips This configuration is supported since the Apache Doris 2.1.4 version :::

Default: 100MB

The maximum size of a (received) message of the thrift server, in bytes. If the size of the message sent by the client exceeds this limit, the Thrift server will reject the request and close the connection. As a result, the client will encounter the error: “connection has been closed by peer.” In this case, you can try increasing this parameter.

`txn_commit_rpc_timeout_ms`

- Description:txn submit rpc timeout
- Default value: 60,000 (ms)

`txn_map_shard_size`

- Description: txn_map_lock fragment size, the value is 2^n, n=0,1,2,3,4. This is an enhancement to improve the performance of managing txn
- Default value: 128

`txn_shard_size`

- Description: txn_lock shard size, the value is 2^n, n=0,1,2,3,4, this is an enhancement function that can improve the performance of submitting and publishing txn
- Default value: 1024

`unused_rowset_monitor_interval`

- Description: Time interval for clearing expired Rowset
- Default value: 30 (s)

`max_client_cache_size_per_host`

- Description: The maximum number of client caches per host. There are multiple client caches in BE, but currently we use the same cache size configuration. If necessary, use different configurations to set up different client-side caches
- Default value: 10

`string_type_length_soft_limit_bytes`

- Type: int32
- Description: The soft limit of the maximum length of String type.
- Default value: 1,048,576

`big_column_size_buffer`

- Type: int64
- Description: When using the odbc external table, if a column type of the odbc source table is HLL, CHAR or VARCHAR, and the length of the column value exceeds this value, the query will report an error 'column value length longer than buffer length'. You can increase this value
- Default value: 65535

`small_column_size_buffer`

- Type: int64
- Description: When using the odbc external table, if a column type of the odbc source table is not HLL, CHAR or VARCHAR, and the length of the column value exceeds this value, the query will report an error 'column value length longer than buffer length'. You can increase this value
- Default value: 100

`jsonb_type_length_soft_limit_bytes`

- Type: int32
- Description: The soft limit of the maximum length of JSONB type.
- Default value: 1,048,576

Query

`fragment_pool_queue_size`

- Description: The upper limit of query requests that can be processed on a single node
- Default value: 4096

`fragment_pool_thread_num_min`

- Description: Query the number of threads. By default, the minimum number of threads is 64.
- Default value: 64

`fragment_pool_thread_num_max`

- Description: Follow up query requests create threads dynamically, with a maximum of 512 threads created.
- Default value: 2048

`doris_max_scan_key_num`

- Type: int
- Description: Used to limit the maximum number of scan keys that a scan node can split in a query request. When a conditional query request reaches the scan node, the scan node will try to split the conditions related to the key column in the query condition into multiple scan key ranges. After that, these scan key ranges will be assigned to multiple scanner threads for data scanning. A larger value

usually means that more scanner threads can be used to increase the parallelism of the scanning operation. However, in high concurrency scenarios, too many threads may bring greater scheduling overhead and system load, and will slow down the query response speed. An empirical value is 50. This configuration can be configured separately at the session level. For details, please refer to the description of `max_scan_key_num` in Variables.
    - When the concurrency cannot be improved in high concurrency scenarios, try to reduce this value and observe the impact.
- Default value: 48

`doris_scan_range_row_count`

- Type: int32
- Description: When BE performs data scanning, it will split the same scanning range into multiple ScanRanges. This parameter represents the scan data range of each ScanRange. This parameter can limit the time that a single OlapScanner occupies the io thread.
- Default value: 524288

`doris_scanner_row_num`

- Description: The maximum number of data rows returned by each scanning thread in a single execution
- Default value: 16384

`doris_scanner_row_bytes`

- Description: single read execute fragment row bytes
    - Note: If there are too many columns in the table, you can adjust this config if you encounter a `select *` stuck
- Default value: 10485760

`doris_scanner_thread_pool_queue_size`

- Type: int32
- Description: The queue length of the Scanner thread pool. In Doris' scanning tasks, each Scanner will be submitted as a thread task to the thread pool waiting to be scheduled, and after the number of submitted tasks exceeds the length of the thread pool queue, subsequent submitted tasks will be blocked until there is a empty slot in the queue.
- Default value: 102400

`doris_scanner_thread_pool_thread_num`

- Type: int32
- Description: The number of threads in the Scanner thread pool. In Doris' scanning tasks, each Scanner will be submitted as a thread task to the thread pool to be scheduled. This parameter determines the size of the Scanner thread pool.
- Default value: Depending on cpu cores. Equal to `max(48, 2 * num_of_cpu_cores)`

`doris_max_remote_scanner_thread_pool_thread_num`

- Type: int32
- Description: Max thread number of Remote scanner thread pool. Remote scanner thread pool is used for scan task of all external data sources.
- Default: 512

`exchg_node_buffer_size_bytes`

- Type: int32
- Description: The size of the Buffer queue of the ExchangeNode node, in bytes. After the amount of data sent from the Sender side is larger than the Buffer size of ExchangeNode, subsequent data sent will block until the Buffer frees up space for writing.
- Default value: 10485760

`max_pushdown_conditions_per_column`

- Type: int
- Description: Used to limit the maximum number of conditions that can be pushed down to the storage engine for a single column in a query request. During the execution of the query plan, the filter conditions on some columns can be pushed down to the storage engine, so that the index information in the storage engine can be used for data filtering, reducing the amount of data that needs to be scanned by the query. Such as equivalent conditions, conditions in IN predicates, etc. In most cases, this parameter only affects queries containing IN predicates. Such as `WHERE colA IN (1,2,3,4, ...)`. A larger number means that more conditions in the IN predicate can be pushed to the storage engine, but too many conditions may cause an increase in random reads, and in some cases may reduce query efficiency. This configuration can be individually configured for session level. For details, please refer to the description of `max_pushdown_conditions_per_column` in Variables.
- Default value: 1024
- Example
    - The table structure is' id INT, col2 INT, col3 varchar (32),... '.
    - The query request is'WHERE id IN (v1, v2, v3, ...) #### `max_send_batch_parallelism_per_job`
- Type: int
- Description: Max send batch parallelism for OlapTableSink. The value set by the user for `send_batch_parallelism` is not allowed to exceed `max_send_batch_parallelism_per_job`, if exceed, the value of `send_batch_parallelism` would be `max_send_batch_parallelism_per_job`.
- Default value: 5

`doris_scan_range_max_mb`

- Type: int32
- Description: The maximum amount of data read by each OlapScanner.
- Default value: 1024

compaction

`disable_auto_compaction`

- Type: bool
- Description: Whether disable automatic compaction task
    - Generally it needs to be turned off. When you want to manually operate the compaction task in the debugging or test environment, you can turn on the configuration.
- Default value: false

`enable_vertical_compaction`

- Type: bool
- Description: Whether to enable vertical compaction
- Default value: true

`vertical_compaction_num_columns_per_group`

- Type: int32
- Description: In vertical compaction, column number for every group
- Default value: 5

`vertical_compaction_max_row_source_memory_mb`

- Type: int32
- Description: In vertical compaction, max memory usage for row_source_buffer,The unit is MB.
- Default value: 200

`vertical_compaction_max_segment_size`

- Type: int32
- Description: In vertical compaction, max dest segment file size, The unit is m bytes.
- Default value: 268435456

`enable_ordered_data_compaction`

- Type: bool
- Description: Whether to enable ordered data compaction
- Default value: true

`ordered_data_compaction_min_segment_size`

- Type: int32
- Description: In ordered data compaction, min segment size for input rowset, The unit is m bytes.
- Default value: 10485760

`max_base_compaction_threads`

git * Type: int32 * Description: The maximum of thread number in base compaction thread pool, -1 means one thread per disk. * Default value: 4

`generate_compaction_tasks_interval_ms`

- Description: Minimal interval (ms) to generate compaction tasks
- Default value: 10 (ms)

`base_compaction_min_rowset_num`

- Description: One of the triggering conditions of BaseCompaction: The limit of the number of Cumulative files to be reached. After reaching this limit, BaseCompaction will be triggered
- Default value: 5

`base_compaction_min_data_ratio`

- Description: One of the trigger conditions of BaseCompaction: Cumulative file size reaches the proportion of Base file
- Default value: 0.3 (30%)

`total_permits_for_compaction_score`

- Type: int64
- Description: The upper limit of “permits” held by all compaction tasks. This config can be set to limit memory consumption for compaction.
- Default value: 10000
- Dynamically modifiable: Yes

`compaction_promotion_size_mbytes`

- Type: int64
- Description: The total disk size of the output rowset of cumulative compaction exceeds this configuration size, and the rowset will be used for base compaction. The unit is m bytes.
    - Generally, if the configuration is less than 2G, in order to prevent the cumulative compression time from being too long, resulting in the version backlog.
- Default value: 1024

`compaction_promotion_ratio`

- Type: double
- Description: When the total disk size of the cumulative compaction output rowset exceeds the configuration ratio of the base version rowset, the rowset will be used for base compaction.
    - Generally, it is recommended that the configuration should not be higher than 0.1 and lower than 0.02.
- Default value: 0.05

`compaction_promotion_min_size_mbytes`

- Type: int64
- Description: If the total disk size of the output rowset of the cumulative compaction is lower than this configuration size, the rowset will not undergo base compaction and is still in the cumulative compaction process. The unit is m bytes.
    - Generally, the configuration is within 512m. If the configuration is too large, the size of the early base version is too small, and base compaction has not been performed.
- Default value: 128

`compaction_min_size_mbytes`

- Type: int64
- Description: When the cumulative compaction is merged, the selected rowsets to be merged have a larger disk size than this configuration, then they are divided and merged according to the level policy. When it is smaller than this configuration, merge directly. The unit is m bytes.
    - Generally, the configuration is within 128m. Over configuration will cause more cumulative compaction write amplification.
- Default value: 64

`default_rowset_type`

- Type: string
- Description: Identifies the storage format selected by BE by default. The configurable parameters are: "**ALPHA**", "**BETA**". Mainly play the following two roles
    - When the storage_format of the table is set to Default, select the storage format of BE through this configuration.
    - Select the storage format of when BE performing Compaction
- Default value: BETA

`cumulative_compaction_min_deltas`

- Description: Cumulative compaction strategy: the minimum number of incremental files
- Default value: 5

`cumulative_compaction_max_deltas`

- Description: Cumulative compaction strategy: the maximum number of incremental files
- Default value: 1000

`base_compaction_trace_threshold`

- Type: int32
- Description: Threshold to logging base compaction's trace information, in seconds
- Default value: 10

Base compaction is a long time cost background task, this configuration is the threshold to logging trace information. Trace information in log file looks like:

```
W0610 11:26:33.804431 56452 storage_engine.cpp:552] execute base compaction cost 0.00319222
BaseCompaction:546859:
  - filtered_rows: 0
   - input_row_num: 10
   - input_rowsets_count: 10
   - input_rowsets_data_size: 2.17 KB
   - input_segments_num: 10
   - merge_rowsets_latency: 100000.510ms
   - merged_rows: 0
   - output_row_num: 10
   - output_rowset_data_size: 224.00 B
   - output_segments_num: 1
0610 11:23:03.727535 (+     0us) storage_engine.cpp:554] start to perform base compaction
0610 11:23:03.728961 (+  1426us) storage_engine.cpp:560] found best tablet 546859
0610 11:23:03.728963 (+     2us) base_compaction.cpp:40] got base compaction lock
0610 11:23:03.729029 (+    66us) base_compaction.cpp:44] rowsets picked
0610 11:24:51.784439 (+108055410us) compaction.cpp:46] got concurrency lock and start to do compaction
0610 11:24:51.784818 (+   379us) compaction.cpp:74] prepare finished
0610 11:26:33.359265 (+101574447us) compaction.cpp:87] merge rowsets finished
0610 11:26:33.484481 (+125216us) compaction.cpp:102] output rowset built
0610 11:26:33.484482 (+     1us) compaction.cpp:106] check correctness finished
0610 11:26:33.513197 (+ 28715us) compaction.cpp:110] modify rowsets finished
0610 11:26:33.513300 (+   103us) base_compaction.cpp:49] compaction finished
0610 11:26:33.513441 (+   141us) base_compaction.cpp:56] unused rowsets have been moved to GC queue
```

`cumulative_compaction_trace_threshold`

- Type: int32
- Description: Threshold to logging cumulative compaction's trace information, in seconds
    - Similar to `base_compaction_trace_threshold`.
- Default value: 2

`compaction_task_num_per_disk`

- Type: int32
- Description: The number of compaction tasks which execute in parallel for a disk(HDD).
- Default value: 4

`compaction_task_num_per_fast_disk`

- Type: int32
- Description: The number of compaction tasks which execute in parallel for a fast disk(SSD).
- Default value: 8

`cumulative_compaction_rounds_for_each_base_compaction_round`

- Type: int32
- Description: How many rounds of cumulative compaction for each round of base compaction when compaction tasks generation.
- Default value: 9

`cumulative_compaction_policy`

- Type: string
- Description: Configure the merge strategy in the cumulative compression phase. Currently, two merge strategies are implemented, num_based and size_based
    - For details, “ordinary” is the initial version of the cumulative compression consolidation policy. After a cumulative compression, the base compression process is directly performed. size_The general policy is the optimized version of the ordinary policy. Version merging can only be performed when the disk volume of the rowset is the same order of magnitude. After merging, qualified rowsets are promoted to the base compaction stage. In the case of a large number of small batch imports, it can reduce the write magnification of base compact, balance the read magnification and space magnification, and reduce the data of file versions.
- Default value: size_based

`max_cumu_compaction_threads`

- Type: int32
- Description: The maximum of thread number in cumulative compaction thread pool, -1 means one thread per disk.
- Default value: -1

`enable_segcompaction`

- Type: bool
- Description: Enable to use segment compaction during loading to avoid -238 error
- Default value: true

`segcompaction_batch_size`

- Type: int32
- Description: Segment compaction is triggered when the number of segments exceeds this threshold. This configuration also limits the maximum number of raw segments in a single segment compaction task.
- Default value: 10

`segcompaction_candidate_max_rows`

- Type: int32
- Description: Max row count allowed in a single source segment, bigger segments will be skipped.
- Default value: 1048576

`segcompaction_candidate_max_bytes`

- Type: int64
- Description: Max file size allowed in a single source segment, bigger segments will be skipped.
- Default value: 104857600

`segcompaction_task_max_rows`

- Type: int32
- Description: Max total row count allowed in a single segcompaction task.
- Default value: 1572864

`segcompaction_task_max_bytes`

- Type: int64
- Description: Max total file size allowed in a single segcompaction task.
- Default value: 157286400

`segcompaction_num_threads`

- Type: int32
- Description: Global segcompaction thread pool size.
- Default value: 5

`disable_compaction_trace_log`

- Type: bool
- Description: disable the trace log of compaction
  - If set to true, the `cumulative_compaction_trace_threshold` and `base_compaction_trace_threshold` won't work and log is disabled.
- Default value: true

`pick_rowset_to_compact_interval_sec`

- Type: int64
- Description: select the time interval in seconds for rowset to be compacted.
- Default value: 86400

`max_single_replica_compaction_threads`

- Type: int32
- Description: The maximum of thread number in single replica compaction thread pool. -1 means one thread per disk.
- Default value: -1

`update_replica_infos_interval_seconds`

- Description: Minimal interval (s) to update peer replica infos
- Default value: 60 (s)

Load

`enable_stream_load_record`

- Type: bool
- Description:Whether to enable stream load record function, the default is false.
- Default value: false

`load_data_reserve_hours`

- Description: Used for mini load. The mini load data file will be deleted after this time
- Default value: 4 (h)

`push_worker_count_high_priority`

- Description: Import the number of threads for processing HIGH priority tasks
- Default value: 3

`push_worker_count_normal_priority`

- Description: Import the number of threads for processing NORMAL priority tasks
- Default value: 3

`enable_single_replica_load`

- Description: Whether to enable the single-copy data import function
- Default value: true

`load_error_log_reserve_hours`

- Description: The load error log will be deleted after this time
- Default value: 48 (h)

`load_error_log_limit_bytes`

- Description: The loading error logs larger than this value will be truncated
- Default value: 209715200 (byte)

`load_process_max_memory_limit_percent`

- Description: The percentage of the upper memory limit occupied by all imported threads on a single node, the default is 50%
    - Set these default values very large, because we don't want to affect load performance when users upgrade Doris. If necessary, the user should set these configurations correctly
- Default value: 50 (%)

`load_process_soft_mem_limit_percent`

- Description: The soft limit refers to the proportion of the load memory limit of a single node. For example, the load memory limit for all load tasks is 20GB, and the soft limit defaults to 50% of this value, that is, 10GB. When the load memory usage exceeds the soft limit, the job with the largest memory consumption will be selected to be flushed to release the memory space, the default is 50%
- Default value: 50 (%)

`routine_load_thread_pool_size`

- Description: The thread pool size of the routine load task. This should be greater than the FE configuration'max_concurrent_task_num_per_be'
- Default value: 10

`slave_replica_writer_rpc_timeout_sec`

- Type: int32
- Description: This configuration is mainly used to modify timeout of brpc between master replica and slave replica, used for single replica load.
- Default value: 60

`max_segment_num_per_rowset`

- Type: int32
- Description: Used to limit the number of segments in the newly generated rowset when importing. If the threshold is exceeded, the import will fail with error -238. Too many segments will cause compaction to take up a lot of memory and cause OOM errors.
- Default value: 200

`high_priority_flush_thread_num_per_store`

- Type: int32
- Description: The number of flush threads per store path allocated for the high priority import task.
- Default value: 1

`routine_load_consumer_pool_size`

- Type: int32
- Description: The number of caches for the data consumer used by the routine load.
- Default value: 10

`multi_table_batch_plan_threshold`

- Type: int32
- Description: For single-stream-multi-table load. When receive a batch of messages from kafka, if the size of batch is more than this threshold, we will request plans for all related tables.
- Default value: 200

`multi_table_max_wait_tables`

- Type: int32
- Description: Used in single-stream-multi-table load. When receiving a batch of messages from Kafka, if the size of the table wait for plan is more than this threshold, we will request plans for all related tables.The param is aimed to avoid requesting and executing too many plans at once. Performing small batch processing on multiple tables during the loaded process can reduce the pressure of a single RPC and improve the real-time processing of data.
- Default value: 5

`single_replica_load_download_num_workers`

- Type: int32
- Description:This configuration is mainly used to modify the number of http worker threads for segment download, used for single replica load. When the load concurrency increases, you can adjust this parameter to ensure that the Slave replica synchronizes data files from the Master replica timely. If needed, `webserver_num_workers` should also be increased for better IO performance.
- Default value: 64

`load_task_high_priority_threshold_second`

- Type: int32
- Description: When the timeout of an import task is less than this threshold, Doris will consider it to be a high priority task. High priority tasks use a separate pool of flush threads.
- Default: 120

`min_load_rpc_timeout_ms`

- Type: int32
- Description: The minimum timeout for each rpc in the load job.
- Default: 20

`kafka_api_version_request`

- Type: bool
- Description: If the dependent Kafka version is lower than 0.10.0.0, this value should be set to false.
- Default: true

`kafka_broker_version_fallback`

- Description: If the dependent Kafka version is lower than 0.10.0.0, the value set by the fallback version kafka_broker_version_fallback will be used if the value of kafka_api_version_request is set to false, and the valid values are: 0.9.0.x, 0.8.x.y.
- Default: 0.10.0

`max_consumer_num_per_group`

- Description: The maximum number of consumers in a data consumer group, used for routine load
- Default: 3

`streaming_load_max_mb`

- Type: int64
- Description: Used to limit the maximum amount of csv data allowed in one Stream load.
    - Stream Load is generally suitable for loading data less than a few GB, not suitable for loading too large data.
- Default value: 10240 (MB)
- Dynamically modifiable: Yes

`streaming_load_json_max_mb`

- Type: int64
- Description: it is used to limit the maximum amount of json data allowed in one Stream load. The unit is MB.
    - Some data formats, such as JSON, cannot be split. Doris must read all the data into the memory before parsing can begin. Therefore, this value is used to limit the maximum amount of data that can be loaded in a single Stream load.
- Default value: 100
- Dynamically modifiable: Yes

`olap_table_sink_send_interval_microseconds`

- Description: While loading data, there's a polling thread keep sending data to corresponding BE from Coordinator's sink node. This thread will check whether there's data to send every `olap_table_sink_send_interval_microseconds` microseconds.
- Default value: 1000

`olap_table_sink_send_interval_auto_partition_factor`

- Description: If we load data to a table which enabled auto partition. the interval of `olap_table_sink_send_interval_microseconds` is too slow. In that case the real interval will multiply this factor.
- Default value: 0.001

Thread

`delete_worker_count`

- Description: Number of threads performing data deletion tasks
- Default value: 3

`clear_transaction_task_worker_count`

- Description: Number of threads used to clean up transactions
- Default value: 1

`clone_worker_count`

- Description: Number of threads used to perform cloning tasks
- Default value: 3

`be_service_threads`

- Type: int32
- Description: The number of execution threads of the thrift server service on BE which represents the number of threads that can be used to execute FE requests.
- Default value: 64

`download_worker_count`

- Description: The number of download threads.
- Default value: 1

`drop_tablet_worker_count`

- Description: Number of threads to delete tablet
- Default value: 3

`flush_thread_num_per_store`

- Description: The number of threads used to refresh the memory table per store
- Default value: 2

`num_threads_per_core`

- Description: Control the number of threads that each core runs. Usually choose 2 times or 3 times the number of cores. This keeps the core busy without causing excessive jitter
- Default value: 3

`num_threads_per_disk`

- Description: The maximum number of threads per disk is also the maximum queue depth of each disk
- Default value: 0

`number_slave_replica_download_threads`

- Description: Number of threads for slave replica synchronize data, used for single replica load.
- Default value: 64

`publish_version_worker_count`

- Description: the count of thread to publish version
- Default value: 8

`upload_worker_count`

- Description: Maximum number of threads for uploading files
- Default value: 1

`webserver_num_workers`

- Description: Webserver default number of worker threads
- Default value: 48

`send_batch_thread_pool_thread_num`

- Type: int32
- Description: The number of threads in the SendBatch thread pool. In NodeChannels' sending data tasks, the SendBatch operation of each NodeChannel will be submitted as a thread task to the thread pool to be scheduled. This parameter determines the size of the SendBatch thread pool.
- Default value: 64

`send_batch_thread_pool_queue_size`

- Type: int32
- Description: The queue length of the SendBatch thread pool. In NodeChannels' sending data tasks, the SendBatch operation of each NodeChannel will be submitted as a thread task to the thread pool waiting to be scheduled, and after the number of submitted tasks exceeds the length of the thread pool queue, subsequent submitted tasks will be blocked until there is a empty slot in the queue.
- Default value: 102400

`make_snapshot_worker_count`

- Description: Number of threads making snapshots
- Default value: 5

`release_snapshot_worker_count`

- Description: Number of threads releasing snapshots
- Default value: 5

Memory

`disable_mem_pools`

- Type: bool

- Description: Whether to disable the memory cache pool.
- Default value: false

`buffer_pool_clean_pages_limit`

- Description: Clean up pages that may be saved by the buffer pool
- Default value: 50%

`buffer_pool_limit`

- Type: string
- Description: The largest allocatable memory of the buffer pool
    - The maximum amount of memory available in the BE buffer pool. The buffer pool is a new memory management structure of BE, which manages the memory by the buffer page and enables spill data to disk. The memory for all concurrent queries will be allocated from the buffer pool. The current buffer pool only works on **AggregationNode** and **ExchangeNode**.
- Default value: 20%

`chunk_reserved_bytes_limit`

- Description: The reserved bytes limit of Chunk Allocator, usually set as a percentage of mem_limit. defaults to bytes if no unit is given, the number of bytes must be a multiple of 2. must larger than 0. and if larger than physical memory size, it will be set to physical memory size. increase this variable can improve performance, but will acquire more free memory which can not be used by other modules.
- Default value: 20%

`madvise_huge_pages`

- Type: bool
- Description: Whether to use linux memory huge pages.
- Default value: false

`max_memory_sink_batch_count`

- Description: The maximum external scan cache batch count, which means that the cache max_memory_cache_batch_count * batch_size row, the default is 20, and the default value of batch_size is 1024, which means that 20 * 1024 rows will be cached
- Default value: 20

`memory_max_alignment`

- Description: Maximum alignment memory
- Default value: 16

`mmap_buffers`

- Description: Whether to use mmap to allocate memory

- Default value: false

`memtable_mem_tracker_refresh_interval_ms`

- Description: Interval in milliseconds between memtable flush mgr refresh iterations
- Default value: 100

`zone_map_row_num_threshold`

- Type: int32
- Description: If the number of rows in a page is less than this value, no zonemap will be created to reduce data expansion
- Default value: 20

`enable_tcmalloc_hook`

- Type: bool
- Description: Whether Hook TCmalloc new/delete, currently consume/release tls mem tracker in Hook.
- Default value: true

`memory_mode`

- Type: string
- Description: Control gc of tcmalloc, in performance mode Doris releases memory of tcmalloc cache when usage >= 90% * mem_limit, otherwise, doris releases memory of tcmalloc cache when usage >= 50% * mem_limit;
- Default value: performance

`max_sys_mem_available_low_water_mark_bytes`

- Type: int64
- Description: The maximum low water mark of the system `/proc/meminfo/MemAvailable`, Unit byte, default 1.6G, actual low water mark=min(1.6G, MemTotal * 10%), avoid wasting too much memory on machines with large memory larger than 16G. Turn up max. On machines with more than 16G memory, more memory buffers will be reserved for Full GC. Turn down max. will use as much memory as possible.
- Default value: 1717986918

`memory_limitation_per_thread_for_schema_change_bytes`

- Description: Maximum memory allowed for a single schema change task
- Default value: 2147483648 (2GB)

`mem_tracker_consume_min_size_bytes`

- Type: int32
- Description: The minimum length of TCMalloc Hook when consume/release MemTracker. Consume size smaller than this value will continue to accumulate to avoid frequent calls to consume/release of

MemTracker. Decreasing this value will increase the frequency of consume/release. Increasing this value will cause MemTracker statistics to be inaccurate. Theoretically, the statistical value of a MemTracker differs from the true value = (mem_tracker_consume_min_size_bytes * the number of BE threads where the MemTracker is located).

- Default value: 1,048,576

`cache_clean_interval`

- Description: File handle cache cleaning interval, used to clean up file handles that have not been used for a long time.Also the clean interval of Segment Cache.
- Default value: 1800 (s)

`min_buffer_size`

- Description: Minimum read buffer size
- Default value: 1024 (byte)

`write_buffer_size`

- Description: The size of the buffer before flashing
  - Imported data is first written to a memory block on the BE, and only written back to disk when this memory block reaches the threshold. The default size is 200MB. too small a threshold may result in a large number of small files on the BE. This threshold can be increased to reduce the number of files. However, too large a threshold may cause RPC timeouts
- Default value: 209,715,200

`remote_storage_read_buffer_mb`

- Type: int32
- Description: The cache size used when reading files on hdfs or object storage.
  - Increasing this value can reduce the number of calls to read remote data, but it will increase memory overhead.
- Default value: 16 (MB)

`file_cache_type`

- Type: string
- Description: Type of cache file.`whole_file_cache`: download the entire segment file, `sub_file_cache`: the segment file is divided into multiple files by size. if set ““, no cache, please set this parameter when caching is required.
- Default value: ““

`file_cache_alive_time_sec`

- Type: int64
- Description: Save time of cache file
- Default value: 604800 (1 week)

`file_cache_max_size_per_disk`

- Type: int64
- Description: The cache occupies the disk size. Once this setting is exceeded, the cache that has not been accessed for the longest time will be deleted. If it is 0, the size is not limited. unit is bytes.
- Default value: 0

`max_sub_cache_file_size`

- Type: int64
- Description: Cache files using sub_ file_ The maximum size of the split file during cache
- Default value: 104857600 (100MB)

`download_cache_thread_pool_thread_num`

- Type: int32
- Description: The number of threads in the DownloadCache thread pool. In the download cache task of FileCache, the download cache operation will be submitted to the thread pool as a thread task and wait to be scheduled. This parameter determines the size of the DownloadCache thread pool.
- Default value: 48

`download_cache_thread_pool_queue_size`

- Type: int32
- Description: The number of threads in the DownloadCache thread pool. In the download cache task of FileCache, the download cache operation will be submitted to the thread pool as a thread task and wait to be scheduled. After the number of submitted tasks exceeds the length of the thread pool queue, subsequent submitted tasks will be blocked until there is a empty slot in the queue.
- Default value: 102400

`generate_cache_cleaner_task_interval_sec`

- Type：int64
- Description: Cleaning interval of cache files, in seconds
- Default: 43200 (12 hours)

`path_gc_check`

- Type：bool
- Description: Whether to enable the recycle scan data thread check
- Default：true

`path_gc_check_interval_second`

- Description: Recycle scan data thread check interval
- Default：86400 (s)

`path_gc_check_step`

- Default：1000

`path_gc_check_step_interval_ms`

- Default：10 (ms)

`path_scan_interval_second`

- Default：86400

`scan_context_gc_interval_min`

- Description: This configuration is used for the context gc thread scheduling cycle. Note: The unit is minutes, and the default is 5 minutes
- Default：5

Storage

`default_num_rows_per_column_file_block`

- Type: int32
- Description: Configure how many rows of data are contained in a single RowBlock.
- Default value: 1024

`disable_storage_page_cache`

- Type: bool
- Description: Disable to use page cache for index caching, this configuration only takes effect in BETA storage format, usually it is recommended to false
- Default value: false

`disk_stat_monitor_interval`

- Description: Disk status check interval
- Default value: 5 (s)

`max_free_io_buffers`

- Description: For each io buffer size, the maximum number of buffers that IoMgr will reserve ranges from 1024B to 8MB buffers, up to about 2GB buffers.
- Default value: 128

`max_garbage_sweep_interval`

- Description: The maximum interval for disk garbage cleaning

- Default value: 3600 (s)

`max_percentage_of_error_disk`

- Type: int32
- Description: The storage engine allows the percentage of damaged hard disks to exist. After the damaged hard disk exceeds the changed ratio, BE will automatically exit.
- Default value: 0

`read_size`

- Description: The read size is the read size sent to the os. There is a trade-off between latency and the whole process, getting to keep the disk busy but not introducing seeks. For 8 MB reads, random io and sequential io have similar performance.
- Default value: 8388608

`min_garbage_sweep_interval`

- Description: The minimum interval between disk garbage cleaning
- Default value: 180 (s)

`pprof_profile_dir`

- Description: pprof profile save directory
- Default value: ${DORIS_HOME}/log

`small_file_dir`

- Description: Save files downloaded by SmallFileMgr
- Default value: ${DORIS_HOME}/lib/small_file/

`user_function_dir`

- Description: udf function directory
- Default value: ${DORIS_HOME}/lib/udf

`storage_flood_stage_left_capacity_bytes`

- Description: The min bytes that should be left of a data dir.
- Default value: 1073741824

`storage_flood_stage_usage_percent`

- Description: The storage_flood_stage_usage_percent and storage_flood_stage_left_capacity_bytes configurations limit the maximum usage of the capacity of the data directory.
- Default value: 90 (90%)

`storage_medium_migrate_count`

- Description: the count of thread to clone
- Default value: 1

`storage_page_cache_limit`

- Description: Cache for storage page size
- Default value: 20%

`storage_page_cache_shard_size`

- Description: Shard size of StoragePageCache, the value must be power of two. It's recommended to set it to a value close to the number of BE cores in order to reduce lock contentions.
- Default value: 16

`index_page_cache_percentage`

- Type: int32
- Description: Index page cache as a percentage of total storage page cache, value range is [0, 100]
- Default value: 10

`segment_cache_capacity`

- Type: int32
- Description: Max number of segment cache (the key is rowset id) entries. -1 is for backward compatibility as fd_number * 2/5.
- Default value: -1

`storage_strict_check_incompatible_old_format`

- Type: bool
- Description: Used to check incompatible old format strictly
- Default value: true
- Dynamically modify: false

`sync_tablet_meta`

- Description: Whether the storage engine opens sync and keeps it to the disk
- Default value: true

`pending_data_expire_time_sec`

- Description: The maximum duration of unvalidated data retained by the storage engine
- Default value: 1800 (s)

`ignore_rowset_stale_unconsistent_delete`

- Type: boolean
- Description:It is used to decide whether to delete the outdated merged rowset if it cannot form a consistent version path.
    - The merged expired rowset version path will be deleted after half an hour. In abnormal situations, deleting these versions will result in the problem that the consistent path of the query cannot be constructed. When the configuration is false, the program check is strict and the program will directly report an error and exit.When configured as true, the program will run normally and ignore this error. In general, ignoring this error will not affect the query, only when the merged version is dispatched by fe, -230 error will appear.
- Default value: false

`create_tablet_worker_count`

- Description: Number of worker threads for BE to create a tablet
- Default value: 3

`check_consistency_worker_count`

- Description: The number of worker threads to calculate the checksum of the tablet
- Default value: 1

`max_tablet_version_num`

- Type: int
- Description: Limit the number of versions of a single tablet. It is used to prevent a large number of version accumulation problems caused by too frequent import or untimely compaction. When the limit is exceeded, the import task will be rejected.
- Default value: 2000

`number_tablet_writer_threads`

- Description: Number of tablet write threads
- Default value: 16

`tablet_map_shard_size`

- Description: tablet_map_lock fragment size, the value is 2^n, n=0,1,2,3,4, this is for better tablet management
- Default value: 4

`tablet_meta_checkpoint_min_interval_secs`

- Description: TabletMeta Checkpoint 线程轮询的时间间隔
- Default value: 600 (s)

`tablet_meta_checkpoint_min_new_rowsets_num`

- Description: The minimum number of Rowsets for storing TabletMeta Checkpoints
- Default value: 10

`tablet_stat_cache_update_interval_second`

- Description: Update interval of tablet state cache
- Default value:300 (s)

`tablet_rowset_stale_sweep_time_sec`

- Type: int64
- Description: It is used to control the expiration time of cleaning up the merged rowset version. When the current time now() minus the max created rowset‘s create time in a version path is greater than tablet_rowset_stale_sweep_time_sec, the current path is cleaned up and these merged rowsets are deleted, the unit is second.
    - When writing is too frequent, Fe may not be able to query the merged version, resulting in a query -230 error. This problem can be avoided by increasing this parameter.
- Default value: 300

`tablet_writer_open_rpc_timeout_sec`

- Description: Update interval of tablet state cache
    - The RPC timeout for sending a Batch (1024 lines) during import. The default is 60 seconds. Since this RPC may involve writing multiple batches of memory, the RPC timeout may be caused by writing batches, so this timeout can be adjusted to reduce timeout errors (such as send batch fail errors). Also, if you increase the write_buffer_size configuration, you need to increase this parameter as well.
- Default value: 60

`tablet_writer_ignore_eovercrowded`

- Type: bool
- Description: Used to ignore brpc error ‘[E1011]The server is overcrowded’ when writing data.
- Default value: false

`streaming_load_rpc_max_alive_time_sec`

- Description: The lifetime of TabletsChannel. If the channel does not receive any data at this time, the channel will be deleted.
- Default value: 1200

`alter_tablet_worker_count`

- Description: The number of threads making schema changes
- Default value: 3

alter_index_worker_count

- Description: The number of threads making index change
- Default value: 3

ignore_load_tablet_failure

- Type: bool
- Description: It is used to decide whether to ignore errors and continue to start be in case of tablet loading failure
- Default value: false

When BE starts, a separate thread will be started for each data directory to load the meta information of the tablet header. In the default configuration, if a data directory fails to load a tablet, the startup process will terminate. At the same time, it will be displayed in the be The following error message is seen in the INFO log:

```
load tablets from header failed, failed tablets size: xxx, path=xxx
```

Indicates how many tablets failed to load in the data directory. At the same time, the log will also contain specific information about the tablets that failed to load. At this time, manual intervention is required to troubleshoot the cause of the error. After troubleshooting, there are usually two ways to restore: - The tablet information cannot be repaired. If the other copies are normal, you can delete the wrong tablet with the meta_tool tool. - Set ignore_load_tablet_failure to true, BE will ignore these faulty tablets and start normally

report_disk_state_interval_seconds

- Description: The interval time for the agent to report the disk status to FE
- Default value: 60 (s)

result_buffer_cancelled_interval_time

- Description: Result buffer cancellation time
- Default value: 300 (s)

snapshot_expire_time_sec

- Description: Snapshot file cleaning interval.
- Default value:172800 (48 hours)

compress_rowbatches

- Type: bool
- Description: enable to use Snappy compression algorithm for data compression when serializing RowBatch
- Default value: true

Log

`sys_log_dir`

- Type: string
- Description: Storage directory of BE log data
- Default value: ${DORIS_HOME}/log

`sys_log_level`

- Description: Log Level: INFO < WARNING < ERROR < FATAL
- Default value: INFO

`sys_log_roll_mode`

- Description: The size of the log split, one log file is split every 1G
- Default value: SIZE-MB-1024

`sys_log_roll_num`

- Description: Number of log files kept
- Default value: 10

`sys_log_verbose_level`

- Description: Log display level, used to control the log output at the beginning of VLOG in the code
- Default value: 10

`sys_log_verbose_modules`

- Description: Log printing module, writing olap will only print the log under the olap module
- Default value: empty

`aws_log_level`

- Type: int32
- Description: log level of AWS SDK,

```
Off = 0,
Fatal = 1,
Error = 2,
Warn = 3,
Info = 4,
Debug = 5,
Trace = 6
```

- Default value: 3

`log_buffer_level`

- Description: The log flushing strategy is kept in memory by default
- Default value: empty

Else

`report_tablet_interval_seconds`

- Description: The interval time for the agent to report the olap table to the FE
- Default value: 60 (s)

`report_task_interval_seconds`

- Description: The interval time for the agent to report the task signature to FE
- Default value: 10 (s)

`periodic_counter_update_period_ms`

- Description: Update rate counter and sampling counter cycle
- Default value: 500 (ms)

`enable_metric_calculator`

- Description: If set to true, the metric calculator will run to collect BE-related indicator information, if set to false, it will not run
- Default value: true

`enable_system_metrics`

- Description: User control to turn on and off system indicators.
- Default value: true

`enable_token_check`

- Description: Used for forward compatibility, will be removed later.
- Default value: true

`max_runnings_transactions_per_txn_map`

- Description: Max number of txns for every txn_partition_map in txn manager, this is a self protection to avoid too many txns saving in manager
- Default value: 2000

`max_download_speed_kbps`

- Description: Maximum download speed limit

- Default value: 50000 (kb/s)

`download_low_speed_time`

- Description: Download time limit
- Default value: 300 (s)

`download_low_speed_limit_kbps`

- Description: Minimum download speed
- Default value: 50 (KB/s)

`enable_batch_download`

:::tip Tips This configuration is supported since the Apache Doris 2.1.8 version :::

- Description: Whether to download files in batch, it is recommended to open it when the binlog is enabled.
- Default value: false

`doris_cgroups`

- Description: Cgroups assigned to doris
- Default value: empty

`priority_queue_remaining_tasks_increased_frequency`

- Description: the increased frequency of priority for remaining tasks in BlockingPriorityQueue
- Default value: 512

`jdbc_drivers_dir`

:::tip Tips This configuration is supported since the Apache Doris 1.2 version :::

- Description: Default dirs to put jdbc drivers.
- Default value: `${DORIS_HOME}/jdbc_drivers`

`enable_simdjson_reader`

:::tip Tips This configuration is supported since the Apache Doris 1.2 version :::

- Description: Whether enable simdjson to parse json while stream load
- Default value: true

`enable_query_memory_overcommit`

- Description: If true, when the process does not exceed the soft mem limit, the query memory will not be limited; when the process memory exceeds the soft mem limit, the query with the largest ratio between

the currently used memory and the exec_mem_limit will be canceled. If false, cancel query when the memory used exceeds exec_mem_limit.
- Default value: true

`user_files_secure_path`

- Description: The storage directory for files queried by `local` table valued functions.
- Default value: `${DORIS_HOME}`

`brpc_streaming_client_batch_bytes`

- Description: The batch size for sending data by brpc streaming client
- Default value: 262144

`grace_shutdown_wait_seconds`

- Description: In cloud native deployment scenario, BE will be add to cluster and remove from cluster very frequently. User's query will fail if there is a fragment is running on the shuting down BE. Users could use stop_be.sh –grace, then BE will wait all running queries to stop to avoiding running query failure, but if the waiting time exceed the limit, then be will exit directly. During this period, FE will not send any queries to BE and waiting for all running queries to stop.
- Default value: 120

`enable_java_support`

- Description: BE Whether to enable the use of java-jni. When enabled, mutual calls between c++ and java are allowed. Currently supports hudi, java-udf, jdbc, max-compute, paimon, preload, avro
- Default value: true

`group_commit_wal_path`

- The `WAL` directory of group commit.
- Default: A directory named `wal` is created under each directory of the `storage_root_path`. Configuration examples:

```
group_commit_wal_path=/data1/storage/wal;/data2/storage/wal;/data3/storage/wal
```

`default_tzfiles_path`

- Description: Doris comes with its own time zone database. If the time zone file is not found in the system directory, the data in that directory is enabled.
- Default: "${DORIS_HOME}/zoneinfo"

`time_series_max_tablet_version_num`

- Type: int

- Description: Limit the number of versions of a single tablet under the time-series compaction policy. It is used to prevent a large number of version accumulation problems caused by too frequent load or delayed compaction. When the limit is exceeded, the load task will be rejected. Supported since version 2.1.11
- Default value: 20000

User Property

User configuration item

This document mainly introduces related configuration items at the User level. The configuration of the User level is mainly effective for a single user. Each user can set their own User property. Does not affect each other.

View configuration items

After the FE is started, on the MySQL client, use the following command to view the User configuration items:

```
SHOW PROPERTY [FOR user] [LIKE key pattern]
```

The specific syntax can be queried through the command: `help show property;`.

Set configuration items

After FE is started, on the MySQL client, modify the User configuration items with the following command:

```
SET PROPERTY [FOR'user'] 'key' = 'value' [,'key' ='value']
```

The specific syntax can be queried through the command: `help set property;`.

User-level configuration items will only take effect for the specified users, and will not affect the configuration of other users.

Application examples

1. Modify the max_user_connections of user Billie

   Use `SHOW PROPERTY FOR 'Billie' LIKE '%max_user_connections%';` to check that the current maximum number of links for Billie users is 100.

   Use `SET PROPERTY FOR 'Billie' 'max_user_connections' = '200';` to modify the current maximum number of connections for Billie users to 200.

Configuration item list

max_user_connections

```
The maximum number of user connections, the default value is 100 In general, this parameter does not need to be changed unless the number of concurrent queries exceeds the default value.
```

max_query_instances

The maximum number of instances that the user can use at a certain point in time, The default value is -1, negative number means use default_max_query_instances config.

resource

quota

default_load_cluster

load_cluster

Overview | System Tables

Overview

Apache Doris cluster has multiple built-in system databases to store metadata information about the Doris system itself.

information_schema

All tables under the `information_schema` database are virtual tables and do not have physical entities. These system tables contain metadata about the Doris cluster and all its database objects, including databases, tables, columns, permissions, etc. They also include functional status information like Workload Group, Task, etc.

There is an `information_schema` database under each Catalog, containing metadata only for the corresponding Catalog's databases and tables.

All tables in the `information_schema` database are read-only, and users cannot modify, drop, or create tables in this database.

By default, all users have read permissions for all tables in this database, but the query results will vary based on the user's actual permission. For example, if User A only has permissions for `db1.table1`, querying the `information_schema.tables` table will only return information related to `db1.table1`.

mysql

All tables under the `mysql` database are virtual tables and do not have physical entities. These system tables contain information such as permissions and are mainly used for MySQL ecosystem compatibility.

There is a `mysql` database under each Catalog, but the content of tables is identical.

All tables in the `mysql` database are read-only, and users cannot modify, delete, or create tables in this database.

__internal_schema

All tables under the `__internal_schema` database are actual tables in Doris, stored similarly to user-created data tables. When a Doris cluster is created, all system tables under this database are automatically created.

By default, common users have read-only permissions for tables in this database. However, once granted, they can modify, delete, or create tables under this database.

active_queries

Overview

Used to view currently executing queries

Database

`information_schema`

Table Information

Column Name	Type	Description
QUERY_ID	varchar(256)	The ID of the query
QUERY_START_TIME	varchar(256)	The start time of the query
QUERY_TIME_MS	bigint	The execution time of the query
WORKLOAD_GROUP_ID	bigint	The ID of the Workload Group to which the query belongs
DATABASE	varchar(256)	The Database where the query was executed
FRONTEND_INSTANCE	varchar(256)	The IP address of the Frontend instance that received the query request
QUEUE_START_TIME	varchar(256)	The start time of queuing; empty if not queued
QUEUE_END_TIME	varchar(256)	The end time of queuing; empty if not queued
QUERY_STATUS	varchar(256)	The status of the query
SQL	text	The text of the query statement

backend_active_tasks

Overview

View the resource usage of Query or Load tasks currently running on Backends.

Database

`information_schema`

Table Information

Column Name	Type	Description
BE_ID	bigint	The ID of the Backend executing the task
FE_HOST	varchar(256)	The address of the Frontend that issued the task
QUERY_ID	varchar(256)	The ID of the query
TASK_TIME_MS	bigint	The task execution time
TASK_CPU_TIME_MS	bigint	The CPU time used by the task
SCAN_ROWS	bigint	The number of rows scanned
SCAN_BYTES	bigint	The number of bytes scanned
BE_PEAK_MEMORY_BYTES	bigint	The peak memory usage
CURRENT_USED_MEMORY_BYTES	bigint	The current memory usage
SHUFFLE_SEND_BYTES	bigint	The number of bytes shuffled and sent
SHUFFLE_SEND_ROWS	bigint	The number of rows shuffled and sent
QUERY_TYPE	varchar(256)	The type of query

catalog_meta_cache_statistics

Overview

View the metadata cache information of the External Catalog in the currently connected FE.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
CATALOG_NAME	text	The name of the Catalog
CACHE_NAME	text	The name of the cache
METRIC_NAME	text	The name of the metric
METRIC_VALUE	text	The value of the metric

Usage Example

```
+---------------------+----------------------------+---------------------+--------------------+
| CATALOG_NAME        | CACHE_NAME                 | METRIC_NAME         | METRIC_VALUE       |
+---------------------+----------------------------+---------------------+--------------------+
| hive_iceberg_minio  | iceberg_table_cache        | eviction_count      | 0                  |
| hive_iceberg_minio  | iceberg_table_cache        | hit_ratio           | 0.8235294117647058 |
| hive_iceberg_minio  | iceberg_table_cache        | average_load_penalty| 5.480102048333334E8|
| hive_iceberg_minio  | iceberg_table_cache        | estimated_size      | 6                  |
| hive_iceberg_minio  | iceberg_table_cache        | hit_count           | 28                 |
| hive_iceberg_minio  | iceberg_table_cache        | read_count          | 34                 |
| hive_iceberg_minio  | iceberg_snapshot_cache     | eviction_count      | 0                  |
| hive_iceberg_minio  | iceberg_snapshot_cache     | hit_ratio           | 0.45454545454545453|
| hive_iceberg_minio  | iceberg_snapshot_cache     | average_load_penalty| 5.604907246666666E8|
| hive_iceberg_minio  | iceberg_snapshot_cache     | estimated_size      | 6                  |
| hive_iceberg_minio  | iceberg_snapshot_cache     | hit_count           | 5                  |
| hive_iceberg_minio  | iceberg_snapshot_cache     | read_count          | 11                 |
```

The METRIC_NAME column contains the following Caffeine cache performance metrics: - eviction_count: The number of entries that have been evicted from the cache - hit_ratio: The ratio of cache requests which were hits (ranges from 0.0 to 1.0) - average_load_penalty: The average time spent loading new values (in nanoseconds) - estimated_size: The approximate number of entries in the cache - hit_count: The number of times cache lookup methods have returned a cached value - read_count: The total number of times cache lookup methods have been called

character_sets

Overview

View the supported character sets. This table is only used for compatibility with MySQL behavior and does not truly reflect the character sets used by Doris.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
CHARACTER_SET_NAME	varchar(512)	The name of the character set
DEFAULT_COLLATE_NAME	varchar(64)	The default collation name
DESCRIPTION	varchar(64)	A detailed description of the character set
MAXLEN	bigint	The maximum number of bytes occupied by a single character in the character set

collations

Overview

View all collation methods for character sets. This table is only used for compatibility with MySQL behavior and holds no practical significance. It does not truly reflect the character collation methods used by Doris.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
COLLATION_NAME	varchar(512)	The name of the character set collation method
CHARACTER_SET_NAME	varchar(64)	The name of the associated character set
ID	bigint	The ID of the collation method
IS_DEFAULT	varchar(64)	Indicates if it is the current default collation method.
IS_COMPILED	varchar(64)	Indicates if it is compiled into the service
SORTLEN	bigint	Related to the memory used by this collation algorithm

column_privileges

Overview

This table is solely used for compatibility with MySQL behavior and is always empty. It does not truly reflect the column permission information of Doris.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
GRANTEE	varchar(128)	
TABLE_CATALOG	varchar(512)	
TABLE_SCHEMA	varchar(64)	
TABLE_NAME	varchar(64)	
COLUMN_NAME	varchar(64)	
PRIVILEGE_TYPE	varchar(64)	
IS_GRANTABLE	varchar(3)	

Column Statistics | Information Schema

Column Statistics

Overview

This table is solely used for compatibility with MySQL behavior and is always empty. It does not truly reflect the statistical information of the data within Doris. To view the statistical information collected by Doris, please refer to the Statistics section.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
SCHEMA_NAME	varchar(64)	
TABLE_NAME	varchar(64)	
COLUMN_NAME	varchar(64)	
HISTOGRAM	json	

columns

Overview

View all column information.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
TABLE_CATALOG	varchar(512)	Catalog name
TABLE_SCHEMA	varchar(64)	Database name
TABLE_NAME	varchar(64)	Table name
COLUMN_NAME	varchar(64)	Column name
ORDINAL_POSITION	bigint	The position of the column in the table
COLUMN_DEFAULT	varchar(1024)	Default value of the column
IS_NULLABLE	varchar(3)	Whether NULL is allowed
DATA_TYPE	varchar(64)	Data type
CHARACTER_MAXIMUM_LENGTH	bigint	Maximum number of characters allowed for character types
CHARACTER_OCTET_LENGTH	bigint	Maximum number of bytes allowed for character types
NUMERIC_PRECISION	bigint	Precision for numeric types
NUMERIC_SCALE	bigint	Scale for numeric types
DATETIME_PRECISION	bigint	Precision for datetime types
CHARACTER_SET_NAME	varchar(32)	Character set name for character types, always NULL
COLLATION_NAME	varchar(32)	Collation algorithm name for character types, always NULL
COLUMN_TYPE	varchar(32)	Column type
COLUMN_KEY	varchar(3)	If ‘UNI’, it indicates that the column is a Unique Key column
EXTRA	varchar(27)	Additional information about the column, including whether it is an auto-increment column, a generated column, etc.
PRIVILEGES	varchar(80)	Always empty
COLUMN_COMMENT	varchar(255)	Comment information for the column

Column Name	Type	Description
COLUMN_SIZE	bigint	Width of the column
DECIMAL_DIGITS	bigint	Number of decimal places for numeric types
GENERATION_EXPRESSION	varchar(64)	Always NULL
SRS_ID	bigint	Always NULL

engines

Overview

This table is solely used for compatibility with MySQL behavior. It is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
ENGINE	varchar(64)	
SUPPORT	varchar(8)	
COMMENT	varchar(80)	
TRANSACTIONS	varchar(3)	
XA	varchar(3)	
SAVEPOINTS	varchar(3)	

events

Overview

This table is solely used for compatibility with MySQL behavior and is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
EVENT_CATALOG	varchar(64)	
EVENT_SCHEMA	varchar(64)	
EVENT_NAME	varchar(64)	
DEFINER	varchar(77)	
TIME_ZONE	varchar(64)	
EVENT_BODY	varchar(8)	
EVENT_DEFINITION	varchar(512)	
EVENT_TYPE	varchar(9)	
EXECUTE_AT	datetime	
INTERVAL_VALUE	varchar(256)	
INTERVAL_FIELD	varchar(18)	
SQL_MODE	varchar(8192)	
STARTS	datetime	
ENDS	datetime	
STATUS	varchar(18)	
ON_COMPLETION	varchar(12)	
CREATED	datetime	
LAST_ALTERED	datetime	
LAST_EXECUTED	datetime	
EVENT_COMMENT	varchar(64)	
ORIGINATOR	int	
CHARACTER_SET_CLIENT	varchar(32)	
COLLATION_CONNECTION	varchar(32)	

Column Name	Type	Description
DATABASE_COLLATION	varchar(32)	

file_cache_statistics

Overview

Used to view the metric information related to data cache on each BE node. The metric information is sourced from the monitoring metrics related to BE's data cache.

> **Tip**
>
> This system table is supported from versions 2.1.6 and 3.0.2.

Database

`information_schema`

Table Information

Column Name	Type	Description
BE_ID	BIGINT	BE node ID
BE_IP	VARCHAR(256)	BE node IP
CACHE_PATH	VARCHAR(256)	BE node cache path
METRIC_NAME	VARCHAR(256)	Metric name
METRIC_VALUE	DOUBLE	Metric value

> **Note**
>
> Doris different version may have different metrics

2.1.x Metrics

> Only important metrics are listed.

- `normal_queue_curr_elements`

  Number of File Blocks currently in the cache.

- `normal_queue_max_elements`

  Maximum number of File Blocks allowed in the cache.

- `normal_queue_curr_size`

  Current cache size.

- `normal_queue_max_size`

  Maximum cache size allowed.

- `hits_ratio`

  Overall cache hit ratio since BE startup. Range 0-1.

- `hits_ratio_5m`

  Cache hit ratio in the last 5 minutes. Range 0-1.

- `hits_ratio_1h`

  Cache hit ratio in the last 1 hour. Range 0-1.

3.0.x Metrics

TODO

Examples

1. Query all cache metrics

```
mysql> select * from information_schema.file_cache_statistics;
+-------+---------------+---------------------------+---------------------------+--------------------+
| BE_ID | BE_IP         | CACHE_PATH                | METRIC_NAME               | METRIC_VALUE       |
+-------+---------------+---------------------------+---------------------------+--------------------+
| 10003 | 172.20.32.136 | /mnt/output/be/file_cache/ | normal_queue_curr_elements |               1392 |
| 10003 | 172.20.32.136 | /mnt/output/be/file_cache/ | normal_queue_curr_size     |          248922234 |
| 10003 | 172.20.32.136 | /mnt/output/be/file_cache/ | normal_queue_max_elements  |             102400 |
| 10003 | 172.20.32.136 | /mnt/output/be/file_cache/ | normal_queue_max_size      |        21474836480 |
| 10003 | 172.20.32.136 | /mnt/output/be/file_cache/ | hits_ratio                 | 0.8539634687001242 |
| 10003 | 172.20.32.136 | /mnt/output/be/file_cache/ | hits_ratio_1h              |                  0 |
| 10003 | 172.20.32.136 | /mnt/output/be/file_cache/ | hits_ratio_5m              |                  0 |
+-------+---------------+---------------------------+---------------------------+--------------------+
```

2. Query cache hit ratio and sort by hit ratio

```
select * from information_schema.file_cache_statistics where METRIC_NAME = "hits_ratio" order by METRIC_VALUE desc;
```

files

Overview

This table is solely used for compatibility with MySQL behavior and is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
FILE_ID	bigint	
FILE_NAME	text	
FILE_TYPE	varchar(256)	
TABLESPACE_NAME	varchar(256)	
TABLE_CATALOG	char(16)	
TABLE_SCHEMA	text	
TABLE_NAME	text	
LOGFILE_GROUP_NAME	varchar(256)	
LOGFILE_GROUP_NUMBER	bigint	
ENGINE	varchar(64)	
FULLTEXT_KEYS	text	
DELETED_ROWS	text	
UPDATE_COUNT	text	
FREE_EXTENTS	bigint	
TOTAL_EXTENTS	bigint	
EXTENT_SIZE	bigint	
INITIAL_SIZE	bigint	
MAXIMUM_SIZE	bigint	
AUTOEXTEND_SIZE	bigint	
CREATION_TIME	text	
LAST_UPDATE_TIME	text	
LAST_ACCESS_TIME	text	
RECOVER_TIME	text	

Column Name	Type	Description
TRANSACTION_COUNTER	text	
VERSION	bigint	
ROW_FORMAT	varchar(256)	
TABLE_ROWS	text	
AVG_ROW_LENGTH	text	
DATA_LENGTH	text	
MAX_DATA_LENGTH	text	
INDEX_LENGTH	text	
DATA_FREE	bigint	
CREATE_TIME	text	
UPDATE_TIME	text	
CHECK_TIME	text	
CHECKSUM	text	
STATUS	varchar(256)	
EXTRA	varchar(256)	

global_variables

Overview

View Global Variables

Database

`information_schema`

Table Information

Column Name	Type	Description
VARIABLE_NAME	varchar(64)	Name of the variable
VARIABLE_VALUE	varchar(1024)	Current value of the variable
DEFAULT_VALUE	varchar(1024)	Default value of the variable
CHANGED	varchar(4)	Indicates if different from default value

key_column_usage

Overview

This table is solely for compatibility with MySQL behavior. It is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
CONSTRAINT_CATALOG	varchar(64)	
CONSTRAINT_SCHEMA	varchar(64)	
CONSTRAINT_NAME	varchar(64)	
TABLE_CATALOG	varchar(64)	
TABLE_SCHEMA	varchar(64)	
TABLE_NAME	varchar(64)	
COLUMN_NAME	varchar(64)	
ORDINAL_POSITION	bigint	
POSITION_IN_UNIQUE_CONSTRAINT	bigint	
REFERENCED_TABLE_SCHEMA	varchar(64)	
REFERENCED_TABLE_NAME	varchar(64)	
REFERENCED_COLUMN_NAME	varchar(64)	

metadata_name_ids

Overview

View ID Information of Objects

Database

`information_schema`

Table Information

Column Name	Type	Description
CATALOG_ID	bigint	ID of the Catalog
CATALOG_NAME	varchar(512)	Name of the Catalog
DATABASE_ID	bigint	ID of the Database
DATABASE_NAME	varchar(64)	Name of the Database
TABLE_ID	bigint	ID of the Table
TABLE_NAME	varchar(64)	Name of the Table

parameters

Overview

This table is solely for compatibility with MySQL behavior and is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
SPECIFIC_CATALOG	varchar(64)	
SPECIFIC_SCHEMA	varchar(64)	
SPECIFIC_NAME	varchar(64)	
ORDINAL_POSITION	varchar(77)	
PARAMETER_MODE	varchar(77)	
PARAMETER_NAME	varchar(77)	
DATA_TYPE	varchar(64)	
CHARACTER_OCTET_LENGTH	varchar(64)	
NUMERIC_PRECISION	varchar(512)	
NUMERIC_SCALE	varchar(64)	
DATETIME_PRECISION	varchar(64)	
CHARACTER_SET_NAME	varchar(256)	
COLLATION_NAME	varchar(64)	
DTD_IDENTIFIER	varchar(64)	
ROUTINE_TYPE	varchar(64)	

partitions

Overview

Show the Partition status of all tables in the database. Before 2.1.7(inclusive), the tables were always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
TABLE_CATALOG	varchar(64)	Catalog name
TABLE_SCHEMA	varchar(64)	Database name
TABLE_NAME	varchar(64)	Table name
PARTITION_NAME	varchar(64)	Partition name
SUBPARTITION_NAME	varchar(64)	Always empty
PARTITION_ORDINAL_POSITION	int	Ordinal position of the partition
SUBPARTITION_ORDINAL_POSITION	int	Always empty
PARTITION_METHOD	varchar(13)	Partition method
SUBPARTITION_METHOD	varchar(13)	Always empty
PARTITION_EXPRESSION	varchar(2048)	Partition expression
SUBPARTITION_EXPRESSION	varchar(2048)	Always empty
PARTITION_DESCRIPTION	text	Partition description
TABLE_ROWS	bigint	
AVG_ROW_LENGTH	bigint	
DATA_LENGTH	bigint	
MAX_DATA_LENGTH	bigint	
INDEX_LENGTH	bigint	
DATA_FREE	bigint	
CREATE_TIME	bigint	
UPDATE_TIME	datetime	
CHECK_TIME	datetime	
CHECKSUM	bigint	
PARTITION_COMMENT	text	

Column Name	Type	Description
NODEGROUP	varchar(256)	
TABLESPACE_NAME	varchar(268)	

processlist

Overview

View All Current Connections

Database

`information_schema`

Table Information

Column Name	Type	Description
CURRENT_CONNECTED	varchar(16)	Deprecated, always No
ID	largeint	Connection ID
USER	varchar(32)	Connected user
HOST	varchar(261)	Connection address
LOGIN_TIME	datetime	Login time
CATALOG	varchar(64)	Current Catalog
DB	varchar(64)	Current Database
COMMAND	varchar(16)	Type of MySQL Command currently sent
TIME	int	Execution time of the last query
STATE	varchar(64)	Status of the last query
QUERY_ID	varchar(256)	ID of the last query
INFO	varchar(65533)	Query statement of the last query
FE	varchar(64)	Connected Front-End (FE)
CLOUD_CLUSTER	varchar(64)	Name of the Cloud Cluster being used

profiling

Overview

This table is solely for the purpose of maintaining compatibility with MySQL behavior. It is always empty.

Database

`information_schema`

Table Information

Column Name	Type	Description
QUERY_ID	int	
SEQ	int	
STATE	varchar(30)	
DURATION	double	
CPU_USER	double	
CPU_SYSTEM	double	
CONTEXT_VOLUNTARY	int	
CONTEXT_INVOLUNTARY	int	
BLOCK_OPS_IN	int	
BLOCK_OPS_OUT	int	
MESSAGES_SENT	int	
MESSAGES_RECEIVED	int	
PAGE_FAULTS_MAJOR	int	
PAGE_FAULTS_MINOR	int	
SWAPS	int	
SOURCE_FUNCTION	varchar(30)	
SOURCE_FILE	varchar(20)	
SOURCE_LINE	int	

referential_constraints

Overview

This table is only used for compatibility with MySQL behavior. It is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
CONSTRAINT_CATALOG	varchar(64)	
CONSTRAINT_SCHEMA	varchar(64)	
CONSTRAINT_NAME	varchar(64)	
UNIQUE_CONSTRAINT_CATALOG	varchar(64)	
UNIQUE_CONSTRAINT_SCHEMA	varchar(64)	
UNIQUE_CONSTRAINT_NAME	varchar(64)	
MATCH_OPTION	varchar(64)	
UPDATE_RULE	varchar(64)	
DELETE_RULE	varchar(64)	
TABLE_NAME	varchar(64)	
REFERENCED_TABLE_NAME	varchar(64)	

routines

Overview

This table is solely for the purpose of maintaining compatibility with MySQL behavior. It is always empty.

Database

`information_schema`

Table Information

Column Name	Type	Description
SPECIFIC_NAME	varchar(64)	
ROUTINE_CATALOG	varchar(64)	
ROUTINE_SCHEMA	varchar(64)	
ROUTINE_NAME	varchar(64)	
ROUTINE_TYPE	varchar(64)	
DTD_IDENTIFIER	varchar(64)	
ROUTINE_BODY	varchar(64)	
ROUTINE_DEFINITION	varchar(64)	
EXTERNAL_NAME	varchar(64)	
EXTERNAL_LANGUAGE	varchar(64)	
PARAMETER_STYLE	varchar(64)	
IS_DETERMINISTIC	varchar(64)	
SQL_DATA_ACCESS	varchar(64)	
SQL_PATH	varchar(64)	
SECURITY_TYPE	varchar(64)	
CREATED	datetime	
LAST_ALTERED	datetime	
SQL_MODE	varchar(64)	
ROUTINE_COMMENT	varchar(64)	
DEFINER	varchar(64)	
CHARACTER_SET_CLIENT	varchar(64)	
COLLATION_CONNECTION	varchar(64)	
DATABASE_COLLATION	varchar(64)	

rowsets

Overview

Returns basic information about the Rowset.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
BACKEND_ID	bigint	The ID of the Backend, which is a unique identifier for the Backend.
ROWSET_ID	varchar(64)	The ID of the Rowset, which is a unique identifier for the Rowset.
TABLET_ID	bigint	The ID of the Tablet, which is a unique identifier for the Tablet.
ROWSET_NUM_ROWS	bigint	The number of data rows contained in the Rowset.
TXN_ID	bigint	The transaction ID that wrote to the Rowset.
NUM_SEGMENTS	bigint	The number of Segments contained in the Rowset.
START_VERSION	bigint	The starting version number of the Rowset.
END_VERSION	bigint	The ending version number of the Rowset.
INDEX_DISK_SIZE	bigint	The storage space for indexes within the Rowset.
DATA_DISK_SIZE	bigint	The storage space for data within the Rowset.
CREATION_TIME	datetime	The creation time of the Rowset.
NEWEST_WRITE_TIMESTAMP	datetime	The most recent write time of the Rowset.
SCHEMA_VERSION	int	The Schema version number of the table corresponding to the Rowset data.

schema_privileges

Overview

View the authorization information of the database.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
GRANTEE	varchar(81)	The authorized user
TABLE_CATALOG	varchar(512)	The Catalog name, always ‘def’
TABLE_SCHEMA	varchar(64)	The Database name
PRIVILEGE_TYPE	varchar(64)	The type of privilege
IS_GRANTABLE	varchar(3)	Whether authorization can be granted to other users

schemata

Overview

View information related to the database.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
CATALOG_NAME	varchar(512)	The name of the Catalog
SCHEMA_NAME	varchar(32)	The name of the Database
DEFAULT_CHARACTER_SET_NAME	varchar(32)	For compatibility with MySQL only, no actual function
DEFAULT_COLLATION_NAME	varchar(32)	For compatibility with MySQL only, no actual function
SQL_PATH	varchar(512)	For compatibility with MySQL only, no actual function
DEFAULT_ENCRYPTION	varchar(3)	For compatibility with MySQL only, no actual function

session_variables

Overview

View session variable information.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
VARIABLE_NAME	varchar(64)	The name of the variable
VARIABLE_VALUE	varchar(1024)	The current value
DEFAULT_VALUE	varchar(1024)	The default value
CHANGED	varchar(4)	Whether the current value differs from the default value

statistics

Overview

This table is solely for compatibility with MySQL behavior. It is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
TABLE_CATALOG	varchar(512)	
TABLE_SCHEMA	varchar(64)	
TABLE_NAME	varchar(64)	
NON_UNIQUE	bigint	
INDEX_SCHEMA	varchar(64)	
INDEX_NAME	varchar(64)	
SEQ_IN_INDEX	bigint	
COLUMN_NAME	varchar(64)	
COLLATION	varchar(1)	
CARDINALITY	bigint	
SUB_PART	bigint	
PACKED	varchar(10)	
NULLABLE	varchar(3)	
INDEX_TYPE	varchar(16)	
COMMENT	varchar(16)	
INDEX_COMMENT	varchar(1024)	

table_constraints

Overview

This table is solely for the purpose of maintaining compatibility with MySQL behavior. It is always empty.

Database

`information_schema`

Table Information

Column Name	Type	Description
CONSTRAINT_CATALOG	varchar(512)	
CONSTRAINT_SCHEMA	varchar(64)	
CONSTRAINT_NAME	varchar(64)	
TABLE_SCHEMA	varchar(64)	
TABLE_NAME	varchar(64)	
CONSTRAINT_TYPE	varchar(64)	

table_options

Overview

This table is solely used for compatibility with MySQL behavior. It is always empty.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
TABLE_CATALOG	varchar(64)	
TABLE_SCHEMA	varchar(64)	
TABLE_NAME	varchar(64)	
TABLE_MODEL	text	
TABLE_MODEL_KEY	text	
DISTRIBUTE_KEY	text	
DISTRIBUTE_TYPE	text	
BUCKETS_NUM	int	
PARTITION_NUM	int	

table_privileges

Overview

View the authorization information of the table.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
GRANTEE	varchar(81)	The authorized user
TABLE_CATALOG	varchar(512)	The name of the Catalog
TABLE_SCHEMA	varchar(64)	The name of the Database
TABLE_NAME	varchar(64)	The name of the Table
PRIVILEGE_TYPE	varchar(64)	The type of privilege
IS_GRANTABLE	varchar(3)	Whether the privilege can be granted to others

table_properties

Overview

Used to view attribute information of tables (including internal and external tables).

Database

`information_schema`

Table Information

Column Name	Type	Description
TABLE_CATALOG	varchar(64)	The Catalog to which the table belongs
TABLE_SCHEMA	varchar(64)	The Database to which the table belongs
TABLE_NAME	varchar(64)	The name of the table
PROPERTY_NAME	string	The name of the property
PROPERTY_VALUE	string	The value of the property

tables

Overview

Stores all table information.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
TABLE_CATALOG	varchar(512)	The Catalog to which the table belongs
TABLE_SCHEMA	varchar(64)	The Database to which the table belongs
TABLE_NAME	varchar(64)	The name of the table
TABLE_TYPE	varchar(64)	The type of the table, including: SYSTEM VIEW, VIEW, BASE TABLE
ENGINE	varchar(64)	The storage engine type of the table
VERSION	bigint	Invalid value
ROW_FORMAT	varchar(10)	Invalid value
TABLE_ROWS	bigint	Estimated number of rows in the table
AVG_ROW_LENGTH	bigint	Average row size of the table
DATA_LENGTH	bigint	Estimated size of the table
MAX_DATA_LENGTH	bigint	Invalid value
INDEX_LENGTH	bigint	Invalid value
DATA_FREE	bigint	Invalid value
AUTO_INCREMENT	bigint	Invalid value
CREATE_TIME	datetime	The time when the table was created
UPDATE_TIME	datetime	The time when the table data was last updated
CHECK_TIME	datetime	Invalid value
TABLE_COLLATION	varchar(32)	Fixed value: utf-8
CHECKSUM	bigint	Invalid value
CREATE_OPTIONS	varchar(255)	Invalid value
TABLE_COMMENT	varchar(2048)	Comments on the table

triggers

Overview

The `triggers` table provides information about triggers. Currently, Apache Doris supports this table for MySQL compatibility, but does not support user-defined triggers. This table is always empty.

Database

`information_schema`

Table Information

Column Name	Type	Description
TRIGGER_CATALOG	varchar(512)	The name of the catalog to which the trigger belongs. Always ‘def’.
TRIGGER_SCHEMA	varchar(64)	The name of the schema (database) to which the trigger belongs.
TRIGGER_NAME	varchar(64)	The name of the trigger.
EVENT_MANIPULATION	varchar(6)	The trigger event (INSERT, UPDATE, DELETE).
EVENT_OBJECT_CATALOG	varchar(512)	The catalog name of the table with which the trigger is associated. Always ‘def’.
EVENT_OBJECT_SCHEMA	varchar(64)	The schema (database) name of the table with which the trigger is associated.
EVENT_OBJECT_TABLE	varchar(64)	The name of the table with which the trigger is associated.
ACTION_ORDER	bigint	The ordinal definition order of the trigger.
ACTION_CONDITION	varchar(512)	null
ACTION_STATEMENT	varchar(512)	The trigger body.
ACTION_ORIENTATION	varchar(9)	Always ‘ROW’.
ACTION_TIMING	varchar(6)	Trigger timing (BEFORE, AFTER).
ACTION_REFERENCE_OLD_TABLE	varchar(64)	null
ACTION_REFERENCE_NEW_TABLE	varchar(64)	null
ACTION_REFERENCE_OLD_ROW	varchar(3)	Always ‘OLD’.
ACTION_REFERENCE_NEW_ROW	varchar(3)	Always ‘NEW’.
CREATED	datetime	The time when the trigger was created.
SQL_MODE	varchar(8192)	The SQL mode in effect when the trigger was created.

Column Name	Type	Description
DEFINER	varchar(77)	The account that created the trigger.
CHARACTER_SET_CLIENT	varchar(32)	The session value of the character_set_client system variable when the trigger was created.
COLLATION_CONNECTION	varchar(32)	The session value of the collation_connection system variable when the trigger was created.
DATABASE_COLLATION	varchar(32)	The collation of the database with which the trigger is associated.

user_privileges

Overview

View user authorization information.

Database

`information_schema`

Table Information

Column Name	Type	Description
GRANTEE	varchar(81)	The user who is granted the privilege
TABLE_CATALOG	varchar(512)	Always ‘def’
PRIVILEGE_TYPE	varchar(64)	Type of privilege
IS_GRANTABLE	varchar(3)	Whether the privilege can be granted to others

views

Overview

Stores all view information.

Database

`information_schema`

Table Information

Column Name	Type	Description
TABLE_CATALOG	varchar(512)	Catalog name
TABLE_SCHEMA	varchar(64)	Database name
TABLE_NAME	varchar(64)	View name
VIEW_DEFINITION	varchar(8096)	View definition statement
CHECK_OPTION	varchar(8)	No practical effect, only for MySQL compatibility
IS_UPDATABLE	varchar(3)	No practical effect, only for MySQL compatibility
DEFINER	varchar(77)	No practical effect, only for MySQL compatibility
SECURITY_TYPE	varchar(7)	No practical effect, only for MySQL compatibility
CHARACTER_SET_CLIENT	varchar(32)	No practical effect, only for MySQL compatibility
COLLATION_CONNECTION	varchar(32)	No practical effect, only for MySQL compatibility

workload_group_privileges

Overview

Stores permission information for Workload Groups.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
GRANTEE	varchar(64)	The user granted permissions
WORKLOAD_GROUP_NAME	varchar(256)	The name of the Workload Group
PRIVILEGE_TYPE	varchar(64)	The type of privilege
IS_GRANTABLE	varchar(3)	Whether it can be granted to other users

workload_group_resource_usage

Overview

Stores usage information for Workload Group resources.

Database

`information_schema`

Table Information

Column Name	Type	Description
BE_ID	bigint	The ID of the Backend
WORKLOAD_GROUP_ID	bigint	The ID of the Workload Group
MEMORY_USAGE_BYTES	bigint	Memory usage in bytes
CPU_USAGE_PERCENT	double	CPU usage percentage
LOCAL_SCAN_BYTES_PER_SECOND	bigint	Local scan data rate in bytes per second
REMOTE_SCAN_BYTES_PER_SECOND	bigint	Remote scan data rate in bytes per second

workload_groups

Overview

Records the configuration information of Workload Groups.

Database

```
information_schema
```

Table Information

Column Name	Type	Description
ID	bigint	ID of the Workload Group
NAME	varchar(256)	Name of the Workload Group
CPU_SHARE	bigint	Soft limit weight for CPU of the Workload Group
MEMORY_LIMIT	varchar(256)	Memory limit for the Workload Group
ENABLE_MEMORY_OVERCOMMIT	varchar(256)	Whether to enable soft limit for memory of the Workload Group
MAX_CONCURRENCY	bigint	Maximum concurrency for the Workload Group
MAX_QUEUE_SIZE	bigint	Maximum queue size for the Workload Group
QUEUE_TIMEOUT	bigint	Queue timeout for the Workload Group
CPU_HARD_LIMIT	varchar(256)	Hard limit size for CPU of the Workload Group
SCAN_THREAD_NUM	bigint	Number of threads for local scan
MAX_REMOTE_SCAN_THREAD_NUM	bigint	Maximum number of threads in the remote scan thread pool
MIN_REMOTE_SCAN_THREAD_NUM	bigint	Minimum number of threads in the remote scan thread pool
SPILL_THRESHOLD_LOW_WATERMARK	varchar(256)	Low watermark for disk spill of the Workload Group
SPILL_THRESHOLD_HIGH_WATERMARK	varchar(256)	High watermark for disk spill of the Workload Group
TAG	varchar(256)	Tag for the Workload Group
READ_BYTES_PER_SECOND	bigint	Bytes scanned per second for local reads
REMOTE_READ_BYTES_PER_SECOND	bigint	Bytes scanned per second for remote reads

workload_policy

Overview

Records the configuration information of Workload Policies.

Database

`information_schema`

Table Information

Column Name	Type	Description
ID	bigint	ID of the Workload Policy
NAME	varchar(256)	Name of the Workload Policy
CONDITION	text	Condition of the Workload Policy
ACTION	text	Action of the Workload Policy
PRIORITY	int	Priority of the Workload Policy
ENABLED	boolean	Whether the Workload Policy is enabled
VERSION	int	Version of the Workload Policy
WORKLOAD_GROUP	text	Name of the Workload Group bound to the Policy

procs_priv

Overview

This table is solely for the purpose of compatibility with MySQL behavior. It is always empty.

Database

`mysql`

Table Information

Column Name	Type	Description
host	char(60)	
db	char(64)	
user	char(32)	
routine_name	char(64)	
routine_type	char(9)	
grantor	char(93)	
proc_priv	char(16)	
timestamp	char(1)	

user

Overview

View all user information.

Database

`mysql`

Table Information

Column Name	Type	Description
host	character(255)	The host from which the user is allowed to connect.
user	char(32)	Username.
node_priv	char(1)	Whether the user has Node privileges.
admin_priv	char(1)	Whether the user has Admin privileges.
grant_priv	char(1)	Whether the user has Grant privileges.
select_priv	char(1)	Whether the user has Select privileges.
load_priv	char(1)	Whether the user has Load privileges.
alter_priv	char(1)	Whether the user has Alter privileges.
create_priv	char(1)	Whether the user has Create privileges.
drop_priv	char(1)	Whether the user has Drop privileges.
usage_priv	char(1)	Whether the user has Usage privileges.
show_view_priv	char(1)	Whether the user has Show View privileges.
cluster_usage_priv	char(1)	Whether the user has Cluster usage privileges.
stage_usage_priv	char(1)	Whether the user has Stage usage privileges.
ssl_type	char(9)	Always empty, for MySQL compatibility only.
ssl_cipher	varchar(65533)	Always empty, for MySQL compatibility only.
x509_issuer	varchar(65533)	Always empty, for MySQL compatibility only.
x509_subject	varchar(65533)	Always empty, for MySQL compatibility only.
max_questions	bigint	Always 0, for MySQL compatibility only.
max_updates	bigint	Always 0, for MySQL compatibility only.
max_connections	bigint	Always 0, for MySQL compatibility only.
max_user_connections	bigint	The maximum number of allowed connections.
plugin	char(64)	Always empty, for MySQL compatibility only.

Column Name	Type	Description
authentication_string	varchar(65533)	Always empty, for MySQL compatibility only.
password_policy.expiration_seconds	varchar(32)	Password expiration time.
password_policy.password_creation_time	varchar(32)	Password creation time.
password_policy.history_num	varchar(32)	Number of historical passwords.
password_policy.history_passwords	varchar(65533)	Historical passwords.
password_policy.num_failed_login	varchar(32)	Allowed number of consecutive login failures.
password_policy.password_lock_seconds	varchar(32)	Password lock time after triggering lock.
password_policy.failed_login_counter	varchar(32)	Login failure count.
password_policy.lock_time	varchar(32)	

audit_log

Overview

Store audit logs

Database

`__internal_schema`

Table Information

Column Name	Type	Description
query_id	varchar(48)	ID of the Query
time	datetime(3)	Time when the query was executed(in milliseconds)
client_ip	varchar(128)	IP address of the client sending the query
user	varchar(128)	User
catalog	varchar(128)	Current Catalog during statement execution
db	varchar(128)	Current Database during statement execution
state	varchar(128)	Execution status of the statement
error_code	int	Error code
error_message	text	Error message
query_time	bigint	Execution time of the statement
scan_bytes	bigint	Amount of data scanned
scan_rows	bigint	Number of rows scanned
return_rows	bigint	Number of rows returned
shuffleSendRows	bigint	The number of rows transferred between nodes during statement execution. Supported since version 3.0.
shuffleSendBytes	bigint	The amount of data transferred between nodes during statement execution. Supported since version 3.0.
scanBytesFromLocalStorage	bigint	The amount of data read from the local disk. Supported since version 3.0.
scanBytesFromRemoteStorage	bigint	The amount of data read from the remote storage. Supported since version 3.0.
stmt_id	bigint	Statement ID
stmt_type	string	Statement type. Supported since version 3.0.
is_query	tinyint	Whether it is a query

Column Name	Type	Description
is_nereids	boolean	Is using Nereids Optimizer.
frontend_ip	varchar(128)	IP address of the connected Frontend
cpu_time_ms	bigint	Cumulative CPU time (in milliseconds) consumed by the Backend for statement execution
sql_hash	varchar(128)	Hash value of the statement
sql_digest	varchar(128)	Digest (signature) of the statement
peak_memory_bytes	bigint	Peak memory usage of the Backend during statement execution
workload_group	text	Workload Group used for statement execution
compute_group	string	In storage and computation decoupled mode, the compute group used by the execution statement. Supported since version 3.0.
trace_id	string	Trace ID set when executing the statement. Removed since version 2.1.7
stmt	text	Statement text

Description

- `client_ip`: If a proxy service is used and the IP pass-through is not enabled, the proxy service IP may be recorded here instead of the real client IP.
- `state`: `EOF` indicates that the query is executed successfully. `OK` indicates that the DDL and DML statements are executed successfully. `ERR` indicates that the statement execution fails.

Column Statistics | Internal Schema

Column Statistics

Overview

Column statistics

Database

`__internal_schema`

Table Information

Column Name	Type	Description
id	varchar(4096)	Unique ID
catalog_id	varchar(64)	ID of the Catalog
db_id	varchar(64)	ID of the Database
tbl_id	varchar(64)	ID of the Table
idx_id	varchar(64)	ID of the Index
col_id	varchar(64)	ID of the column, currently storing column names
part_id	varchar(64)	ID of the Partition, always empty
count	bigint	Number of rows
ndv	bigint	Number of distinct values
null_count	bigint	Number of NULLs
min	varchar(65533)	Minimum value
max	varchar(65533)	Maximum value
data_size_in_bytes	bigint	Data size in bytes
update_time	datetime	Update time of current statistics

partition_statistics

Overview

Partition statistics

Database

`__internal_schema`

Table Information

Column Name	Type	Description
catalog_id	varchar(64)	ID of the Catalog
db_id	varchar(64)	ID of the Database
tbl_id	varchar(64)	ID of the Table
idx_id	varchar(64)	ID of the Index
part_name	varchar(64)	Name of the Partition
part_id	bigint	ID of the Partition
col_id	varchar(64)	ID of the column, currently storing column names
count	bigint	Number of rows
ndv	hll	Number of distinct values
null_count	bigint	Number of NULLs
min	varchar(65533)	Minimum value
max	varchar(65533)	Maximum value
data_size_in_bytes	bigint	Data size in bytes
update_time	datetime	Last update time of current statistics

Overview | Memory Management

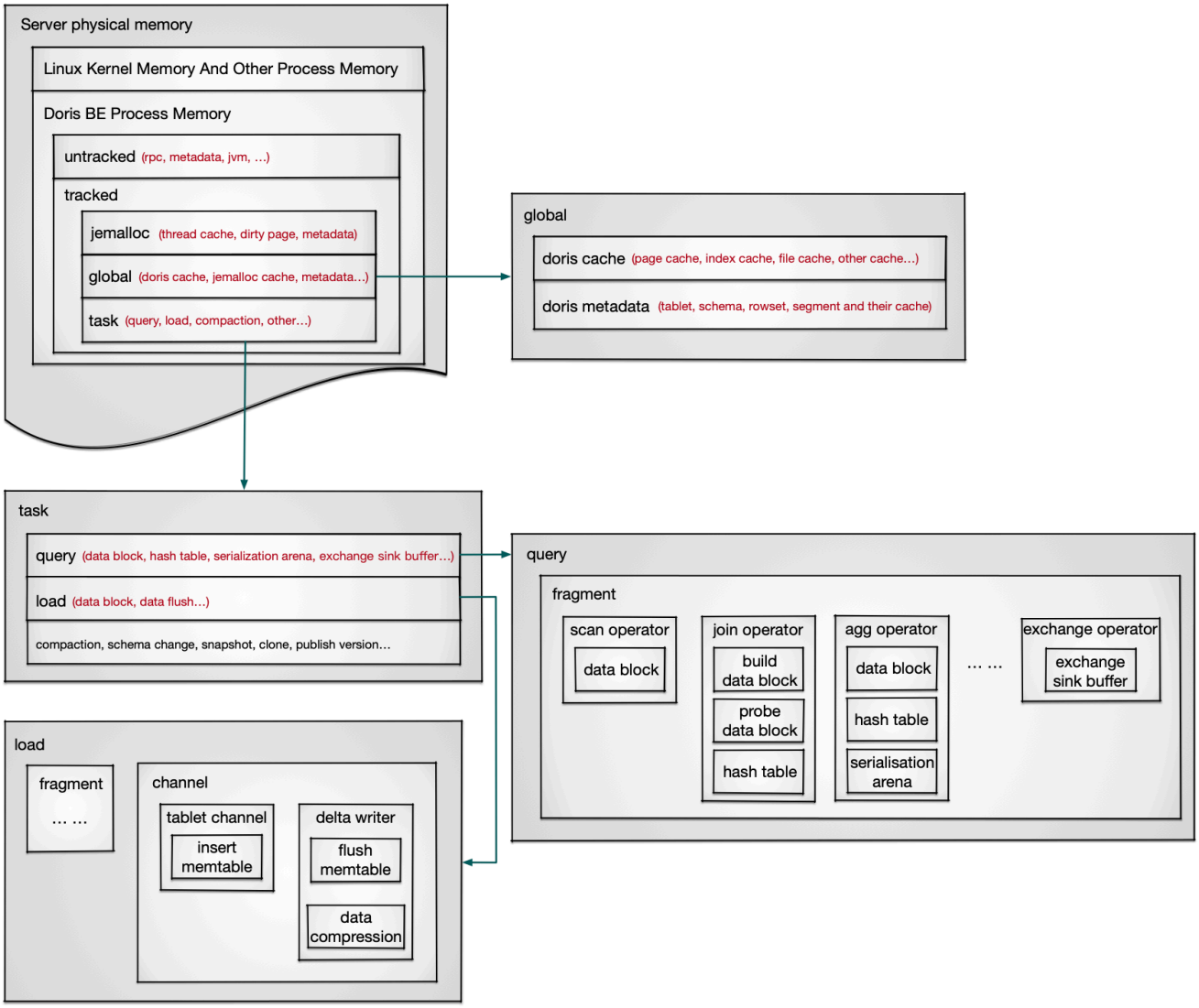
Overview

Memory management is one of the most important components of Doris. During the operation of Doris, both load and query rely on a large number of memory operations. The quality of memory management directly affects the stability and performance of Doris.

As an OLAP database based on the MPP architecture, Apache Doris will stream and calculate between operators after loading data from disk to memory, and store the intermediate results of calculation in memory. This method reduces frequent disk I/O operations and makes full use of the parallel computing capabilities of multiple machines and multiple cores, which can show huge advantages in performance.

When facing complex calculations and large-scale operations with huge memory resource consumption, effective memory allocation, statistics, and control play a very critical role in the stability of the system - faster memory allocation speed will effectively improve query performance, and memory allocation, tracking and restriction can ensure that there is no memory hotspot, timely and accurately respond to insufficient memory and avoid OOM and query failure as much as possible. This series of mechanisms will significantly improve system stability; at the same time, more accurate memory statistics are also the basis for large queries to fall to disk.

Doris BE memory structure



Memory Structure

```
Server physical memory: The physical memory used by all processes on the server, MemTotal seen by `cat /proc/meminfo` or `free -h`.
    |
    |---> Linux Kernel Memory And Other Process Memory: Memory used by the Linux kernel and other processes.
    |
    |---> Doris BE Process Memory: The memory used by the Doris BE process. The upper limit is the server physical memory minus the
memory used by the Linux kernel and other processes, or the memory size configured by the Cgroup.
           |
           |---> Untracked: Memory that is not tracked and managed, including RPC, JVM, some metadata, etc. JVM is used when
accessing external tables or using Java UDF.
           |
           |---> tracked: Memory that is tracked and managed, allowing real-time viewing, automatic memory recycling, and size
control through parameters.
                  |
                  |---> jemalloc: The cache and metadata managed by jemalloc support parameter control and are automatically
recycled when memory is insufficient.
                  |
                  |---> global: Doris globally shared memory, mainly including cache and metadata.
                  |       |
                  |       |---> doris cache: Doris manages its own cache, which supports controlling capacity and elimination time
through parameters separately, and automatically reclaims when memory is insufficient.
                  |       |
                  |       |---> doris metadata: metadata of data stored on BE, including a series of memory data structures such
```

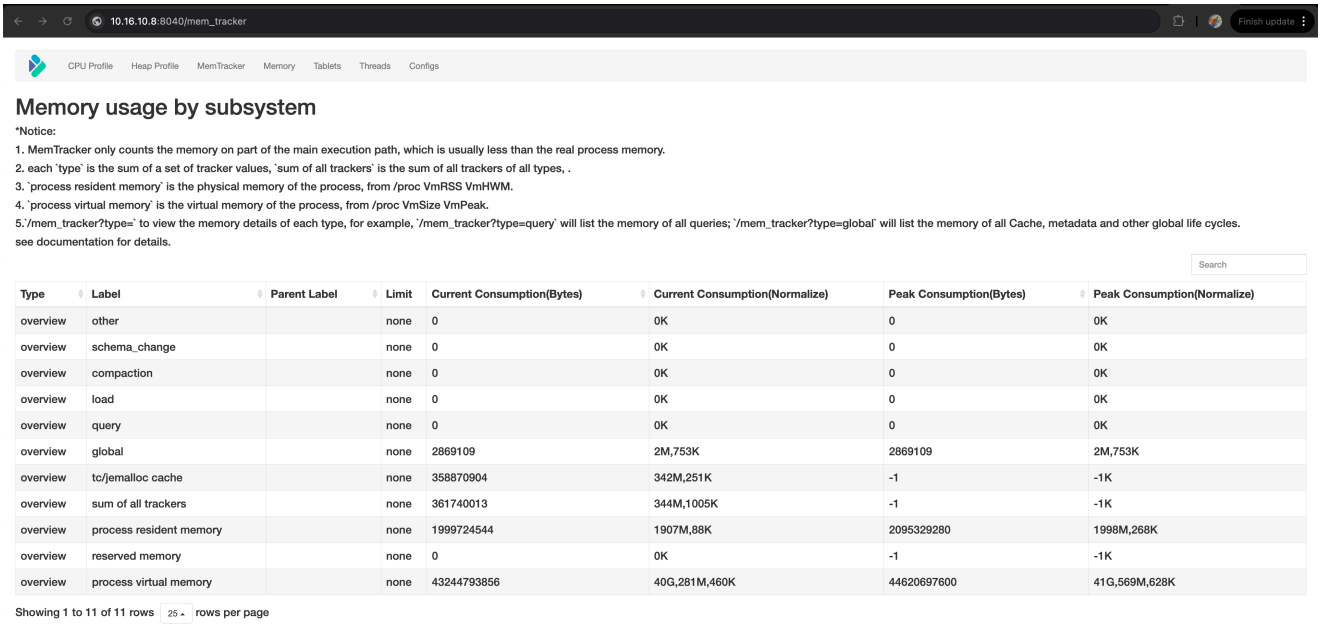
```
as data schema and their caches.
                    |
                    |---> Task: The memory used by the tasks executed on Doris, which is expected to be released after the task is
completed, including query, load, compaction, etc.
                    |       |
                    |       |---> query: Memory used during the query. A query is split into multiple fragments and executed
separately, connected by data shuffle.
                    |       |       |
                    |       |       |---> Fragment: A fragment is split into multiple operators and executed in the form of a
pipeline.
                    |       |       |       |
                    |       |       |       |---> operator: includes memory data structures such as data block, hash table, arena,
exchange sink buffer, etc.
                    |       |
                    |       |---> load: Memory used during data load. Data load includes two stages: fragment reading and channel
writing data.
                    |       |       |
                    |       |       |---> fragment: Same as query fragment execution, stream load usually only has scan operator.
                    |       |       |
                    |       |       |---> channel: The tablet channel writes data to a temporary data structure called memtable, and
then the delta writer compresses the data and writes it to the file.
```

Memory View

Doris BE uses Memory Tracker to record process memory usage, supports Web page viewing, and prints to BE logs when memory-related errors are reported, for memory analysis and troubleshooting.

Real-time Memory Statistics

Real-time memory statistics can be viewed through Doris BE's Web page `http://{be_host}:{be_web_server_port}/mem_tracker`, which displays the current memory size and peak memory size tracked by Memory Tracker of `type=overview`, including Query/Load/Compaction/Global, etc. `be_web_server_port` defaults to 8040.



image

Memory Tracker is divided into different types. Among the Memory Tracker of type=overview, except for `process resident memory`, `process virtual memory`, and `sum of all trackers`, the details of other Memory Trackers of type=overview can be viewed through `http://{be_host}:{be_web_server_port}/mem_tracker?type=Label`.

Memory Tracker has the following properties:

1. Label: the name of the Memory Tracker
2. Current Consumption(Bytes): the current memory value, in B.
3. Current Consumption(Normalize): the .G.M.K formatted output of the current memory value.
4. Peak Consumption (Bytes): The peak memory value after the BE process is started, in units of B, and reset after the BE is restarted.
5. Peak Consumption (Normalize): The .G.M.K formatted output of the peak memory value after the BE process is started, and reset after the BE is restarted.
6. Parent Label: Used to indicate the parent-child relationship between two memory trackers. The memory recorded by the Child Tracker is a subset of the Parent Tracker. The memory recorded by different trackers with the same Parent may have an intersection.

For more information about Memory Tracker, refer to Memory Tracker.

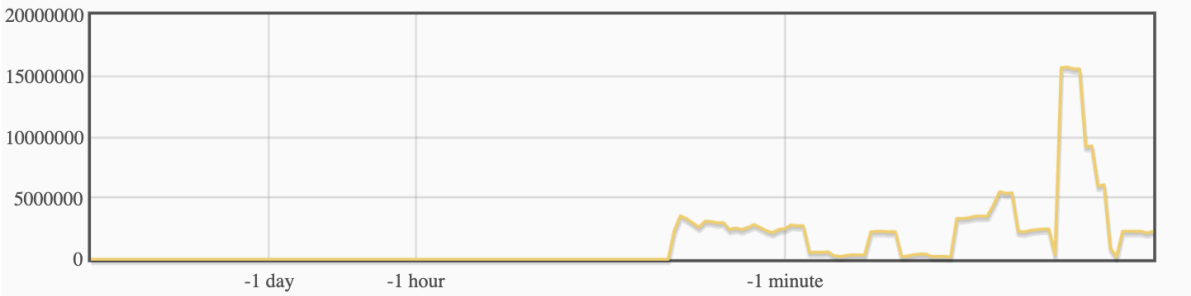
Historical memory statistics

Historical memory statistics can be viewed through Doris BE's Bvar page `http://{be_host}:{brpc_port}/vars/*memory_*`. Use the real-time memory statistics page `http://{be_host}:{be_web_server_port}/mem_tracker` to search for the Bvar page under the Memory Tracker Label to get the memory size change trend tracked by the corresponding Memory Tracker. `brpc_port` defaults to 8060.

status | vars | connections | flags | rpcz | cpu | heap | growth | co

Search :

doris_column_reader_memory_bytes : 0
doris_index_reader_memory_bytes : 0
doris_ordinal_index_memory_bytes : 0
doris_primary_key_index_memory_bytes : 0
doris_short_key_index_memory_bytes : 0
doris_zone_map_memory_bytes : 0
meminfo_process_memory_usage : 2102392648
memory_all_trackers_sum_bytes : 217250632
memory_arbitrator_process_usage_bytes : 2102392648
memory_arbitrator_refresh_interval_growth_bytes : 0
memory_arbitrator_reserve_memory_bytes : 0
memory_arbitrator_sys_available_bytes : 135568199680
memory_cgroup_usage_bytes : 8883695616
memory_compaction_trackers_sum_bytes : 0
memory_global_trackers_sum_bytes : 0
memory_jemalloc_cache_bytes : 113617080
memory_jemalloc_dirty_pages_bytes : 2277376



memory_jemalloc_metadata_bytes : 118172416
memory_jemalloc_virtual_bytes : 2086457344
memory_load_trackers_sum_bytes : 0
memory_memtrackerlimiter_cnt : 14
memory_other_trackers_sum_bytes : 0
memory_query_trackers_sum_bytes : 0
memory_schema_change_trackers_sum_bytes : 0
memory_sys_available_bytes : 135568199680
normal_memory_used : 0
process_memory_data_and_stack : 23652200448
process_memory_resident : 2216009728
process_memory_shared : 74240000
process_memory_text : 487174144
process_memory_virtual : 25661272064

Bvar Memory

When the error process memory exceeds the limit or the available memory is insufficient, you can find the `Memory Tracker Summary` in the `be/log/be.INFO` log, which contains all the Memory Trackers of `Type=overview` and `Type=global`, to help users analyze the memory status at that time. For details, please refer to Memory Log Analysis

Memory Analysis

Correspond the Memory Tracker of `type=overview` to each part of the memory under `tracked` in the above memory structure:

```
Doris BE Process Memory
    |
    |---> tracked: corresponds to `MemTrackerLimiter Label=sum of all trackers, Type=overview`, which is all the memory counted by the Memory Tracker, that is, the sum of the Current Consumption of other Memory Trackers with `type=overview` except `Label=process resident memory` and `Label=process virtual memory`.
            |
            |---> jemalloc
            |       |
            |       |---> jemalloc cache: corresponds to `MemTrackerLimiter Label=tc/jemalloc_cache, Type=overview`. The Jemalloc cache includes two parts: Dirty Page and Thread Cache.
            |       |
            |       |---> jemalloc metadata: Corresponding to `MemTrackerLimiter Label=tc/jemalloc_metadata, Type=overview`, Metadata of Jemalloc.
            |
            |---> global: corresponds to `MemTrackerLimiter Label=global, Type=overview`, including global Memory Trackers with the same life cycle and process such as Cache, metadata, and decompression. The web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=global` displays all Memory Trackers of `type=global`.
            |
            |---> task
            |       |
            |       |---> query: corresponds to `MemTrackerLimiter Label=query, Type=overview`, which is the sum of Current Consumption of all Query Memory Trackers. The web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=query` displays all Memory Trackers of `type=query`.
            |       |
            |       |---> load: corresponds to `MemTrackerLimiter Label=load, Type=overview`, the sum of Current Consumption of all Load Memory Trackers. The web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=load` displays all Memory Trackers of `type=load`.
            |       |
            |       |---> reserved: corresponds to `MemTrackerLimiter Label=reserved_memory, Type=overview`. The reserved memory is used when query the Hash Table and other memory-intensive behaviors. Before query the Hash Table, the memory of the Hash Table to be constructed will be reserved from the Memory Tracker to ensure that subsequent memory requests can be met.
            |       |
            |       |---> compaction: corresponds to `MemTrackerLimiter Label=compaction, Type=overview`, the sum of Current Consumption of all Compaction Memory Trackers. The web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=compaction` displays all Memory Trackers of `type=compaction`.
            |       |
            |       |---> schema_change: corresponds to `MemTrackerLimiter Label=schema_change, Type=overview`, the sum of Current Consumption of all Schema Change Memory Trackers. The web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=schema_change` displays all Memory Trackers of `type=schema_change`.

            |       |
            |       |---> other: corresponds to `MemTrackerLimiter Label=other, Type=overview`, the total memory of other tasks except the above, such as EngineAlterTabletTask, EngineCloneTask, CloudEngineCalcDeleteBitmapTask, SnapshotManager, etc. The web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=other` displays all Memory Trackers of `type=other`.
    |
    |---> Doris BE process physical memory, corresponding to `MemTrackerLimiter Label=process resident memory, Type=overview`, Current Consumption is taken from VmRSS in `/proc/self/status`, Peak Consumption is taken from VmHWM in `/proc/self/status`.
    |
    |---> Doris BE process virtual memory, corresponding to `MemTrackerLimiter Label=process virtual memory, Type=overview`, Current Consumption is taken from VmSize in `/proc/self/status`, and Peak Consumption is taken from VmPeak in `/proc/self/status`.
```

Analysis methods for each part of the memory in the above memory structure:

1. Jemalloc memory analysis
2. Global memory analysis

3. Query memory analysis
4. Load memory analysis

Memory problem FAQ

Refer to Memory problem FAQ to analyze common memory problems.

Memory control strategy

Refer to Memory Control Strategy for an introduction to memory allocation, monitoring, and recycling, which ensure the efficient and controllable memory of the Doris BE process.

Memory Issue FAQ

Doris BE process memory analysis mainly uses `be/log/be.INFO` log, BE process memory monitoring (Metrics), Doris Bvar statistics. If OOM Killer is triggered, you need to collect `dmesg -T` execution results. If you analyze the memory of query or load tasks, you need to collect Query Profile. Analyze common memory problems based on this information. If you cannot solve the problem by yourself, you need to ask Doris developers for help. No matter which way you use (submit issue on Github, create issue on Doris forum, email or WeChat), please add the above information to your problem description.

First locate which memory problem the currently observed phenomenon belongs to, and further investigate. Usually, you need to analyze the process memory log first. Refer to Memory Log Analysis. Common memory problems are listed below.

1 Query and load memory limit error

When `MEM_LIMIT_EXCEEDED` appears in the query and load error message, it means that the task is canceled because the process has insufficient available memory or the task exceeds the memory limit for a single execution.

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.8)[MEM_LIMIT_EXCEEDED] xxxx .
```

If the error message contains `Process memory not enough`, it means that the process has insufficient available memory. Refer to Query or load error process has insufficient available memory for further analysis.

If `memory tracker limit exceeded` appears in the error message, it means that the task exceeds the single execution memory limit. Refer to Query or load error exceeds single execution memory limit for further analysis.

2 Doris BE OOM Crash

If there is no error message in `log/be.out` after the BE process crashes, execute `dmesg -T`. If you see `Out of memory: Killed process {pid} (doris_be)`, it means that the OOM Killer is triggered. Refer to OOM Killer Crash Analysis for further analysis.

3 Memory Leak

> If you encounter a suspected memory leak, the best solution is to upgrade to the latest three-digit version. If you are using Doris 2.0, upgrade to the latest version of Doris 2.0.x, because it is likely that other people have encountered the same phenomenon, and most memory leaks are fixed in version iterations.

If the following phenomenon is observed, it means that there may be a memory leak:

- Doris Grafana or server monitoring finds that the memory of the Doris BE process has been growing linearly, and the memory does not decrease after the task on the cluster stops.

- Memory Tracker has missing statistics, refer to Memory Tracker for analysis.

Memory leaks are usually accompanied by missing statistics in Memory Tracker, so the analysis method also refers to the [Memory Tracker] section.

4 Doris BE process memory does not decrease or continues to increase

If Doris Grafana or server monitoring finds that the memory of the Doris BE process has been increasing linearly, and the memory does not decrease after the task on the cluster is stopped, first refer to Memory Tracker in Memory Tracker to analyze whether there is missing statistics in Memory Tracker. If there is missing statistics in Memory Tracker, further analyze the cause.

If Memory Tracker does not have missing statistics and has counted most of the memory, refer to Overview to analyze the reasons why different parts of the Doris BE process occupy too much memory and how to reduce its memory usage.

5 Large virtual memory usage

`Label=process virtual memory` Memory Tracker displays the real-time virtual memory size, which is the same as the virtual memory of the Doris BE process seen by `top -p {pid}`.

```
MemTrackerLimiter Label=process virtual memory, Type=overview, Limit=-1.00 B(-1 B), Used=44.25 GB(47512956928 B), Peak=44.25
GB(47512956928 B)
```

Doris currently still has the problem of Doris BE process virtual memory being too large, usually because Jemalloc retains a large number of virtual memory mappings, which will also cause Jemalloc Metadata to occupy too much memory. Refer to Jemalloc Memory Analysis for the analysis of Jemalloc Metadata memory.

In addition, it is known that Doris's Join Operator and Column lack memory reuse, which will cause more virtual memory to be requested in some scenarios and eventually cached in Jemalloc Retained. There is currently no good solution. It is recommended to restart the Doris BE process regularly.

6 The process memory is very large after the BE process is just started

This is usually because the metadata memory loaded when the BE process starts is too large. Refer to Metadata Memory Analysis to view Doris BE Bvar.

- If `doris_total_tablet_num` is too much, it is usually because the number of partitions and buckets of the table is too large. Check `{fe_host}:{fe_http_port}/System?path=//dbs` to find a table with a large number of tablets. The number of tablets of a table is equal to the number of its partitions multiplied by the number of buckets. Try to reduce the number of its partitions and buckets. Or delete outdated tables or partitions that will not be used.
- If `doris_total_rowset_num` is too large but the number of tablets is small, refer to the `SHOW-PROC` document to find tables with many rowsets but few tablets, and then manually trigger compaction, or wait for automatic compaction to complete. For details, refer to the metadata management related documents. It is normal for metadata to occupy several GB when there are hundreds of thousands of rowsets.

- If `tablet_meta_schema_columns_count` is too large, hundreds or thousands of times larger than `doris_total_tablet_schema_num`, it means that there are large wide tables with hundreds or thousands of columns in the cluster. At this time, the same number of tablets will occupy more memory.

7 Query does not have complex operators but simply scans data, but it uses a lot of memory

It may be the memory occupied by the Column Reader and Index Read opened when reading the Segment. Refer to Metadata Memory Analysis to view the changes of `doris_total_segment_num`, `doris_column_reader_num`, `doris_ordinal_index_memory_bytes`, `doris_zone_map_memory_bytes`, and `doris_short_key_index_memory_bytes` in Doris BE Bvar. This phenomenon is also common when reading large wide tables. When hundreds of thousands of Column Readers are opened, the memory may occupy tens of GB.

If you see the `Segment` and `ColumnReader` fields in the call stack with a large memory share in the Heap Profile, it can be basically confirmed that a large amount of memory is occupied when reading the Segment.

At this time, you can only modify the SQL to reduce the amount of data scanned, or reduce the bucket size specified when creating the table, so as to open fewer segments.

8. Query Cancel stuck

> Common before Doris 2.1.3

Most of the memory requested during query execution needs to be released when the query ends. When the process memory is sufficient, there is usually no need to pay attention to how fast or slow the query ends. However, when the process memory is insufficient, some queries are often canceled according to a certain strategy to release their memory and avoid the process triggering OOM Killer. At this time, if the query cancel process is stuck and the memory cannot be released in time, in addition to increasing the risk of triggering OOM Killer, it may also cause more queries to be canceled due to insufficient process memory.

If a query is known to be canceled, the following is to analyze whether it is stuck in the cancel process based on this QueryID. First, execute `grep {queryID} be/log/be.INFO` to find the first log containing the `Cancel` keyword. The corresponding time point is the time when the query is canceled. Find the log containing the keyword `deregister query/load memory tracker`. The corresponding time point is the time when the query cancel is completed. If the OOM Killer is finally triggered, and no log containing the keyword `deregister query/load memory tracker` is found, it means that the query has not been canceled until the OOM Killer occurs. Usually, if the query cancel process takes more than 3s, the query is stuck in the cancel process, and the query execution log needs to be further analyzed.

In addition, after executing `grep {queryID} be/log/be.INFO`, if you see a log containing the keyword `tasks is being canceled and has not been completed yet`, the QueryID list behind it means that when the Memory GC is executed, it is found that these queries are being canceled but not canceled. At this time, these queries will be skipped and memory will continue to be released elsewhere. This can be used to determine whether the behavior of the Memory GC is as expected.

Jemalloc Memory Analysis

Doris uses Jemalloc as the general memory allocator by default. The memory occupied by Jemalloc itself includes Cache and Metadata. Cache includes Thread Cache and Dirty Page. You can view the original profile of the memory allocator in real time at http://{be_host}:{be_web_server_port}/memz.

Jemalloc Cache Memory Analysis

If you see a large value of `Label=tc/jemalloc_cache, Type=overview` Memory Trakcer, it means that Jemalloc or TCMalloc Cache uses a lot of memory. Doris uses Jemalloc as the default Allocator, so here we only analyze the situation where Jemalloc Cache uses a lot of memory.

```
MemTrackerLimiter Label=tc/jemalloc_cache, Type=overview, Limit=-1.00 B(-1 B), Used=410.44 MB(430376896 B), Peak=-1.00 B(-1 B)
```

> Before Doris 2.1.6, `Label=tc/jemalloc_cache` also includes Jemalloc Metadata, and it is likely that the large memory usage of Jemalloc Metadata causes `Label=tc/jemalloc_cache` to be too large. Refer to the analysis of `Label=tc/jemalloc_metadata` Memory Tracker.

During the running of the BE process, Jemalloc Cache consists of two parts.

- Thread Cache, cache a specified number of Pages in Thread Cache, refer to Jemalloc opt.tcache.
- Dirty Page, all memory Pages that can be reused in Arena.

Jemalloc Cache View Method

View Doris BE's Web page `http://{be_host}:{be_web_server_port}/memz` (webserver_port defaults to 8040) to obtain Jemalloc Profile, and interpret the use of Jemalloc Cache based on several sets of key information.

- `tcache_bytes` in Jemalloc Profile is the total number of bytes of Jemalloc Thread Cache. If the `tcache_bytes` value is large, it means that the memory used by Jemalloc Thread Cache is too large.
- The sum of the values of the `dirty` column in the `extents` table in the Jemalloc Profile is large, indicating that the memory used by the Jemalloc Dirty Page is too large.

Thread Cache Memory is Too Large

It may be that the Thread Cache caches a large number of large pages, because the upper limit of the Thread Cache is the number of pages, not the total number of bytes of the pages.

Consider reducing `lg_tcache_max` in `JEMALLOC_CONF` in `be.conf`. `lg_tcache_max` is the upper limit of the byte size of the page allowed to be cached. The default value is 15, that is, 32 KB (2^15). Pages exceeding this size will not be cached in the Thread Cache. `lg_tcache_max` corresponds to `Maximum thread-cached size class` in the Jemalloc Profile.

> Before Doris 2.1, the default value of `lg_tcache_max` in `JEMALLOC_CONF` in `be.conf` is 20, which will cause the Jemalloc Cache to be too large in some scenarios. After Doris 2.1, it has been changed back to the default value of Jemalloc 15.

This is usually because the query or load in the BE process is applying for a large number of memory pages of large size classes, or after executing a large memory query or load, a large number of memory pages of large size classes are cached in the Thread Cache. There are two times to clean up the Thread Cache. One is to recycle the memory blocks that have not been used for a long time when the memory application and release reach a certain number of times; the other is to recycle all pages when the thread exits. At this time, there is a Bad Case. If the thread has not executed new queries or loads in the future, it will no longer allocate memory and fall into a so-called `idle` state. Users expect that the memory can be released after the query is completed, but in fact, in this scenario, if the thread does not exit, the Thread Cache will not be cleaned.

However, there is usually no need to pay attention to the Thread Cache. When the available memory of the process is insufficient, if the size of the Thread Cache exceeds 1G, Doris will manually flush the Thread Cache.

Dirty Page Memory Too Large

```
extents:       size ind      ndirty       dirty      nmuzzy       muzzy    nretained    retained      ntotal        total
               4096   0           7       28672           1        4096           21       86016          29       118784
               8192   1          11       90112           2       16384           11       90112          24       196608
              12288   2           2       24576           4       49152           45      552960          51       626688
              16384   3           0           0           1       16384            6       98304           7       114688
              20480   4           0           0           1       20480            5      102400           6       122880
              24576   5           0           0          43     1056768            2       49152          45      1105920
              28672   6           0           0           0           0           13      372736          13       372736
              32768   7           0           0           1       32768           13      425984          14       458752
              40960   8           0           0          31     1150976           35     1302528          66      2453504
              49152   9           4      196608           2       98304            3      139264           9       434176
              57344  10           0           0           1       57344            9      512000          10       569344
              65536  11           3      184320           0           0            6      385024           9       569344
              81920  12           2      147456           3      241664           38     2809856          43      3198976
              98304  13           0           0           1       86016            6      557056           7       643072
             114688  14           1      102400           1      106496           15     1642496          17      1851392
```

Reduce `dirty_decay_ms` of `JEMALLOC_CONF` in `be.conf` to 2000 ms or less. The default `dirty_decay_ms` in `be.conf` is 5000 ms. Jemalloc will release dirty pages according to a smooth gradient curve within the time specified by `dirty_decay_ms`. For reference, Jemalloc opt.dirty_decay_ms. When the BE process has insufficient available memory and triggers Minor GC or Full GC, it will actively release all dirty pages according to a certain strategy.

> Before Doris 2.1, the default value of `dirty_decay_ms` in `JEMALLOC_CONF` in `be.conf` is 15000, which will cause the Jemalloc Cache to be too large in some scenarios. After Doris 2.1, the default value is 5000.

`extents` in Jemalloc Profile contains the statistical values of buckets of different page sizes in all Jemalloc `arena`, where `ndirty` is the number of dirty pages and `dirty` is the total memory of dirty pages. Refer to `stats.arenas.<i>.extents.<j>.{extent_type}_bytes` in Jemalloc and add up the `dirty` of all Page Sizes to get the memory byte size of the Dirty Page in Jemalloc.

Jemalloc Metadata Memory Analysis

If the value of `Label=tc/jemalloc_metadata, Type=overview` Memory Trakcer is large, it means that Jemalloc or TCMalloc Metadata uses a lot of memory. Doris uses Jemalloc as the default Allocator, so here we only analyze the situation where Jemalloc Metadata uses a lot of memory.

```
MemTrackerLimiter Label=tc/jemalloc_metadata, Type=overview, Limit=-1.00 B(-1 B), Used=144 MB(151759440 B), Peak=-1.00 B(-1 B)
```

> `Label=tc/jemalloc_metadata` Memory Tracker was added after Doris 2.1.6. In the past, Jemalloc Metadata was included in `Label=tc/jemalloc_cache` Memory Tracker.

How to view Jemalloc Metadata

You can get the Jemalloc Profile by viewing the Doris BE web page `http://{be_host}:{be_web_server_port}/memz` (webserver_port defaults to 8040). Find the overall memory statistics of Jemalloc in the Jemalloc Profile as follows, where `metadata` is the memory size of Jemalloc Metadata.

`Allocated: 2401232080, active: 2526302208, metadata: 535979296 (n_thp 221), resident: 2995621888, mapped: 3221979136, retained: 131542581248`

- `Allocated` The total number of bytes of memory allocated by Jemalloc for the BE process.
- `active` The total number of bytes of all pages allocated by Jemalloc for the BE process, which is a multiple of Page Size and is usually greater than or equal to `Allocated`.
- `metadata` The total number of bytes of Jemalloc metadata, which is related to the number of allocated and cached pages, memory fragmentation and other factors. Refer to the document Jemalloc stats.metadata
- `retained` The size of the virtual memory mapping retained by Jemalloc, which is not returned to the operating system through munmap or similar methods, and is not strongly associated with physical memory. Reference document Jemalloc stats.retained

Jemalloc Metadata memory is too large

The size of Jemalloc Metadata is positively correlated with the size of process virtual memory. Usually, the virtual memory of Doris BE process is large because Jemalloc retains a large number of virtual memory mappings, that is, the above `retained`. The virtual memory returned to Jemalloc is cached in Retained by default, waiting to be reused, and will not be released automatically or manually.

The fundamental reason for the large size of Jemalloc Retained is that the memory reuse at the Doris code level is insufficient, resulting in the need to apply for a large amount of virtual memory, which enters Jemalloc Retained after being released. Usually, the ratio of virtual memory to Jemalloc Metadata size is between 300-500, that is, if there is 10T of virtual memory, Jemalloc Metadata may occupy 20G.

If you encounter problems with Jemalloc Metadata and Retained continuing to increase, and the process virtual memory is too large, it is recommended to consider restarting the Doris BE process regularly. Usually this only occurs after Doris BE has been running for a long time, and only a few Doris clusters will encounter it. There is

currently no way to reduce the virtual memory mapping retained by Jemalloc Retained without losing performance. Doris is continuously optimizing memory usage.

If the above problems occur frequently, refer to the following methods.

1. A fundamental solution is to turn off the Jemalloc Retained cache virtual memory mapping, add `retain:false` after `JEMALLOC_CONF` in `be.conf`, and restart BE. However, query performance may be significantly reduced, and the performance of the TPC-H Benchmark test will be reduced by about 3 times.

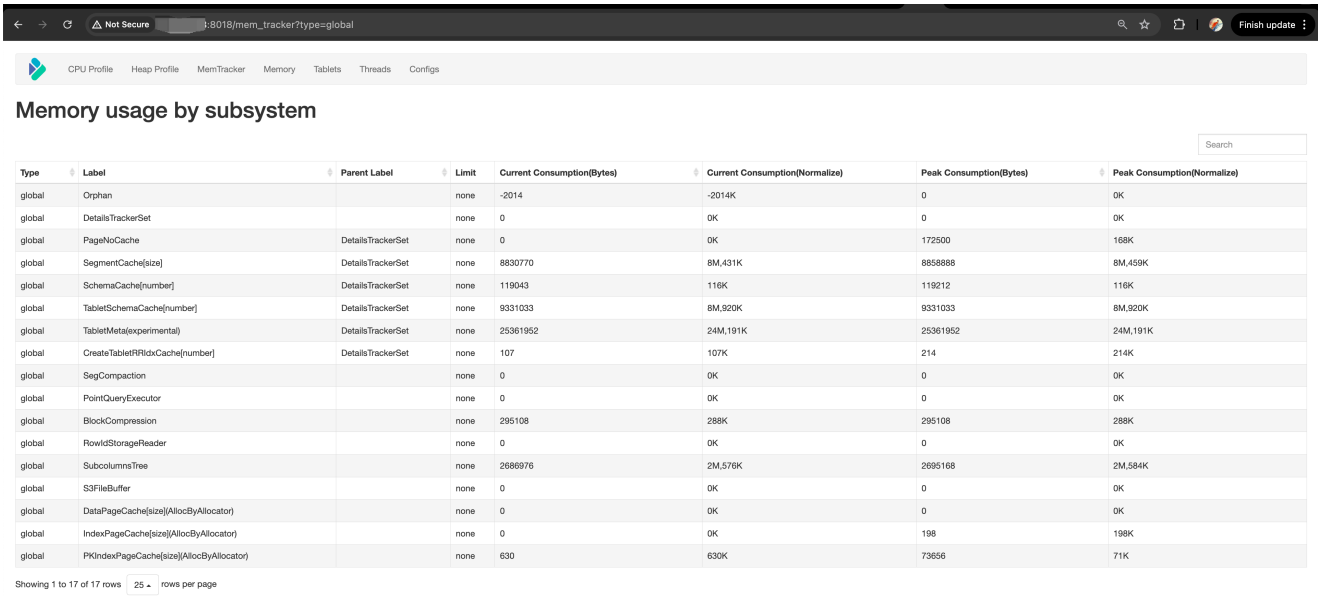
2. On Doris 2.1, you can turn off Pipelinex and Pipeline by executing `set global experimental_enable_pipeline_engine=false; set global experimental_enable_pipeline_x_engine=false;`, because pipelinex and pipeline will apply for more virtual memory. This will also lead to reduced query performance.

Global Memory Analysis

GLobal Memory is Doris's globally shared memory, mainly including Cache and Metadata.

Global Memory View Method

The web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=global` displays all Memory Trackers of `type=global`.



image

```
- Orphan: Collects memory that does not know where it belongs, and ideally it is expected to be equal to 0.
- DataPageCache\[size\](AllocByAllocator): The size of the data Page cache.
- IndexPageCache\[size\](AllocByAllocator): The size of the index cache of the data Page.
- PKIndexPageCache\[size\](AllocByAllocator): Primary key index of data Page.
- DetailsTrackerSet: Contains some memory that is not currently tracked accurately. These memories will not be counted in Global memory, including some Cache and metadata memory, etc. By default, only Memory Trackers with Peak Consumption not equal to 0 are displayed, mainly including the following:
- SegmentCache[size]: Caches the memory size of the opened Segment, such as index information.
- SchemaCache[number]: Caches the number of entries of Rowset Schema.
- TabletSchemaCache[number]: Caches the number of entries of Tablet Schema.
- TabletMeta(experimental): Memory size of all Tablet Schema.
- CreateTabletRRIdxCache[number]: Caches the number of entries of create tabelt index.
- PageNoCache: If page cache is turned off, this Memory Trakcer will track the sum of all page memory used by all Queries.
- IOBufBlockMemory: The total IOBuf memory used by BRPC.
- PointQueryLookupConnectionCache[number]: The number of cached Point Query Lookup Connection entries.
- AllMemTableMemory: The total Memtable memory of all loads cached in memory waiting to be flushed.
- MowTabletVersionCache[number]: The number of cached Mow Tablet Version entries.
- MowDeleteBitmapAggCache[size]: The cached Mow DeleteBitmap memory size.
- SegCompaction: The total memory allocated from `Doris Allocator` by all SegCompaction tasks.
- PointQueryExecutor: Some memory shared by all Point Queries.
- BlockCompression: Some memory used in the decompression process shared by all Queries.
- RowIdStorageReader: All Multiget Data requests use memory in RowIdStorageReader.
- SubcolumnsTree: Some memory used by Point Query in SubcolumnsTree.
- S3FileBuffer: Memory allocated by the File Buffer when reading S3.
```

Some of the Memory Tracker tags have suffixes, which mean:

- `[size]` means that the Cache Tracker records the memory size.
- `[number]` means that the Cache Tracker records the number of cached entries, which is usually because the memory cannot be accurately counted at present.
- `(AllocByAllocator)` means that the Tracker value is tracked by Doris Allocator.
- `(experimental)` means that this Memory Tracker is still experimental and the value may not be accurate.

Global Memory occupies a lot

```
MemTrackerLimiter Label=global, Type=overview, Limit=-1.00 B(-1 B), Used=199.37 MB(209053204 B), Peak=199.37 MB(209053204 B)
```

The value of Global Memory Tracker `Label=global, Type=overview` is equal to the sum of all Memory Trackers with `Type=global` and `Parent Label != DetailsTrackerSet`, mainly including Cache and metadata, which are shared between different tasks.

Cache analysis method

Refer to Doris Cache Memory Analysis

Metadata analysis method

Refer to Metadata Memory Analysis

Orphan analysis method

If the Orphan Memory Tracker value is too large, it means that the Memory Tracker statistics are missing. Refer to the analysis in [Memory Tracker Statistics Missing] in Memory Tracker.

Cache Memory Analysis

The caches managed by Doris are currently all LRU elimination strategies, and all support controlling the capacity and elimination time through parameters separately.

Doris Cache Type

1. Page Cache

Used to speed up data scanning.

```
- DataPageCache: Cache data Page.
- IndexPageCache: Cache data Page index.
- PKIndexPageCache: Cache Page primary key index.
```

2. Metadata Cache

Used to speed up metadata reading.

```
- SegmentCache: Cache open Segments, such as index information.
- SchemaCache: Cache Rowset Schema.
- TabletSchemaCache: Cache Tablet Schema.
- CreateTabletRRIdxCache: Cache Create Tabelt index.
- MowTabletVersionCache: Cache Mow Tablet Version.
- MowDeleteBitmapAggCache: Cache Mow DeleteBitmap.
```

3. Cloud Cache

A dedicated cache on the cloud.

```
- CloudTabletCache: Cache Tablet on the Cloud.
- CloudTxnDeleteBitmapCache: Cache DeleteBitmap on Cloud.
```

4. Inverted Index Cache

Speed up inverted index.

```
- InvertedIndexSearcherCache
- InvertedIndexQueryCache
```

5. Point Query Cache

Speed up point query execution, mainly used for log analysis.

```
- PointQueryRowCache
- PointQueryLookupConnectionCache
```

6. Other Cache

```
- FileCache: File cache used by external table queries and Cloud.
- CommonObjLRUCache
- LastSuccessChannelCache
```

Doris Cache View Method

There are three ways to view Doris Cache related indicators.

1. Doris BE Metrics

Web page `http://{be_host}:{be_web_server_port}/metrics` can see BE process memory monitoring (Metrics), including each cache capacity, usage, number of elements, search and hit times and other indicators.

```
- `doris_be_cache_capacity{name="TabletSchemaCache"} 102400`: Cache capacity, two limiting methods: memory size or number of
elements.
- `doris_be_cache_usage{name="TabletSchemaCache"} 40838`: Cache usage, memory size or number of elements, corresponding to the limit
of cache capacity.
- `doris_be_cache_usage_ratio{name="TabletSchemaCache"} 0.398809`: Cache usage, equal to `(cache_usage / cache_capacity)`.
- `doris_be_cache_element_count{name="TabletSchemaCache"} 1628`: Number of cache elements, equal to Cache Usage when the cache
capacity limits the number of elements.
- `doris_be_cache_lookup_count{name="TabletSchemaCache"} 63393`: Number of cache lookups.
- `doris_be_cache_hit_count{name="TabletSchemaCache"} 61765`: Number of hits when looking up the cache.
- `doris_be_cache_hit_ratio{name="TabletSchemaCache"} 0.974319`: Hit ratio, equal to `(hit_count / lookup_count)`
```

2. Doris BE Bvar

Web page `http://{be_host}:{brpc_port}/vars/*cache*` can show some unique metrics of some caches.

> In the future, the indicators in Doris BE Metrics will be moved to Doris BE Bvar.

3. Memory Trakcer

To view the memory size occupied by each cache in real time, refer to Global Memory Analysis. When there is a memory error, you can find the `Memory Tracker Summary` in the `be/log/be.INFO` log, which contains the cache memory size at that time.

Doris Cache Memory Analysis

There are various caches when Doris BE is running. Usually, there is no need to pay attention to the cache memory, because when the BE process has insufficient available memory, the memory GC will be triggered to clean up the cache first.

However, if the cache is too large, it will increase the pressure of memory GC, increase the risk of insufficient available memory for query or load error processes, and increase the risk of BE process OOM Crash. So if the memory is constantly tight, you can consider lowering the upper limit of the cache, closing the cache, or reducing the survival time of the cache entry. A smaller cache may reduce query performance in some scenarios, but it is usually tolerable in a production environment. After adjustment, you can observe the query and load performance for a period of time.

> Before Doris 2.1, Memory GC was not perfect. When the memory was insufficient, the cache might not be released in time. If the memory was constantly tight, you often had to consider manually lowering the cache limit.

After Doris 2.1.6, if you want to manually clean up all caches during BE operation, execute `curl http://{be_host}:{be_web_server_port}/api/clear_cache/all`, and the released memory size will be returned.

The following analyzes the situation where different caches use more memory.

DataPageCache uses more memory

- After Doris 2.1.6, execute `curl http://{be_host}:{be_web_server_port}/api/clear_cache/DataPageCache` to manually clean up during BE operation.
- Execute `curl -X POST http://{be_host}:{be_web_server_port}/api/update_config?disable_storage_page_cache=true` to disable DataPageCache for the running BE, and clear it after a maximum of 10 minutes by default. However, this is a temporary method. DataPageCache will take effect again after BE restarts.
- If you are sure that you want to reduce the memory usage of DataPageCache for a long time, refer to BE Configuration Items, reduce `storage_page_cache_limit` in `conf/be.conf` to reduce the capacity of DataPageCache, or reduce `data_page_cache_stale_sweep_time_sec` to reduce the effective time of DataPageCache cache, or increase `disable_storage_page_cache=true` to disable DataPageCache, and then restart the BE process.

SegmentCache uses a lot of memory

- After Doris 2.1.6, execute `curl http://{be_host}:{be_web_server_port}/api/clear_cache/SegmentCache` to manually clean up during BE operation.
- Execute `curl -X POST http:/{be_host}:{be_web_server_port}/api/update_config?disable_segment_cache=true` to disable SegmentCache for the running BE, and clear it after a maximum of 10 minutes by default, but this is a temporary method, and SegmentCache will take effect again after BE restarts.
- If you are sure that you want to reduce the memory usage of SegmentCache for a long time, refer to BE Configuration Items, adjust `segment_cache_capacity` or `segment_cache_memory_percentage` in `conf/be.conf` to reduce the capacity of SegmentCache, or reduce `tablet_rowset_stale_sweep_time_sec` to reduce the effective time of SegmentCache cache, or add `disable_segment_cache=true` in `conf/be.conf` to disable SegmentCache and restart the BE process.

PKIndexPageCache uses a lot of memory

- After Doris 2.1.6, execute `curl http://{be_host}:{be_web_server_port}/api/clear_cache/PKIndexPageCache` to manually clean up during BE operation.
- Refer to BE Configuration Items, reduce the capacity of PKIndexPageCache by lowering `pk_storage_page_cache_limit` in `conf/be.conf`, or reduce the effective time of PKIndexPageCache by

lowering `pk_index_page_cache_stale_sweep_time_sec`, or add `disable_pk_storage_page_cache=true` in `conf/be.conf` to disable PKIndexPageCache, and then restart the BE process.

Metadata Memory Analysis

The metadata of Doris BE in memory includes data structures such as `Tablet`, `Rowset`, `Segment`, `TabletSchema`, `ColumnReader`, `PrimaryKeyIndex`, and `BloomFilterIndex`. For more information about Doris BE metadata, refer to the document Analysis of Doris Storage Structure Design.

Metadata Viewing Method

Currently, Memory Tracker does not accurately count the metadata memory size of Doris BE. You can estimate the metadata memory size by viewing some counters in Doris BE Bvar and Doris BE Metrics, or use Heap Profile for further analysis.

Doris BE Bvar

You can see some metadata-related indicators counted by Bvar on the web page `http://{be_host}:{brpc_port}/vars`.

```
- `doris_total_tablet_num`: The number of all tablets.
- `doris_total_rowset_num`: The number of all rowsets.
- `doris_total_segment_num`: The number of all open segments.
- `doris_total_tablet_schema_num`: The number of all tablet schemas.
- `tablet_schema_cache_count`: The number of cached tablet schemas.
- `tablet_meta_schema_columns_count`: The number of columns in all tablet schemas.
- `tablet_schema_cache_columns_count`: The number of cached columns in a tablet schema.
- `doris_column_reader_num`: The number of open column readers.
- `doris_column_reader_memory_bytes`: the number of bytes occupied by the opened Column Reader.
- `doris_ordinal_index_memory_bytes`: the number of bytes occupied by the opened Ordinal Index.
- `doris_zone_map_memory_bytes`: the number of bytes occupied by the opened ZoneMap Index.
- `doris_short_key_index_memory_bytes`: the number of bytes occupied by the opened Short Key Index.
- `doris_pk/index_reader_bytes`: the cumulative number of bytes occupied by the Primary Key Index Reader. This is not a real-time
statistical value and is expected to be fixed.
- `doris_pk/index_reader_pages`: Same as above, the cumulative value of the statistics.
- `doris_pk/index_reader_cached_pages`: Same as above, the cumulative value of the statistics.
- `doris_pk/index_reader_pk_pages`: Same as above, cumulative value of statistics.
- `doris_primary_key_index_memory_bytes`: Same as above, cumulative value of statistics.
```

Doris BE Metrics

Web page `http://{be_host}:{be_web_server_port}/metrics` can see some metadata indicators in BE process memory monitoring (Metrics). Among them, Metadata Cache related indicators refer to Doris Cache Memory Analysis.

```
- `doris_be_all_rowsets_num`: The number of all Rowsets.
- `doris_be_all_segments_num`: The number of all Segments.
```

Use Heap Profile to analyze metadata memory

If the memory location cannot be accurately located using Doris BE Bvar and Metrics above, if the cluster can be easily restarted and the phenomenon can be reproduced, refer to Heap Profile Memory Analysis to analyze

Metadata memory.

After the phenomenon is reproduced, if you see the `Tablet`, `Segment`, `TabletSchema`, and `ColumnReader` fields in the call stack with a large memory usage in Heap Profile, it can be basically confirmed that metadata occupies a large amount of memory.

Query Memory Analysis

Usually, we first use Query Profile to analyze query memory usage. If the sum of the memory of each operator counted in Query Profile is much smaller than the memory counted by Query Memory Tracker, it means that the operator memory counted by Query Profile is much different from the actual memory used. Then, we often need to use Heap Profile for further analysis. If Query is Canceled due to memory limit overrun and cannot be completed, Query Profile is incomplete and may not be analyzed accurately. Usually, we directly use Heap Profile to analyze query memory usage.

Query Memory View

If you see a large value of `Label=query, Type=overview` Memory Tracker anywhere, it means that the query memory usage is high.

```
MemTrackerLimiter Label=query, Type=overview, Limit=-1.00 B(-1 B), Used=83.32 MB(87369024 B), Peak=88.33 MB(92616000 B)
```

If you already know the query to be analyzed, skip this section and continue with the analysis below. Otherwise, refer to the following method to locate the large memory query.

First locate the QueryID of the large memory query. In the BE web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=query`, you can see the real-time large memory query by sorting by `Current Consumption`. You can find the QueryID in `label`.

When the error process memory exceeds the limit or the available memory is insufficient, the lower part of the `Memory Tracker Summary` in the `be.INFO` log contains the Memory Tracker of the top 10 tasks (query/load/compaction, etc.) with the highest memory usage. The format is `MemTrackerLimiter Label=Query#Id=xxx, Type=query`. Usually, the QueryID of the large memory query can be located in the top 10 tasks.

The memory statistics of historical queries can be viewed in `peakMemoryBytes` of each query in `fe/log/fe.audit.log`, or search `Deregister query/load memory tracker, queryId` in `be/log/be.INFO` to view the peak memory of each query on a single BE.

Use Query Profile to analyze query memory usage

Find query information including SQL in `fe/log/fe.audit.log` based on QueryID, get the query plan by `explain SQL`, and get the query profile by executing SQL after `set enable_profile=true`. For a detailed introduction to Query Profile, refer to the document Query Profile. Here we only introduce the memory-related content in Query Profile, and locate the Operator and data structure that use a lot of memory based on it.

1. Locate Operators or memory data structures that use a lot of memory

Query Profile is divided into two parts:

- `MergedProfile`

MergedProfile is the aggregated result of all Instance Profiles of Query, which shows the sum, avg, max, and min of memory usage of each Operator in each Pipeline of each Fragment on all Instances, including the Operator's peak memory `PeakMemoryUsage` and the peak memory of major memory data structures such as `HashTable` and `Arena`. Based on this, Operators and data structures that use a lot of memory can be located.

```
MergedProfile
          Fragments:
              Fragment  0:
                  Pipeline  :  0(instance_num=1):
                      RESULT_SINK_OPERATOR  (id=0):
                            -  MemoryUsage:  sum  ,  avg  ,  max  ,  min
                          EXCHANGE_OPERATOR  (id=20):
                                -  MemoryUsage:  sum  ,  avg  ,  max  ,  min
                                    -  PeakMemoryUsage:  sum  1.16  KB,  avg  1.16  KB,  max  1.16  KB,  min  1.16  KB
              Fragment  1:
                  Pipeline  :  1(instance_num=12):
                      AGGREGATION_SINK_OPERATOR  (id=18  ,  nereids_id=1532):
                            -  MemoryUsage:  sum  ,  avg  ,  max  ,  min
                                -  HashTable:  sum  96.00  B,  avg  8.00  B,  max  24.00  B,  min  0.00
                                -  PeakMemoryUsage:  sum  1.58  MB,  avg  134.67  KB,  max  404.02  KB,  min  0.00
                                -  SerializeKeyArena:  sum  1.58  MB,  avg  134.67  KB,  max  404.00  KB,  min  0.00
                          EXCHANGE_OPERATOR  (id=17):
                                -  MemoryUsage:  sum  ,  avg  ,  max  ,  min
                                    -  PeakMemoryUsage:  sum  2.25  KB,  avg  192.00  B,  max  768.00  B,  min  0.00
```

- `Execution Profile`

`Execution Profile` is the result of each specific Instance Profile of Query. Usually, after locating the Operator and data structure that use a lot of memory based on `MergedProfile`, you can analyze the reasons for their memory usage based on the query plan after `explain SQL`. If you need to analyze the memory value of Query in a certain BE node or a certain Instance in some scenarios, you can further locate it based on `Execution Profile`.

```
Execution  Profile  36ca4f8b97834449-acae910fbee8c670:(ExecTime:  48sec201ms)
    Fragments:
        Fragment  0:
            Fragment  Level  Profile:    (host=TNetworkAddress(hostname:10.16.10.8,  port:9013)):(ExecTime:  48sec111ms)
            Pipeline  :1    (host=TNetworkAddress(hostname:10.16.10.8,  port:9013)):
                PipelineTask  (index=80):(ExecTime:  6sec267ms)
                DATA_STREAM_SINK_OPERATOR  (id=17,dst_id=17):(ExecTime:  1.634ms)
                -  MemoryUsage:
                    -  PeakMemoryUsage:  1.50  KB
                STREAMING_AGGREGATION_OPERATOR  (id=16  ,  nereids_id=1526):(ExecTime:  41.269ms)
                    -  MemoryUsage:
                        -  HashTable:  168.00  B
                        -  PeakMemoryUsage:  404.16  KB
                        -  SerializeKeyArena:  404.00  KB
                HASH_JOIN_OPERATOR  (id=15  ,  nereids_id=1520):(ExecTime:  6sec150ms)
                        -  MemoryUsage:
                            -  PeakMemoryUsage:  3.22  KB
                            -  ProbeKeyArena:  3.22  KB
                    LOCAL_EXCHANGE_OPERATOR  (PASSTHROUGH)  (id=-12):(ExecTime:  67.950ms)
                            -  MemoryUsage:
                                -  PeakMemoryUsage:  1.41  MB
```

2. `HASH_JOIN_SINK_OPERATOR` takes up too much memory

```
HASH_JOIN_SINK_OPERATOR  (id=12  ,  nereids_id=1304):(ExecTime:  1min14sec)
    -  JoinType:  INNER_JOIN
```

```
    - BroadcastJoin: true
    - BuildRows: 600.030257M (600030257)
    - InputRows: 600.030256M (600030256)
    - MemoryUsage:
        - BuildBlocks: 15.65 GB
        - BuildKeyArena: 0.00
        - HashTable: 6.24 GB
        - PeakMemoryUsage: 21.89 GB
```

It can be seen that the `BuildBlocks` and `HashTable` in the Hash Join Build phase mainly use memory. Usually, the Hash Join Build phase uses too much memory. First, confirm whether the Join Reorder order is reasonable. Usually, the correct order is to use small tables for Hash Join Build and large tables for Hash Join Probe. This can minimize the overall memory usage of Hash Join and usually has better performance.

In order to confirm whether the Join Reorder order is reasonable, we find the profile of `HASH_JOIN_OPERATOR` with id=12. We can see that `ProbeRows` has only 196240 rows. Therefore, the correct order of this Hash Join Reorder should be to swap the positions of the left and right tables. You can `set disable_join_reorder=true` to turn off Join Reorder and manually specify the order of the left and right tables and then perform Query verification. For further information, please refer to the relevant documents on Join Reorder in the query optimizer.

```
HASH_JOIN_OPERATOR (id=12 , nereids_id=1304):(ExecTime: 8sec223ms)
    - BlocksProduced: 227
    - MemoryUsage:
        - PeakMemoryUsage: 0.00
        - ProbeKeyArena: 0.00
    - ProbeRows: 196.24K (196240)
    - RowsProduced: 786.22K (786220)
```

Use Heap Profile to analyze query memory usage

If the query profile above cannot accurately locate the memory usage location, if the cluster can be easily restarted and the phenomenon can be reproduced, refer to Heap Profile Memory Analysis to analyze the query memory.

Dump the Heap Profile once before the query is executed, and dump the Heap Profile again during the query execution. By using `jeprof --dot lib/doris_be --base=heap_dump_file_1 heap_dump_file_2` to compare the memory changes between the two Heap Profiles, you can get the memory usage ratio of each function in the code during the query execution. Compare the code to locate the memory usage location. Because the memory changes in real time during the query execution, you may need to dump the Heap Profile multiple times during the query execution and compare and analyze.

Load Memory Analysis

Doris data load is divided into two stages: fragment reading and channel writing. The execution logic of fragment and query fragment is the same, but Stream Load usually has only Scan Operator. Channel mainly writes data to the temporary data structure Memtable, and then Delta Writer compresses the data and writes it to the file.

Load memory view

If you see a large value of `Label=load, Type=overview` Memory Tracker anywhere, it means that the load memory is used a lot.

```
MemTrackerLimiter Label=load, Type=overview, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
```

The memory load by Doris is divided into two parts. The first part is the memory used by fragment execution, and the second part is the memory used in the construction and flushing process of MemTable.

The Memory Tracker with `Label=AllMemTableMemory, Parent Label=DetailsTrackerSet` found in the BE web page `http://{be_host}:{be_web_server_port}/mem_tracker?type=global` is the memory used by all load tasks to construct and flush `MemTable` on this BE node. When the error process memory exceeds the limit or the available memory is insufficient, this Memory Tracker can also be found in the `Memory Tracker Summary` in the `be.INFO` log.

```
MemTracker Label=AllMemTableMemory, Parent Label=DetailsTrackerSet, Used=25.08 MB(26303456 B), Peak=25.08 MB(26303456 B)
```

Load Memory Analysis

If the value of `Label=AllMemTableMemory` is small, the main memory used by the load task is the execution fragment. The analysis method is the same as Query Memory Analysis, so it will not be repeated here.

If the value of `Label=AllMemTableMemory` is large, MemTable may not be flushed in time. You can consider reducing the values of `load_process_max_memory_limit_percent` and `load_process_soft_mem_limit_percent` in `be.conf`. This can make MemTable flush more frequently, so that fewer MemTables are cached in memory, but the number of files written will increase. If too many small files are written, the pressure of compaction will increase. If compaction is not timely, the metadata memory will increase, the query will slow down, and even the load will report an error after the number of files exceeds the limit.

During the load execution process, check the BE web page `/mem_tracker?type=load`. According to the values of the two groups of memory trackers `Label=MemTableManualInsert` and `Label=MemTableHookFlush`, you can locate `LoadID` and `TabletID` with large `MemTable` memory usage.

Query error Process Memory Not Enough

When `MEM_LIMIT_EXCEEDED` appears in the error message of the query and contains `Process memory not enough`, it means that the process is canceled due to insufficient available memory.

First, parse the error message to confirm the reason for Cancel, the memory size used by the query itself when Canceled, and the memory status of the process. There are usually three reasons for the Cancel of the query:

1. The memory of the Canceled Query itself is too large.
2. The memory of the Canceled Query itself is small, and there are other queries with larger memory.
3. The memory of the globally shared Cahce, metadata, etc. is too large, or the memory of other tasks other than query and load tasks is too large

Error message analysis

There are two situations when the available memory of the process is insufficient. One is that the current memory of the process exceeds the configured memory limit, and the other is that the remaining available memory of the system is lower than the watermark. There are three paths to cancel tasks such as queries:

- If the error message contains `cancel top memory used`, it means that the task is canceled in the memory Full GC.
- If the error message contains `cancel top memory overcommit`, it means that the task was canceled in the memory Minor GC.
- If the error message contains `Allocator sys memory check failed`, it means that the task was canceled after failing to apply for memory from `Doris Allocator`.

After analyzing the error message below,

- If the memory used by the query and load itself accounts for a large proportion of the process memory, refer to [Query own memory is too large] to analyze the memory usage of the query and load, and try to adjust parameters or optimize SQL to reduce the memory required for execution.
- If the task itself uses very little memory, refer to [Process memory other than query and load is too large] to try to reduce the memory usage of other locations in the process, so as to reserve more memory for query and other task execution.

For more information about memory limits, watermark calculation methods, and memory GC, see Memory Control Strategy

1 Canceled in memory Full GC

If the BE process memory exceeds the process memory upper limit (MemLimit) or the system's remaining available memory is lower than the memory low watermark (LowWaterMark), Full GC is triggered. At this time, the task with the largest memory will be canceled first.

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.8)[MEM_LIMIT_EXCEEDED]Process memory not enough, cancel top memory used query: <Query#Id=e862471398b14e71-9361a1ab8153cb29> consumption 866.97 MB, backend 10.16.10.8, process memory used 3.12 GB exceed limit 3.01 GB or sys available memory 191.25 GB less than low water mark 3.20 GB. Execute again after enough memory, details see be.INFO.
```

Error message analysis:

1. `(10.16.10.8)`: BE node with insufficient memory during query.
2. `query: <Query#Id=e862471398b14e71-9361a1ab8153cb29> consumption 866.97 MB`: The currently canceled queryID, the query itself uses 866.97 MB of memory.
3. `process memory used 3.12 GB exceed limit 3.01 GB or sys available memory 191.25 GB less than low water mark 3.20 GB` The reason for the process memory exceeding the limit is that the physical memory used by the BE process, 3.12 GB, exceeds the MemLimit of 3.01 GB. The current operating system has 191.25 GB of memory available for BE to use, which is still higher than the LowWaterMark of 3.20 GB.

2 Canceled in Minor GC of memory

If the Doris BE process memory exceeds the process memory soft limit (SoftMemLimit) or the system's remaining available memory is lower than the memory warning watermark (WarningWaterMark), Minor GC is triggered. At this time, the query with the largest memory limit ratio will be canceled first.

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.8)[MEM_LIMIT_EXCEEDED]Process memory not enough, cancel top memory overcommit query: <Query#Id=e862471398b14e71-9361a1ab8153cb29> consumption 866.97 MB, backend 10.16.10.8, process memory used 2.12 GB exceed soft limit 2.71 GB or sys available memory 3.25 GB less than warning water mark 6.40 GB. Execute again after enough memory, details see be.INFO.
```

Error message analysis: `process memory used 3.12 GB exceed soft limit 6.02 GB or sys available memory 3.25 GB less than warning water mark 6.40 GB` The reason for the process memory exceeding the limit is that the remaining memory available to BE in the current operating system is 3.25 GB, which is lower than the WarningWaterMark of 6.40 GB, and the physical memory used by the BE process is 2.12 GB, which does not exceed the SoftMemLimit of 2.71 GB.

3 Failed to apply for memory from Allocator

Doris BE's large memory requests will be allocated through `Doris Allocator`, and the memory size will be checked during allocation. If the process has insufficient available memory, an exception will be thrown and the current query will be attempted to be canceled.

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.8)[MEM_LIMIT_EXCEEDED]PreCatch error code:11, [E11] Allocator sys memory check failed: Cannot alloc:4294967296, consuming tracker:<Query#Id=457efb1fdae74d3 b-b4fffdcfd4baaf32>, peak used 405956032, current used 386704704, exec node:<>, process memory used 2.23 GB exceed limit 3.01 GB or sys available memory 181.67 GB less than low water mark 3.20 GB.
```

Error message analysis:

1. `consuming tracker:<Query#Id=457efb1fdae74d3b-b4fffdcfd4baaf32>, peak used 405956032, current used 386704704, exec node:VAGGREGATION_NODE (id=7)>`: The queryID currently being canceled, the query

currently uses 386704704 Bytes of memory, the query memory peak is 405956032 Bytes, and the operator being executed is `VAGGREGATION_NODE (id=7)`.

2. `Cannot alloc:4294967296`: The current application for 4 GB of memory failed because the current process memory of 2.23 GB plus 4 GB will exceed the MemLimit of 3.01 GB.

The memory usage of the Canceled Query is too large

Refer to Query Memory Analysis or Load Memory Analysis to analyze the memory usage of queries and loads, and try to adjust parameters or optimize SQL to reduce the memory required for execution.

It should be noted that if the task fails to apply for memory from the Allocator and is Canceled, `Cannot alloc` or `try alloc` will show that the memory currently being applied for by the Query is too large. At this time, you need to pay attention to whether the memory application here is reasonable. Search `Allocator sys memory check failed` in `be/log/be.INFO` to find the stack of memory application.

The Canceled Query itself has a small memory, and there are other queries with larger memory

Usually, it is because the query with larger memory is stuck in the Cancel stage and cannot release the memory in time. Full GC will first cancel queries in order of memory usage, and then cancel loads in order of memory usage. If a query is canceled in memory Full GC, but there are other queries in the BE process that use more memory than the currently canceled query, you need to pay attention to whether these queries with larger memory usage are stuck during the cancel process.

First, execute `grep {queryID} be/log/be.INFO` to find the time when the query is canceled, and then search `Memory Tracker Summary` in the context to find the process memory statistics log. If there is a query that uses more memory in `Memory Tracker Summary`. Run `grep {queryID with larger memory} be/log/be.INFO` to check if there is a log with the keyword `Cancel`. The corresponding time point is the time when the query is canceled. If the query is also canceled, and the time point when the query with larger memory is canceled is different from the time point when the current query is canceled, refer to [Query Cancel process stuck] in Memory Issue FAQ to analyze whether the query with larger memory is stuck in the cache process. For the analysis of `Memory Tracker Summary`, refer to Memory Log Analysis.

Process memory outside query and load tasks is too large

Try to locate the memory location and consider reducing memory usage to reserve more memory for query and load execution.

The time when the task was canceled due to insufficient available memory can be found in the process memory statistics log in `be/log/be.INFO`. Run `grep queryID be/log/be.INFO` to find the time when the query was canceled, and then search `Memory Tracker Summary` in the context to find the process memory statistics log. Then refer to the [Process Memory Statistics Log Analysis] section in Memory Log Analysis for further analysis. Before analysis, refer to the [Memory Tracker Statistics Missing] section in Memory Tracker to analyze whether the Memory Tracker has statistics missing.

If the Memory Tracker has statistics missing, refer to the [Memory Tracker Statistics Missing] section for further analysis. Otherwise, Memory Tracker counts most of the memory and there is no missing statistics. Refer to

Overview to analyze the reasons why different parts of the Doris BE process occupy too much memory and how to reduce its memory usage.

Query error Memory Tracker Limit Exceeded

When `MEM_LIMIT_EXCEEDED` appears in the query or load error message and contains `memory tracker limit exceeded`, it means that the task exceeds the single execution memory limit.

Error message analysis

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.8)[MEM_LIMIT_EXCEEDED]PreCatch error code:11, [E11] Allocator mem tracker check failed, [MEM_LIMIT_EXCEEDED] failed alloc size 1.03 MB, memory tracker limit exceeded, tracker label :Query#Id=f78208b15e064527-a84c5c0b04c04fcf, type:query, limit 100.00 MB, peak used 99.29 MB, current used 99.25 MB. backend 10.16.10.8, process memory used 2.65 GB. exec node:<execute:<ExecNode:VHASH_JOIN _NODE (id=4)>>, can `set exec_mem_limit=8G` to change limit, details see be.INFO.
```

The error message is divided into two parts:

1. `failed alloc size 1.03 MB, memory tracker limit exceeded, tracker label:Query#Id=f78208b15e064527-a84c5c0b04c04fcf, type:query, limit 100.00 MB, peak used 99.29 MB, current used 99.25 MB`: Query `f78208b15e064527-a84c5c0b04c04fcf` is currently being executed. When trying to apply for 1.03 MB of memory, it was found that the query exceeded the memory limit for a single execution. The query memory limit is 100 MB (`exec_mem_limit` in Session Variables). Currently, 99.25 MB has been used, and the peak memory is 99.29. MB.

2. `backend 10.16.10.8, process memory used 2.65 GB. exec node:<execute:<ExecNode:VHASH_JOIN_NODE (id=4)>>, can set exec_mem_limit=8G to change limit, details see be.INFO.`: The location of this memory application is `VHASH_JOIN_NODE (id=4)`, and it is prompted that the memory limit of a single query can be increased by `set exec_mem_limit`.

Single execution memory limit and memory over-issuance

`show variables;` You can view Doris Session Veriable, where `exec_mem_limit` is the execution memory limit for a single query and load, but since Doris 1.2, query memory over-issuance (overcommit) is supported, aiming to allow queries to set more flexible memory limits. When there is sufficient memory, even if the query memory exceeds the upper limit, it will not be Cancel, so users usually do not need to pay attention to query memory usage. Until the memory is insufficient, the query will wait for a while while trying to allocate new memory. At this time, based on certain rules, the query with a larger ratio of `mem_used` to `exec_mem_limit` will be canceled first. If the amount of memory released during the waiting process meets the requirements, the query will continue to execute, otherwise an exception will be thrown and the query will be terminated.

If you want to turn off query memory over-issuance, refer to BE Configuration Items and add `enable_query_memory_overcommit=false` in `conf/be.conf`. At this time, a single query and load memory exceeding `exec_mem_limit` will be canceled. If you want to avoid the negative impact of large queries on cluster stability, or want to accurately control the execution of tasks on the cluster to ensure sufficient stability, then you can consider turning off query memory over-issuance.

Query Memory Analysis

If you need to analyze the memory usage of queries, refer to Query Memory Analysis.

After `set enable_profile=true` is used to enable Query Profile, when a task exceeds the memory limit for a single execution, the call stack of the query requesting memory will be printed in `be/log/be.INFO`, and the current memory used and peak value of each operator in the query can be seen. Refer to Memory Log Analysis to analyze `Process Memory Summary` and `Memory Tracker Summary` to help confirm whether the current query memory usage meets expectations.

```
Allocator mem tracker check failed, [MEM_LIMIT_EXCEEDED]failed alloc size 32.00 MB, memory tracker limit exceeded, tracker
label:Query#I
d=41363cb6ba734ad5-bc8720bdf9b3090d, type:query, limit 100.00 MB, peak used 75.32 MB, current used 72.62 MB. backend 10.16.10.8,
process memory used 2.33 GB. exec node:<>, can `set exec_mem_limit=8G`
 to change limit, details see be.INFO.
Process Memory Summary:
    os physical memory 375.81 GB. process memory used 2.33 GB(= 2.60 GB[vm/rss] - 280.53 MB[tc/jemalloc_cache] + 0[reserved] +
0B[waiting_refresh]), limit 338.23 GB, soft limit 304.41 GB. sys availab
le memory 337.33 GB(= 337.33 GB[proc/available] - 0[reserved] - 0B[waiting_refresh]), low water mark 6.40 GB, warning water mark
12.80 GB.
Memory Tracker Summary:    MemTrackerLimiter Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d, Type=query, Limit=100.00 MB(104857600
B), Used=72.62 MB(76146688 B), Peak=75.32 MB(78981248 B)
    MemTracker Label=HASH_JOIN_SINK_OPERATOR, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d, Used=122.00 B(122 B),
Peak=122.00 B(122 B)
    MemTracker Label=VDataStreamRecvr:41363cb6ba734ad5-bc8720bdf9b309fe, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d,
Used=0(0 B), Peak=384.00 B(384 B)
    MemTracker Label=local data queue mem tracker, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d, Used=0(0 B), Peak=384.00
B(384 B)
    MemTracker Label=HASH_JOIN_SINK_OPERATOR, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d, Used=21.73 MB(22790276 B),
Peak=21.73 MB(22790276 B)
    MemTracker Label=VDataStreamRecvr:41363cb6ba734ad5-bc8720bdf9b309fe, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d,
Used=0(0 B), Peak=2.23 MB(2342912 B)
    MemTracker Label=local data queue mem tracker, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d, Used=0(0 B), Peak=2.23
MB(2342912 B)
    MemTracker Label=HASH_JOIN_SINK_OPERATOR, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d, Used=24.03 MB(25201284 B),
Peak=24.03 MB(25201284 B)
    MemTracker Label=VDataStreamRecvr:41363cb6ba734ad5-bc8720bdf9b309fe, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d,
Used=1.08 MB(1130496 B), Peak=7.17 MB(7520256 B)
    MemTracker Label=local data queue mem tracker, Parent Label=Query#Id=41363cb6ba734ad5-bc8720bdf9b3090d, Used=1.08 MB(1130496 B),
Peak=7.17 MB(7520256 B)
```

OOM Killer Crash Analysis

If there is no error message in `log/be.out` after the BE process crashes, execute `dmesg -T`. If you see the following log, it means that the OOM Killer has been triggered. It can be seen that at `20240718 15:03:59`, the physical memory (anon-rss) of the doris_be process with pid 360303 is about 60 GB.

```
[Thu Jul 18 15:03:59 2024] Out of memory: Killed process 360303 (doris_be) total-vm:213416916kB, anon-rss:62273128kB, file-rss:0kB,
shmem-rss:0kB, UID:0 pgtables:337048kB oom_score_adj:0
```

Ideally, Doris will regularly detect the remaining available memory of the operating system, and take a series of actions including blocking subsequent memory requests and triggering memory GC to avoid triggering OOM Killer when memory is insufficient. However, refreshing memory status and memory GC have a certain lag, and it is difficult to completely catch all large memory requests. When the cluster pressure is too high, there is still a certain probability of triggering OOM Killer, causing the BE process to crash. In addition, if the memory status of the process is abnormal, the memory GC cannot release the memory, resulting in a decrease in the actual available memory of the process, which will increase the memory pressure of the cluster.

If the OOM Killer is triggered, first analyze the memory status and task execution of the BE process before the OOM Killer is triggered based on the log, and then adjust the parameters in a targeted manner to restore the cluster to stability.

Find the memory log before the OOM Killer is triggered

When the OOM Killer is triggered, it means that the process has insufficient available memory. Refer to Memory Log Analysis to find the last printed `Memory Tracker Summary` keyword from bottom to top at the time when the OOM Killer is triggered in `be/log/be.INFO` and analyze the main memory location of the BE process.

> `less be/log/be.INFO` After opening the file, first jump to the log corresponding to the time when OOM Killer was triggered. Taking the result of `dmesg -T` above as an example, enter `/20240718 15:03:59` and press Enter to search for the corresponding time. If it cannot be found, it may be that the time when OOM Killer was triggered is somewhat deviated. You can search for `/20240718 15:03:`. After the log jumps to the corresponding time, enter `/Memory Tracker Summary` and press Enter to search for keywords. By default, it will search downward in the log. If it cannot be found or the time does not match, you need to press `shift + n` to search upward first to find the last printed `Memory Tracker Summary` and the `Process Memory Summary` memory logs printed at the same time.

Excessive cluster memory pressure triggers OOM Killer

If the following phenomenon is met, it can be considered that the cluster memory pressure is too high, resulting in the process memory status not being refreshed in time at a certain moment, and the memory GC failing to release the memory in time, resulting in the failure to effectively control the BE process memory.

> Before Doris 2.1, Memory GC was not perfect, and when the memory was constantly tight, it was often easier to trigger the OOM Killer.

- Analysis of `Memory Tracker Summary` found that the memory usage of queries and other tasks, various caches, metadata, etc. is reasonable.
- BE process memory monitoring in the corresponding time period shows that the memory usage rate is maintained at a high level for a long time, and there is no sign of memory leak
- Locate the memory log before the OOM Killer time point in `be/log/be.INFO`, search the `GC` keyword from bottom to top, and find that the BE process frequently executes memory GC.

At this time, refer to BE Configuration Items to reduce `mem_limit` and increase `max_sys_mem_available_low_water_mark_bytes` in `be/conf/be.conf`. For more information about memory limits, watermark calculation methods, and memory GC, see Memory Control Strategy.

In addition, other parameters can be adjusted to control memory status refresh and GC, including `memory_gc_sleep_time_ms`, `soft_mem_limit_frac`, `memory_maintenance_sleep_time_ms`, `process_minor_gc_size`, `process_full_gc_size`, `enable_query_memory_overcommit`, `thread_wait_gc_max_milliseconds`, etc.

Some abnormal problems trigger OOM Killer

If the cluster memory pressure is too high, the memory status may be abnormal at this time, and the memory GC may not be able to release the memory in time. The following are some common abnormal problems that trigger OOM Killer.

Memory Tracker Statistics Missing

If the difference between `Label=process resident memory` Memory Tracker and `Label=sum of all trackers` Memory Tracker in the log `Memory Tracker Summary` is large, or the Orphan Memory Tracker value is too large, it means that there is a statistical missing in the Memory Tracker. Refer to the Memory Tracker Statistics Missing section in Memory Tracker for further analysis.

Query Cancel stuck

Locate the time point of OOM Killer in the `be/log/be.INFO` log, and then search `Memory Tracker Summary` in the context to find the process memory statistics log. If there is a query that uses a large amount of memory in the `Memory Tracker Summary`, execute `grep {queryID} be/log/be.INFO` to confirm whether there is a log with the keyword `Cancel`. The corresponding time point is the time when the query was canceled. If the query has been canceled, and the time point when the query was canceled is a long time away from the time point when the OOM Killer was triggered, refer to the analysis of [Query Cancel process stuck] in Memory Problem FAQ. For analysis of `Memory Tracker Summary`, refer to Memory Log Analysis.

Jemalloc Metadata has a large memory footprint

Memory GC currently cannot release Jemalloc Metadata. Refer to the analysis of `Label=tc/jemalloc_metadata` Memory Tracker in Memory Tracker to reduce memory usage.

Jemalloc Cache has a large memory footprint

> Common in Doris 2.0

The default value of `lg_tcache_max` in `JEMALLOC_CONF` in `be.conf` of Doris 2.0 is 20, which will cause the Jemalloc Cache to be too large and unable to be automatically released in some scenarios. Refer to Jemalloc Memory Analysis to reduce the memory footprint of Jemalloc Cache.

Memory Log Analysis

The process memory logs in `be/log/be.INFO` are mainly divided into two categories. One is the process memory status log, including the process memory size and the remaining available memory size of the system. The other is a more detailed process memory statistics log, including the memory size counted by Memory Tracker.

Process memory status log analysis

The process memory status of Doris BE will be printed in the `log/be.INFO` log every time the process memory increases or decreases by 256 MB. In addition, when the process memory is insufficient, the process memory status will also be printed along with other logs.

```
os physical memory 375.81 GB. process memory used 4.09 GB(= 3.49 GB[vm/rss] - 410.44 MB[tc/jemalloc_cache] + 1 GB[reserved] + 0B[waiting_refresh]), limit 3.01 GB, soft limit 2.71 GB. sys available memory 134.41 GB(= 1 35.41 GB[proc/available] - 1 GB[reserved] - 0B[waiting_refresh]), low water mark 3.20 GB, warning water mark 6.40 GB.
```

1. `os physical memory 375.81 GB` refers to the system physical memory 375.81 GB.

2. `process memory used 4.09 GB (= 3.49 GB [vm/rss] - 410.44 MB [tc/jemalloc_cache] + 1 GB [reserved] + 0B [waiting_refresh])`

- Currently we think that the BE process uses 4.09 GB of memory, and the actual physical memory used by the BE process `vm/rss` is 3.49 GB,
- 410.44 MB of which is `tc/jemalloc_cache`. This part of the cache will be reused first in the subsequent execution process, so it is not counted as BE process memory here.
- `reserved` is the memory reserved during the execution process. Usually, before building HashTable and other operations that consume a lot of memory, the memory of HashTable is reserved in advance to ensure that the process of building HashTable will not be terminated due to insufficient memory. This part of the reserved memory is calculated in the BE process memory, even if it has not actually been allocated.
- `waiting_refresh` is the large memory requested in the interval between two memory status refreshes. The default interval for Doris memory status refreshes is 100ms. To avoid a large number of memory requests in the interval between two memory status refreshes, the memory GC is not detected and triggered in time after the memory exceeds the limit. Therefore, the large memory requested in the interval is calculated in the BE process memory. `waiting_refresh` will be cleared to 0 after each memory status refresh.

3. `sys available memory 134.41 GB (= 135.41 GB [proc/available] - 1 GB [reserved] - 0B [waiting_refresh])`

- The current remaining available memory for the BE process is 134.41 GB, and the actual memory `proc/available` available to the BE process in the system is 135.41 GB.
- 1GB of the memory has been reserved, so `reserved` is subtracted when calculating the remaining available memory of the BE process. Regarding `reserved` and `waiting_refresh` For an introduction to `limit 3.01 GB, soft limit 2.71 GB` and `low water mark 3.20 GB, warning water mark 6.40 GB`, for more information about MemLimit and WaterMark, see [Memory limit and watermark calculation method].

Process memory statistics log analysis

When the available memory of the process is insufficient, most memory requests in BE will be aware of it and try to make a predetermined callback method, including triggering Memory GC or Cancel query, etc., and print the process memory statistics log. The default printing interval is 1s. The log is divided into two parts: `Process Memory Summary` and `Memory Tracker Summary`. You can find it in `be/log/be.INFO` to confirm whether the current process memory usage is in line with expectations.

```
Process Memory Summary:
    os physical memory 375.81 GB. process memory used 4.09 GB(= 3.49 GB[vm/rss] - 410.44 MB[tc/jemalloc_cache] + 1 GB[reserved] +
0B[waiting_refresh]), limit 3.01 GB, soft limit 2.71 GB. sys available memory 134.41 GB(= 135.41 GB[proc/available] - 1 GB[reserved]
- 0B[waiting_refresh]), low water mark 3.20 GB, warning water mark 6.40 GB.
Memory Tracker Summary:
    MemTrackerLimiter Label=other, Type=overview, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=schema_change, Type=overview, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=compaction, Type=overview, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=load, Type=overview, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=query, Type=overview, Limit=-1.00 B(-1 B), Used=83.32 MB(87369024 B), Peak=88.33 MB(92616000 B)
    MemTrackerLimiter Label=global, Type=overview, Limit=-1.00 B(-1 B), Used=199.37 MB(209053204 B), Peak=199.37 MB(209053204 B)
    MemTrackerLimiter Label=tc/jemalloc_cache, Type=overview, Limit=-1.00 B(-1 B), Used=410.44 MB(430376896 B), Peak=-1.00 B(-1 B)
    MemTrackerLimiter Label=tc/jemalloc_metadata, Type=overview, Limit=-1.00 B(-1 B), Used=144 MB(151759440 B), Peak=-1.00 B(-1 B)
    MemTrackerLimiter Label=sum of all trackers, Type=overview, Limit=-1.00 B(-1 B), Used=693.14 MB(726799124 B), Peak=-1.00 B(-1 B)
    MemTrackerLimiter Label=process resident memory, Type=overview, Limit=-1.00 B(-1 B), Used=3.49 GB(3743289344 B), Peak=3.49
GB(3743289344 B)
    MemTrackerLimiter Label=reserved_memory, Type=overview, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=-1.00 B(-1 B)
    MemTrackerLimiter Label=process virtual memory, Type=overview, Limit=-1.00 B(-1 B), Used=44.25 GB(47512956928 B), Peak=44.25
GB(47512956928 B)
    MemTrackerLimiter Label=Orphan, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=DetailsTrackerSet, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTracker Label=IOBufBlockMemory, Parent Label=DetailsTrackerSet, Used=1.41 MB(1474560 B), Peak=1.41 MB(1474560 B)
    MemTracker Label=SegmentCache[size], Parent Label=DetailsTrackerSet, Used=1.64 MB(1720543 B), Peak=18.78 MB(19691997 B)
    MemTracker Label=SchemaCache[number], Parent Label=DetailsTrackerSet, Used=9.21 KB(9428 B), Peak=9.21 KB(9428 B)
    MemTracker Label=TabletSchemaCache[number], Parent Label=DetailsTrackerSet, Used=9.29 MB(9738798 B), Peak=9.29 MB(9738798 B)
    MemTracker Label=TabletMeta(experimental), Parent Label=DetailsTrackerSet, Used=25.08 MB(26303456 B), Peak=25.08 MB(26303456 B)
    MemTracker Label=RuntimeFilterMergeControllerEntity(experimental), Parent Label=DetailsTrackerSet, Used=32.00 B(32 B),
Peak=32.00 B(32 B)
    MemTrackerLimiter Label=SegCompaction, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=PointQueryExecutor, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=BlockCompression, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=RowIdStorageReader, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=SubcolumnsTree, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=S3FileBuffer, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=DataPageCache[size](AllocByAllocator), Type=global, Limit=-1.00 B(-1 B), Used=198.70 MB(208357157 B),
Peak=198.73 MB(208381892 B)
    MemTrackerLimiter Label=IndexPageCache[size](AllocByAllocator), Type=global, Limit=-1.00 B(-1 B), Used=679.73 KB(696047 B),
Peak=679.73 KB(696047 B)
    MemTrackerLimiter Label=PKIndexPageCache[size](AllocByAllocator), Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=Query#Id=529e3cb37dff464c-93bd9eafa8944ea6, Type=query, Limit=2.00 GB(2147483648 B), Used=83.32
MB(87369024 B), Peak=88.33 MB(92616000 B)
    MemTrackerLimiter Label=MemTableTrackerSet, Type=load, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTrackerLimiter Label=SnapshotManager, Type=other, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
    MemTracker Label=AllMemTableMemory, Parent Label=DetailsTrackerSet, Used=0(0 B), Peak=0(0 B)
```

`Process Memory Summary` is the process memory status, refer to Process Memory Status Log Analysis above.

`Memory Tracker Summary` is a summary of the process Memory Tracker, including all Memory Trackers of `Type=overview` and `Type=global`, to help users analyze the current memory status. Refer to Overview to analyze the meaning of each part of memory.

Heap Profile Memory Analysis

Heap Profile supports real-time viewing of process memory usage and call stacks, so this usually requires some understanding of the code. It should be noted that Heap Profile records virtual memory. You need to modify the configuration and restart the Doris BE process, and the phenomenon can be reproduced.

Doris uses Jemalloc as the default Allocator. Refer to the following method to use Heap Profile.

1. Change `prof_active:false` of `JEMALLOC_CONF` in `be.conf` to `prof_active:true` and restart Doris BE.
2. After executing `curl http://be_host:8040/jeheap/dump`, you will see the generated `profile` file in the `${DORIS_HOME}/log` directory.
3. After executing `jeprof --dot ${DORIS_HOME}/lib/doris_be ${DORIS_HOME}/log/profile_file`, paste the text output by the terminal to the online dot drawing website to generate a memory allocation graph.

The above process is based on Doris 2.1.8 and 3.0.4 and later versions, which are used for real-time memory analysis. If you need to observe memory for a long time or observe the cumulative value of memory application, please refer to Jemalloc Heap Profile for more information about the use of Jemalloc Heap Profile.

If you see the `Segment`, `TabletSchema`, and `ColumnReader` fields in the call stack with a large memory share of Heap Profile, it means that the metadata occupies a large amount of memory.

If the BE memory does not decrease when the cluster is idle after running for a period of time, then you can see fields such as `Agg`, `Join`, `Filter`, `Sort`, and `Scan` in the call stack with a large memory share in the Heap Profile. If the BE process memory monitoring in the corresponding time period shows a continuous upward trend, then there is reason to suspect that there is a memory leak. Continue to analyze the code based on the call stack.

If you see fields such as `Agg`, `Join`, `Filter`, `Sort`, and `Scan` in the call stack with a large memory share in the Heap Profile during task execution on the cluster, and the memory is released normally after the task is completed, it means that most of the memory is used by the running tasks and there is no leak. If the value of `Label=query, Type=overview` Memory Tracker accounts for a smaller proportion of the total memory than the memory call stack containing the above fields in the Heap Profile, it means that the statistics of `Label=query, Type=overview` Memory Tracker are inaccurate, and you can provide timely feedback in the community.

Memory Tracker

Doris BE uses Memory Tracker to record process memory usage, including memory used in the life cycle of tasks such as query, load, compaction, schema change, and various caches. It supports real-time viewing of web pages and prints them to BE logs when memory-related errors are reported, which is used for memory analysis and troubleshooting of memory problems.

The viewing methods of Memory Tracker, the reasons for excessive memory usage represented by different Memory Trackers, and the analysis methods for reducing their memory usage have been introduced in Overview in conjunction with the Doris BE memory structure. This article only introduces the principles, structure, and some common problems of Memory Tracker.

Memory Tracking Principle

Memory Tracker relies on Doris Allocator to track each application and release of memory. For an introduction to Doris Allocator, refer to Memory Control Strategy.

Process memory: Doris BE will periodically obtain Doris BE process memory from the system and is compatible with Cgroup.

Task memory: Each query, load, compaction and other tasks will create its own unique Memory Tracker when initialized, and put the Memory Tracker into TLS (Thread Local Storage) during execution. Doris's main memory data structures are inherited from Allocator. Allocator will record each application and release of memory in TLS's Memory Tracker.

Operator memory: Different execution operators of tasks will also create their own Memory Trakcer, such as Join/Agg/Sink, etc., which support manual memory tracking or put it into TLS and recorded by `Doris Allocator` for execution logic control and analysis of memory usage of different operators in Query Profile.

Global memory: Global memory mainly includes Cache and metadata, which are shared between different tasks. Each Cache has its own unique Memory Tracker, which is tracked by `Doris Allocator` or manually; metadata memory is not fully counted at present, and more analysis depends on various metadata Counters counted by Metrics and Bvar.

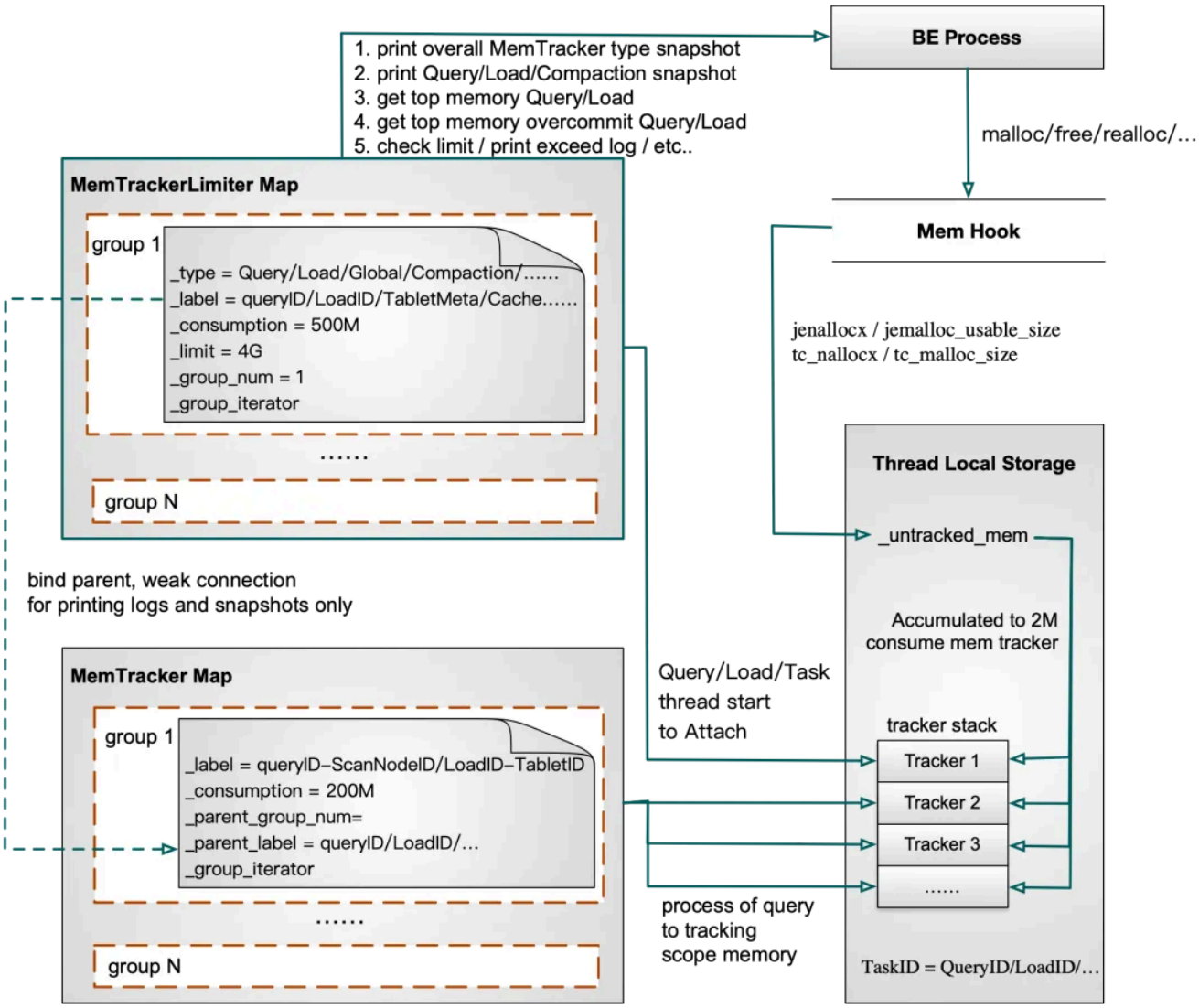
Among them, the Doris BE process memory is taken from the operating system and can be considered completely accurate. Due to implementation limitations, the memory tracked by other Memory Trackers is usually only a subset of the real memory, resulting in the sum of all Memory Trackers being less than the physical memory of the Doris BE process in most cases. There are certain omissions, but the memory recorded by Memory Tracker is more reliable in most cases and can be used for memory analysis with confidence. In addition, Memory Tracker actually tracks virtual memory, not the physical memory that is usually more concerned, and there are certain errors between them.

Memory Tracker Structure

Based on the usage, Memory Tracker is divided into two categories. The first category, Memory Tracker Limiter, is unique in each query, load, Compaction and other tasks and global Cache, TabletMeta, and is used to observe and control memory usage; the second category, Memory Tracker, is mainly used to track memory

hotspots during query execution, such as HashTable in Join/Aggregation/Sort/window functions, serialized intermediate data, etc., to analyze the memory usage of different operators in the query, and for memory control of load data flushing.

The parent-child relationship between the two is only used for snapshot printing, and is associated with the label name, which is equivalent to a soft link. It does not rely on the parent-child relationship for simultaneous consumption, and the life cycle does not affect each other, reducing the cost of understanding and use. All memory trackers are stored in a set of maps, and provide methods such as printing snapshots of all memory tracker types, printing snapshots of tasks such as query/load/compaction, obtaining the group of queries/loads that currently use the most memory, and obtaining the group of queries/loads that currently use the most excessive memory.



Memory Tracker Implement

Memory Tracker Statistics Missing

The phenomenon of Memory Tracker Statistics Missing is different in versions before and after Doris 2.1.

Memory Tracker Statistics Missing Phenomenon

1. There are two phenomena of Memory Tracker Statistics Missing after Doris 2.1.

- The difference between `Label=process resident memory` Memory Tracker and `Label=sum of all trackers` Memory Tracker is too large.
- The value of Orphan Memory Tracker is too large.

2. Before Doris 2.1, the value of Orphan Memory Tracker is too large, which means that the Memory Tracker statistics are missing.

Memory Tracker Statistics Missing Analysis

> In versions prior to Doris 2.1.5, if Memory Tracker statistics are missing or BE process memory does not decrease, please refer to Cache Memory Analysis to analyze SegmentCache memory usage, and try to close Segment Cache before continuing the test.

> In versions prior to Doris 2.1.5, Segment Cache Memory Tacker is inaccurate. This is because some Index memory statistics, including Primary Key Index, are inaccurate, resulting in Segment Cache memory not being effectively limited, often occupying too much memory, especially in large wide tables with hundreds or thousands of columns. Refer to Metadata Memory Analysis If you find that `doris_be_cache_usage{name="SegmentCache"}` in Doris BE Metrics is not large, but `doris_column_reader_num` in Doris BE Bvar is large, you need to suspect the memory usage of Segment Cache. If you see `Segment` and `ColumnReader` fields in the call stack with a large memory usage in Heap Profile, it can be basically confirmed that Segment Cache occupies a large amount of memory.

If the above phenomenon is observed, if the cluster can be easily restarted and the phenomenon can be reproduced, refer to Heap Profile Memory Analysis to use Jemalloc Heap Profile to analyze process memory.

Otherwise, you can first refer to Metadata Memory Analysis to analyze the metadata memory of Doris BE.

Reasons for missing Memory Tracker statistics

The following introduces the reasons for missing Memory Tracker statistics, which involve the implementation of Memory Tracker and usually do not need to be paid attention to.

Doris 2.1 and later

1. The difference between `Label=process resident memory` Memory Tracker and `Label=sum of all trackers` Memory Tracker is too large.

If the value of `Label=sum of all trackers` Memory Tracker accounts for more than 70% of `Label=process resident memory` Memory Tracker, it usually means that Memory Tracker has counted most of the memory of the Doris BE process, and usually only needs to analyze Memory Tracker to locate the memory location.

If the value of `Label=sum of all trackers` Memory Tracker accounts for less than 70% of `Label=process resident memory` Memory Tracker, it means that Memory Tracker statistics are missing, and Memory Tracker may not be able to accurately locate the memory location.

The difference between `Label=process resident memory` Memory Tracker and `Label=sum of all trackers` Memory Tracker is the memory not allocated by `Doris Allocator`. The main memory data structures of Doris are inherited from `Doris Allocator`, but there is still a part of memory not allocated by `Doris Allocator`, including metadata memory, RPC memory, etc., which may also be a memory leak. In this case, in addition to analyzing the Memory Tracker with a large memory value, it is usually necessary to pay attention to whether the metadata memory is reasonable and whether there is a memory leak.

2. Orphan Memory Tracker value is too large

```
MemTrackerLimiter Label=Orphan, Type=global, Limit=-1.00 B(-1 B), Used=0(0 B), Peak=0(0 B)
```

Orphan Memory Tracker is the default Memory Tracker. Positive or negative values mean that the memory allocated by Doris Allocator is not accurately tracked. The larger the value, the lower the credibility of the overall statistical results of Memory Tracker. Its statistical value comes from two sources:

- If the Memory Tracker is not bound to TLS when the thread starts, Doris Allocator will record the memory to Orphan Memory Tracker by default, which means that this part of the memory is unknown. For the principle of Doris Allocator recording memory, please refer to the above Memory Tracking Principle.
- If the value of the Memory Tracker of a task such as Query or Load is not equal to 0 when it is destroyed, it usually means that this part of the memory has not been released. The remaining memory will be recorded in the Orphan Memory Tracker, which is equivalent to letting the remaining memory continue to be tracked by the Orphan Memory Tracker. This ensures that the sum of the Orphan Memory Tracker and other Memory Trackers is equal to all the memory allocated by Doris Allocator.

Ideally, the value of the Orphan Memory Tracker is expected to be close to 0. So we hope that all threads will attach a Memory Tracker other than Orphan at the beginning, such as Query or Load Memory Tracker. And all Query or Load Memory Trackers are equal to 0 when they are destroyed, which means that the memory used during the execution of Query or Load has been released when it is destroyed.

If the Orphan Memory Tracker is not equal to 0 and has a large value, it means that a large amount of unknown memory has not been released, or a large amount of memory has not been released after the query and load are executed.

Before Doris 2.1

Before Doris 2.1, all unknown memory was counted in the `Label=Orphan` Memory Tracker, so a large value of the Orphan Memory Tracker means that the Memory Tracker statistics are missing.

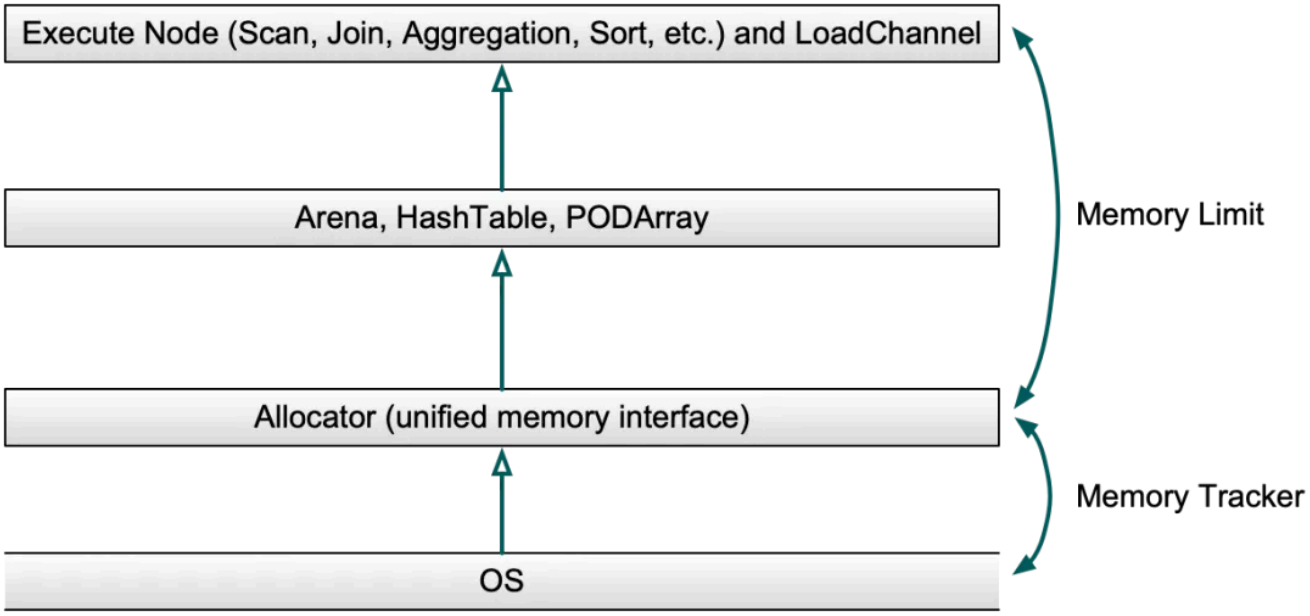
Memory Control Strategy

Doris Allocator is the unified entry point for large-block memory applications in the system. It intervenes in the process of limiting memory allocation at the right time to ensure efficient and controllable memory applications.

Doris MemoryArbitrator is a memory arbitrator that monitors the memory usage of the Doris BE process in real time, and regularly updates the memory status and collects snapshots of memory-related statistics.

Doris MemoryReclamation is a memory reclaimer that triggers memory GC to reclaim part of the memory when the available memory is insufficient, ensuring the stability of most task executions on the cluster.

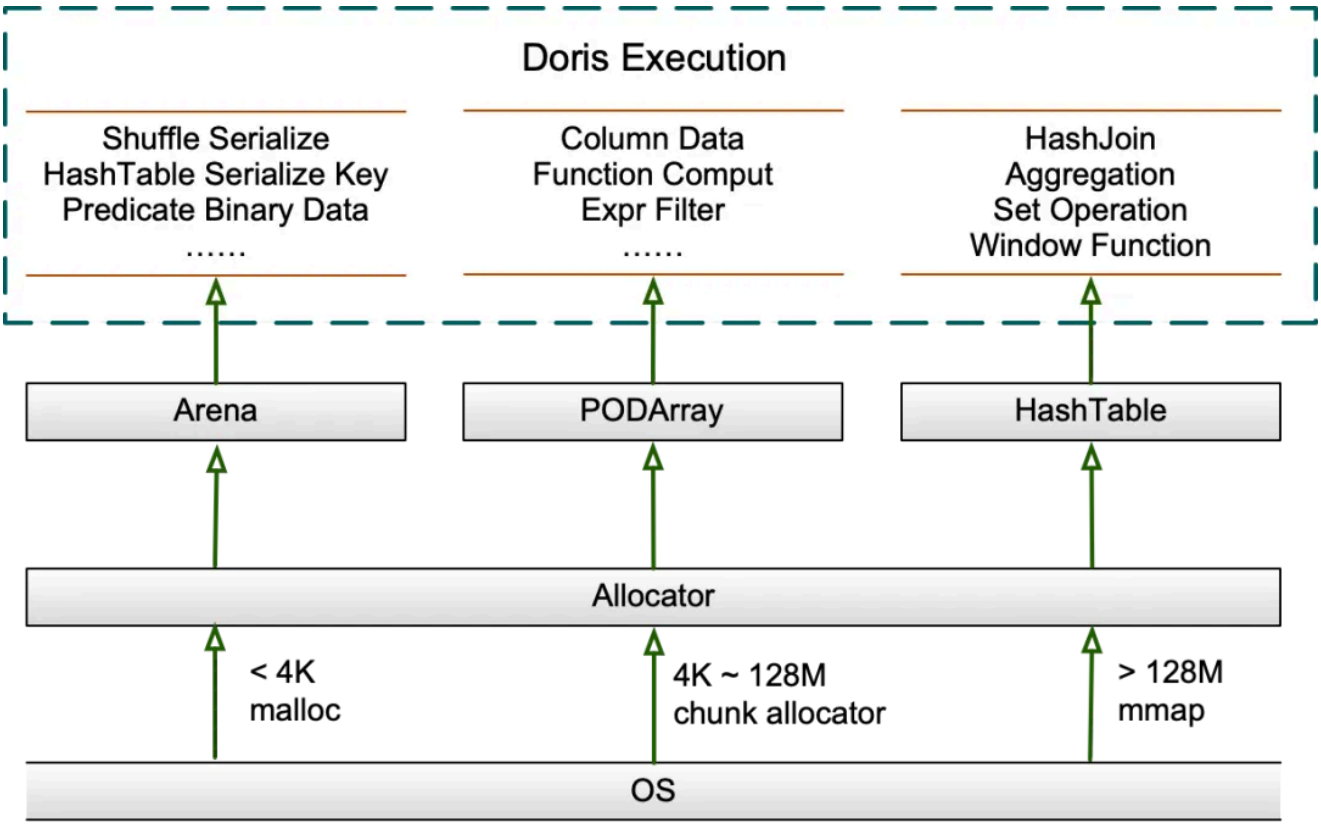
Doris Allocator



Memory Management Overview

Allocator applies for memory from the system and uses MemTracker to track the size of memory applications and releases during the application process. The large memory required for executing operators in batches will be managed by different data structures.

During the query execution process, the allocation of large blocks of memory is mainly managed by Arena, HashTable, and PODArray data structures. Allocator serves as the unified memory interface of Arena, PODArray, and HashTable to achieve unified memory management and local memory reuse.



Memory Allocator

Allocator uses a general memory allocator to apply for memory. In the choice of Jemalloc and TCMalloc, Doris previously used the Spin Lock of CentralFreeList in TCMalloc to account for 40% of the total query time in high-concurrency tests. Although turning off aggressive memory decommit can effectively improve performance, it will waste a lot of memory. For this reason, a separate thread has to be used to periodically recycle the TCMalloc cache. Jemalloc outperforms TCMalloc under high concurrency and is mature and stable. In Doris 1.2.2, Jemalloc was switched to. After tuning, the performance is on par with TCMalloc in most scenarios, and it uses less memory. The performance of high-concurrency scenarios is also significantly improved.

Arena

Arena is a memory pool that maintains a list of memory blocks and allocates memory from them to respond to alloc requests, thereby reducing the number of times memory is requested from the system to improve performance. The memory block is called Chunk, which exists throughout the life cycle of the memory pool and is uniformly released during destruction, which is usually the same as the query life cycle. It also supports memory alignment and is mainly used to save serialized/deserialized data during the Shuffle process, serialized Key in HashTable, etc.

Chunk is initially 4096 bytes, and uses a cursor to record the allocated memory location internally. If the remaining size of the current Chunk cannot meet the current memory request, a new Chunk is requested and added to the list. To reduce the number of times memory is requested from the system, when the current Chunk is less than 128M, the size of each newly requested Chunk is doubled. When the current Chunk is greater than 128M, the size of the newly requested Chunk is allocated up to 128M on the premise of meeting the current memory request, to avoid wasting too much memory. By default, the previous Chunk will no longer participate in subsequent alloc.

HashTable

HashTable in Doris is mainly used in Hash Join, aggregation, set operation, and window function. The mainly used PartitionedHashTable contains up to 16 sub-HashTables, supports parallel merging of two HashTables, and each sub-Hash Join is independently expanded, which is expected to reduce the use of total memory, and the delay during expansion will also be amortized.

When the HashTable is less than 8M, it will be expanded by a multiple of 4. When the HashTable is greater than 8M, it will be expanded by a multiple of 2. When the HashTable is less than 2G, the expansion factor is 50%, that is, the expansion is triggered when the HashTable is filled to 50%. After the HashTable is greater than 2G, the expansion factor is adjusted to 75%. In order to avoid wasting too much memory, the HashTable is usually pre-expanded according to the amount of data before building it. In addition, Doris designed different HashTables for different scenarios, such as using PHmap to optimize concurrent performance in aggregation scenarios.

PODArray

PODArray is a dynamic array of POD type. It is different from std::vector in that it does not initialize elements, supports some std::vector interfaces, supports memory alignment and expansion in multiples of 2. When PODArray is destructed, it does not call the destructor of each element, but directly releases the entire block of memory. It is mainly used to save data in columns such as String. In addition, it is also widely used in function calculation and expression filtering.

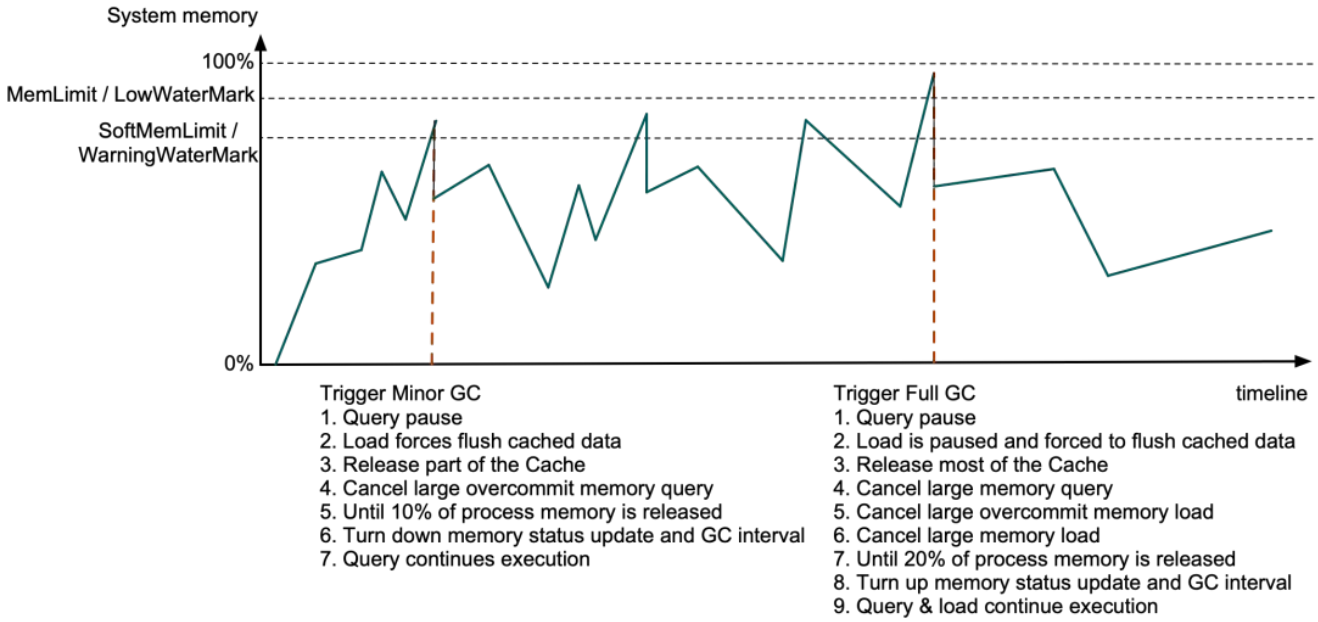
Memory reuse

Doris has done a lot of memory reuse in the execution layer, and the visible memory hotspots are basically blocked. For example, the reuse of data blocks Block runs through the execution of Query; for example, the Sender end of Shuffle always keeps one Block to receive data, and one Block is used alternately in RPC transmission; and the storage layer reuses the predicate column to read, filter, copy to the upper layer Block, and Clear when reading a Tablet; when load the Aggregate Key table, the MemTable of cached data reaches a certain size and then shrinks after pre-aggregation and continues to write, etc.

In addition, Doris will pre-allocate a batch of Free Blocks based on the number of Scanners and threads before the data Scan starts. Each time the Scanner is scheduled, a Block will be obtained from it and passed to the storage layer to read the data. After the reading is completed, the Block will be placed in the producer queue for consumption by the upper-layer operator and subsequent calculations. After the upper-layer operator copies the data, it will put the Block back into the Free Block for the next Scanner scheduling, thereby realizing memory reuse. After the data Scan is completed, the Free Block will be uniformly released in the previously pre-allocated threads to avoid the extra overhead caused by memory application and release not being in the same thread. The number of Free Blocks also controls the concurrency of data Scan to a certain extent.

Memory GC

Doris BE will periodically obtain the physical memory of the process and the current available memory of the system from the system, and collect snapshots of all query, load, and compaction task MemTracker. When the BE process memory exceeds the limit or the system has insufficient available memory, Doris will release the cache and terminate some queries or loads to release memory. This process is executed regularly by a separate GC thread.



Memory GC

Minor GC is triggered when the Doris BE process memory exceeds SoftMemLimit (81% of the total system memory by default) or the remaining available system memory is lower than the Warning watermark (usually no more than 3.2GB). At this time, the query will be paused when the Allocator allocates memory, and the data in the forced cache will be load and some Data Page Cache and expired Segment Cache will be released. If the released memory is less than 10% of the process memory, if query memory over-issuance is enabled, the query with a large memory over-issuance ratio will be canceled until 10% of the process memory is released or no query can be canceled, and then the system memory status acquisition interval and GC interval will be lowered. Other queries will continue to execute after the remaining memory is found.

If the BE process memory exceeds MemLimit (90% of the total system memory by default) or the remaining available system memory is lower than the Low watermark (usually no more than 1.6GB), Full GC is triggered. In addition to the above operations, the load will also be paused when the cache data is forced to be flushed, and all Data Page Cache and most other caches will be released. If the released memory is less than 20%, it will start to search in the MemTracker list of all queries and loads according to a certain strategy, and cancel queries with large memory usage, loads with large memory over-issuance ratios, and loads with large memory usage in turn until 20% of the process memory is released. Then, the system memory status acquisition interval and GC interval are increased, and other queries and loads will continue to execute. The GC time is usually between hundreds of us and tens of ms.

Memory limit and watermark calculation method

- Process memory limit MemLimit = `be.conf/mem_limit * PhysicalMemory`, the default is 90% of the total system memory, for details, please refer to .
- Process memory soft limit SoftMemLimit = `be.conf/mem_limit * PhysicalMemory * be.conf/soft_mem_limit_frac`, the default is 81% of the total system memory.
- System remaining available memory low water mark LowWaterMark = `be.conf/max_sys_mem_available_low_water_mark_bytes`, the default is -1, then LowWaterMark = `min(PhysicalMemory - MemLimit, PhysicalMemory * 0.05)`, on a machine with 64G memory, the value of

LowWaterMark is slightly less than 3.2 GB (because the actual value of `PhysicalMemory` is often less than 64G).

- System remaining available memory warning water mark WarningWaterMark = `2 * LowWaterMark`, on a machine with 64G memory, `WarningWaterMark` defaults to slightly less than 6.4 GB.

Calculation of remaining available memory in the system

When the available memory in the error message is less than the low water mark, it is also treated as a process memory overrun. The value of the available memory in the system comes from `MemAvailable` in `/proc/meminfo`. When `MemAvailable` is insufficient, continuing to request memory may return std::bad_alloc or cause BE process OOM. Because refreshing process memory statistics and BE memory GC have a certain lag, a small part of the memory buffer is reserved as the low water mark to avoid OOM as much as possible.

Among them, `MemAvailable` is the total amount of memory that can be provided to the user process without triggering swap as much as possible, given by the operating system based on factors such as the current free memory, buffer, cache, and memory fragmentation. A simple calculation formula is: `MemAvailable = MemFree - LowWaterMark + (PageCache - min(PageCache / 2, LowWaterMark))`, which is the same as the `available` value seen by cmd `free`. For details, please refer to:

why-is-memavailable-a-lot-less-than-memfreebufferscached

Linux MemAvailable

The default low water mark is 3.2G (1.6G before 2.1.5), which is calculated based on `MemTotal`, `vm/min_free_kbytes`, `confg::mem_limit`, `config::max_sys_mem_available_low_water_mark_bytes`, and avoids wasting too much memory. Among them, `MemTotal` is the total system memory, and the value is also taken from `/proc/meminfo`; `vm/min_free_kbytes` is the buffer reserved by the operating system for the memory GC process, and the value is usually between 0.4% and 5%. On some cloud servers, `vm/min_free_kbytes` may be 5%, which will make the system available memory appear to be less than the actual value; increasing `config::max_sys_mem_available_low_water_mark_bytes` will reserve more memory buffer for Full GC on machines with more than 64G memory, otherwise reducing it will make the best possible use of memory.

Compaction

Compaction

Doris writes data through a structure similar to LSM-Tree, and continuously merges small files into large ordered files through compaction in the background. Compaction handles operations such as deletion and updating.

Appropriately adjusting the compaction strategy can greatly improve load and query efficiency. Doris provides the following compaction strategies for tuning:

Vertical compaction

Vertical compaction is a new compaction algorithm implemented in Doris 1.2.2, which is used to optimize compaction execution efficiency and resource overhead in large-scale and wide table scenarios. It can effectively reduce the memory overhead of compaction and improve the execution speed of compaction. The test results show that the memory consumption by vertical compaction is only 1/10 of the original compaction algorithm, and the compaction rate is increased by 15%.

In vertical compaction, merging by row is changed to merging by column group. The granularity of each merge is changed to column group, which reduces the amount of data involved in single compaction and reduces the memory usage during compaction.

BE configuration: - `enable_vertical_compaction = true` will turn on vertical compaction - `vertical_compaction_num_columns_per_group = 5` The number of columns contained in each column group, by testing, the efficiency and memory usage of a group of 5 columns by default is more friendly - `vertical_compaction_max_segment_size` is used to configure the size of the disk file after vertical compaction, the default value is 268435456 (bytes)

Segment compaction

Segment compaction mainly deals with the large-scale data load. Segment compaction operates during the load process and compact segments inside the job, which is different from normal compaction and vertical compaction. This mechanism can effectively reduce the number of generated segments and avoid the -238 (OLAP_ERR_TOO_MANY_SEGMENTS) errors.

The following features are provided by segment compaction: - reduce the number of segments generated by load - the compacting process is parallel to the load process, which will not increase the load time - memory consumption and computing resources will increase during loading, but the increase is relatively low because it is evenly distributed throughout the long load process. - data after segment compaction will have resource and performance advantages in subsequent queries and normal compaction.

BE configuration: - `enable_segcompaction=true` turn it on. - `segcompaction_batch_size` is used to configure the interval for merging. The default value 10 means that every 10 segment files will trigger a segment compaction. It is recommended to set between 10 - 30. The larger value will increase the memory usage of segment compaction.

Situations where segment compaction is recommended:

- Loading large amounts of data fails at OLAP_ ERR_ TOO_ MANY_ SEGMENTS (errcode - 238) error. Then it is recommended to turn on segment compaction to reduce the quantity of segments during the load process.
- Too many small files are generated during the load process: although the amount of loading data is reasonable, the generation of a large number of small segment files may also fail the load job because of low cardinality or memory constraints that trigger memtable to be flushed in advance. Then it is recommended to turn on this function.
- Query immediately after loading. When the load is just finished and the standard compaction has not finished, large number of segment files will affect the efficiency of subsequent queries. If the user needs to query immediately after loading, it is recommended to turn on this function.
- The pressure of normal compaction is high after loading: segment compaction evenly puts part of the pressure of normal compaction on the loading process. At this time, it is recommended to enable this function.

Situations where segment compaction is not recommended: - When the load operation itself has exhausted memory resources, it is not recommended to use the segment compaction to avoid further increasing memory pressure and causing the load job to fail.

Refer to this link for more information about implementation and test results.

Single replica compaction

> **Deprecated**
>
> Single replica compaction has been deprecated and is removed since Doris 4.1.2, because its peer selection carries a data correctness risk: the replicas that pull the compacted files choose which peer and version to fetch from based on replica information that is only refreshed periodically, while each replica advances through versions independently. This selection can therefore act on a stale view of the cluster and lead to subtle data inconsistencies. Enabling this feature is not recommended. The `enable_single_replica_compaction` property no longer exists in Doris 4.x.

By default, compaction for multiple replicas is performed independently, with each replica consuming CPU and IO resources. When single replica compaction is enabled, only one replica performs the compaction. Afterward, the other replicas pull the compacted files from this replica, resulting in CPU resources being consumed only once, saving N - 1 times CPU usage (where N is the number of replicas).

Single replica compaction is specified in the table's PROPERTIES via the parameter `enable_single_replica_compaction`, which is false by default (disabled). To enable it, set the parameter to true.

This parameter can be specified when creating the table or modified later using:

```
ALTER TABLE table_name SET("enable_single_replica_compaction" = "true");
```

Compaction strategy

The compaction strategy determines when and which small files are merged into larger files. Doris currently offers two compaction strategies, specified by the `compaction_policy` parameter in the table properties.

Size-based compaction strategy

The size-based compaction strategy is the default strategy and is suitable for most scenarios.

```
"compaction_policy" = "size_based"
```

Time series compaction strategy

The time series compaction strategy is optimized for scenarios like logs and time-series data. It leverages the time locality of time-series data, merging small files written in adjacent times into larger files. Each file participates in compaction only once, reducing write amplification from repeated compaction.

```
"compaction_policy" = "time_series"
```

The time series compaction strategy is triggered when any of the following conditions are met: - The size of unmerged files exceeds `time_series_compaction_goal_size_mbytes` (default 1 GB). - The number of unmerged files exceeds `time_series_compaction_file_count_threshold` (default 2000). - The time since the last compaction exceeds `time_series_compaction_time_threshold_seconds` (default 1 hour).

These parameters are set in the table's PROPERTIES and can be specified when creating the table or modified later using:

```
ALTER TABLE table_name SET("name" = "value");
```

Compaction concurrency control

Compaction runs in the background and consumes CPU and IO resources. The resource consumption can be controlled by adjusting the number of concurrent compaction threads.

The number of concurrent compaction threads is configured in the BE configuration file, including the following parameters: - `max_base_compaction_threads`: Number of base compaction threads, default is 4. - `max_cumu_compaction_threads`: Number of cumulative compaction threads, default is -1, which mean that 1 thread per disk. - `max_single_replica_compaction_threads`: Number of threads for fetching data files during single replica compaction, default is 10.

Metadata Operations and Maintenance

> **Avoid using metadata_failure_recovery unless absolutely necessary. Using it may cause metadata truncation, loss, and split-brains. Use cautiously to prevent irreversible data damage from improper operations.**

This document focuses on how to manage Doris metadata in a real production environment. It includes the proposed deployment of FE nodes, some commonly used operational methods, and common error resolution methods.

For the time being, read the Doris metadata design document to understand how Doris metadata works.

Important tips

- Current metadata design is not backward compatible. That is, if the new version has a new metadata structure change (you can see whether there is a new VERSION in the `FeMetaVersion.java` file in the FE code), it is usually impossible to roll back to the old version after upgrading to the new version. Therefore, before upgrading FE, be sure to test metadata compatibility according to the operations in the Upgrade Document.

Metadata catalog structure

Let's assume that the path of `meta_dir` specified in fe.conf is `path/to/doris-meta`. In a normal Doris cluster, the directory structure of metadata should be as follows:

```
/path/to/doris-meta/
            |-- bdb/
            |   |-- 00000000.jdb
            |   |-- je.config.csv
            |   |-- je.info.0
            |   |-- je.info.0.lck
            |   |-- je.lck
            |   `-- je.stat.csv
            `-- image/
                |-- ROLE
                |-- VERSION
                `-- image.xxxx
```

1. bdb

   We use bdbje as a distributed kV system to store metadata journal. This BDB directory is equivalent to the "data directory" of bdbje.

   The `.jdb` suffix is the data file of bdbje. These data files will increase with the increasing number of metadata journals. When Doris regularly completes the image, the old log is deleted. So normally, the total size of these data files varies from several MB to several GB (depending on how Doris is used, such as import frequency). When the total size of the data file is larger than 10GB, you may need to

wonder whether the image failed or the historical journals that failed to distribute the image could not be deleted.

`je.info.0` is the running log of bdbje. The time in this log is UTC + 0 time zone. We may fix this in a later version. From this log, you can also see how some bdbje works.

2. image directory

   The image directory is used to store metadata mirrors generated regularly by Doris. Usually, you will see a `image.xxxxx` mirror file. Where `xxxxx` is a number. This number indicates that the image contains all metadata journal before `xxxx`. And the generation time of this file (viewed through `ls -al`) is usually the generation time of the mirror.

   You may also see a `image.ckpt` file. This is a metadata mirror being generated. The `du -sh` command should show that the file size is increasing, indicating that the mirror content is being written to the file. When the mirror is written, it automatically renames itself to a new `image.xxxxx` and replaces the old image file.

   Only FE with a Master role will actively generate image files on a regular basis. After each generation, FE is pushed to other non-Master roles. When it is confirmed that all other FEs have received this image, Master FE deletes the metadata journal in bdbje. Therefore, if image generation fails or image push fails to other FEs, data in bdbje will accumulate.

   `ROLE` file records the type of FE (FOLLOWER or OBSERVER), which is a text file.

   `VERSION` file records the cluster ID of the Doris cluster and the token used to access authentication between nodes, which is also a text file.

   `ROLE` file and `VERSION` file may only exist at the same time, or they may not exist at the same time (e.g. at the first startup).

Basic operations

Start single node FE

Single node FE is the most basic deployment mode. A complete Doris cluster requires at least one FE node. When there is only one FE node, the type of the node is Follower and the role is Master.

1. First start-up
    1. Suppose the path of `meta_dir` specified in fe.conf is `path/to/doris-meta`.
    2. Ensure that `path/to/doris-meta` already exists, that the permissions are correct and that the directory is empty.
    3. Start directly through `sh bin/start_fe.sh`.
    4. After booting, you should be able to see the following log in fe.log:
        - Palo FE starting…
        - image does not exist: /path/to/doris-meta/image/image.0
        - transfer from INIT to UNKNOWN

- transfer from UNKNOWN to MASTER
- the very first time to open bdb, dbname is 1
- start fencing, epoch number is 1
- finish replay in xxx msec
- QE service start
- thrift server started

The above logs are not necessarily strictly in this order, but they are basically similar.

5. The first start-up of a single-node FE usually does not encounter problems. If you haven't seen the above logs, generally speaking, you haven't followed the document steps carefully, please read the relevant wiki carefully.

2. Restart

    1. Stopped FE nodes can be restarted by using `sh bin/start_fe.sh`.

    2. After restarting, you should be able to see the following log in fe.log:

        - Palo FE starting…
        - finished to get cluster id: xxxx, role: FOLLOWER and node name: xxxx
        - If no image has been generated before reboot, you will see:
        - image does not exist: /path/to/doris-meta/image/image.0
        - If an image is generated before the restart, you will see:
        - start load image from /path/to/doris-meta/image/image.xxx. is ckpt: false
        - finished load image in xxx ms
        - transfer from INIT to UNKNOWN
        - replayed journal id is xxxx, replay to journal id is yyyy
        - transfer from UNKNOWN to MASTER
        - finish replay in xxx msec
        - master finish replay journal, can write now.
        - begin to generate new image: image.xxxx
        - start save image to /path/to/doris-meta/image/image.ckpt. is ckpt: true
        - finished save image /path/to/doris-meta/image/image.ckpt in xxx ms. checksum is xxxx
        - push image.xxx to other nodes. totally xx nodes, push succeeded xx nodes
        - QE service start
        - thrift server started

The above logs are not necessarily strictly in this order, but they are basically similar.

3. Common problems

   For the deployment of single-node FE, start-stop usually does not encounter any problems. If you have any questions, please refer to the relevant Wiki and check your operation steps carefully.

Add FE

Adding FE processes is described in detail in the Elastic Expansion Documents and will not be repeated. Here are some points for attention, as well as common problems.

1. Notes

   - Before adding a new FE, make sure that the current Master FE runs properly (connection is normal, JVM is normal, image generation is normal, bdbje data directory is too large, etc.)
   - The first time you start a new FE, you must make sure that the `--helper` parameter is added to point to Master FE. There is no need to add `--helper` when restarting. (If `--helper` is specified, FE will directly ask the helper node for its role. If not, FE will try to obtain information from `ROLE` and `VERSION` files in the `doris-meta/image/` directory.
   - The first time you start a new FE, you must make sure that the `meta_dir` of the FE is created, has correct permissions and is empty.
   - Starting a new FE and executing the `ALTER SYSTEM ADD FOLLOWER/OBSERVER` statement adds FE to metadata in a sequence that is not required. If a new FE is started first and no statement is executed, the `current node is not added to the group. Please add it first.` in the new FE log. When the statement is executed, it enters the normal process.
   - Make sure that after the previous FE is added successfully, the next FE is added.
   - Connect to MASTER FE and execute `ALTER SYSTEM ADD FOLLOWER/OBSERVER` stmt.

2. Common problems

   1. this need is DETACHED

      When you first start a FE to be added, if the data in doris-meta/bdb on Master FE is large, you may see the words `this node is DETACHED.` in the FE log to be added. At this point, bdbje is copying data, and you can see that the `bdb/` directory of FE to be added is growing. This process usually takes several minutes (depending on the amount of data in bdbje). Later, there may be some bdbje-related error stack information in fe. log. If `QE service start` and `thrift server start` are displayed in the final log, the start is usually successful. You can try to connect this FE via mysql-client. If these words do not appear, it may be the problem of bdbje replication log timeout. At this point, restarting the FE directly will usually solve the problem.

   2. Failure to add due to various reasons

      - If OBSERVER is added, because OBSERVER-type FE does not participate in the majority of metadata writing, it can theoretically start and stop at will. Therefore, for the case of adding OBSERVER failure. The process of OBSERVER FE can be killed directly. After clearing the metadata directory of OBSERVER, add the process again.

      - If FOLLOWER is added, because FOLLOWER is mostly written by participating metadata. So it is possible that FOLLOWER has joined the bdbje electoral team. If there are only two

FOLLOWER nodes (including MASTER), then stopping one FE may cause another FE to quit because it cannot write most of the time. At this point, we should first delete the newly added FOLLOWER node from the metadata through the `ALTER SYSTEM DROP FOLLOWER` command, then kill the FOLLOWER process, empty the metadata and re-add the process.

Delete FE

The corresponding type of FE can be deleted by the `ALTER SYSTEM DROP FOLLOWER/OBSERVER` command. The following points should be noted:

- For OBSERVER type FE, direct DROP is enough, without risk.
- For FOLLOWER type FE. First, you should make sure that you start deleting an odd number of FOLLOWERs (three or more).
    1. If the FE of non-MASTER role is deleted, it is recommended to connect to MASTER FE, execute DROP command, and then kill the process.
    2. If you want to delete MASTER FE, first confirm that there are odd FOLLOWER FE and it works properly. Then kill the MASTER FE process first. At this point, a FE will be elected MASTER. After confirming that the remaining FE is working properly, connect to the new MASTER FE and execute the DROP command to delete the old MASTER FE.

Advanced Operations

Failure recovery

FE may fail to start bdbje and synchronize between FEs for some reasons. Phenomena include the inability to write metadata, the absence of MASTER, and so on. At this point, we need to manually restore the FE. The general principle of manual recovery of FE is to start a new MASTER through metadata in the current `meta_dir`, and then add other FEs one by one. Please follow the following steps strictly:

1. First, **stop all FE processes and all business access**. Make sure that during metadata recovery, external access will not lead to other unexpected problems.(if not, this may cause split-brain problem)
2. Identify which FE node's metadata is up-to-date:
    - First of all, **be sure to back up all FE's** `meta_dir` **directories first.**
    - Usually, Master FE's metadata is up to date. You can see the suffix of image.xxxx file in the `meta_dir/image` directory. The larger the number, the newer the metadata.
    - Usually, by comparing all FOLLOWER FE image files, you can find the latest metadata.
    - After that, we use the FE node with the latest metadata to recover.
    - If using metadata of OBSERVER node to recover will be more troublesome, it is recommended to choose FOLLOWER node as far as possible.
3. The following operations are performed on the FE nodes selected in step 2.
    1. Modify fe.conf
        - If the node is an OBSERVER, first change the `role=OBSERVER` in the `meta_dir/image/ROLE` file to `role=FOLLOWER`. (Recovery from the OBSERVER node will be more cumbersome, first follow the steps here, followed by a separate description)

        - If fe.version < 2.0.2, add configuration in fe.conf: `metadata_failure_recovery=true`.
    2. Run `sh bin/start_fe.sh --metadata_failure_recovery --daemon` to start the FE. (If you are recovering from an OBSERVER node, jump to the subsequent OBSERVER document after this step.)
    3. If normal, the FE will start in the role of MASTER, similar to the description in the previous section `Start a single node FE`. You should see the words `transfer from XXXX to MASTER` in fe.log.
    4. After the start-up is completed, connect to the FE first, and execute some query imports to check whether normal access is possible. If the operation is not normal, it may be wrong. It is recommended to read the above steps carefully and try again with the metadata previously backed up. If not, the problem may be more serious.
    5. If successful, through the `show frontends;` command, you should see all the FEs you added before, and the current FE is master.
    6. **If FE version < 2.0.2**, delete the `metadata_failure_recovery=true` configuration item in fe.conf, or set it to `false`, and restart the FE (**Important**).

    :::tip If you are recovering metadata from an OBSERVER node, after completing the above steps, you will find that the current FE role is OBSERVER, but `IsMaster` appears as `true`. This is because the "OBSERVER" seen here is recorded in Doris's metadata, but whether it is master or not, is recorded in bdbje's metadata. Because we recovered from an OBSERVER node, there was inconsistency. Please take the following steps to fix this problem (we will fix it in a later version):

    1. First, all FE nodes except this "OBSERVER" are DROPed out.

    2. A new FOLLOWER FE is added through the `ADD FOLLOWER` command, assuming that it is on hostA.

    3. Start a new FE on hostA and join the cluster by `helper`.

    4. After successful startup, you should see two FEs through the `show frontends;` statement, one is the previous OBSERVER, the other is the newly added FOLLOWER, and the OBSERVER is the master.

    5. After confirming that the new FOLLOWER is working properly, the new FOLLOWER metadata is used to perform a failure recovery operation again.

    6. The purpose of the above steps is to manufacture a metadata of FOLLOWER node artificially, and then use this metadata to restart fault recovery. This avoids inconsistencies in recovering metadata from OBSERVER.

    The meaning of `metadata_failure_recovery` is to empty the metadata of `bdbje`. In this way, bdbje will not contact other FEs before, but start as a separate FE. This parameter needs to be set to true only when restoring startup. After recovery, it must be set to false. Otherwise, once restarted, the metadata of bdbje will be emptied again, which will make other FEs unable to work properly. :::

4. After the successful execution of step 3, we delete the previous FEs from the metadata by using the `ALTER SYSTEM DROP FOLLOWER/OBSERVER` command and add them again by adding new FEs.

5. If the above operation is normal, it will be restored.

FE type change

If you need to change the existing FOLLOWER/OBSERVER type FE to OBSERVER/FOLLOWER type, please delete FE in the way described above, and then add the corresponding type FE.

FE Migration

If you need to migrate one FE from the current node to another, there are several scenarios.

1. FOLLOWER, or OBSERVER migration for non-MASTER nodes

   After adding a new FOLLOWER / OBSERVER directly, delete the old FOLLOWER / OBSERVER.

2. Single-node MASTER migration

   When there is only one FE, refer to the `Failure Recovery` section. Copy the doris-meta directory of FE to the new node and start the new MASTER in Step 3 of the `Failure Recovery` section

3. A set of FOLLOWER migrates from one set of nodes to another set of new nodes

   Deploy FE on the new node and add the new node first by adding FOLLOWER. The old nodes can be dropped by DROP one by one. In the process of DROP-by-DROP, MASTER automatically selects the new FOLLOWER node.

Replacement of FE port

FE currently has the following ports

- Ed_log_port: bdbje's communication port
- http_port: http port, also used to push image
- rpc_port: thrift server port of Frontend
- query_port: Mysql connection port
- arrow_flight_sql_port: Arrow Flight SQL connection port

1. edit_log_port

   If this port needs to be replaced, it needs to be restored with reference to the operations in the `Failure Recovery` section. Because the port has been persisted into bdbje's own metadata (also recorded in Doris's own metadata), it is necessary to clear bdbje's metadata by setting `metadata_failure_recovery` when Fe start.

2. http_port

   All FE http_ports must be consistent. So if you want to modify this port, all FEs need to be modified and restarted. Modifying this port will be more complex in the case of multiple FOLLOWER deployments (involving laying eggs and laying hens…), so this operation is not recommended. If necessary, follow the operation in the `Failure Recovery` section directly.

3. rpc_port

   After modifying the configuration, restart FE directly. Master FE informs BE of the new port through heartbeat. Only this port of Master FE will be used. However, it is still recommended that all FE ports be

consistent.

4. query_port

   After modifying the configuration, restart FE directly. This only affects mysql's connection target.

5. arrow_flight_sql_port

   After modifying the configuration, restart FE directly. This only affects arrow flight sql server connection target.

Recover metadata from FE memory

In some extreme cases, the image file on the disk may be damaged, but the metadata in the memory is intact. At this point, we can dump the metadata from the memory and replace the image file on the disk to recover the metadata. the entire non-stop query service operation steps are as follows:

1. Stop all Load, Create, Alter operations.
2. Execute the following command to dump metadata from the Master FE memory: (hereafter called image_mem)

```
curl -u $root_user:$password http://$master_hostname:8030/dump
```

3. Replace the image file in the `meta_dir/image` directory on the OBSERVER FE node with the image_mem file, restart the OBSERVER FE node, and verify the integrity and correctness of the image_mem file. You can check whether the DB and Table metadata are normal on the FE Web page, whether there is an exception in `fe.log`, whether it is in a normal replayed jour.

   Since 1.2.0, it is recommended to use following method to verify the `image_mem` file:

   ```
   sh start_fe.sh --image path_to_image_mem
   ```

   > Notice: `path_to_image_mem` is the path of `image_mem`.
   >
   > If verify succeed, it will print: `Load image success. Image file /absolute/path/to/image.xxxxxx is valid`.
   >
   > If verify failed, it will print: `Load image failed. Image file /absolute/path/to/image.xxxxxx is invalid`.

4. Replace the image file in the `meta_dir/image` directory on the FOLLOWER FE node with the image_mem file in turn, restart the FOLLOWER FE node, and confirm that the metadata and query services are normal.
5. Replace the image file in the `meta_dir/image` directory on the Master FE node with the image_mem file, restart the Master FE node, and then confirm that the FE Master switch is normal and The Master FE node can generate a new image file through checkpoint.
6. Recover all Load, Create, Alter operations.

Note: If the Image file is large, the entire process can take a long time, so during this time, make sure Master FE does not generate a new image file via checkpoint. When the image.ckpt file in the meta_dir/image directory on the Master FE node is observed to be as large as the image.xxx file, the image.ckpt file can be deleted directly.

View data in BDBJE

The metadata log of FE is stored in BDBJE in the form of Key-Value. In some abnormal situations, FE may not be started due to metadata errors. In this case, Doris provides a way to help users query the data stored in BDBJE to facilitate troubleshooting.

First, you need to add configuration in fe.conf: `enable_bdbje_debug_mode=true`, and then start FE through `sh start_fe.sh --daemon`.

At this time, FE will enter the debug mode, only start the http server and MySQL server, and open the BDBJE instance, but will not load any metadata and other subsequent startup processes.

This is, we can view the data stored in BDBJE by visiting the web page of FE, or after connecting to Doris through the MySQL client, through `show proc "/bdbje";`.

```
mysql> show proc "/bdbje";
+----------+---------------+---------+
| DbNames  | JournalNumber | Comment |
+----------+---------------+---------+
| 110589   | 4273          |         |
| epochDB  | 4             |         |
| metricDB | 430694        |         |
+----------+---------------+---------+
```

The first level directory will display all the database names in BDBJE and the number of entries in each database.

```
mysql> show proc "/bdbje/110589";
+-----------+
| JournalId |
+-----------+
| 1         |
| 2         |

...
| 114858    |
| 114859    |
| 114860    |
| 114861    |
+-----------+
4273 rows in set (0.06 sec)
```

Entering the second level, all the entry keys under the specified database will be listed.

```
mysql> show proc "/bdbje/110589/114861";
+-----------+--------------+--------------------------------------------+
| JournalId | OpType       | Data                                       |
+-----------+--------------+--------------------------------------------+
| 114861    | OP_HEARTBEAT | org.apache.doris.persist.HbPackage@6583d5fb |
```

```
+-----------+--------------+-------------------------------------------+
1 row in set (0.05 sec)
```

The third level can display the value information of the specified key.

Best Practices

The deployment recommendation of FE is described in the Installation and Deployment Document. Here are some supplements.

- **If you don't know the operation logic of FE metadata very well, or you don't have enough experience in the operation and maintenance of FE metadata, we strongly recommend that only one FOLLOWER-type FE be deployed as MASTER in practice, and the other FEs are OBSERVER, which can reduce many complex operation and maintenance problems.** Don't worry too much about the failure of MASTER single point to write metadata. First, if you configure it properly, FE as a java process is very difficult to hang up. Secondly, if the MASTER disk is damaged (the probability is very low), we can also use the metadata on OBSERVER to recover manually through `fault recovery`.
- The JVM of the FE process must ensure sufficient memory. We **strongly recommend** that FE's JVM memory should be at least 10GB and 32GB to 64GB. And deploy monitoring to monitor JVM memory usage. Because if OOM occurs in FE, metadata writing may fail, resulting in some failures that **cannot recover**!
- FE nodes should have enough disk space to prevent the excessive metadata from causing insufficient disk space. At the same time, FE logs also take up more than a dozen gigabytes of disk space.

Other common problems

1. Output `meta out of date. current time: xxx, synchronized time: xxx, has log: xxx, fe type: xxx` in fe.log

   This is usually because the FE cannot elect Master. For example, if three FOLLOWERs are configured, but only one FOLLOWER is started, this FOLLOWER will cause this problem. Usually, just start the remaining FOLLOWER. If the problem has not been solved after the start-up, manual recovery may be required in accordance with the way in the `Failure Recovery` section.

2. `Clock delta: xxxx ms. between Feeder: xxxx and this Replica exceeds max permissible delta: xxxx ms.`

   Bdbje requires that clock errors between nodes should not exceed a certain threshold. If exceeded, the node will exit abnormally. The default threshold is 5000ms, which is controlled by FE parameter `max_bdbje_clock_delta_ms', and can be modified as appropriate. But we suggest using NTP and other clock synchronization methods to ensure the clock synchronization of Doris cluster hosts.

3. Mirror files in the `image/` directory have not been updated for a long time

   Master FE generates a mirror file by default for every 50,000 metadata journal. In a frequently used cluster, a new image file is usually generated every half to several days. If you find that the image file has not been updated for a long time (for example, more than a week), you can see the reasons in sequence as follows:

   1. Search for `memory is not enough to do checkpoint. Committed memory XXXX Bytes, used memory XXXX Bytes.` in the fe.log of Master FE. If found, it indicates that the current FE's JVM memory is

insufficient for image generation (usually we need to reserve half of the FE memory for image generation). Then you need to add JVM memory and restart FE before you can observe. Each time Master FE restarts, a new image is generated directly. This restart method can also be used to actively generate new images. Note that if there are multiple FOLLOWER deployments, then when you restart the current Master FE, another FOLLOWER FE will become MASTER, and subsequent image generation will be the responsibility of the new Master. Therefore, you may need to modify the JVM memory configuration of all FOLLOWER FE.

2. Search for `begin to generate new image: image.xxxx` in the fe.log of Master FE. If it is found, then the image is generated. Check the subsequent log of this thread, and if `checkpoint finished save image.xxxx` appears, the image is written successfully. If `Exception when generating new image file` occurs, the generation fails and specific error messages need to be viewed.

4. The size of the `bdb/` directory is very large, reaching several Gs or more.

   The BDB directory will remain large for some time after eliminating the error that the new image cannot be generated. Maybe it's because Master FE failed to push image. You can search `push image.XXXX to other nodes. totally XX nodes, push succeeded YY nodes` in the fe. log of Master FE. If YY is smaller than xx, then some FEs are not pushed successfully. You can see the specific error `Exception when pushing image file.url = xxx` in the fe. log.

   At the same time, you can add the configuration in the FE configuration file: `edit_log_roll_num = xxxx`. This parameter sets the number of metadata journals and makes an image once. The default is 50000. This number can be reduced appropriately to make images more frequent, thus speeding up the deletion of old journals.

5. FOLLOWER FE hangs up one after another

   Because Doris's metadata adopts the majority writing strategy, that is, a metadata journal must be written to at least a number of FOLLOWER FEs (for example, three FOLLOWERs, two must be written successfully) before it can be considered successful. If the write fails, the FE process exits on its own initiative. So suppose there are three FOLLOWERs: A, B and C. C hangs up first, and then B hangs up, then A will hang up. So as described in the `Best Practices`section, if you don't have extensive experience in metadata operations and maintenance, it's not recommended to deploy multiple FOLLOWERs.

6. fe.log 中出现 `get exception when try to close previously opened bdb database. ignore it`

   If there is the word `ignore it` behind it, there is usually no need to deal with it. If you are interested, you can search for this error in `BDBEnvironment.java`, and see the annotations.

7. From `show frontends;` Look, the `Join` of a FE is listed as `true`, but actually the FE is abnormal.

   Through `show frontends;` see the `Join` information. If the column is `true`, it only means that the FE **has joined the** cluster. It does not mean that it still exists normally in the cluster. If `false`, it means that the FE **has never joined the** cluster.

8. Configuration of FE `master_sync_policy`, `replica_sync_policy`, and `txn_rollback_limit`.

   `master_sync_policy` is used to specify whether fsync (), `replica_sync_policy` is called when Leader FE writes metadata log, and `replica_sync_policy` is used to specify whether other Follower FE calls fsync () when FE HA deploys synchronous metadata. In earlier versions of Doris, these two parameters

defaulted to `WRITE_NO_SYNC`, i.e., fsync () was not called. In the latest version of Doris, the default has been changed to `SYNC`, that is, fsync () is called. Calling fsync () significantly reduces the efficiency of metadata disk writing. In some environments, IOPS may drop to several hundred and the latency increases to 2-3ms (but it's still enough for Doris metadata manipulation). Therefore, we recommend the following configuration:

1. For a single Follower FE deployment, `master_sync_policy` is set to `SYNC`, which prevents the loss of metadata due to the downtime of the FE system.
2. For multi-Follower FE deployment, we can set `master_sync_policy` and `replica_sync_policy` to `WRITE_NO_SYNC`, because we think that the probability of simultaneous outage of multiple systems is very low.

If `master_sync_policy` is set to `WRITE_NO_SYNC` in a single Follower FE deployment, then a FE system outage may occur, resulting in loss of metadata. At this point, if other Observer FE attempts to restart, it may report an error:

```
Node xxx must rollback xx total commits(numPassedDurableCommits of which were durable) to the earliest point indicated by transaction xxxx in order to rejoin the replication group, but the transaction rollback limit of xxx prohibits this.
```

This means that some transactions that have been persisted need to be rolled back, but the number of entries exceeds the upper limit. Here our default upper limit is 100, which can be changed by setting `txn_rollback_limit`. This operation is only used to attempt to start FE normally, but lost metadata cannot be recovered.

FE Lock Management

FE Lock Management is used to detect potential deadlocks and slow locks within the FE process, making it easier for users to identify and troubleshoot issues in production, as well as monitor lock usage.

> **This feature is experimental and is supported starting from version 2.1.6.**

Deadlock Detection

The FE Lock Management module offers deadlock detection, a feature designed to automatically identify deadlocks.

By default, this feature is disabled but can be enabled via configuration settings. When enabled, deadlock detection will periodically check for deadlocks, with a default interval of 5 minutes. You can adjust this interval by setting the `deadlock_detection_interval_minute` parameter.

The results of deadlock detection will be logged, and if any deadlocks are detected, corresponding warning logs will be generated. To check for deadlocks, search the logs for the keyword `Deadlocks detected`.

Log Content Example (Original Log in a Single Line of Text):

```
2024-08-15 12:55:46  [ pool-1-thread-1:1034 ] - [ WARN ]  Find dead lock, info ["Thread-0" prio=5 Id=15 WAITING on
java.util.concurrent.locks.ReentrantLock$NonfairSync@5b7e0fca owned by "Thread-1" Id=16
    at java.base@17.0.6/jdk.internal.misc.Unsafe.park(Native Method)
    -  waiting on java.util.concurrent.locks.ReentrantLock$NonfairSync@5b7e0fca
    at java.base@17.0.6/java.util.concurrent.locks.LockSupport.park(LockSupport.java:211)
    at java.base@17.0.6/java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:715)
    at java.base@17.0.6/java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:938)
    at java.base@17.0.6/java.util.concurrent.locks.ReentrantLock$Sync.lock(ReentrantLock.java:153)
    at java.base@17.0.6/java.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:322)
    at app//org.example.lock.MonitoredReentrantLock.lock(MonitoredReentrantLock.java:22)
    at app//org.example.Main.lambda$testDeadLock$3(Main.java:79)
    ...

    Number of locked synchronizers = 1
    - java.util.concurrent.locks.ReentrantLock$NonfairSync@9abbac5

, "Thread-1" prio=5 Id=16 WAITING on java.util.concurrent.locks.ReentrantLock$NonfairSync@9abbac5 owned by "Thread-0" Id=15
    at java.base@17.0.6/jdk.internal.misc.Unsafe.park(Native Method)
    -  waiting on java.util.concurrent.locks.ReentrantLock$NonfairSync@9abbac5
    at java.base@17.0.6/java.util.concurrent.locks.LockSupport.park(LockSupport.java:211)
    at java.base@17.0.6/java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:715)
    at java.base@17.0.6/java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:938)
    at java.base@17.0.6/java.util.concurrent.locks.ReentrantLock$Sync.lock(ReentrantLock.java:153)
    at java.base@17.0.6/java.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:322)
    at app//org.example.lock.MonitoredReentrantLock.lock(MonitoredReentrantLock.java:22)
    at app//org.example.Main.lambda$testDeadLock$4(Main.java:93)
    ...

    Number of locked synchronizers = 1
    - java.util.concurrent.locks.ReentrantLock$NonfairSync@5b7e0fca
```

```
]
```

Configuration Parameters

The following are the deadlock detection-related parameters in the `fe.conf` configuration file:

Parameter Name	Description	Default Value
enable_deadlock_detection	Enables deadlock detection	false
deadlock_detection_interval_minute	Deadlock detection interval (in minutes)	5

Slow Lock Detection

The FE Lock Management module also includes slow lock detection. This feature monitors all locks related to databases, tables, and transactions. If a lock is held for longer than a specified threshold (10 seconds by default), a corresponding warning log will be generated. To check for slow locks, search the logs for the keyword `Lock held for`.

Log Content Example (Original Log in a Single Line of Text):

```
2024-08-12 16:38:51,004 INFO (mysql-nio-pool-0|242) [StreamEncoder.writeBytes():234] 2024-08-12 16:38:51  [ mysql-nio-pool-0:47482 ] - [ WARN ]  Thread ID: 242, Thread Name: mysql-nio-pool-0 - Lock held for 1923 ms, exceeding hold timeout of 1923 ms Thread stack trace:   at java.base/java.lang.Thread.getStackTrace(Thread.java:1610)
    at org.apache.doris.common.lock.AbstractMonitoredLock.afterUnlock(AbstractMonitoredLock.java:59)
    at org.apache.doris.common.lock.MonitoredReentrantLock.unlock(MonitoredReentrantLock.java:59)
    at org.apache.doris.datasource.InternalCatalog.unlock(InternalCatalog.java:370)
    at org.apache.doris.datasource.InternalCatalog.createDb(InternalCatalog.java:443)
    at org.apache.doris.catalog.Env.createDb(Env.java:3150)
    at org.apache.doris.qe.DdlExecutor.execute(DdlExecutor.java:168)
    at org.apache.doris.qe.StmtExecutor.handleDdlStmt(StmtExecutor.java:3066)
    at org.apache.doris.qe.StmtExecutor.executeByLegacy(StmtExecutor.java:1059)
    at org.apache.doris.qe.StmtExecutor.execute(StmtExecutor.java:644)
    at org.apache.doris.qe.StmtExecutor.queryRetry(StmtExecutor.java:562)
    at org.apache.doris.qe.StmtExecutor.execute(StmtExecutor.java:552)
    at org.apache.doris.qe.ConnectProcessor.executeQuery(ConnectProcessor.java:385)
    at org.apache.doris.qe.ConnectProcessor.handleQuery(ConnectProcessor.java:237)
    at org.apache.doris.qe.MysqlConnectProcessor.handleQuery(MysqlConnectProcessor.java:272)
    at org.apache.doris.qe.MysqlConnectProcessor.dispatch(MysqlConnectProcessor.java:300)
    at org.apache.doris.qe.MysqlConnectProcessor.processOnce(MysqlConnectProcessor.java:359)
    at org.apache.doris.mysql.ReadListener.lambda$handleEvent$0(ReadListener.java:52)
    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
    at java.base/java.lang.Thread.run(Thread.java:833)
```

Configuration Parameters

The following are the slow lock detection-related parameters in the `fe.conf` configuration file:

Parameter Name	Description	Default Value
max_lock_hold_threshold_seconds	Threshold for slow lock warnings (in seconds)	10

Tablet Local Debug

Tablet Local Debug

During the online operation of Doris, various bugs may occur due to various reasons. For example: the replica is inconsistent, the data exists in the version diff, etc.

At this time, it is necessary to copy the copy data of the tablet online to the local environment for reproduction, and then locate the problem.

1. Get information about the tablet

The tablet id can be confirmed by the BE log, and then the information can be obtained by the following command (assuming the tablet id is 10020).

Get information such as DbId/TableId/PartitionId where the tablet is located.

```
mysql> show tablet 10020\G
*************************** 1. row ***************************
       DbName: default_cluster:db1
    TableName: tbl1
PartitionName: tbl1
    IndexName: tbl1
         DbId: 10004
      TableId: 10016
  PartitionId: 10015
      IndexId: 10017
       IsSync: true
        Order: 1
    DetailCmd: SHOW PROC '/dbs/10004/10016/partitions/10015/10017/10020';
```

Execute `DetailCmd` in the previous step to obtain information such as BackendId/SchemaHash.

```
mysql>  SHOW PROC '/dbs/10004/10016/partitions/10015/10017/10020'\G
*************************** 1. row ***************************
        ReplicaId: 10021
        BackendId: 10003
          Version: 3
LstSuccessVersion: 3
 LstFailedVersion: -1
    LstFailedTime: NULL
       SchemaHash: 785778507
    LocalDataSize: 780
   RemoteDataSize: 0
         RowCount: 2
            State: NORMAL
            IsBad: false
     VersionCount: 3
         PathHash: 7390150550643804973
          MetaUrl: http://192.168.10.1:8040/api/meta/header/10020
 CompactionStatus: http://192.168.10.1:8040/api/compaction/show?tablet_id=10020
```

Create tablet snapshot and get table creation statement

```
mysql> admin copy tablet 10020 properties("backend_id" = "10003", "version" = "2")\G
*************************** 1. row ***************************
         TabletId: 10020
        BackendId: 10003
               Ip: 192.168.10.1
             Path: /path/to/be/storage/snapshot/20220830101353.2.3600
ExpirationMinutes: 60
  CreateTableStmt: CREATE TABLE `tbl1` (
  `k1` int(11) NULL,
  `k2` int(11) NULL
) ENGINE=OLAP
DUPLICATE KEY(`k1`, `k2`)
DISTRIBUTED BY HASH(k1) BUCKETS 1
PROPERTIES (
"replication_num" = "1",
"version_info" = "2"
);
```

The `admin copy tablet` command can generate a snapshot file of the corresponding replica and version for the specified tablet. Snapshot files are stored in the `Path` directory of the BE node indicated by the `Ip` field.

There will be a directory named tablet id under this directory, which will be packaged as a whole for later use. (Note that the directory is kept for a maximum of 60 minutes, after which it is automatically deleted).

```
cd /path/to/be/storage/snapshot/20220830101353.2.3600
tar czf 10020.tar.gz 10020/
```

The command will also generate the table creation statement corresponding to the tablet at the same time. Note that this table creation statement is not the original table creation statement, its bucket number and replica number are both 1, and the `versionInfo` field is specified. This table building statement is used later when loading the tablet locally.

So far, we have obtained all the necessary information, the list is as follows:

1. Packaged tablet data, such as 10020.tar.gz.
2. Create a table statement.

2. Load Tablet locally

1. Build a local debugging environment

   Deploy a single-node Doris cluster (1FE, 1BE) locally, and the deployment version is the same as the online cluster. If the online deployment version is DORIS-1.1.1, the local environment also deploys the DORIS-1.1.1 version.

2. Create a table

   Create a table in the local environment using the create table statement from the previous step.

3. Get the tablet information of the newly created table

   Because the number of buckets and replicas of the newly created table is 1, there will only be one tablet with one replica:

```
mysql> show tablets from tbl1\G
*************************** 1. row ***************************
              TabletId: 10017
             ReplicaId: 10018
             BackendId: 10003
            SchemaHash: 44622287
               Version: 1
      LstSuccessVersion: 1
       LstFailedVersion: -1
          LstFailedTime: NULL
          LocalDataSize: 0
         RemoteDataSize: 0
               RowCount: 0
                  State: NORMAL
LstConsistencyCheckTime: NULL
           CheckVersion: -1
           VersionCount: -1
               PathHash: 7390150550643804973
                MetaUrl: http://192.168.10.1:8040/api/meta/header/10017
       CompactionStatus: http://192.168.10.1:8040/api/compaction/show?tablet_id=10017
```

```
mysql> show tablet 10017\G
*************************** 1. row ***************************
       DbName: default_cluster:db1
    TableName: tbl1
PartitionName: tbl1
    IndexName: tbl1
         DbId: 10004
      TableId: 10015
  PartitionId: 10014
      IndexId: 10016
       IsSync: true
        Order: 0
    DetailCmd: SHOW PROC '/dbs/10004/10015/partitions/10014/10016/10017';
```

Here we will record the following information:

- TableId
- PartitionId
- TabletId
- SchemaHash

At the same time, we also need to go to the data directory of the BE node in the debugging environment to confirm the shard id where the new tablet is located:

```
cd /path/to/storage/data/*/10017 && pwd
```

This command will enter the directory where the tablet 10017 is located and display the path. Here we will see a path similar to the following:

```
/path/to/storage/data/0/10017
```

where `0` is the shard id.

4. Modify Tablet Data

Unzip the tablet data package obtained in the first step. The editor opens the 10017.hdr.json file, and modifies the following fields to the information obtained in the previous step:

```
"table_id":10015
"partition_id":10014
"tablet_id":10017
"schema_hash":44622287
"shard_id":0
```

5. Load the tablet

   First, stop the debug environment's BE process (./bin/stop_be.sh). Then copy all the .dat files in the same level directory of the 10017.hdr.json file to the `/path/to/storage/data/0/10017/44622287` directory. This directory is the directory where the debugging environment tablet we obtained in step 3 is located. `10017/44622287` are the tablet id and schema hash respectively.

   Delete the original tablet meta with the `meta_tool` tool. The tool is located in the `be/lib` directory.

   ```
   ./lib/meta_tool --root_path=/path/to/storage --operation=delete_meta --tablet_id=10017 --schema_hash=44622287
   ```

   Where `/path/to/storage` is the data root directory of BE. If the deletion is successful, the delete successfully log will appear.

   Load the new tablet meta via the `meta_tool` tool.

   ```
   ./lib/meta_tool --root_path=/path/to/storage --operation=load_meta --json_meta_path=/path/to/10017.hdr.json
   ```

   If the load is successful, the load successfully log will appear.

6. Verification

   Restart the debug environment's BE process (./bin/start_be.sh). Query the table, if correct, you can query the data of the loaded tablet, or reproduce the online problem.

Tablet metadata management tool

Tablet metadata management tool

Background

In the latest version of the code, we introduced RocksDB in BE to store meta-information of tablet, in order to solve various functional and performance problems caused by storing meta-information through header file. Currently, each data directory (root path) has a corresponding RocksDB instance, in which all tablets on the corresponding root path are stored in the key-value manner.

To facilitate the maintenance of these metadata, we provide an online HTTP interface and an offline meta tool to complete related management operations.

The HTTP interface is only used to view tablet metadata online, and can be used when the BE process is running.

However, meta tool is only used for off-line metadata management operations. BE must be stopped before it can be used.

The meta tool tool is stored in the Lib / directory of BE.

Operation

View Tablet Meta

Viewing Tablet Meta information can be divided into online and offline methods

Online

Access BE's HTTP interface to obtain the corresponding Tablet Meta information:

api:

```
http://{host}:{port}/api/meta/header/{tablet_id}/{schema_hash}
```

> Host: be Hostname
>
> port: BE's HTTP port
>
> tablet id: tablet id
>
> schema hash: tablet schema hash

Give an example:

`http://be_host:8040/api/meta/header/14156/2458238340`

If the final query is successful, the Tablet Meta will be returned as json.

Offline

Get Tablet Meta on a disk based on the meta tool tool.

Command:

```
./lib/meta_tool --root_path=/path/to/root_path --operation=get_meta --tablet_id=xxx --schema_hash=xxx
```

> root_path: The corresponding root_path path path configured in be.conf.

The result is also a presentation of Tablet Meta in JSON format.

Load header

The function of loading header is provided to realize manual migration of tablet. This function is based on Tablet Meta in JSON format, so if changes in the shard field and version information are involved, they can be changed directly in the JSON content of Tablet Meta. Then use the following commands to load.

Command:

```
./lib/meta_tool --operation=load_meta --root_path=/path/to/root_path --json_meta_path=path
```

Delete header

In order to realize the function of deleting a tablet meta from a disk of a BE. Support single delete and batch delete.

Single delete:

```
./lib/meta_tool --operation=delete_meta --root_path=/path/to/root_path --tablet_id=xxx --schema_hash=xxx`
```

Batch delete:

```
./lib/meta_tool --operation=batch_delete_meta --tablet_file=/path/to/tablet_file.txt
```

Each line in `tablet_file.txt` represents the information of a tablet. The format is:

`root_path,tablet_id,schema_hash`

Each column are separated by comma.

`tablet_file` example:

```
/output/be/data/,14217,352781111
/output/be/data/,14219,352781111
/output/be/data/,14223,352781111
/output/be/data/,14227,352781111
/output/be/data/,14233,352781111
/output/be/data/,14239,352781111
```

Batch delete will skip the line with incorrect tablet information format in `tablet_file`. And after the execution is completed, the number of successful deletions and the number of errors are displayed.

TabletMeta in Pb format

This command is to view the old file-based management PB format Tablet Meta, and to display Tablet Meta in JSON format.

Command:

```
./lib/meta_tool --operation=show_meta --root_path=/path/to/root_path --pb_header_path=path
```

Segment meta in Pb format

This command is to view the PB format segment meta, and to display segment meta in JSON format.

Command:

```
./meta_tool --operation=show_segment_footer --file=/path/to/segment/file
```

Repairing Data

Repairing Data

For the Unique Key Merge on Write table, there are bugs in some Doris versions, which may cause errors when the system calculates the delete bitmap, resulting in duplicate primary keys. At this time, the full compaction function can be used to repair the data. This function is invalid for non-Unique Key Merge on Write tables.

This feature requires Doris version 2.0+.

To use this function, it is necessary to stop the import as much as possible, otherwise problems such as import timeout may occur.

Brief principle explanation

After the full compaction is executed, the delete bitmap will be recalculated, and the wrong delete bitmap data will be deleted to complete the data restoration.

Instructions for use

`POST /api/compaction/run?tablet_id={int}&compact_type=full`

or

`POST /api/compaction/run?table_id={int}&compact_type=full`

Note that only one tablet_id and table_id can be specified, and cannot be specified at the same time. After specifying table_id, full_compaction will be automatically executed for all tablets under this table.

Example of use

```
curl -X POST "http://127.0.0.1:8040/api/compaction/run?tablet_id=10015&compact_type=full"
curl -X POST "http://127.0.0.1:8040/api/compaction/run?table_id=10104&compact_type=full"
```

Overview | Open Api

Overview

As a supplement to Apache Doris operation and maintenance management, OPEN API is mainly used by database administrators to perform some management operations.

> **The OPEN API is currently unstable and is only recommended for developers to test and use. We may change the interface behavior in subsequent versions.**
>
> In a production environment, it is recommended to use SQL commands to complete operations.

Security certification

The security authentication of FE BE API can be enabled through the following configuration:

Configuration	Configuration File	Default Value	Description
`enable_all_http_auth`	`be.conf`	`false`	Enable authentication for BE HTTP port (default 8040). After enabling, access to BE's HTTP API requires ADMIN user login.
`enable_brpc_builtin_services`	`be.conf`	true	Whether to open brpc built-in service to the outside world (default is 8060). If disabled, HTTP port 8060 will be inaccessible. (Supported since version 2.1.7)
`enable_all_http_auth`	`fe.conf`	`false`	Enable authentication for the FE HTTP port (default 8030). After enabling, access to the FE HTTP API requires corresponding user permissions.

> **NOTE**
>
> The permission requirements for the HTTP API of FE and BE vary from version to version. Please refer to the corresponding API documentation for details.

Config Action

Config Action

Request

```
GET /rest/v1/config/fe/
```

Description

Config Action is used to obtain current FE configuration information.

Path parameters

None

Query parameters

- `conf_item`

  Optional parameters. Return specified item in FE configuration.

Request body

None

Response


```
{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": ["Name", "Value"],
        "rows": [{
            "Value": "DAY",
            "Name": "sys_log_roll_interval"
        }, {
            "Value": "23",
            "Name": "consistency_check_start_time"
        }, {
            "Value": "4096",
            "Name": "max_mysql_service_task_threads_num"
        }, {
            "Value": "1000",
            "Name": "max_unfinished_load_job"
        }, {
            "Value": "100",
            "Name": "max_routine_load_job_num"
        }, {
```

```
            "Value": "SYNC",
            "Name": "master_sync_policy"
        }]
    },
    "count": 0
}
```

The returned result is the same as `System Action`. Is a description of the table.

HA Action

HA Action

Request

```
GET /rest/v1/ha
```

Description

HA Action is used to obtain the high availability group information of the FE cluster.

Path parameters

None

Query parameters

None

Request body

None

Response


```
{
    "msg": "success",
    "code": 0,
    "data": {
        "Observernodes": [],
        "CurrentJournalId": [{
            "Value": 433648,
            "Name": "FrontendRole"
        }],
        "Electablenodes": [{
            "Value": "host1",
            "Name": "host1"
        }],
        "allowedFrontends": [{
            "Value": "name: 192.168.1.1_9213_1597652404352, role: FOLLOWER, 192.168.1.1:9213",
            "Name": "192.168.1.1_9213_1597652404352"
        }],
        "removedFrontends": [],
        "CanRead": [{
            "Value": true,
            "Name": "Status"
        }],
        "databaseNames": [{
```

```
            "Value": "433436 ",
            "Name": "DatabaseNames"
        }],
        "FrontendRole": [{
            "Value": "MASTER",
            "Name": "FrontendRole"
        }],
        "CheckpointInfo": [{
            "Value": 433435,
            "Name": "Version"
        }, {
            "Value": "2020-09-03T02:07:37.000+0000",
            "Name": "lastCheckPointTime"
        }]
    },
    "count": 0
}
```

Hardware Info Action

Hardware Info Action

Request

```
GET /rest/v1/hardware_info/fe/
```

Description

Hardware Info Action is used to obtain the hardware information of the current FE.

Path parameters

None

Query parameters

None

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "VersionInfo": {
            "Git": "git://host/core@5bc28f4c36c20c7b424792df662fc988436e679e",
            "Version": "trunk",
            "BuildInfo": "cmy@192.168.1",
            "BuildTime": "Tuesday, 05 September 2019 11:07:42 CST"
        },
        "HardwareInfo": {
            "NetworkParameter": "...",
            "Processor": "...",
            "OS": "...",
            "Memory": "...",
            "FileSystem": "...",
            "NetworkInterface": "...",
            "Processes": "...",
            "Disk": "..."
        }
    },
    "count": 0
}
```

- The contents of each value in the `HardwareInfo` field are all hardware information text displayed in html format.

Help Action

Help Action

Request

`GET /rest/v1/help`

Description

Used to obtain help through fuzzy query.

Path parameters

None

Query parameters

- `query`

  Keywords to be matched, such as array and select.

Request body

None

Response


```
{
    "msg":"success",
    "code":0,
    "data":{"fuzzy":"No Fuzzy Matching Topic","matching":"No Matching Category"},
    "count":0
}
```

Log Action

Log Action

Request

```
GET /rest/v1/log
```

Description

GET is used to obtain the latest part of Doris's WARNING log, and the POST method is used to dynamically set the log level of FE.

Path parameters

None

Query parameters

- `add_verbose`

  Optional parameters for the POST method. Enable the DEBUG level log of the specified package.

- `del_verbose`

  Optional parameters for the POST method. Turn off the DEBUG level log of the specified package.

Request body

None

Response

```
GET /rest/v1/log

{
    "msg": "success",
    "code": 0,
    "data": {
        "LogContents": {
            "logPath": "/home/disk1/cmy/git/doris/core-for-ui/output/fe/log/fe.warn.log",
            "log": "<pre>2020-08-26 15:54:30,081 WARN (UNKNOWN 10.81.85.89_9213_1597652404352(-1)|1)
[Catalog.notifyNewFETypeTransfer():2356] notify new FE type transfer: UNKNOWN</br>2020-08-26 15:54:32,089 WARN (RepNode
10.81.85.89_9213_1597652404352(-1)|61) [Catalog.notifyNewFETypeTransfer():2356] notify new FE type transfer: MASTER</br>2020-08-26
15:54:35,121 WARN (stateListener|73) [Catalog.replayJournal():2510] replay journal cost too much time: 2975 replayedJournalId:
232383</br>2020-08-26 15:54:48,117 WARN (leaderCheckpointer|75) [Catalog.replayJournal():2510] replay journal cost too much time:
2812 replayedJournalId: 232383</br></pre>",
            "showingLast": "603 bytes of log"
```

```
        },
        "LogConfiguration": {
            "VerboseNames": "org",
            "AuditNames": "slow_query,query",
            "Level": "INFO"
        }
    },
    "count": 0
}
```

Among them, `data.LogContents.log` means the log content in the latest part of `fe.warn.log`.

```
POST /rest/v1/log?add_verbose=org

{
    "msg": "success",
    "code": 0,
    "data": {
        "LogConfiguration": {
            "VerboseNames": "org",
            "AuditNames": "slow_query,query",
            "Level": "INFO"
        }
    },
    "count": 0
}
```

Login Action

Login Action

Request

`POST /rest/v1/login`

Description

Used to log in to the service.

Path parameters

None

Query parameters

None

Request body

None

Response

- Login success

```
{
    "msg": "Login success!",
    "code": 200
}
```

- Login failure

```
{
    "msg": "Error msg...",
    "code": xxx,
    "data": "Error data...",
    "count": 0
}
```

Logout Action

Logout Action

Request

```
POST /rest/v1/logout
```

Description

Logout Action is used to log out of the current login.

Path parameters

None

Query parameters

None

Request body

None

Response

```
{
    "msg": "OK",
    "code": 0
}
```

Query Profile Action | Fe Http

Query Profile Action

Request

```
GET /rest/v1/query_profile/<query_id>
```

Description

The Query Profile Action is used to obtain the Query profile.

Path parameters

- `<query_id>`

  Optional parameters. When not specified, the latest query list is returned. When specified, return the profile of the specified query.

Query parameters

None

Request body

None

Response

- Not specify `<query_id>`

```
GET /rest/v1/query_profile/
{
    "msg": "success",
    "code": 0,
    "data": {
        "href_column": ["Query ID"],
        "column_names": ["Query ID", "User", "Default Db", "Sql Statement", "Query Type", "Start Time", "End Time",
"Total", "Query State"],
        "rows": [{
            "User": "root",
            "__hrefPath": ["/query_profile/d73a8a0b004f4b2f-b4829306441913da"],
            "Query Type": "Query",
            "Total": "5ms",
            "Default Db": "default_cluster:db1",
            "Sql Statement": "select * from tbl1",
            "Query ID": "d73a8a0b004f4b2f-b4829306441913da",
            "Start Time": "2020-09-03 10:07:54",
            "Query State": "EOF",
```

```
            "End Time": "2020-09-03 10:07:54"
        }, {
            "User": "root",
            "__hrefPath": ["/query_profile/fd706dd066824c21-9d1a63af9f5cb50c"],
            "Query Type": "Query",
            "Total": "6ms",
            "Default Db": "default_cluster:db1",
            "Sql Statement": "select * from tbl1",
            "Query ID": "fd706dd066824c21-9d1a63af9f5cb50c",
            "Start Time": "2020-09-03 10:07:54",
            "Query State": "EOF",
            "End Time": "2020-09-03 10:07:54"
        }]
    },
    "count": 3
}
```

The returned result is the same as `System Action`, which is a table description.

- Specify `<query_id>`

```
GET /rest/v1/query_profile/<query_id>

{
    "msg": "success",
    "code": 0,
    "data": "Query:</br>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;Summary:</br>...",
    "count": 0
}
```

`data` is the text content of the profile.

Session Action

Session Action

Request

`GET /rest/v1/session`

`GET /rest/v1/session/all`

Description

Session Action is used to obtain the current session information.

Path parameters

None

Query parameters

None

Request body

None

Obtain the current session information

`GET /rest/v1/session`

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": ["Id", "User", "Host", "Cluster", "Db", "Command", "Time", "State", "Info"],
        "rows": [{
            "User": "root",
            "Command": "Sleep",
            "State": "",
            "Cluster": "default_cluster",
            "Host": "10.81.85.89:31465",
            "Time": "230",
            "Id": "0",
            "Info": "",
            "Db": "db1"
        }]
    },
```

```
    "count": 2
}
```

Obtain all FE session information

```
GET /rest/v1/session/all
```

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": ["FE", "Id", "User", "Host", "Cluster", "Db", "Command", "Time", "State", "Info"],
        "rows": [{
            "FE": "10.14.170.23",
            "User": "root",
            "Command": "Sleep",
            "State": "",
            "Cluster": "default_cluster",
            "Host": "10.81.85.89:31465",
            "Time": "230",
            "Id": "0",
            "Info": "",
            "Db": "db1"
        },
        {
            "FE": "10.14.170.24",
            "User": "root",
            "Command": "Sleep",
            "State": "",
            "Cluster": "default_cluster",
            "Host": "10.81.85.88:61465",
            "Time": "460",
            "Id": "1",
            "Info": "",
            "Db": "db1"
        }]
    },
    "count": 2
}
```

The returned result is the same as `System Action`. Is a description of the table.

System Action

System Action

Request

```
GET /rest/v1/system
```

Description

System Action is used for information about the Proc system built in Doris.

Path parameters

None

Query parameters

- `path`

  Optional parameter, specify the path of proc

Request body

None

Response

Take `/dbs/10003/10054/partitions/10053/10055` as an example:


```
{
    "msg": "success",
    "code": 0,
    "data": {
        "href_columns": ["TabletId", "MetaUrl", "CompactionStatus"],
        "column_names": ["TabletId", "ReplicaId", "BackendId", "SchemaHash", "Version", "VersionHash", "LstSuccessVersion",
"LstSuccessVersionHash", "LstFailedVersion", "LstFailedVersionHash", "LstFailedTime", "DataSize", "RowCount", "State",
"LstConsistencyCheckTime", "CheckVersion", "CheckVersionHash", "VersionCount", "PathHash", "MetaUrl", "CompactionStatus"],
        "rows": [{
            "SchemaHash": "1294206575",
            "LstFailedTime": "\\N",
            "LstFailedVersion": "-1",
            "MetaUrl": "URL",
            "__hrefPaths": ["http://192.168.100.100:8030/rest/v1/system?path=/dbs/10003/10054/partitions/10053/10055/10056",
"http://192.168.100.100:8043/api/meta/header/10056", "http://192.168.100.100:8043/api/compaction/show?tablet_id=10056"],
            "CheckVersionHash": "-1",
            "ReplicaId": "10057",
            "VersionHash": "4611804212003004639",
            "LstConsistencyCheckTime": "\\N",
```

```
            "LstSuccessVersionHash": "4611804212003004639",
            "CheckVersion": "-1",
            "Version": "6",
            "VersionCount": "2",
            "State": "NORMAL",
            "BackendId": "10032",
            "DataSize": "776",
            "LstFailedVersionHash": "0",
            "LstSuccessVersion": "6",
            "CompactionStatus": "URL",
            "TabletId": "10056",
            "PathHash": "-3259732870068082628",
            "RowCount": "21"
        }]
    },
    "count": 1
}
```

The `column_names` in the data part is the header information, and `href_columns` indicates which columns in the table are hyperlink columns. Each element in the `rows` array represents a row. Among them, `__hrefPaths` is not the table data, but the link URL of the hyperlink column, which corresponds to the column in `href_columns` one by one.

Colocate Meta Action

Colocate Meta Action

Request

`GET /api/colocate` `POST/DELETE /api/colocate/group_stable` `POST /api/colocate/bucketseq`

Description

Used to obtain or modify colocate group information.

Path parameters

None

Query parameters

- `db_id`

  Specify database id

- `group_id`

  Specify group id

Request body

None

Response

TO DO

Meta Action

Meta Action

Request

```
GET /image
GET /info
GET /version
GET /put
GET /journal_id
GET /role
GET /check
GET /dump
```

Description

This is a set of APIs related to FE metadata, except for `/dump`, they are all used for internal communication between FE nodes.

Path parameters

TODO

Query parameters

TODO

Request body

TODO

Response

TODO

Examples

TODO

Cluster Action

Cluster Action

Request

`GET /rest/v2/manager/cluster/cluster_info/conn_info`

Cluster Connection Information

`GET /rest/v2/manager/cluster/cluster_info/conn_info`

Description

Used to get cluster http, mysql connection information.

Path parameters

None

Query parameters

None

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "http": [
            "fe_host:http_ip"
        ],
        "mysql": [
            "fe_host:query_ip"
        ]
    },
    "count": 0
}
```

Examples

```
GET /rest/v2/manager/cluster/cluster_info/conn_info

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "http": [
            "127.0.0.1:8030"
        ],
        "mysql": [
            "127.0.0.1:9030"
        ]
    },
    "count": 0
}
```

Node Action

Request

GET /rest/v2/manager/node/frontends

GET /rest/v2/manager/node/backends

GET /rest/v2/manager/node/brokers

GET /rest/v2/manager/node/configuration_name

GET /rest/v2/manager/node/node_list

POST /rest/v2/manager/node/configuration_info

POST /rest/v2/manager/node/set_config/fe

POST /rest/v2/manager/node/set_config/be

POST /rest/v2/manager/node/{action}/be

POST /rest/v2/manager/node/{action}/fe

Get information about fe, be, broker nodes

GET /rest/v2/manager/node/frontends

GET /rest/v2/manager/node/backends

GET /rest/v2/manager/node/brokers

Description

Used to get cluster to get fe, be, broker node information.

Response

```
frontends:
{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": [
            "Name",
            "IP",
            "HostName",
            "EditLogPort",
            "HttpPort",
            "QueryPort",
            "RpcPort",
            "ArrowFlightSqlPort",
```

```
            "Role",
            "IsMaster",
            "ClusterId",
            "Join",
            "Alive",
            "ReplayedJournalId",
            "LastHeartbeat",
            "IsHelper",
            "ErrMsg",
            "Version"
        ],
        "rows": [
            [
                ...
            ]
        ]
    },
    "count": 0
}
```

```
backends:
{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": [
            "BackendId",
            "Cluster",
            "IP",
            "HostName",
            "HeartbeatPort",
            "BePort",
            "HttpPort",
            "BrpcPort",
            "LastStartTime",
            "LastHeartbeat",
            "Alive",
            "SystemDecommissioned",
            "ClusterDecommissioned",
            "TabletNum",
            "DataUsedCapacity",
            "AvailCapacity",
            "TotalCapacity",
            "UsedPct",
            "MaxDiskUsedPct",
            "ErrMsg",
            "Version",
            "Status"
        ],
        "rows": [
            [
                ...
            ]
        ]
    },
    "count": 0
}
```

```
brokers:
{
    "msg": "success",
    "code": 0,
```

```
    "data": {
        "column_names": [
            "Name",
            "IP",
            "HostName",
            "Port",
            "Alive",
            "LastStartTime",
            "LastUpdateTime",
            "ErrMsg"
        ],
        "rows": [
            [
                ...
            ]
        ]
    },
    "count": 0
}
```

Get node configuration information

`GET /rest/v2/manager/node/configuration_name`

`GET /rest/v2/manager/node/node_list`

`POST /rest/v2/manager/node/configuration_info`

Description

configuration_name Used to get the name of the node configuration item.
node_list Get the list of nodes.
configuration_info to get the node configuration details.

Query parameters

`GET /rest/v2/manager/node/configuration_name`
none

`GET /rest/v2/manager/node/node_list`
none

`POST /rest/v2/manager/node/configuration_info`

- type The value is fe or be, which specifies to get the configuration information of fe or the configuration information of be.

Request body

`GET /rest/v2/manager/node/configuration_name`
none

`GET /rest/v2/manager/node/node_list`
none

POST /rest/v2/manager/node/configuration_info

```
{
    "conf_name": [
        ""
    ],
    "node": [
        ""
    ]
}

If no body is included, the parameters in the body use the default values.
conf_name specifies which configuration items to return, the default is all configuration items.
node is used to specify which node's configuration information is returned, the default is all fe nodes or be nodes configuration
information.
```

Response

GET /rest/v2/manager/node/configuration_name

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "backend":[
            ""
        ],
        "frontend":[
            ""
        ]
    },
    "count": 0
}
```

GET /rest/v2/manager/node/node_list

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "backend": [
            ""
        ],
        "frontend": [
            ""
        ]
    },
    "count": 0
}
```

POST /rest/v2/manager/node/configuration_info?type=fe

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": [
            "Configuration Item",
```

```
            "Node",
            "Node Type",
            "Configuration Value Type",
            "MasterOnly",
            "Configuration Value",
            "Modifiable"
        ],
        "rows": [
            [
                ""
            ]
        ]
    },
    "count": 0
}
```

POST /rest/v2/manager/node/configuration_info?type=be

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": [
            "Configuration Item",
            "Node",
            "Node Type",
            "Configuration Value Type",
            "Configuration Value",
            "Modifiable"
        ],
        "rows": [
            [
                ""
            ]
        ]
    },
    "count": 0
}
```

Examples

1. Get the fe agent_task_resend_wait_time_ms configuration information:

   POST /rest/v2/manager/node/configuration_info?type=fe
   body:

   ```
   {
       "conf_name":[
           "agent_task_resend_wait_time_ms"
       ]
   }
   ```

   Response:

   ```
   {
   "msg": "success",
   "code": 0,
   "data": {
   ```

```
    "column_names": [
        "Configuration Item",
        "Node",
        "Node Type",
        "Configuration Value Type",
        "MasterOnly",
        "Configuration Value",
        "Modifiable"
    ],
    "rows": [
        [
            "agent_task_resend_wait_time_ms",
            "127.0.0.1:8030",
            "FE",
            "long",
            "true",
            "50000",
            "true"
        ]
    ]
},
"count": 0
}
```

Modify configuration values

`POST /rest/v2/manager/node/set_config/fe`

`POST /rest/v2/manager/node/set_config/be`

Description

Used to modify fe or be node configuration values

Request body

```
{
    "config_name":{
        "node":[
            ""
        ],
        "value":"",
        "persist":
    }
}

config_name is the corresponding configuration item.
node is a keyword indicating the list of nodes to be modified;
value is the value of the configuration.
persist is true for permanent modification and false for temporary modification. persist means permanent modification, false means
temporary modification. permanent modification takes effect after reboot, temporary modification fails after reboot.
```

Response

`GET /rest/v2/manager/node/configuration_name`

```
{
    "msg": "",
    "code": 0,
    "data": {
        "failed":[
            {
                "config_name":"name",
                "value="",
                "node":"",
                "err_info":""
            }
        ]
    },
    "count": 0
}

failed Indicates a configuration message that failed to be modified.
```

Examples

1. Modify the agent_task_resend_wait_time_ms and alter_table_timeout_second configuration values in the fe 127.0.0.1:8030 node:

   POST /rest/v2/manager/node/set_config/fe body:

   ```
   {
       "agent_task_resend_wait_time_ms":{
           "node":[
               "127.0.0.1:8030"
           ],
           "value":"10000",
           "persist":"true"
       },
       "alter_table_timeout_second":{
           "node":[
               "127.0.0.1:8030"
           ],
           "value":"true",
           "persist":"true"
       }
   }
   ```

   Response: ``` { “msg”: “success”, “code”: 0, “data”: { “failed”: [{ “config_name”: “alter_table_timeout_second”, “node”: “10.81.85.89:8837”, “err_info”: “Unsupported configuration value type.”, “value”: “true” }] }, “count”: 0 }

   gent_task_resend_wait_time_ms configuration value modified successfully, alter_table_timeout_second modification failed. ```

Operate be node

```
POST /rest/v2/manager/node/{action}/be
```

Description

Used to add/drop/offline be node

action：ADD/DROP/DECOMMISSION

Request body

```
{
    "hostPorts": ["127.0.0.1:9050"],
    "properties": {
        "tag.location": "test"
    }
}

hostPorts A set of be node addresses to be operated, ip:heartbeat_port
properties The configuration passed in when adding a node is only used to configure the tag. If not, the default tag is used
```

Response

```
{
    "msg": "Error",
    "code": 1,
    "data": "errCode = 2, detailMessage = Same backend already exists[127.0.0.1:9050]",
    "count": 0
}

msg Success/Error
code 0/1
data ""/Error message
```

Examples

1. add be node

   post /rest/v2/manager/node/ADD/be Request body

   ```
   {
       "hostPorts": ["127.0.0.1:9050"]
   }
   ```

   Response

   ```
   {
       "msg": "success",
       "code": 0,
       "data": null,
       "count": 0
   }
   ```

2. drop be node

   post /rest/v2/manager/node/DROP/be Request body

```
{
    "hostPorts": ["127.0.0.1:9050"]
}
```

Response

```
{
    "msg": "success",
    "code": 0,
    "data": null,
    "count": 0
}
```

3. offline be node

   post /rest/v2/manager/node/DECOMMISSION/be Request body `json { "hostPorts": ["127.0.0.1:9050"] }`

   Response `json { "msg": "success", "code": 0, "data": null, "count": 0 }`

Operate fe node

`POST /rest/v2/manager/node/{action}/fe`

Description

Used to add/drop fe node

action：ADD/DROP

Request body

```
{
    "role": "FOLLOWER",
    "hostPort": "127.0.0.1:9030"
}

role FOLLOWER/OBSERVER
hostPort The address of the fe node to be operated, ip:edit_log_port
```

Response

```
{
    "msg": "Error",
    "code": 1,
    "data": "errCode = 2, detailMessage = frontend already exists name: 127.0.0.1:9030_1670495889415, role: FOLLOWER,
127.0.0.1:9030",
    "count": 0
}

msg Success/Error
```

```
code 0/1
data ""/Error message
```

Examples

1. add FOLLOWER node

   post /rest/v2/manager/node/ADD/fe Request body `json { "role": "FOLLOWER", "hostPort": "127.0.0.1:9030" }`

   Response `json { "msg": "success", "code": 0, "data": null, "count": 0 }`

2. drop FOLLOWER node

   post /rest/v2/manager/node/DROP/fe Request body `json { "role": "FOLLOWER", "hostPort": "127.0.0.1:9030" }`

   Response `json { "msg": "success", "code": 0, "data": null, "count": 0 }`

Query Profile Action | Fe Http

Query Profile Action

Request

```
GET /rest/v2/manager/query/query_info

GET /rest/v2/manager/query/trace_id/{trace_id}

GET /rest/v2/manager/query/sql/{query_id}

GET /rest/v2/manager/query/profile/text/{query_id}

GET /rest/v2/manager/query/profile/graph/{query_id}

GET /rest/v2/manager/query/profile/json/{query_id}

GET /rest/v2/manager/query/profile/fragments/{query_id}

GET /rest/v2/manager/query/current_queries

GET /rest/v2/manager/query/kill/{query_id}
```

Get the query information

```
GET /rest/v2/manager/query/query_info
```

Description

Gets information about select queries for all fe nodes in the cluster.

Query parameters

- `query_id`

  Optional, specifies the query ID of the query to be returned, default returns information for all queries.

- `search`

  Optional, specifies that query information containing strings is returned, currently only string matches are performed.

- `is_all_node`

  Optional, if true, returns query information for all fe nodes, if false, returns query information for the current fe node. The default is true.

Response

```
{
   "msg": "success",
    "code": 0,
    "data": {
        "column_names": [
            "Query ID",
            "FE Node",
            "Query User",
            "Execution Database",
            "Sql",
            "Query Type",
            "Start Time",
            "End Time",
            "Execution Duration",
            "Status"
        ],
        "rows": [
            [
                ...
            ]
        ]
    },
    "count": 0
}
```

> **Note**
>
> Since Doris Version 1.2, Admin and Root users can view all queries. Regular users can only view their own submitted queries.

Examples

```
GET /rest/v2/manager/query/query_info

{
    "msg": "success",
    "code": 0,
    "data": {
        "column_names": [
            "Query ID",
            "FE Node",
            "Query User",
            "Execution Database",
            "Sql",
            "Query Type",
            "Start Time",
            "End Time",
            "Execution Duration",
            "Status"
        ],
        "rows": [
            [
                "d7c93d9275334c35-9e6ac5f295a7134b",
                "127.0.0.1:8030",
                "root",
```

```
                "default_cluster:testdb",
                "select c.id, c.name, p.age, p.phone, c.date, c.cost from cost c join people p on c.id = p.id where p.age > 20 order
by c.id",
                "Query",
                "2021-07-29 16:59:12",
                "2021-07-29 16:59:12",
                "109ms",
                "EOF"
            ]
        ]
    },
    "count": 0
}
```

Get Query Id By Trace Id

`GET /rest/v2/manager/query/trace_id/{trace_id}`

Description

Get query id by trance id.

Before executing a Query, set a unique trace id:

```
set session_context="trace_id:your_trace_id";
```

After executing the Query within the same Session, the query id can be obtained through the trace id.

Path parameters

- `{trace_id}`

  User specific trace id.

Query parameters

Response

```
{
    "msg": "success",
    "code": 0,
    "data": "fb1d9737de914af1-a498d5c5dec638d3",
    "count": 0
}
```

> **Info**
>
> Since Doris version 1.2, admin and root user can view all queries. Ordinary users can only view the Query sent by themselves. If the specified trace id does not exist or has no permission, it will return Bad Request:

```
{
    "msg": "Bad Request",
    "code": 403,
    "data": "error messages",
    "count": 0
}
```

Get the sql and text profile for the specified query

`GET /rest/v2/manager/query/sql/{query_id}`

`GET /rest/v2/manager/query/profile/text/{query_id}`

Description

Get the sql and profile text for the specified query id.

Path parameters

- `query_id`

  The query id.

Query parameters

- `is_all_node`

  Optional, if true then query for the specified query id in all fe nodes, if false then query for the specified query id in the currently connected fe nodes. The default is true.

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "sql": ""
    },
    "count": 0
}
```

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "profile": ""
    },
    "count": 0
}
```

Info

Since Doris version 1.2, admin and root user can view all queries. Ordinary users can only view the Query sent by themselves. If the specified trace id does not exist or has no permission, it will return Bad Request:

```
{
    "msg": "Bad Request",
    "code": 403,
    "data": "error messages",
    "count": 0
}
```

Examples

1. get sql.

```
GET /rest/v2/manager/query/sql/d7c93d9275334c35-9e6ac5f295a7134b

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "sql": "select c.id, c.name, p.age, p.phone, c.date, c.cost from cost c join people p on c.id = p.id where p.age >
20 order by c.id"
    },
    "count": 0
}
```

Get the specified query fragment and instance information

`GET /rest/v2/manager/query/profile/fragments/{query_id}`

Since 2.1.1, this API is deprecated. You can still download profile from http://:/QueryProfile

Description

Get the fragment name, instance id, host ip/port and execution time for the specified query id.

Path parameters

- `query_id`

  The query id.

Query parameters

- `is_all_node`

  Optional, if true then query for the specified query id in all fe nodes, if false then query for the specified query id in the currently connected fe nodes. The default is true.

Response

```
{
    "msg": "success",
    "code": 0,
    "data": [
        {
            "fragment_id": "",
            "time": "",
            "instance_id": {
                "": {
                  "host": "",
                  "active_time": ""
                }
            }
        }
    ],
    "count": 0
}
```

> **Info**
>
> Since Doris version 1.2, admin and root user can view all queries. Ordinary users can only view the Query sent by themselves. If the specified trace id does not exist or has no permission, it will return Bad Request:
>
> ```
> {
>     "msg": "Bad Request",
>     "code": 403,
>     "data": "error messages",
>     "count": 0
> }
> ```

Examples

```
GET /rest/v2/manager/query/profile/fragments/d7c93d9275334c35-9e6ac5f295a7134b

Response:
{
    "msg": "success",
    "code": 0,
    "data": [
        {
            "fragment_id": "0",
            "time": "36.169ms",
            "instance_id": {
                "d7c93d9275334c35-9e6ac5f295a7134e": {
```

```
                    "host": "172.19.0.4:9060",
                    "active_time": "36.169ms"
                }
            }
        },
        {
            "fragment_id": "1",
            "time": "20.710ms",
            "instance_id": {
                "d7c93d9275334c35-9e6ac5f295a7134c": {
                    "host": "172.19.0.5:9060",
                    "active_time": "20.710ms"
                }
            }
        },
        {
            "fragment_id": "2",
            "time": "7.83ms",
            "instance_id": {
                "d7c93d9275334c35-9e6ac5f295a7134d": {
                    "host": "172.19.0.6:9060",
                    "active_time": "7.83ms"
                },
                "d7c93d9275334c35-9e6ac5f295a7134f": {
                    "host": "172.19.0.7:9060",
                    "active_time": "10.873ms"
                }
            }
        }
    ],
    "count": 0
}
```


Get the specified query id tree profile information

`GET /rest/v2/manager/query/profile/graph/{query_id}`

Description

Get the tree profile information of the specified query id, same as `show query profile` command.

Path parameters

- `query_id`

  The query id.

Query parameters

- `fragment_id` and `instance_id`

  Optional, both parameters must be specified or not.
  If both are not specified, a simple tree of profiles is returned, equivalent to `show query profile '/query_id'`;
  If both are specified, a detailed profile tree is returned, which is equivalent to `show query profile '/query_id/fragment_id/instance_id'`.

- is_all_node

  Optional, if true then query information about the specified query id in all fe nodes, if false then query information about the specified query id in the currently connected fe nodes. The default is true.

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "graph":""
    },
    "count": 0
}
```

> **Info**
>
> Since Doris version 1.2, admin and root user can view all queries. Ordinary users can only view the Query sent by themselves. If the specified trace id does not exist or has no permission, it will return Bad Request:
>
> ```
> {
>     "msg": "Bad Request",
>     "code": 403,
>     "data": "error messages",
>     "count": 0
> }
> ```

Current running queries

`GET /rest/v2/manager/query/current_queries`

Description

Same as `show proc "/current_query_stmts"`, return current running queries.

Path parameters

Query parameters

- is_all_node

  Optional. Return current running queries from all FE if set to true. Default is true.

Response

```
{
    "msg": "success",
```

```
    "code": 0,
    "data": {
        "columnNames": ["Frontend", "QueryId", "ConnectionId", "Database", "User", "ExecTime", "SqlHash", "Statement"],
        "rows": [
            ["172.19.0.3", "108e47ab438a4560-ab1651d16c036491", "2", "", "root", "6074", "1a35f62f4b14b9d7961b057b77c3102f", "select
sleep(60)"],
            ["172.19.0.11", "3606cad4e34b49c6-867bf6862cacc645", "3", "", "root", "9306", "1a35f62f4b14b9d7961b057b77c3102f",
"select sleep(60)"]
        ]
    },
    "count": 0
}
```


Cancel query

`POST /rest/v2/manager/query/kill/{query_id}`

Description

Cancel query of specified connection.

Path parameters

- `{query_id}`

  query id. You can get query id by `trance_id` api.

Query parameters

Response


```
{
    "msg": "success",
    "code": 0,
    "data": null,
    "count": 0
}
```

Backends Action

Backends Action

Request

```
GET /api/backends
```

Description

Backends Action returns the Backends list, including Backend's IP, PORT and other information.

Path parameters

None

Query parameters

- `is_alive`

  Optional parameters. Whether to return the surviving BE nodes. The default is false, which means that all BE nodes are returned.

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "backends": [
            {
                "ip": "192.1.1.1",
                "http_port": 8040,
                "is_alive": true
            }
        ]
    },
    "count": 0
}
```

Bootstrap Action

Bootstrap Action

Request

`GET /api/bootstrap`

Description

It is used to judge whether the FE has started. When no parameters are provided, only whether the startup is successful is returned. If `token` and `cluster_id` are provided, more detailed information is returned.

Path parameters

none

Query parameters

- `cluster_id`

  The cluster id. It can be viewed in the file `doris-meta/image/VERSION`.

- `token`

  Cluster token. It can be viewed in the file `doris-meta/image/VERSION`.

Request body

none

Response

- No parameters provided

  ```
  {
      "msg": "OK",
      "code": 0,
      "data": null,
      "count": 0
  }
  ```

  A code of 0 means that the FE node has started successfully. Error codes other than 0 indicate other errors.

- Provide `token` and `cluster_id`

```
{
    "msg": "OK",
    "code": 0,
    "data": {
        "queryPort": 9030,
        "rpcPort": 9020,
        "arrowFlightSqlPort": 9040,
        "maxReplayedJournal": 17287
    },
    "count": 0
}
```

- `queryPort` is the MySQL protocol port of the FE node.
- `rpcPort` is the thrift RPC port of the FE node.
- `maxReplayedJournal` represents the maximum metadata journal id currently played back by the FE node.
- `arrowFlightSqlPort` is the Arrow Flight SQL port of the FE node.

Examples

1. No parameters

```
GET /api/bootstrap

Response:
{
    "msg": "OK",
    "code": 0,
    "data": null,
    "count": 0
}
```

2. Provide `token` and `cluster_id`

```
GET /api/bootstrap?cluster_id=935437471&token=ad87f6dd-c93f-4880-bcdb-8ca8c9ab3031

Response:
{
    "msg": "OK",
    "code": 0,
    "data": {
        "queryPort": 9030,
        "rpcPort": 9020,
        "arrowFlightSqlPort": 9040,
        "maxReplayedJournal": 17287
    },
    "count": 0
}
```

Cancel Load Action

Cancel Load Action

Request

`POST /api/<db>/_cancel`

Description

Used to cancel the load transaction of the specified label. RETURN VALUES Return a JSON format string: Status: Success: cancel succeed Others: cancel failed Message: Error message if cancel failed

Path parameters

- `<db>`

  Specify the database name

Query parameters

- `<label>`

  Specify the load label

Request body

None

Response

- Cancel success

```
{
    "msg": "OK",
    "code": 0,
    "data": null,
    "count": 0
}
```

- Cancel failed

```
{
    "msg": "Error msg...",
    "code": 1,
    "data": null,
```

```
    "count": 0
}
```

Examples

1. Cancel the load transaction of the specified label

```
POST /api/example_db/_cancel?label=my_label1

Response:
{
    "msg": "OK",
    "code": 0,
    "data": null,
    "count": 0
}
```

Check Decommission Action

Check Decommission Action

Request

`GET /api/check_decommission`

Description

Used to determine whether the specified BE can be decommissioned. For example, after the node being decommissioned, whether the remaining nodes can meet the space requirements and the number of replicas.

Path parameters

None

Query parameters

- `host_ports`

  Specify one or more BEs, separated by commas. Such as: `ip1:port1,ip2:port2,....`

  Where port is the heartbeat port of BE.

Request body

None

Response

Return a list of nodes that can be decommissioned

```
{
    "msg": "OK",
    "code": 0,
    "data": ["192.168.10.11:9050", "192.168.10.12:9050"],
    "count": 0
}
```

Examples

1. Check whether the specified BE node can be decommissioned

   ```
   GET /api/check_decommission?host_ports=192.168.10.11:9050,192.168.10.12:9050
   ```

Response:

```
{
    "msg": "OK",
    "code": 0,
    "data": ["192.168.10.11:9050", "192.168.10.12:9050"],
    "count": 0
}
```

Check Storage Type Action

Check Storage Type Action

Request

```
GET /api/_check_storagetype
```

Description

It is used to check whether the storage format of the table under the specified database is the row storage format. (The row format is deprecated)

Path parameters

None

Query parameters

- `db`

  Specify the database

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "tbl2": {},
        "tbl1": {}
    },
    "count": 0
}
```

If there is content after the table name, the base or rollup table whose storage format is row storage will be displayed.

Examples

1. Check whether the storage format of the following table of the specified database is row format

```
GET /api/_check_storagetype

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "tbl2": {},
        "tbl1": {}
    },
    "count": 0
}
```

Connection Action

Connection Action

Request

`GET /api/connection`

Description

Given a connection id, return the query id that is currently being executed for this connection or the last execution completed.

The connection id can be viewed through the id column in the MySQL command `show processlist;`.

Path parameters

无

Query parameters

- `connection_id`

  Specified connection id

Request body

None

Response

```
{
    "msg": "OK",
    "code": 0,
    "data": {
        "query_id": "b52513ce3f0841ca-9cb4a96a268f2dba"
    },
    "count": 0
}
```

Examples

1. Get the query id of the specified connection id

   ```
   GET /api/connection?connection_id=101

   Response:
   ```

```
{
    "msg": "OK",
    "code": 0,
    "data": {
        "query_id": "b52513ce3f0841ca-9cb4a96a268f2dba"
    },
    "count": 0
}
```

Extra Basepath Action

Extra Basepath Action

Request

`GET /api/basepath`

Description

Used to obtain http basepath.

Path parameters

None

Query parameters

None

Request body

None

Response


```
{
    "msg":"success",
    "code":0,
    "data":{"enable":false,"path":""},
    "count":0
}
```

Fe Version Info Action

Fe Version Info Action

Request

`GET /api/fe_version_info`

Description

Get fe version info from fe host.

Path parameters

None.

Query parameters

None.

Request body

None.

Response

```
{
    "msg":"success",
    "code":0,
    "data":{
        "feVersionInfo":{
            "dorisBuildVersionPrefix":"doris",
            "dorisBuildVersionMajor":0,
            "dorisBuildVersionMinor":0,
            "dorisBuildVersionPatch":0,
            "dorisBuildVersionRcVersion":"trunk",
            "dorisBuildVersion":"doris-0.0.0-trunk",
            "dorisBuildHash":"git://4b7b503d1cb3/data/doris/doris/be/../@a04f9814fe5a09c0d9e9399fe71cc4d765f8bff1",
            "dorisBuildShortHash":"a04f981",
            "dorisBuildTime":"Fri, 09 Sep 2022 07:57:02 UTC",
            "dorisBuildInfo":"root@4b7b503d1cb3",
            "dorisJavaCompileVersion":"openjdk full version \"1.8.0_332-b09\""
        }
    },
    "count":0
}
```

Examples

```
GET /api/fe_version_info

Response:
{
    "msg":"success",
    "code":0,
    "data":{
        "feVersionInfo":{
            "dorisBuildVersionPrefix":"doris",
            "dorisBuildVersionMajor":0,
            "dorisBuildVersionMinor":0,
            "dorisBuildVersionPatch":0,
            "dorisBuildVersionRcVersion":"trunk",
            "dorisBuildVersion":"doris-0.0.0-trunk",
            "dorisBuildHash":"git://4b7b503d1cb3/data/doris/doris/be/../@a04f9814fe5a09c0d9e9399fe71cc4d765f8bff1",
            "dorisBuildShortHash":"a04f981",
            "dorisBuildTime":"Fri, 09 Sep 2022 07:57:02 UTC",
            "dorisBuildInfo":"root@4b7b503d1cb3",
            "dorisJavaCompileVersion":"openjdk full version \"1.8.0_332-b09\""
        }
    },
    "count":0
}
```

Get DDL Statement Action

Get DDL Statement Action

Request

`GET /api/_get_ddl`

Description

Used to get the table creation statement, partition creation statement and rollup statement of the specified table.

Path parameters

None

Query parameters

- `db`

  Specify database

- `table`

  Specify table

Request body

None

Response

```
{
    "msg": "OK",
    "code": 0,
    "data": {
        "create_partition": ["ALTER TABLE `tbl1` ADD PARTITION ..."],
        "create_table": ["CREATE TABLE `tbl1` ..."],
        "create_rollup": ["ALTER TABLE `tbl1` ADD ROLLUP ..."]
    },
    "count": 0
}
```

Examples

1. Get the DDL statement of the specified table

```
GET /api/_get_ddl?db=db1&table=tbl1

Response
{
    "msg": "OK",
    "code": 0,
    "data": {
        "create_partition": [],
        "create_table": ["CREATE TABLE `tbl1` (\n  `k1` int(11) NULL COMMENT \"\",\n  `k2` int(11) NULL COMMENT \"\"\n)
ENGINE=OLAP\nDUPLICATE KEY(`k1`, `k2`)\nCOMMENT \"OLAP\"\nDISTRIBUTED BY HASH(`k1`) BUCKETS 1\nPROPERTIES
(\n\"replication_num\" = \"1\",\n\"version_info\" = \"1,0\",\n\"in_memory\" = \"false\",\n\"storage_format\" =
\"DEFAULT\"\n);"],
        "create_rollup": []
    },
    "count": 0
}
```

Get Load Info Action

Get Load Info Action

Request

`GET /api/<db>/_load_info`

Description

Used to obtain the information of the load job of the specified label.

Path parameters

- `<db>`

  Specify database

Query parameters

- `label`

  Specify load label

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "dbName": "default_cluster:db1",
        "tblNames": ["tbl1"],
        "label": "my_label",
        "clusterName": "default_cluster",
        "state": "FINISHED",
        "failMsg": "",
        "trackingUrl": ""
    },
    "count": 0
}
```

Examples

1. Get the load job information of the specified label

```
GET /api/example_db/_load_info?label=my_label

Response
{
    "msg": "success",
    "code": 0,
    "data": {
        "dbName": "default_cluster:db1",
        "tblNames": ["tbl1"],
        "label": "my_label",
        "clusterName": "default_cluster",
        "state": "FINISHED",
        "failMsg": "",
        "trackingUrl": ""
    },
    "count": 0
}
```

Get Load State

Get Load State

Request

`GET /api/<db>/get_load_state`

Description

Returns the status of the load transaction of the specified label Return of JSON format string of the status of specified transaction: Label: The specified label. Status: Success or not of this request. Message: Error messages State: UNKNOWN/PREPARE/COMMITTED/VISIBLE/ABORTED

Path parameters

- `<db>`

  Specify database

Query parameters

- `label`

  Specify label

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": "VISIBLE",
    "count": 0
}
```

If label does not exist, return:

```
{
    "msg": "success",
    "code": 0,
    "data": "UNKNOWN",
    "count": 0
}
```

Examples

1. Get the status of the load transaction of the specified label.

```
GET /api/example_db/get_load_state?label=my_label

{
    "msg": "success",
    "code": 0,
    "data": "VISIBLE",
    "count": 0
}
```

Get FE log file

Get FE log file

Request

`HEAD /api/get_log_file`

`GET /api/get_log_file`

Description

Users can obtain FE log files through the HTTP interface.

The HEAD request is used to obtain the log file list of the specified log type. GET request is used to download the specified log file.

Path parameters

None

Query parameters

- `type`

  Specify the log type. The following types are supported:

  - `fe.audit.log`: Audit log of Frontend.

- `file`

  Specify file name

Request body

None

Response

- `HEAD`

  ```
  HTTP/1.1 200 OK
  file_infos: {"fe.audit.log":24759,"fe.audit.log.20190528.1":132934}
  content-type: text/html
  connection: keep-alive
  ```

  The returned header lists all current log files of the specified type and the size of each file.

- GET

  Download the specified log file in text form

Examples

1. Get the log file list of the corresponding type

```
HEAD /api/get_log_file?type=fe.audit.log

Response:

HTTP/1.1 200 OK
file_infos: {"fe.audit.log":24759,"fe.audit.log.20190528.1":132934}
content-type: text/html
connection: keep-alive
```

   In the returned header, the `file_infos` field displays the file list and the corresponding file size (in bytes) in json format

2. Download log file

```
GET /api/get_log_file?type=fe.audit.log&file=fe.audit.log.20190528.1

Response:

< HTTP/1.1 200
< Vary: Origin
< Vary: Access-Control-Request-Method
< Vary: Access-Control-Request-Headers
< Content-Disposition: attachment;fileName=fe.audit.log
< Content-Type: application/octet-stream;charset=UTF-8
< Transfer-Encoding: chunked

... File Content ...
```

Get Small File Action

Get Small File

Request

`GET /api/get_small_file`

Description

Through the file id, download the file in the small file manager.

Path parameters

None

Query parameters

- `token`

  The token of the cluster. It can be viewed in the file `doris-meta/image/VERSION`.

- `file_id`

  The file id displayed in the file manager. The file id can be viewed with the `SHOW FILE` command.

Request body

None

Response

```
< HTTP/1.1 200
< Vary: Origin
< Vary: Access-Control-Request-Method
< Vary: Access-Control-Request-Headers
< Content-Disposition: attachment;fileName=ca.pem
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked

... File Content ...
```

If there is an error, it returns:

```
{
    "msg": "File not found or is not content",
    "code": 1,
    "data": null,
```

```
    "count": 0
}
```

Examples

1. Download the file with the specified id

```
GET /api/get_small_file?token=98e8c0a6-3a41-48b8-a72b-0432e42a7fe5&file_id=11002

Response:

< HTTP/1.1 200
< Vary: Origin
< Vary: Access-Control-Request-Method
< Vary: Access-Control-Request-Headers
< Content-Disposition: attachment;fileName=ca.pem
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked

... File Content ...
```

Get WAL size

Request

```
GET fe_host:fe_http_port/api/get_wal_size?host_ports=host1:port1,host2:port2...
```

Description

Through this HTTP interface, users can get the number of WAL files of a specified BE. If no BE is specified, the number of WAL files of all BEs is returned by default.

Path parameters

None

Query parameters

- `host_ports`

  The IP and HTTP port of the BE.

Request body

None

Response

```
{
"msg": "OK",
"code": 0,
"data": ["192.168.10.11:9050:1", "192.168.10.11:9050:0"],
"count": 0
}
```

Examples

1. Get the number of WAL files of all BEs.

   ```
   curl -u root: "127.0.0.1:8038/api/get_wal_size"

   Response:
   {
   "msg": "OK",
   "code": 0,
   "data": ["192.168.10.11:9050:1", "192.168.10.11:9050:0"],
   "count": 0
   }
   ```

In the returned result, the number following each BE is the WAL file count of that BE.

2. Get the number of WAL files of a specified BE.

```
curl -u root: "127.0.0.1:8038/api/get_wal_size?host_ports=192.168.10.11:9050"

Response:
{
"msg": "OK",
"code": 0,
"data": ["192.168.10.11:9050:1"],
"count": 0
}
```

In the returned result, the number following each BE is the WAL file count of that BE.

Health Action

Health Action

Request

`GET /api/health`

Description

Returns the number of BE nodes currently surviving in the cluster and the number of BE nodes that are down.

Path parameters

None

Query parameters

None

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "online_backend_num": 10,
        "total_backend_num": 10
    },
    "count": 0
}
```

Meta Info Action | Fe Http

Meta Info Action

Meta Info Action is used to obtain metadata information in the cluster. Such as database list, table structure, etc.

List Database

Request

```
GET /api/meta/namespaces/<ns_name>/databases
```

Description

Get a list of all database names, arranged in alphabetical order.

Path parameters

None

Query parameters

- `limit`

  Limit the number of result rows returned

- `offset`

  Pagination information, need to be used with `limit`

Request body

None

Response

```
{
    "msg": "OK",
    "code": 0,
    "data": [
       "db1", "db2", "db3", ...
    ],
    "count": 3
}
```

- The data field returns a list of database names.

List Table

Request

```
GET /api/meta/namespaces/<ns_name>/databases/<db_name>/tables
```

Description

Get a list of tables in the specified database, arranged in alphabetical order.

Path parameters

- `<db_name>`

  Specify database

Query parameters

- `limit`

  Limit the number of result rows returned

- `offset`

  Pagination information, need to be used with `limit`

Request body

None

Response

```
{
    "msg": "OK",
    "code": 0,
    "data": [
       "tbl1", "tbl2", "tbl3", ...
    ],
    "count": 0
}
```

- The data field returns a list of table names.

Schema Info

Request

```
GET /api/meta/namespaces/<ns_name>/databases/<db_name>/tables/<tbl_name>/schema
```

Description

Get the table structure information of the specified table in the specified database.

Path parameters

- `<db_name>`

  Specify the database name

- `<tbl_name>`

  Specify table name

Query parameters

- `with_mv`

  Optional. If not specified, the table structure of the base table is returned by default. If specified, all rollup index will also be returned.

Request body

None

Response

```
GET /api/meta/namespaces/default/databases/db1/tables/tbl1/schema

{
    "msg": "success",
    "code": 0,
    "data": {
        "tbl1": {
            "schema": [{
                    "Field": "k1",
                    "Type": "INT",
                    "Null": "Yes",
                    "Extra": "",
                    "Default": null,
                    "Key": "true"
                },
                {
                    "Field": "k2",
                    "Type": "INT",
                    "Null": "Yes",
                    "Extra": "",
```

```
                    "Default": null,
                    "Key": "true"
                }
            ],
            "is_base": true
        }
    },
    "count": 0
}
```

```
GET /api/meta/namespaces/default/databases/db1/tables/tbl1/schema?with_mv=1

{
    "msg": "success",
    "code": 0,
    "data": {
        "tbl1": {
            "schema": [{
                    "Field": "k1",
                    "Type": "INT",
                    "Null": "Yes",
                    "Extra": "",
                    "Default": null,
                    "Key": "true"
                },
                {
                    "Field": "k2",
                    "Type": "INT",
                    "Null": "Yes",
                    "Extra": "",
                    "Default": null,
                    "Key": "true"
                }
            ],
            "is_base": true
        },
        "rollup1": {
            "schema": [{
                "Field": "k1",
                "Type": "INT",
                "Null": "Yes",
                "Extra": "",
                "Default": null,
                "Key": "true"
            }],
            "is_base": false
        }
    },
    "count": 0
}
```

- The data field returns the table structure information of the base table or rollup table.

Meta Replay State Action

Meta Replay State Action

(Not Implemented)

Request

```
GET /api/_meta_replay_state
```

Description

Get the status of FE node metadata replay.

Path parameters

None

Query parameters

None

Request body

None

Response

TODO

Examples

TODO

Metrics Action

Metrics Action

Request

`GET /api/metrics`

Description

Used to obtain doris metrics infomation.

Path parameters

None

Query parameters

- `type`

  Optional parameter. Display all metrics by default, and with following values:

  - `core` Output core metrics
  - `json` Output metrics in json format

Request body

None

Response

TO DO

Profile Action

Profile Action

Request

> **Note:**
>
> Admin privileges (admin_priv) are required to use this API. We recommend using query profile action to view profiles instead.

`GET /api/profile` `GET /api/profile/text`

Description

Used to obtain the query profile of the specified query id. If query_id is not exists, return 404 NOT FOUND ERROR If query_id exists, return query profile like this

```
Query:
  Summary:
     - Query ID: a0a9259df9844029-845331577440a3bd
     - Start Time: 2020-06-15 14:10:05
     - End Time: 2020-06-15 14:10:05
     - Total: 8ms
     - Query Type: Query
     - Query State: EOF
     - Doris Version: trunk
     - User: root
     - Default Db: default_cluster:test
     - Sql Statement: select * from table1
  Execution Profile a0a9259df9844029-845331577440a3bd:(Active: 7.315ms, % non-child: 100.00%)
    Fragment 0:
      Instance a0a9259df9844029-845331577440a3be (host=TNetworkAddress(hostname:172.26.108.176, port:9560)):(Active: 1.523ms, % non-
child: 0.24%)
         - MemoryLimit: 2.00 GB
         - PeakUsedReservation: 0.00
         - PeakMemoryUsage: 72.00 KB
         - RowsProduced: 5
         - AverageThreadTokens: 0.00
         - PeakReservation: 0.00
        BlockMgr:
           - BlocksCreated: 0
           - BlockWritesOutstanding: 0
           - BytesWritten: 0.00
           - TotalEncryptionTime: 0ns
           - BufferedPins: 0
           - TotalReadBlockTime: 0ns
           - TotalBufferWaitTime: 0ns
           - BlocksRecycled: 0
           - TotalIntegrityCheckTime: 0ns
           - MaxBlockSize: 8.00 MB
        DataBufferSender (dst_fragment_instance_id=a0a9259df9844029-845331577440a3be):
```

```
           - AppendBatchTime: 9.23us
             - ResultSendTime: 956ns
             - TupleConvertTime: 5.735us
           - NumSentRows: 5
         OLAP_SCAN_NODE (id=0):(Active: 1.506ms, % non-child: 20.59%)
           - TotalRawReadTime: 0ns
           - CompressedBytesRead: 6.47 KB
           - PeakMemoryUsage: 0.00
           - RowsPushedCondFiltered: 0
           - ScanRangesComplete: 0
           - ScanTime: 25.195us
           - BitmapIndexFilterTimer: 0ns
           - BitmapIndexFilterCount: 0
           - NumScanners: 65
           - RowsStatsFiltered: 0
           - VectorPredEvalTime: 0ns
           - BlockSeekTime: 1.299ms
           - RawRowsRead: 1.91K (1910)
           - ScannerThreadsVoluntaryContextSwitches: 0
           - RowsDelFiltered: 0
           - IndexLoadTime: 911.104us
           - NumDiskAccess: 1
           - ScannerThreadsTotalWallClockTime: 0ns
             - MaterializeTupleTime: 0ns
             - ScannerThreadsUserTime: 0ns
             - ScannerThreadsSysTime: 0ns
           - TotalPagesNum: 0
           - RowsReturnedRate: 3.319K /sec
           - BlockLoadTime: 539.289us
           - CachedPagesNum: 0
           - BlocksLoad: 384
           - UncompressedBytesRead: 0.00
           - RowsBloomFilterFiltered: 0
           - TabletCount : 1
           - RowsReturned: 5
           - ScannerThreadsInvoluntaryContextSwitches: 0
           - DecompressorTimer: 0ns
           - RowsVectorPredFiltered: 0
           - ReaderInitTime: 6.498ms
           - RowsRead: 5
           - PerReadThreadRawHdfsThroughput: 0.0 /sec
           - BlockFetchTime: 4.318ms
           - ShowHintsTime: 0ns
           - TotalReadThroughput: 0.0 /sec
           - IOTimer: 1.154ms
           - BytesRead: 48.49 KB
           - BlockConvertTime: 97.539us
           - BlockSeekCount: 0
```

If it is a text interface, simply return the plain text content of the profile.

Path parameters

None

Query parameters

- query_id

  Specify query id

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "profile": "query profile ..."
    },
    "count": 0
}
```

Examples

1. Get the query profile of the specified query id

   ```
   GET /api/profile?query_id=f732084bc8e74f39-8313581c9c3c0b58

   Response:
   {
       "msg": "success",
       "code": 0,
       "data": {
           "profile": "query profile ..."
       },
       "count": 0
   }
   ```

2. Get the query profile text of the specified query id ``` GET /api/profile/text?query_id=f732084bc8e74f39-8313581c9c3c0b58

   Response: Summary: - Profile ID: 48bdf6d75dbb46c9-998b9c0368f4561f - Task Type: QUERY - Start Time: 2023-12-20 11:09:41 - End Time: 2023-12-20 11:09:45 - Total: 3s680ms - Task State: EOF - User: root - Default Db: tpcds - Sql Statement: with customer_total_return as select sr_customer_sk as ctr_customer_sk ,sr_store_sk as ctr_store_sk ,sum(SR_FEE) as ctr_total_return … ```

Query Schema Action

Query Schema Action

Request

```
POST /api/query_schema/<ns_name>/<db_name>
```

Description

The Query Schema Action can return the table creation statement for the given SQL-related table. Can be used to test some query scenarios locally.

The API was released in version 1.2.

Path parameters

- `<db_name>`

  Specify the database name. This database will be considered as the default database for the current session, and will be used if the table name in SQL does not qualify the database name.

Query parameters

None

Request body

```
text/plain

sql
```

- “sql” field is the SQL string.

Response

- Return value

```
CREATE TABLE `tbl1` (
  `k1` int(11) NULL,
  `k2` int(11) NULL
) ENGINE=OLAP
DUPLICATE KEY(`k1`, `k2`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`k1`) BUCKETS 3
PROPERTIES (
```

```
"replication_allocation" = "tag.location.default: 1",
"in_memory" = "false",
"storage_format" = "V2",
"disable_auto_compaction" = "false"
);

CREATE TABLE `tbl2` (
  `k1` int(11) NULL,
  `k2` int(11) NULL
) ENGINE=OLAP
DUPLICATE KEY(`k1`, `k2`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`k1`) BUCKETS 3
PROPERTIES (
"replication_allocation" = "tag.location.default: 1",
"in_memory" = "false",
"storage_format" = "V2",
"disable_auto_compaction" = "false"
);
```

Example

1. Write the SQL in local file 1.sql

```
select tbl1.k2 from tbl1 join tbl2 on tbl1.k1 = tbl2.k1;
```

2. Use curl to get the table creation statement.

```
curl -X POST -H 'Content-Type: text/plain'  -uroot: http://127.0.0.1:8030/api/query_schema/internal/db1 -d@1.sql
```

Query Stats Action

Request

```
View
get api/query_stats/<catalog_name>
get api/query_stats/<catalog_name>/<db_name>
get api/query_stats/<catalog_name>/<db_name>/<tbl_name>

Clear
delete api/query_stats/<catalog_name>/<db_name>
delete api/query_stats/<catalog_name>/<db_name>/<tbl_name>
```

Description

Get or delete the statistics information of the specified catalog database or table, if it is a doris catalog, you can use default_cluster

Path parameters

- `<catalog_name>` specified catalog name
-
- `<db_name>` specified database name
- `<tbl_name>` specified table name

Query parameters

- `summary` if true, only return summary information, otherwise return all the detailed statistics information of the table, only used in get

Request body

```
GET /api/query_stats/default_cluster/test_query_db/baseall?summary=false
{
    "msg": "success",
    "code": 0,
    "data": {
        "summary": {
            "query": 2
        },
        "detail": {
            "baseall": {
                "summary": {
                    "query": 2
                }
            }
        }
    },
```

```
    "count": 0
}
```

Response

- return statistics information

Example

2. use curl

```
curl --location -u root: 'http://127.0.0.1:8030/api/query_stats/default_cluster/test_query_db/baseall?summary=false'
```

Row Count Action

Row Count Action

Request

`GET /api/rowcount`

Description

Used to manually update the row count statistics of the specified table. While updating the statistics of the number of rows, the table and the number of rows corresponding to the rollup will also be returned in JSON format

Path parameters

None

Query parameters

- `db`

  Specify database

- `table`

  Specify table

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "tbl1": 10000
    },
    "count": 0
}
```

Examples

1. Update and get the number of rows in the specified Table

```
GET /api/rowcount?db=example_db&table=tbl1

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "tbl1": 10000
    },
    "count": 0
}
```

Set Config Action

Set Config Action

Request

`GET /api/_set_config`

Description

Used to dynamically set the configuration of FE. This command is passed through the `ADMIN SET FRONTEND CONFIG` command. But this command will only set the configuration of the corresponding FE node. And it will not automatically forward the `MasterOnly` configuration item to the Master FE node.

Path parameters

None

Query parameters

- `confkey1=confvalue1`

  Specify the configuration name to be set, and its value is the configuration value to be modified.

- `persist`

  Whether to persist the modified configuration. The default is false, which means it is not persisted. If it is true, the modified configuration item will be written into the `fe_custom.conf` file and will still take effect after FE is restarted.

- `reset_persist` Whether or not to clear the original persist configuration only takes effect when the persist parameter is true. For compatibility with the original version, reset_persist defaults to true.
  If persist is set to true and reset_persist is not set or reset_persist is true, the configuration in the `fe_custom.conf` file will be cleared before this modified configuration is written to `fe_custom.conf`.
  If persist is set to true and reset_persist is false, this modified configuration item will be incrementally added to `fe_custom.conf`.

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
```

```
    "data": {
        "set": {
            "key": "value"
        },
        "err": [
            {
               "config_name": "",
               "config_value": "",
               "err_info": ""
            }
        ],
        "persist":""
    },
    "count": 0
}
```

The `set` field indicates the successfully set configuration. The `err` field indicates the configuration that failed to be set. The `persist` field indicates persistent information.

Examples

1. Set the values of `storage_min_left_capacity_bytes`, `replica_ack_policy` and `agent_task_resend_wait_time_ms`.

```
GET /api/_set_config?
storage_min_left_capacity_bytes=1024&replica_ack_policy=SIMPLE_MAJORITY&agent_task_resend_wait_time_ms=true

Response:
{
"msg": "success",
"code": 0,
"data": {
    "set": {
        "storage_min_left_capacity_bytes": "1024"
    },
    "err": [
        {
            "config_name": "replica_ack_policy",
            "config_value": "SIMPLE_MAJORITY",
            "err_info": "Not support dynamic modification."
        },
        {
            "config_name": "agent_task_resend_wait_time_ms",
            "config_value": "true",
            "err_info": "Unsupported configuration value type."
        }
    ],
    "persist": ""
},
"count": 0
}

storage_min_left_capacity_bytes  Successfully;
replica_ack_policy  Failed, because the configuration item does not support dynamic modification.
agent_task_resend_wait_time_ms  Failed, failed to set the boolean type because the configuration item is of type long.
```

2. Set `max_bytes_per_broker_scanner` and persist it.

```
GET /api/_set_config?max_bytes_per_broker_scanner=21474836480&persist=true&reset_persist=false

Response:
{
"msg": "success",
"code": 0,
"data": {
    "set": {
        "max_bytes_per_broker_scanner": "21474836480"
    },
    "err": [],
    "persist": "ok"
},
"count": 0
}
```

The fe/conf directory generates the fe_custom.conf file: `#THIS IS AN AUTO GENERATED CONFIG FILE. #You can modify this file manually, and the configurations in this file #will overwrite the configurations in fe.conf #Wed Jul 28 12:43:14 CST 2021 max_bytes_per_broker_scanner=21474836480`

Show Data Action

Show Data Action

Request

`GET /api/show_data`

Description

Used to get the total data volume of the cluster or the data volume of the specified database. Unit byte.

Path parameters

None

Query parameters

- `db`

  Optional. If specified, get the data volume of the specified database.

Request body

None

Response

1. Specify the amount of data in the database.

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "default_cluster:db1": 381
    },
    "count": 0
}
```

2. Total data

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "__total_size": 381
    },
```

```
    "count": 0
}
```

Examples

1. Get the data volume of the specified database

```
GET /api/show_data?db=db1

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "default_cluster:db1": 381
    },
    "count": 0
}
```

2. Get the total data volume of the cluster

```
GET /api/show_data

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "__total_size": 381
    },
    "count": 0
}
```

Show Meta Info Action

Show Meta Info Action

Request

`GET /api/show_meta_info`

Description

Used to display some metadata information

Path parameters

None

Query parameters

- action

  Specify the type of metadata information to be obtained. Currently supports the following:

  - `SHOW_DB_SIZE`

    Get the data size of the specified database, in bytes.

  - `SHOW_HA`

    Obtain the playback status of FE metadata logs and the status of electable groups.

Request body

None

Response

- `SHOW_DB_SIZE`

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "default_cluster:information_schema": 0,
        "default_cluster:db1": 381
    },
    "count": 0
}
```

- SHOW_HA

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "can_read": "true",
        "role": "MASTER",
        "is_ready": "true",
        "last_checkpoint_version": "1492",
        "last_checkpoint_time": "1596465109000",
        "current_journal_id": "1595",
        "electable_nodes": "",
        "observer_nodes": "",
        "master": "10.81.85.89"
    },
    "count": 0
}
```

Examples

1. View the data size of each database in the cluster

```
GET /api/show_meta_info?action=show_db_size

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "default_cluster:information_schema": 0,
        "default_cluster:db1": 381
    },
    "count": 0
}
```

2. View the FE election group situation

```
GET /api/show_meta_info?action=show_ha

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "can_read": "true",
        "role": "MASTER",
        "is_ready": "true",
        "last_checkpoint_version": "1492",
        "last_checkpoint_time": "1596465109000",
        "current_journal_id": "1595",
        "electable_nodes": "",
        "observer_nodes": "",
        "master": "10.81.85.89"
    },
    "count": 0
}
```

Show Proc Action

Show Proc Action

Request

`GET /api/show_proc`

Description

Used to obtain PROC information.

Path parameters

None

Query parameters

- path

  Specify Proc Path

- forward

  Whether to forward to Master FE for execution

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": [
        proc infos ...
    ],
    "count": 0
}
```

Examples

1. View `/statistic` information

   ```
   GET /api/show_proc?path=/statistic
   ```

```
Response:
{
    "msg": "success",
    "code": 0,
    "data": [
        ["10003", "default_cluster:db1", "2", "3", "3", "3", "3", "0", "0", "0"],
        ["10013", "default_cluster:doris_audit_db__", "1", "4", "4", "4", "4", "0", "0", "0"],
        ["Total", "2", "3", "7", "7", "7", "7", "0", "0", "0"]
    ],
    "count": 0
}
```

2. Forward to Master for execution

```
GET /api/show_proc?path=/statistic&forward=true

Response:
{
    "msg": "success",
    "code": 0,
    "data": [
        ["10003", "default_cluster:db1", "2", "3", "3", "3", "3", "0", "0", "0"],
        ["10013", "default_cluster:doris_audit_db__", "1", "4", "4", "4", "4", "0", "0", "0"],
        ["Total", "2", "3", "7", "7", "7", "7", "0", "0", "0"]
    ],
    "count": 0
}
```

Show Runtime Info Action

Show Runtime Info Action

Request

```
GET /api/show_runtime_info
```

Description

Used to obtain Runtime information of FE JVM

Path parameters

None

Query parameters

None

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "free_mem": "855642056",
        "total_mem": "1037959168",
        "thread_cnt": "98",
        "max_mem": "1037959168"
    },
    "count": 0
}
```

Examples

1. Get the JVM information of the current FE node

```
GET /api/show_runtime_info

Response:
{
    "msg": "success",
    "code": 0,
```

```
    "data": {
        "free_mem": "855642056",
        "total_mem": "1037959168",
        "thread_cnt": "98",
        "max_mem": "1037959168"
    },
    "count": 0
}
```

Show Table Data Action

Show Table Data Action

Request

`GET /api/show_table_data`

Description

Used to get the data size of all tables in all databases under all internal catalog, or the data size of the specified database or table. Unit byte.

Path parameters

NULL

Query parameters

- `db`

  Optional. If specified, get the data size of the tables under the specified database.

- `table`

  Optional. If specified, get the data size of the specified table.

- `single_replica`

  Optional. If specified, get the data size of the single replica of the table.

Request body

NULL

Response

1. The data size of all tables in the specified database.

```
{
    "msg":"success",
    "code":0,
    "data":{
        "tpch":{
            "partsupp":9024548244,
            "revenue0":0,
            "customer":1906421482
        }
    },
```

```
    "count":0
}
```

2. The data size of the specified table of the specified db.

```
{
    "msg":"success",
    "code":0,
    "data":{
        "tpch":{
            "partsupp":9024548244
        }
    },
    "count":0
}
```

3. The data size of the single replica of the specified table of the specified db.

```
{
    "msg":"success",
    "code":0,
    "data":{
        "tpch":{
            "partsupp":3008182748
        }
    },
    "count":0
}
```

Examples

1. The data size of all tables in the specified database.

```
GET /api/show_table_data?db=tpch

Response:
{
    "msg":"success",
    "code":0,
    "data":{
        "tpch":{
            "partsupp":9024548244,
            "revenue0":0,
            "customer":1906421482
        }
    },
    "count":0
}
```

2. The data size of the specified table of the specified db.

```
GET /api/show_table_data?db=tpch&table=partsupp

Response:
{
    "msg":"success",
    "code":0,
```

```
    "data":{
        "tpch":{
            "partsupp":9024548244
        }
    },
    "count":0
}
```

3. The data size of the single replica of the specified table of the specified db.

```
GET /api/show_table_data?db=tpch&table=partsupp&single_replica=true

Response:
{
    "msg":"success",
    "code":0,
    "data":{
        "tpch":{
            "partsupp":3008182748
        }
    },
    "count":0
}
```

Statement Execution Action

Statement Execution Action

Request

```
POST /api/query/<ns_name>/<db_name>
```

Description

Statement Execution Action is used to execute a statement and return the result.

Path parameters

- `<ns_name>`

  Specify the namespace. Currently only supports `default_cluster`.

- `<db_name>`

  Specify the database name. This database will be regarded as the default database of the current session. If the table name in SQL does not qualify the database name, this database will be used.

Query parameters

None

Request body

```
{
    "stmt" : "select * from tbl1"
}
```

- The `stmt` field is the specific SQL statement to be executed.

Response

- Return result set

  ```
  {
      "msg": "success",
      "code": 0,
      "data": {
          "type": "result_set",
          "data": [
              [1],
              [2]
  ```

```
        ],
        "meta": [{
            "name": "k1",
            "type": "INT"
        }],
        "status": {},
        "time": 10
    },
    "count": 0
}
```

- The type field is `result_set`, which means the result set is returned. The results need to be obtained and displayed based on the meta and data fields. The meta field describes the column information returned. The data field returns the result row. The column type in each row needs to be judged by the content of the meta field. The status field returns some information of MySQL, such as the number of alarm rows, status code, etc. The time field return the execution time, unit is millisecond.

- Return execution result

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "type": "exec_status",
        "status": {}
    },
    "count": 0,
    "time": 10
}
```

- The type field is `exec_status`, which means the execution result is returned. At present, if the return result is received, it means that the statement was executed successfully.

Table Query Plan Action

Table Query Plan Action

Request

`POST /api/<db>/<table>/_query_plan`

Description

Given a SQL, it is used to obtain the query plan corresponding to the SQL.

This interface is currently used in Spark-Doris-Connector, Spark obtains Doris' query plan.

Path parameters

- `<db>`

  Specify database

- `<table>`

  Specify table

Query parameters

None

Request body

```
{
    "sql": "select * from db1.tbl1;"
}
```

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "partitions": {
            "10039": {
                "routings": ["10.81.85.89:9062"],
                "version": 2,
                "versionHash": 982459448378619656,
                "schemaHash": 1294206575
            }
        },
```

```
        "opaqued_query_plan":
"DAABDAACDwABDAAAAAEIAAEAAAAACAACAAAAAAgAAwAAAAAKAAT/////////w8ABQgAAAABAAAAA8ABgIAAAABAAIACAAMABIIAAEAAAAADwACCwAAAAIAAAACazEAAAA
CazIPAAMIAAAAAgAAAAUAAAAFAgAEAQAAAA8ABAwAAAACDwABDAAAAAEIAAEAAAAQDAACDwABDAAAAAEIAAEAAAAADAACCAABAAAABQAAAAgABAAAAAAMAA8IAAEAAAAACAA
CAAAAAAAIABT/////CAAX/////wAADwABDAAAAAEIAAEAAAAQDAACDwABDAAAAAEIAAEAAAAADAACCAABAAAABQAAAAgABAAAAAAMAA8IAAEAAAABCAACAAAAAAAIABT////
/CAAX/////wAADAAFCAABAAAABgwACAAADAAGCAABAAAAAA8AAgwAAAAAAAoABwAAAAAAAAAACgAIAAAAAAAAAAAADQACCgwAAAABAAAAAAAAJzcKAAEAAAAAAAAnNwoAAgA
AAAAAAAACCgADDaJlqbrVdwgIAARNJAZvAAwAAw8AAQwAAAACCAABAAAAAAgAAgAAAAAMAAMPAAEMAAAAAQgAAQAAAAAMAAIIAAEAAAAFAAAACAAE/////wgABQAAAAQIAAY
AAAAACAAHAAAAAAsACAAAAAJrMQgACQAAAAACAAoBAAgAAQAAAAEIAAIAAAAADAADDwABDAAAAAEIAAEAAAAADAACCAABAAAABQAAAAgABP////8IAAUAAAAICAAGAAAAAAg
ABwAAAAELAAgAAAACazIIAAkAAAABAgAKAQAPAAIMAAAAAQgAAQAAAAAIAAIAAAAMCAADAAAAAQoABAAAAAAAACc1CAAFAAAAAgAPAAMMAAAAAQoAAQAAAAAACc1CAACAAA
AAQgAAwAAAAIIAAQAAAAACwAHAAAABHRibDELAAgAAAAADAALCwABAAAABHRibDEAAAAMAAQKAAFfL5rpxl1I4goAArgs6f+h6eMxAAA=",
        "status": 200
    },
    "count": 0
}
```

Among them, `opaqued_query_plan` is the binary format of the query plan.

Examples

1. Get the query plan of the specified SQL

```
POST /api/db1/tbl1/_query_plan
{
    "sql": "select * from db1.tbl1;"
}

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "partitions": {
            "10039": {
                "routings": ["192.168.1.1:9060"],
                "version": 2,
                "versionHash": 982459448378619656,
                "schemaHash": 1294206575
            }
        },
        "opaqued_query_plan": "DAABDAACDwABD...",
        "status": 200
    },
    "count": 0
}
```

Table Row Count Action

Table Row Count Action

Request

`GET /api/<db>/<table>/_count`

Description

Used to obtain statistics about the number of rows in a specified table. This interface is currently used in Spark-Doris-Connector. Spark obtains Doris table statistics.

Path parameters

- `<db>`

  Specify database

- `<table>`

  Specify table

Query parameters

None

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "size": 1,
        "status": 200
    },
    "count": 0
}
```

The `data.size` field indicates the number of rows in the specified table.

Examples

1. Get the number of rows in the specified table.

```
GET /api/db1/tbl1/_count

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "size": 1,
        "status": 200
    },
    "count": 0
}
```

Table Schema Action

Table Schema Action

Request

`GET /api/<db>/<table>/_schema`

Description

Used to obtain the table structure information of the specified table. This interface is currently used in Spark/Flink Doris Connector. obtains Doris table structure information.

Path parameters

- `<db>`

  Specify database

- `<table>`

  Specify table

Query parameters

None

Request body

None

Response

- The http interface returns as follows:

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "properties": [{
            "type": "INT",
            "name": "k1",
            "comment": "",
            "aggregation_type":""
        }, {
            "type": "INT",
            "name": "k2",
            "comment": "",
            "aggregation_type":"MAX"
```

```
        }],
        "keysType":UNIQUE_KEYS,
        "status": 200
    },
    "count": 0
}
```

- The http v2 interface returns as follows:

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "properties": [{
            "type": "INT",
            "name": "k1",
            "comment": ""
        }, {
            "type": "INT",
            "name": "k2",
            "comment": ""
        }],
        "keysType":UNIQUE_KEYS,
        "status": 200
    },
    "count": 0
}
```

Note: The difference is that the `http` method returns more `aggregation_type` fields than the `http v2` method. The `http v2` is enabled by setting `enable_http_server_v2`. For detailed parameter descriptions, see fe parameter settings

Examples

1. Get the table structure information of the specified table via http interface.

```
GET /api/db1/tbl1/_schema

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "properties": [{
            "type": "INT",
            "name": "k1",
            "comment": "",
            "aggregation_type":""
        }, {
            "type": "INT",
            "name": "k2",
            "comment": "",
            "aggregation_type":"MAX"
        }],
        "keysType":UNIQUE_KEYS,
        "status": 200
    },
    "count": 0
}
```

2. Get the table structure information of the specified table via http v2 interface.

```
GET /api/db1/tbl1/_schema

Response:
{
    "msg": "success",
    "code": 0,
    "data": {
        "properties": [{
            "type": "INT",
            "name": "k1",
            "comment": ""
        }, {
            "type": "INT",
            "name": "k2",
            "comment": ""
        }],
        "keysType":UNIQUE_KEYS,
        "status": 200
    },
    "count": 0
}
```

Upload Action

Upload Action

Upload Action currently mainly serves the front-end page of FE, and is used for users to load small test files.

Upload load file

Used to upload a file to the FE node, which can be used to load the file later. Currently only supports uploading files up to 100MB.

Request

```
POST /api/<namespace>/<db>/<tbl>/upload
```

Path parameters

- `<namespace>`

  Namespace, currently only supports `default_cluster`

- `<db>`

  Specify database

- `<tbl>`

  Specify table

Query parameters

- `column_separator`

  Optional, specify the column separator of the file. Default is `\t`

- `preview`

  Optional, if set to `true`, up to 10 rows of data rows split according to `column_separator` will be displayed in the returned result.

Request body

The content of the file to be uploaded, the Content-type is `multipart/form-data`

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "id": 1,
        "uuid": "b87824a4-f6fd-42c9-b9f1-c6d68c5964c2",
        "originFileName": "data.txt",
        "fileSize": 102400,
        "absPath": "/path/to/file/data.txt",
        "maxColNum" : 5
    },
    "count": 1
}
```

Load the uploaded file

Request

```
PUT /api/<namespace>/<db>/<tbl>/upload
```

Path parameters

- `<namespace>`

  Namespace, currently only supports `default_cluster`

- `<db>`

  Specify database

- `<tbl>`

  Specify table

Query parameters

- `file_id`

  Specify the load file id, which is returned by the API that uploads the file.

- `file_uuid`

  Specify the file uuid, which is returned by the API that uploads the file.

Header

The options in the header are the same as those in the header in the Stream Load request.

Request body

None

Response

```
{
    "msg": "success",
    "code": 0,
    "data": {
        "TxnId": 7009,
        "Label": "9dbdfb0a-120b-47a2-b078-4531498727cb",
        "Status": "Success",
        "Message": "OK",
        "NumberTotalRows": 3,
        "NumberLoadedRows": 3,
        "NumberFilteredRows": 0,
        "NumberUnselectedRows": 0,
        "LoadBytes": 12,
        "LoadTimeMs": 71,
        "BeginTxnTimeMs": 0,
        "StreamLoadPutTimeMs": 1,
        "ReadDataTimeMs": 0,
        "WriteDataTimeMs": 13,
        "CommitAndPublishTimeMs": 53
    },
    "count": 1
}
```

Example

```
PUT /api/default_cluster/db1/tbl1/upload?file_id=1&file_uuid=b87824a4-f6fd-42c9-b9f1-c6d68c5964c2
```

Import Action

Import Action

Request

`POST /api/import/file_review`

Description

View the contents of the file in CSV or PARQUET format.

Path parameters

None

Query parameters

None

Request body

TO DO

Response

TO DO

Meta Info Action | Fe Http

Meta Info Action

Request

`GET /api/meta/namespaces/<ns>/databases GET /api/meta/namespaces/<ns>/databases/<db>/tables GET /api/meta/namespaces/<ns>/databases/<db>/tables/<tbl>/schema`

Description

Used to obtain metadata information about the cluster, including the database list, table list, and table schema.

Path parameters

- `ns`

  Specify cluster name.

- `db`

  Specify database name.

- `tbl`

  Specify table name.

Query parameters

None

Request body

None

Response

```
{
    "msg":"success",
    "code":0,
    "data":["database list" / "table list" / "table schema"],
    "count":0
}
```

Debug Point

Debug Point

Debug point is a piece of code, inserted into FE or BE code, when program running into this code,

it can change variables or behaviors of the program.

It is mainly used for unit test or regression test when it is impossible to trigger some exceptions through normal means.

Each debug point has a name, the name can be whatever you want, there are swithes to enable and disable debug points,

and you can also pass data to debug points.

Both FE and BE support debug point, and after inserting debug point code, recompilation of FE or BE is needed.

Code Example

FE example

```
private Status foo() {
    // dbug_fe_foo_do_nothing is the debug point name
    // when it's active, DebugPointUtil.isEnable("dbug_fe_foo_do_nothing") returns true
    if (DebugPointUtil.isEnable("dbug_fe_foo_do_nothing")) {
        return Status.Nothing;
    }

    do_foo_action();

    return Status.Ok;
}
```

BE example

```
void Status foo() {
     // dbug_be_foo_do_nothing is the debug point name
     // when it's active, DBUG_EXECUTE_IF will execute the code block
     DBUG_EXECUTE_IF("dbug_be_foo_do_nothing",  { return Status.Nothing; });

     do_foo_action();

     return Status.Ok;
}
```

Global Config

To enable debug points globally, we need to set `enable_debug_points` to true,

`enable_debug_points` is located in FE's fe.conf and BE's be.conf.

Activate A Specified Debug Point

After debug points are enabled globally, a http request with a debug point name should be send to FE or BE node,
only after that, when the program running into the specified debug point, related code can be executed.

API

```
POST /api/debug_point/add/{debug_point_name}[?timeout=<int>&execute=<int>]
```

Query Parameters

- `debug_point_name` Debug point name. Mandatory parameter.
- `timeout` Timeout in seconds. When timeout, the debug point will be deactivated. Default is -1, never timeout. Optional.
- `execute` After activating, the max times the debug point can be executed. Default is -1, unlimited times. Optional.

Request body

None

Response

```
{
    msg: "OK",
    code: 0
}
```

Examples

After activating debug point `foo`, executed no more than five times.

```
curl -X POST "http://127.0.0.1:8030/api/debug_point/add/foo?execute=5"
```

Pass Custom Parameters

When activating debug point, besides "timeout" and "execute" mentioned above, passing custom parameters is also allowed.
A parameter is a key-value pair in the form of "key=value" in url path, after debug point name glued by character '?'.
See examples below.

API

```
POST /api/debug_point/add/{debug_point_name}[?k1=v1&k2=v2&k3=v3...]
```

- k1=v1
  k1 is parameter name
  v1 is parameter value
  multiple key-value pairs are concatenated by `&`

Request body

None

Response

```
{
    msg: "OK",
    code: 0
}
```

Examples

Assuming a FE node with configuration http_port=8030 in fe.conf,
the following http request activates a debug point named `foo` in FE node and passe parameter `percent` and `duration`: >NOTE: User name and password may be needed.

```
curl -u root: -X POST "http://127.0.0.1:8030/api/debug_point/add/foo?percent=0.5&duration=3"
```

```
NOTE:
1. Inside FE and BE code, names and values of parameters are taken as strings.
2. Parameter names and values are case sensitive in http request and FE/BE code.
3. FE and BE share same url paths of REST API, it's just their IPs and Ports are different.
```

Use parameters in FE and BE code

Following request activates debug point `OlapTableSink.write_random_choose_sink` in FE and passes parameter `needCatchUp` and `sinkNum`:

```
curl -u root: -X POST "http://127.0.0.1:8030/api/debug_point/add/OlapTableSink.write_random_choose_sink?needCatchUp=true&sinkNum=3"
```

The code in FE checks debug point `OlapTableSink.write_random_choose_sink` and gets parameter values:

```
private void debugWriteRandomChooseSink(Tablet tablet, long version, Multimap<Long, Long> bePathsMap) {
    DebugPoint debugPoint = DebugPointUtil.getDebugPoint("OlapTableSink.write_random_choose_sink");
    if (debugPoint == null) {
        return;
    }
    boolean needCatchup = debugPoint.param("needCatchUp", false);
```

```
    int sinkNum = debugPoint.param("sinkNum", 0);
    ...
}
```

Following request activates debug point `TxnManager.prepare_txn.random_failed` in BE and passes parameter `percent`:

```
curl -X POST "http://127.0.0.1:8040/api/debug_point/add/TxnManager.prepare_txn.random_failed?percent=0.7
```

The code in BE checks debug point `TxnManager.prepare_txn.random_failed` and gets parameter value:

```
DBUG_EXECUTE_IF("TxnManager.prepare_txn.random_failed",
        {if (rand() % 100 < (100 * dp->param("percent", 0.5))) {
                LOG_WARNING("TxnManager.prepare_txn.random_failed random failed");
                return Status::InternalError("debug prepare txn random failed");
        }}
);
```

Disable Debug Point

API

```
    POST /api/debug_point/remove/{debug_point_name}
```

Query Parameters

- `debug_point_name` Debug point name. Mandatory parameter.

Request body

None

Response

```
{
    msg: "OK",
    code: 0
}
```

Examples

Disable debug point `foo`.

```
curl -X POST "http://127.0.0.1:8030/api/debug_point/remove/foo"
```

Clear Debug Points

API

```
POST /api/debug_point/clear
```

Request body

None

Response

```
{
    msg: "OK",
    code: 0
}
```

Examples

```
curl -X POST "http://127.0.0.1:8030/api/debug_point/clear"
```

Debug Points in Regression Test

> In community's CI system, `enable_debug_points` configuration of FE and BE are true by default.

The Regression test framework also provides methods to activate and deactivate a particular debug point, they are declared as below:

```
// "name" is the debug point to activate, "params" is a list of key-value pairs passed to debug point
def enableDebugPointForAllFEs(String name, Map<String, String> params = null);
def enableDebugPointForAllBEs(String name, Map<String, String> params = null);
// "name" is the debug point to deactivate
def disableDebugPointForAllFEs(String name);
def disableDebugPointForAllFEs(String name);
```

`enableDebugPointForAllFEs()` or `enableDebugPointForAllBEs()` needs to be called before the test actions you want to generate error,
and `disableDebugPointForAllFEs()` or `disableDebugPointForAllBEs()` needs to be called afterward.

Concurrent Issue

Enabled debug points affects FE or BE globally, which could cause other concurrent tests to fail unexpectedly in your pull request.
To avoid this, there's a convention that regression tests using debug points must be in directory regression-test/suites/fault_injection_p0,

and their group name must be “nonConcurrent”, as these regression tests will be executed serially by pull request workflow.

Examples

```
// .groovy file of the test case must be in regression-test/suites/fault_injection_p0
// and the group name must be 'nonConcurrent'
suite('debugpoint_action', 'nonConcurrent') {
    try {
        // Activate debug point named "PublishVersionDaemon.stop_publish" in all FE
        // and pass parameter "timeout"
        // "execute" and "timeout" are pre-existing parameters, usage is mentioned above
        GetDebugPoint().enableDebugPointForAllFEs('PublishVersionDaemon.stop_publish', [timeout:1])

        // Activate debug point named "Tablet.build_tablet_report_info.version_miss" in all BE
        // and pass parameter "tablet_id", "version_miss" and "timeout"
        GetDebugPoint().enableDebugPointForAllBEs('Tablet.build_tablet_report_info.version_miss',
                                                  [tablet_id:'12345', version_miss:true, timeout:1])

        // Test actions which will run into debug point and generate error
        sql """CREATE TABLE tbl_1 (k1 INT, k2 INT)
               DUPLICATE KEY (k1)
               DISTRIBUTED BY HASH(k1)
               BUCKETS 3
               PROPERTIES ("replication_allocation" = "tag.location.default: 1");
            """
        sql "INSERT INTO tbl_1 VALUES (1, 10)"
        sql "INSERT INTO tbl_1 VALUES (2, 20)"
        order_qt_select_1_1 'SELECT * FROM tbl_1'

    } finally {
        // Deactivate debug points
        GetDebugPoint().disableDebugPointForAllFEs('PublishVersionDaemon.stop_publish')
        GetDebugPoint().disableDebugPointForAllBEs('Tablet.build_tablet_report_info.version_miss')
    }
}
```

Statistic Action

Statistic Action

Request

```
GET /rest/v2/api/cluster_overview
```

Description

Obtain cluster statistics such as the number of databases and tables.

Path parameters

None

Query parameters

None

Request body

None

Response


```
{
    "msg":"success",
    "code":0,
    "data":{"diskOccupancy":0,"remainDisk":5701197971457,"feCount":1,"tblCount":27,"beCount":1,"dbCount":2},
    "count":0
}
```

Check Stub Cache

CHECK Stub Cache

Request

`GET /api/check_rpc_channel/{host_to_check}/{remot_brpc_port}/{payload_size}`

Description

Check whether the connection cache is available

Path parameters

- `host_to_check`

  Host to check

- `remot_brpc_port`

  Remote brpc port

- `payload_size`

  Load size, unit: B, value range 1~1024000.

Request body

None

Response

```
```
{
    "msg":"success",
    "code":0,
    "data": "open brpc connection to {host_to_check}:{remot_brpc_port} success.",
    "count":0
}
```
```

Examples

```
```
curl http://127.0.0.1:8040/api/check_rpc_channel/127.0.0.1/8060/1024000
```
```

Reset Stub Cache

Reset Stub Cache

Request

`GET /api/reset_rpc_channel/{endpoints}`

Description

Reset the connection cache of brpc

Path parameters

- `endpoints`
  - `all`: clear all caches
  - `host1:port1,host2:port2`: clear cache of the specified target

Request body

None

Response

```
{
    "msg":"success",
    "code":0,
    "data": "no cached channel.",
    "count":0
}
```

Examples

```
curl http://127.0.0.1:8040/api/reset_rpc_channel/all
```

```
curl http://127.0.0.1:8040/api/reset_rpc_channel/1.1.1.1:8080,2.2.2.2:8080
```

View Compaction Status

View Compaction Status

Request

`GET /api/compaction/run_status GET /api/compaction/show?tablet_id={int}`

Description

It is used to view the overall compaction status of a BE node and the compaction status of a specified tablet.

Query parameters

- `tablet_id` ID of the tablet

Request body

None

Response

The overall compaction status of the node

```
{
  "CumulativeCompaction": {
         "/home/disk1" : [10001, 10002],
         "/home/disk2" : [10003]
  },
  "BaseCompaction": {
         "/home/disk1" : [10001, 10002],
         "/home/disk2" : [10003]
  }
}
```

This structure represents the id of the tablet that is performing the compaction task in a certain data directory, and the type of compaction.

Specify the compaction status of the tablet

```
{
    "cumulative policy type": "SIZE_BASED",
    "cumulative point": 50,
    "last cumulative failure time": "2019-12-16 18:13:43.224",
    "last base failure time": "2019-12-16 18:13:23.320",
    "last cumu success time": ,
    "last base success time": "2019-12-16 18:11:50.780",
    "rowsets": [
```

```
        "[0-48] 10 DATA OVERLAPPING 574.00 MB",
        "[49-49] 2 DATA OVERLAPPING 574.00 B",
        "[50-50] 0 DELETE NONOVERLAPPING 574.00 B",
        "[51-51] 5 DATA OVERLAPPING 574.00 B"
    ],
    "missing_rowsets": [],
    "stale version path": [
        {
            "path id": "2",
            "last create time": "2019-12-16 18:11:15.110 +0800",
            "path list": "2-> [0-24] -> [25-48]"
        },
        {
            "path id": "1",
            "last create time": "2019-12-16 18:13:15.110 +0800",
            "path list": "1-> [25-40] -> [40-48]"
        }
    ]
}
```

Explanation of results:

- cumulative policy type: The cumulative compaction policy type which is used by current tablet.
- cumulative point: The version boundary between base and cumulative compaction. Versions before (excluding) points are handled by base compaction. Versions after (inclusive) are handled by cumulative compaction.
- last cumulative failure time: The time when the last cumulative compaction failed. After 10 minutes by default, cumulative compaction is attempted on the this tablet again.
- last base failure time: The time when the last base compaction failed. After 10 minutes by default, base compaction is attempted on the this tablet again.
- rowsets: The current rowsets collection of this tablet. [0-48] means a rowset with version 0-48. The second number is the number of segments in a rowset. The DELETE indicates the delete version. OVERLAPPING and NONOVERLAPPING indicates whether data between segments is overlap.
- missing_rowset: The missing rowsets.
- stale version path: The merged version path of the rowset collection currently merged in the tablet. It is an array structure and each element represents a merged path. Each element contains three attributes: path id indicates the version path id, and last create time indicates the creation time of the most recent rowset on the path. By default, all rowsets on this path will be deleted after half an hour at the last create time.

Examples

```
curl http://192.168.10.24:8040/api/compaction/show?tablet_id=10015
```

Manually Trigger Compaction

Manually Trigger Compaction

Request

`POST /api/compaction/run?tablet_id={int}&compact_type={enum}` `POST /api/compaction/run?table_id={int}&compact_type=full` Note that table_id=xxx will take effect only when compact_type=full is specified. `GET /api/compaction/run_status?tablet_id={int}`

Description

Used to manually trigger the comparison and show status.

Query parameters

- `tablet_id`
  - ID of the tablet
- `table_id`
  - ID of table. Note that table_id=xxx will take effect only when compact_type=full is specified, and only one tablet_id and table_id can be specified, and cannot be specified at the same time. After specifying table_id, full_compaction will be automatically executed for all tablets under this table.
- `compact_type`
  - The value is `base` or `cumulative` or `full`. For usage scenarios of full_compaction, please refer to Data Recovery.

Request body

None

Response

Trigger Compaction

If the tablet does not exist, an error in JSON format is returned:

```
{
    "status": "Fail",
    "msg": "Tablet not found"
}
```

If the tablet exists and the tablet is not running, JSON format is returned:

```
{
    "status": "Fail",
```

```
    "msg": "fail to execute compaction, error = -2000"
}
```

If the tablet exists and the tablet is running, JSON format is returned:

```
{
    "status": "Success",
    "msg": "compaction task is successfully triggered."
}
```

Explanation of results:

- status: Trigger task status, when it is successfully triggered, it is Success; when for some reason (for example, the appropriate version is not obtained), it returns Fail.
- msg: Give specific success or failure information.

Show Status

If the tablet does not exist, an error in JSON format is returned:

```
{
    "status": "Fail",
    "msg": "Tablet not found"
}
```

If the tablet exists and the tablet is not running, JSON format is returned:

```
{
    "status" : "Success",
    "run_status" : false,
    "msg" : "this tablet_id is not running",
    "tablet_id" : 11308,
    "schema_hash" : 700967178,
    "compact_type" : ""
}
```

If the tablet exists and the tablet is running, JSON format is returned:

```
{
    "status" : "Success",
    "run_status" : true,
    "msg" : "this tablet_id is running",
    "tablet_id" : 11308,
    "schema_hash" : 700967178,
    "compact_type" : "cumulative"
}
```

Explanation of results:

- run_status: Get the current manual compaction task execution status.

Examples

```
curl -X POST "http://127.0.0.1:8040/api/compaction/run?tablet_id=10015&compact_type=cumulative"
```

View Tablet Compaction Score

View Tablet Compaction Score

Request

`GET /api/compaction_score?top_n={int}`

Description

It is used to view tablet compaction scores on a BE node. The compaction score is calculated as the sum of the compaction scores of all rowsets in each tablet. The response is sorted by `compaction_score` in descending order.

Query parameters

- `top_n`

  Optional. A non-negative integer. If specified, only the top N tablets with the highest compaction scores are returned. If not specified, all tablets on the BE node are returned. `top_n=0` returns an empty array.

Request body

None

Response

Returns a JSON array. Each element contains the following fields:

- `tablet_id`: Tablet ID. The value is returned as a string.
- `compaction_score`: Tablet compaction score. The value is returned as a string.

Example response:

```
[
    {
        "compaction_score": "5",
        "tablet_id": "42595"
    },
    {
        "compaction_score": "4",
        "tablet_id": "10034"
    }
]
```

Error handling

If `top_n` is not a valid non-negative integer, the API returns HTTP 400 with an error such as `invalid argument: top_n=wrong`.

Examples

View the top 10 tablets with the highest compaction scores:

```
curl "http://127.0.0.1:8040/api/compaction_score?top_n=10"
```

View Meta

View Meta

Request

`GET /api/meta/header/{tablet_id}?byte_to_base64={bool}`

Description

View meta of a tablet

Path parameters

- `tablet_id` ID of the tablet

Query parameters

- `byte_to_base64` Whether to encode by base64. Optional with default `false`.

Request body

None

Response

```
{
    "table_id": 148107,
    "partition_id": 148104,
    "tablet_id": 148193,
    "schema_hash": 2090621954,
    "shard_id": 38,
    "creation_time": 1673253868,
    "cumulative_layer_point": -1,
    "tablet_state": "PB_RUNNING",
    ...
}
```

Examples

```
curl "http://127.0.0.1:8040/api/meta/header/148193?byte_to_base64=true"
```

Make Snapshot

Make Snapshot

Request

`GET /api/snapshot?tablet_id={int}&schema_hash={int}"`

Description

Make snapshot of a tablet

Query parameters

- `tablet_id` ID of the tablet
- `schema_hash` Schema hash

Request body

None

Response

```
/path/to/snapshot
```

Examples

```
curl "http://127.0.0.1:8040/api/snapshot?tablet_id=123456&schema_hash=1111111"

```

Check All Tablet Segment Lost

Check All Tablet Segment Lost

Request

`GET /api/check_tablet_segment_lost?repair={bool}`

Description

There may be some exceptions that cause segment to be lost on BE node. However, the metadata shows that the tablet is normal. This abnormal replica is not detected by FE and cannot be automatically repaired. When query comes, exception information is thrown that `failed to initialize storage reader`. The function of this interface is to check all tablets on the current BE node that have lost segment.

Query parameters

- `repair`
  - `true`: tablets with lost segment will be set to SHUTDOWN status and treated as bad replica, which can be detected and repaired by FE.
  - `false`: all tablets with missing segment are returned and nothing is done.

Request body

None

Response

```
The return is all tablets on the current BE node that have lost segment:

```
{
    status: "Success",
    msg: "Succeed to check all tablet segment",
    num: 3,
    bad_tablets: [
        11190,
        11210,
        11216
    ],
    set_bad: true,
    host: "172.3.0.101"
}
```
```

Examples

```
curl http://127.0.0.1:8040/api/check_tablet_segment_lost?repair=false
```

Config of BE

Config of BE

Request

`GET /api/show_config POST /api/update_config?{key}={val}`

Description

Query and update the config of BE

Query parameters

- `persist` Whether to persist. Optional with default `false`.
- `key` Config item name
- `val` Config item value

Request body

None

Response

Query

```
[["agent_task_trace_threshold_sec","int32_t","2","true"], ...]
```

Update

```
[
    {
        "config_name": "agent_task_trace_threshold_sec",
        "status": "OK",
        "msg": ""
    }
]
```

```
[
    {
        "config_name": "agent_task_trace_threshold_sec",
        "status": "OK",
        "msg": ""
    },
    {
```

```
        "config_name": "enable_segcompaction",
        "status": "BAD",
        "msg": "set enable_segcompaction=false failed, reason: [NOT_IMPLEMENTED_ERROR]'enable_segcompaction' is not support to
modify."
    },
    {
        "config_name": "enable_time_lut",
        "status": "BAD",
        "msg": "set enable_time_lut=false failed, reason: [NOT_IMPLEMENTED_ERROR]'enable_time_lut' is not support to modify."
    }
]
```

Examples

```
curl "http://127.0.0.1:8040/api/show_config"
```

```
curl -X POST "http://127.0.0.1:8040/api/update_config?agent_task_trace_threshold_sec=2&persist=true"
```

```
curl -X POST "http://127.0.0.1:8040/api/update_config?
agent_task_trace_threshold_sec=2&enable_merge_on_write_correctness_check=true&persist=true"
```

Metrics

Metrics

Request

`GET /metrics?type={enum}&with_tablet={bool}`

Description

Provided for prometheus

Query parameters

- `type` Output style, Optional with default `all` and the following values:
  - `core`: Only core items
  - `json`: Json format
- `with_tablet` Whether to output tablet-related items, Optional with default `false`.

Request body

None

Response

````
```
doris_be__max_network_receive_bytes_rate LONG 60757
doris_be__max_network_send_bytes_rate LONG 16232
doris_be_process_thread_num LONG 1120
doris_be_process_fd_num_used LONG 336
...

```
````

Examples

````
```
    curl "http://127.0.0.1:8040/metrics?type=json&with_tablet=true"
```
````

View Tablet Distribution

View Tablet Distribution

Request

`GET /api/tablets_distribution?group_by={enum}&partition_id={int}`

Description

Get the distribution of tablets under each partition between different disks on BE node

Query parameters

- `group_by` only supports `partition`
- `partition_id` ID of the specified partition, Optional with default all partition.

Request body

None

Response

```
```
{
    msg: "OK",
    code: 0,
    data: {
        host: "***",
        tablets_distribution: [
            {
                partition_id:***,
                disks:[
                    {
                        disk_path:"***",
                        tablets_num:***,
                        tablets:[
                            {
                                tablet_id:***,
                                schema_hash:***,
                                tablet_size:***
                            },

                            ...

                        ]
                    },

                    ...

                ]
```
```

```
            }
        ]
    },
    count: ***
}
```

Examples

```
curl "http://127.0.0.1:8040/api/tablets_distribution?group_by=partition&partition_id=123"

```

Migration Tablet

Migration Tablet

Request

`GET /api/tablet_migration?goal={enum}&tablet_id={int}&schema_hash={int}&disk={string}`

Description

Migrate a tablet to the specified disk.

Query parameters

- `goal`
    - `run`: submit the migration task
    - `status`: show the status of migration task
- `tablet_id` ID of the tablet
- `schema_hash` Schema hash
- `disk` The specified disk.

Request body

None

Response

Submit Task

```
{
    status: "Success",
    msg: "migration task is successfully submitted."
}
```

Or

```
{
    status: "Fail",
    msg: "Migration task submission failed"
}
```

Show Status

```
    {
        status: "Success",
        msg: "migration task is running",
        dest_disk: "xxxxxx"
    }
```

Or

```
    {
        status: "Success",
        msg: "migration task has finished successfully",
        dest_disk: "xxxxxx"
    }
```

Or

```
    {
        status: "Success",
        msg: "migration task failed.",
        dest_disk: "xxxxxx"
    }
```

Examples

```
```
curl "http://127.0.0.1:8040/api/tablet_migration?goal=run&tablet_id=123&schema_hash=333&disk=/disk1"

```
```

View Tablet Info

View Tablet Info

Request

`GET /tablets_json?limit={int}`

Description

View Tablet Info, including ID and schema hash.

Query parameters

- `limit` Number of tablets output, Optional with default 1000. Take `all` to output all tablets.

Request body

None

Response

```
{
    msg: "OK",
    code: 0,
    data: {
        host: "10.38.157.107",
        tablets: [
            {
                tablet_id: 11119,
                schema_hash: 714349777
            },

                ...

            {
                tablet_id: 11063,
                schema_hash: 714349777
            }
        ]
    },
    count: 30
}
```

Examples

```
curl http://127.0.0.1:8040/tablets_json?limit=all

```

Checksum

Checksum

Request

`GET /api/checksum?tablet_id={int}&version={int}&schema_hash={int}`

Description

Checksum

Query parameters

- `tablet_id` ID of the tablet to be checked
- `version` Version of the tablet to be verified
- `schema_hash` Schema hash

Request body

None

Response

```
1843743562
```

Examples

```
curl "http://127.0.0.1:8040/api/checksum?tablet_id=1&version=1&schema_hash=-1"

```

Download Log about Load Error

Download Log about Load Error

Request

`GET /api/_load_error_log?token={string}&file={string}`

Description

Download log file about load error

Query parameters

- `file` Path of log
- `token` token

Request body

None

Response

```
File of log
```

Examples

```
curl "http://127.0.0.1:8040/api/_load_error_log?file=a&token=1"
```

Pad Rowset

Pad Rowset

Request

`POST /api/pad_rowset?tablet_id={int}&start_version={int}&end_version={int}`

Description

Pad one empty rowset as one substitute for error replica.

Query parameters

- `tablet_id` ID of the tablet
- `start_version` Start version
- `end_version` End version

Request body

None

Response

```
{
    msg: "OK",
    code: 0
}
```

Examples

```
curl -X POST "http://127.0.0.1:8040/api/pad_rowset?tablet_id=123456&start_version=1111111&end_version=1111112"

```

Be Version Info

Be Version Info

Request

`GET /api/be_version_info`

Description

Provide BE version info

Query parameters

None

Request body

None

Response

```
{
    "msg":"success",
    "code":0,
    "data":{
        "beVersionInfo":{
            "dorisBuildVersionPrefix":"doris",
            "dorisBuildVersionMajor":0,
            "dorisBuildVersionMinor":0,
            "dorisBuildVersionPatch":0,
            "dorisBuildVersionRcVersion":"trunk",
            "dorisBuildVersion":"doris-0.0.0-trunk",
            "dorisBuildHash":"git://4b7b503d1cb3/data/doris/doris/be/../@a04f9814fe5a09c0d9e9399fe71cc4d765f8bff1",
            "dorisBuildShortHash":"a04f981",
            "dorisBuildTime":"Fri, 09 Sep 2022 07:57:02 UTC",
            "dorisBuildInfo":"root@4b7b503d1cb3"
        }
    },
    "count":0
}
```

Examples

```
curl http://127.0.0.1:8040/api/be_version_info
```

```
```

Check Alive

Check Alive

Request

`GET /api/health`

Description

Provided for the monitoring service to Check whether the BE is alive, Be will respond if alive.

Query parameters

None

Request body

None

Response

```
{"status": "OK","msg": ""}
```

Examples

```
curl http://127.0.0.1:8040/api/health
```

Reload Tablet

Reload Tablet

Request

`GET /api/reload_tablet?tablet_id={int}&schema_hash={int}&path={string}"`

Description

Reload tablet

Query parameters

- `tablet_id` ID of the tablet
- `schema_hash` Schema hash
- `path` Path of file

Request body

None

Response

```
load header succeed
```

Examples

```
curl "http://127.0.0.1:8040/api/reload_tablet?tablet_id=123456&schema_hash=1111111&path=/abc"
```

Restore Tablet

Restore Tablet

Request

`POST /api/restore_tablet?tablet_id={int}&schema_hash={int}"`

Description

To restore the tablet data from trash dir on BE

Query parameters

- `tablet_id` ID of the tablet
- `schema_hash` Schema hash

Request body

None

Response

```
{
    msg: "OK",
    code: 0
}
```

Examples

```
curl -X POST "http://127.0.0.1:8040/api/restore_tablet?tablet_id=123456&schema_hash=1111111"
```

Modify BE VLOG

Request

`POST /api/glog/adjust?module=<module_name>&level=<level_number>`

Description

This function is used to dynamically adjust the VLOG log on the BE side.

Query parameters

- `module_name` Module to set up VLOG, corresponding to BE without suffix filename
- `level_number` VLOG level, from 1 to 10. And -1 for off

Request body

None

Response

````
```json
{
    msg: "adjust vlog of xxx from -1 to 10 succeed",
    code: 0
}
```
````

Examples

````
```bash
curl -X POST "http://127.0.0.1:8040/api/glog/adjust?module=vrow_distribution&level=-1"
```
````

Spark Doris Connector

Spark Doris Connector

Spark Doris Connector can support reading data stored in Doris and writing data to Doris through Spark.

Github: https://github.com/apache/doris-spark-connector

- Support reading data in batch mode from `Doris` through `RDD`, `DataFrame` and `Spark SQL`. It is recommended to use `DataFrame` or `Spark SQL`
- Support writing data to `Doris` in batch or streaming mode with DataFrame API and Spark SQL.
- You can map the `Doris` table to`DataFrame` or `RDD`, it is recommended to use`DataFrame`.
- Support the completion of data filtering on the `Doris` side to reduce the amount of data transmission.

Version Compatibility

Connector	Spark	Doris	Java	Scala
25.2.0	3.5 - 3.1, 2.4	1.0 +	8	2.12, 2.11
25.1.0	3.5 - 3.1, 2.4	1.0 +	8	2.12, 2.11
25.0.1	3.5 - 3.1, 2.4	1.0 +	8	2.12, 2.11
25.0.0	3.5 - 3.1, 2.4	1.0 +	8	2.12, 2.11
24.0.0	3.5 ~ 3.1, 2.4	1.0 +	8	2.12, 2.11
1.3.2	3.4 ~ 3.1, 2.4, 2.3	1.0 ~ 2.1.6	8	2.12, 2.11
1.3.1	3.4 ~ 3.1, 2.4, 2.3	1.0 ~ 2.1.0	8	2.12, 2.11
1.3.0	3.4 ~ 3.1, 2.4, 2.3	1.0 ~ 2.1.0	8	2.12, 2.11
1.2.0	3.2, 3.1, 2.3	1.0 ~ 2.0.2	8	2.12, 2.11
1.1.0	3.2, 3.1, 2.3	1.0 ~ 1.2.8	8	2.12, 2.11
1.0.1	3.1, 2.3	0.12 - 0.15	8	2.12, 2.11

How To Use

Maven

```
<dependency>
    <groupId>org.apache.doris</groupId>
    <artifactId>spark-doris-connector-spark-3.5</artifactId>
    <version>25.2.0</version>
</dependency>
```

Starting from version 24.0.0, the naming rules of the Doris connector package have been adjusted: 1. No longer contains Scala version information. 2. For Spark 2.x versions, use the package named `spark-doris-`

connector-spark-2 uniformly, and by default only compile based on Scala 2.11 version. If you need Scala 2.12 version, please compile it yourself. 3. For Spark 3.x versions, use the package named spark-doris-connector-spark-3.x according to the specific Spark version. Applications based on Spark 3.0 version can use the package spark-doris-connector-spark-3.1.

Note

1. Please replace the corresponding Connector version according to different Spark and Scala versions.
2. You can also download the relevant version jar package from here.

Compile

When compiling, you can directly run sh build.sh, for details, please refer to here.

After successful compilation, the target jar package will be generated in the dist directory, such as: spark-doris-connector-spark-3.5-25.2.0.jar. Copy this file to the ClassPath of Spark to use Spark-Doris-Connector. For example, for Spark running in Local mode, put this file in the jars/ folder. For Spark running in Yarn cluster mode, put this file in the pre-deployment package. You can also

Execute in the source code directory:

sh build.sh

Enter the Scala and Spark versions you need to compile according to the prompts.

After successful compilation, the target jar package will be generated in the dist directory, such as: spark-doris-connector-spark-3.5-25.2.0.jar. Copy this file to the ClassPath of Spark to use Spark-Doris-Connector.

For example, if Spark is running in Local mode, put this file in the jars/ folder. If Spark is running in Yarn cluster mode, put this file in the pre-deployment package.

For example, upload spark-doris-connector-spark-3.5-25.2.0.jar to hdfs and add the Jar package path on hdfs to the spark.yarn.jars parameter

```
1. Upload `spark-doris-connector-spark-3.5-25.2.0.jar` to hdfs.

hdfs dfs -mkdir /spark-jars/
hdfs dfs -put /your_local_path/spark-doris-connector-spark-3.5-25.2.0.jar /spark-jars/

2. Add the `spark-doris-connector-spark-3.5-25.2.0.jar` dependency in the cluster.
spark.yarn.jars=hdfs:///spark-jars/spark-doris-connector-spark-3.5-25.2.0.jar
```

Example

Batch Read

RDD

```
import org.apache.doris.spark._
```

```
val dorisSparkRDD = sc.dorisRDD(
  tableIdentifier = Some("$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME"),
  cfg = Some(Map(
    "doris.fenodes" -> "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT",
    "doris.request.auth.user" -> "$YOUR_DORIS_USERNAME",
    "doris.request.auth.password" -> "$YOUR_DORIS_PASSWORD"
  ))
)

dorisSparkRDD.collect()
```

DataFrame

```
val dorisSparkDF = spark.read.format("doris")
  .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
  .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
  .option("user", "$YOUR_DORIS_USERNAME")
  .option("password", "$YOUR_DORIS_PASSWORD")
  .load()

dorisSparkDF.show(5)
```

Spark SQL

```
CREATE TEMPORARY VIEW spark_doris
USING doris
OPTIONS(
  "table.identifier"="$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME",
  "fenodes"="$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT",
  "user"="$YOUR_DORIS_USERNAME",
  "password"="$YOUR_DORIS_PASSWORD"
);

SELECT * FROM spark_doris;
```

pySpark

```
dorisSparkDF = spark.read.format("doris")
  .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
  .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
  .option("user", "$YOUR_DORIS_USERNAME")
  .option("password", "$YOUR_DORIS_PASSWORD")
  .load()
// show 5 lines data
dorisSparkDF.show(5)
```

Reading via Arrow Flight SQL

Starting from version 24.0.0, data can be read via Arrow Flight SQL (Doris version >= 2.1.0 is required).

Set `doris.read.mode` to arrow, set `doris.read.arrow-flight-sql.port` to the Arrow Flight SQL port configured by FE.

For server configuration, refer to High-speed data transmission link based on Arrow Flight SQL.

```
val df = spark.read.format("doris")
        .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
        .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
        .option("doris.user", "$YOUR_DORIS_USERNAME")
        .option("doris.password", "$YOUR_DORIS_PASSWORD")
        .option("doris.read.mode", "arrow")
        .option("doris.read.arrow-flight-sql.port", "12345")
        .load()

df.show()
```

Batch Write

DataFrame

```
val mockDataDF = List(
  (3, "440403001005", "21.cn"),
  (1, "4404030013005", "22.cn"),
  (33, null, "23.cn")
).toDF("id", "mi_code", "mi_name")
mockDataDF.show(5)

mockDataDF.write.format("doris")
  .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
  .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
  .option("user", "$YOUR_DORIS_USERNAME")
  .option("password", "$YOUR_DORIS_PASSWORD")
  //other options
  //specify the fields to write
  .option("doris.write.fields", "$YOUR_FIELDS_TO_WRITE")
  // Support setting Overwrite mode to overwrite data
  // .mode(SaveMode.Overwrite)
  .save()
```

Spark SQL

```
CREATE TEMPORARY VIEW spark_doris
USING doris
OPTIONS(
  "table.identifier"="$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME",
  "fenodes"="$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT",
  "user"="$YOUR_DORIS_USERNAME",
  "password"="$YOUR_DORIS_PASSWORD"
);

INSERT INTO spark_doris VALUES ("VALUE1", "VALUE2", ...);
-- insert into select
INSERT INTO spark_doris SELECT * FROM YOUR_TABLE;
-- insert overwrite
INSERT OVERWRITE SELECT * FROM YOUR_TABLE;
```

Streaming Write

DataFrame

Write structured data

```
val df = spark.readStream.format("your_own_stream_source").load()

df.writeStream
      .format("doris")
      .option("checkpointLocation", "$YOUR_CHECKPOINT_LOCATION")
      .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
      .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
      .option("user", "$YOUR_DORIS_USERNAME")
      .option("password", "$YOUR_DORIS_PASSWORD")
      .start()
      .awaitTermination()
```

Write directly

If the first column of data in the data stream is formatted data that conforms to the `Doris` table structure, such as CSV format data with the same column order, or JSON format data with the same field name, it can be written directly to `Doris` by setting the `doris.sink.streaming.passthrough` option to `true` without converting to `DataFrame`.

Taking kafka as an example.

And assuming the table structure to be written is:

```
CREATE TABLE `t2` (
`c0` int NULL,
`c1` varchar(10) NULL,
`c2` date NULL
) ENGINE=OLAP
DUPLICATE KEY(`c0`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`c0`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

The value of the message is `{"c0":1,"c1":"a","dt":"2024-01-01"}` in json format.

```
val kafkaSource = spark.readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "$YOUR_KAFKA_SERVERS")
  .option("startingOffsets", "latest")
  .option("subscribe", "$YOUR_KAFKA_TOPICS")
  .load()

// Select the value of the message as the first column of the DataFrame.
kafkaSource.selectExpr("CAST(value as STRING)")
  .writeStream
  .format("doris")
  .option("checkpointLocation", "$YOUR_CHECKPOINT_LOCATION")
```

```
  .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
  .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
  .option("user", "$YOUR_DORIS_USERNAME")
  .option("password", "$YOUR_DORIS_PASSWORD")
  // Set this option to true, and the first column will be written directly without processing.
  .option("doris.sink.streaming.passthrough", "true")
  .option("doris.sink.properties.format", "json")
  .start()
  .awaitTermination()
```

Write in JSON format

Set `doris.sink.properties.format` to json

```
val df = spark.readStream.format("your_own_stream_source").load()
df.write.format("doris")
        .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
        .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
        .option("user", "$YOUR_DORIS_USERNAME")
        .option("password", "$YOUR_DORIS_PASSWORD")
        .option("doris.sink.properties.format", "json")
        .save()
```

Spark Doris Catalog

Since version 24.0.0, support accessing doris through Spark Catalog.

Catalog Config

Key	Required	Comment
spark.sql.catalog.your_catalog_name	true	Set class name of catalog provider, the only valid value for Doris is `org.apache.doris.spark.catalog.DorisTableCatalog`
spark.sql.catalog.your_catalog_name.doris.fenodes	true	Set Doris FE node in the format fe_ip:fe_http_port
spark.sql.catalog.your_catalog_name.doris.query.port	false	Set Doris FE query port, this option is unnecessary if `spark.sql.catalog.your_catalog_name.doris.fe.auto.fetch` is set to true
spark.sql.catalog.your_catalog_name.doris.user	true	Set Doris user
spark.sql.catalog.your_catalog_name.doris.password	true	Set Doris password
spark.sql.defaultCatalog	false	Set Spark SQL default catalog

> **All connector parameters that apply to DataFrame and Spark SQL can be set for catalog.**

> For example, if you want to write data in json format, you can set the option `spark.sql.catalog.your_catalog_name.doris.sink.properties.format` to `json`.

DataFrame

```
val conf = new SparkConf()
conf.set("spark.sql.catalog.your_catalog_name", "org.apache.doris.spark.catalog.DorisTableCatalog")
conf.set("spark.sql.catalog.your_catalog_name.doris.fenodes", "192.168.0.1:8030")
conf.set("spark.sql.catalog.your_catalog_name.doris.query.port", "9030")
conf.set("spark.sql.catalog.your_catalog_name.doris.user", "root")
conf.set("spark.sql.catalog.your_catalog_name.doris.password", "")
val spark = builder.config(conf).getOrCreate()
spark.sessionState.catalogManager.setCurrentCatalog("your_catalog_name")

// show all databases
spark.sql("show databases")

// use databases
spark.sql("use your_doris_db")

// show tables in test
spark.sql("show tables")

// query table
spark.sql("select * from your_doris_table")

// write data
spark.sql("insert into your_doris_table values(xxx)")
```

Spark SQL

Start Spark SQL CLI with necessary config.

```
spark-sql \
--conf "spark.sql.catalog.your_catalog_name=org.apache.doris.spark.catalog.DorisTableCatalog" \
--conf "spark.sql.catalog.your_catalog_name.doris.fenodes=192.168.0.1:8030" \
--conf "spark.sql.catalog.your_catalog_name.doris.query.port=9030" \
--conf "spark.sql.catalog.your_catalog_name.doris.user=root" \
--conf "spark.sql.catalog.your_catalog_name.doris.password=" \
--conf "spark.sql.defaultCatalog=your_catalog_name"
```

Execute query in Spark SQL CLI.

```
-- show all databases
show databases;

-- use databases
use your_doris_db;

-- show tables in test
show tables;

-- query table
select * from your_doris_table;

-- write data
```

```
insert into your_doris_table values(xxx);
insert into your_doris_table select * from your_source_table;

-- access table with full name
select * from your_catalog_name.your_doris_db.your_doris_table;
insert into your_catalog_name.your_doris_db.your_doris_table values(xxx);
insert into your_catalog_name.your_doris_db.your_doris_table select * from your_source_table;
```

Configuration

General

Key	Default Value	Comment
doris.fenodes	–	Doris FE http address, support multiple addresses, separated by commas
doris.table.identifier	–	Doris table identifier, eg, db1.tbl1
doris.user	–	Doris username
doris.password	Empty string	Doris password
doris.request.retries	3	Number of retries to send requests to Doris

Key	DefaultValue	Comment
doris.request.connect.timeout.ms	30000	Connection timeout for sending requests to Doris
doris.request.read.timeout.ms	300000	Read timeout for sending request to Doris
doris.request.query.timeout.s	21600	Query the timeout time of doris, the default is 6 hour, -1 means no timeout limit
doris.request.tablet.size	1	The number of Doris Tablets corresponding to an RDD Partition. The smaller this value is set, the more partitions will be generated. This will increase the parallelism on the Spark side, but at the same time will cause greater pressure on Doris.
doris.read.field	–	List of column names in the Doris table, separated by commas
doris.batch.size	4064	The maximum number of rows to read data from BE at one time. Increasing this value can reduce the number of connections between Spark and Doris. Thereby reducing the extra time overhead caused by network delay.
doris.exec.mem.limit	8589934592	Memory limit for a single query. The default is 8GB, in bytes.

Key	DefaultValue	Comment
doris.write.fields	–	Specifies the fields (or the order of the fields) to write to the Doris table, fileds separated by commas. By default, all fields are written in the order of Doris table fields.
doris.sink.batch.size	500000	Maximum number of lines in a single write BE
doris.sink.max-retries	0	Number of retries after writing BE, Since version 1.3.0, the default value is 0, which means no retries are performed by default. When this parameter is set greater than 0, batch-level failure retries will be performed, and data of the configured size of `doris.sink.batch.size` will be cached in the Spark Executor memory. The memory allocation may need to be appropriately increased.
doris.sink.retry.interval.ms	10000	After configuring the number of retries, the interval between each retry, in ms
doris.sink.properties.format	–	Data format of the stream load. Supported formats: csv, json, arrow More Multi-parameter details
doris.sink.properties.*	–	Import parameters for Stream Load. For example: Specify column separator: `'doris.sink.properties.column_separator' = ','`. More parameter details
doris.sink.task.partition.size	–	The number of partitions corresponding to the Writing task. After filtering and other operations, the number of partitions written in Spark RDD may be large, but the number of records corresponding to each Partition is relatively small, resulting in increased writing frequency and waste of computing resources. The smaller this value is set, the less Doris write frequency and less Doris merge pressure. It is generally used with doris.sink.task.use.repartition.
doris.sink.task.use.repartition	false	Whether to use repartition mode to control the number of partitions written by Doris. The default value is false, and coalesce is used (note: if there is no Spark action before the write, the whole computation will be less parallel). If it is set to true, then repartition is used (note: you can set the final number of partitions at the cost of shuffle).

Key	DefaultValue	Comment
doris.sink.batch.interval.ms	0	The interval time of each batch sink, unit ms.
doris.sink.enable-2pc	false	Whether to enable two-stage commit. When enabled, transactions will be committed at the end of the job, and all pre-commit transactions will be rolled back when some tasks fail.
doris.sink.auto-redirect	true	Whether to redirect StreamLoad requests. After being turned on, StreamLoad will write through FE and no longer obtain BE information explicitly.
doris.enable.https	false	Whether to enable FE Https request.
doris.https.key-store-path	-	Https key store path.
doris.https.key-store-type	JKS	Https key store type.
doris.https.key-store-password	-	Https key store password.

Key	DefaultValue	Comment
doris.read.mode	thrift	Doris read mode, with optional `thrift` and `arrow`.
doris.read.arrow-flight-sql.port	-	Arrow Flight SQL port of Doris FE. When `doris.read.mode` is `arrow`, it is used to read data via Arrow Flight SQL. For server configuration, see High-speed data transmission link based on Arrow Flight SQL
doris.sink.label.prefix	spark-doris	The import label prefix when writing in Stream Load mode.
doris.thrift.max.message.size	2147483647	The maximum size of a message when reading data via Thrift.
doris.fe.auto.fetch	false	Whether to automatically obtain FE information. When set to true, all FE node information will be requested according to the nodes configured by `doris.fenodes`. There is no need to configure multiple nodes and configure `doris.read.arrow-flight-sql.port` and `doris.query.port` separately.

Key	Default Value	Comment
doris.read.bitmap-to-string	false	Whether to convert the Bitmap type to a string composed of array indexes for reading. For the specific result format, see the function definition BITMAP_TO_STRING.
doris.read.bitmap-to-base64	false	Whether to convert the Bitmap type to a Base64-encoded string for reading. For the specific result format, see the function definition BITMAP_TO_BASE64.
doris.query.port	-	Doris FE query port, used for overwriting and obtaining metadata of the Catalog.

SQL & Dataframe Configuration

Key	Default Value	Comment
doris.filter.query.in.max.count	100	In the predicate pushdown, the maximum number of elements in the in expression value list. If this number is exceeded, the in-expression conditional filtering is processed on the Spark side.

Structured Streaming Configuration

Key	Default Value	Comment
doris.sink.streaming.passthrough	false	Write the value of the first column directly without processing.

RDD Configuration

Key	Default Value	Comment
doris.request.auth.user	–	Doris username
doris.request.auth.password	–	Doris password

Key	Default Value	Comment
doris.filter.query	–	Filter expression of the query, which is transparently transmitted to Doris. Doris uses this expression to complete source-side data filtering.

Data Type Mapping from Doris to Spark

Doris Type	Spark Type
NULL_TYPE	DataTypes.NullType
BOOLEAN	DataTypes.BooleanType
TINYINT	DataTypes.ByteType
SMALLINT	DataTypes.ShortType
INT	DataTypes.IntegerType
BIGINT	DataTypes.LongType
FLOAT	DataTypes.FloatType
DOUBLE	DataTypes.DoubleType
DATE	DataTypes.DateType
DATETIME	DataTypes.TimestampType
DECIMAL	DecimalType
CHAR	DataTypes.StringType
LARGEINT	DecimalType
VARCHAR	DataTypes.StringType
STRING	DataTypes.StringType
JSON	DataTypes.StringType
VARIANT	DataTypes.StringType
TIME	DataTypes.DoubleType
HLL	DataTypes.StringType
Bitmap	DataTypes.StringType

Data Type Mapping from Spark to Doris

Spark Type	Doris Type
BooleanType	BOOLEAN
ShortType	SMALLINT
IntegerType	INT
LongType	BIGINT
FloatType	FLOAT

Spark Type	Doris Type
DoubleType	DOUBLE
DecimalType	DECIMAL
StringType	VARCHAR/STRING
DateType	DATE
TimestampType	DATETIME
ArrayType	ARRAY
MapType	MAP/JSON
StructType	STRUCT/JSON

> **Since version 24.0.0, the return type of the Bitmap type is string type, and the default return value is string value** `Read unsupported`**.**

FAQ

1. How to write Bitmap type

   In Spark SQL, when writing data through insert into, if the target table of doris contains data of type `BITMAP` or `HLL`, you need to set the option `doris.ignore-type` to the corresponding type and map the columns through `doris.write.fields`. The usage is as follows:

   BITMAP
   ```
   sql CREATE TEMPORARY VIEW spark_doris USING doris OPTIONS(
   "table.identifier"="$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME",
   "fenodes"="$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT", "user"="$YOUR_DORIS_USERNAME",
   "password"="$YOUR_DORIS_PASSWORD" "doris.ignore-type"="bitmap",
   "doris.write.fields"="col1,col2,col3,bitmap_col2=to_bitmap(col2),bitmap_col3=bitmap_hash(col3)" );
   ```
   HLL
   ```
   sql CREATE TEMPORARY VIEW spark_doris USING doris OPTIONS(
   "table.identifier"="$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME",
   "fenodes"="$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT", "user"="$YOUR_DORIS_USERNAME",
   "password"="$YOUR_DORIS_PASSWORD" "doris.ignore-type"="hll",
   "doris.write.fields"="col1,hll_col1=hll_hash(col1)" );
   ```

   Since version 24.0.0, `doris.ignore-type` has been deprecated and there is no need to add this parameter when writing.

2. **How to use overwrite to write?**

   Since version 1.3.0, overwrite mode writing is supported (only supports data overwriting at the full table level). The specific usage is as follows:

   DataFrame
   ```
   scala resultDf.format("doris")
   .option("doris.fenodes","$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT") // your own options
   .mode(SaveMode.Overwrite) .save()
   ```

SQL `sparksql INSERT OVERWRITE your_target_table SELECT * FROM your_source_table`

3. **How to read Bitmap type**

   Starting from version 24.0.0, it supports reading converted Bitmap data through Arrow Flight SQL (Doris version >= 2.1.0 is required).

   Bitmap to string

   `DataFrame` example is as follows, set `doris.read.bitmap-to-string` to true. For the specific result format, see the option definition.

   ```
   spark.read.format("doris")
   .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
   .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
   .option("user", "$YOUR_DORIS_USERNAME")
   .option("password", "$YOUR_DORIS_PASSWORD")
   .option("doris.read.bitmap-to-string","true")
   .load()
   ```

   Bitmap to base64

   `DataFrame` example is as follows, set `doris.read.bitmap-to-base64` to true. For the specific result format, see the option definition.

   ```
   spark.read.format("doris")
   .option("doris.table.identifier", "$YOUR_DORIS_DATABASE_NAME.$YOUR_DORIS_TABLE_NAME")
   .option("doris.fenodes", "$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT")
   .option("user", "$YOUR_DORIS_USERNAME")
   .option("password", "$YOUR_DORIS_PASSWORD")
   .option("doris.read.bitmap-to-base64","true")
   .load()
   ```

4. **An error occurs when writing in DataFrame mode:** `org.apache.spark.sql.AnalysisException: TableProvider implementation doris cannot be written with ErrorIfExists mode, please use Append or Overwrite modes instead.`

   Need to add save mode to append. `scala resultDf.format("doris") .option("doris.fenodes","$YOUR_DORIS_FE_HOSTNAME:$YOUR_DORIS_FE_RESFUL_PORT") // your own options .mode(SaveMode.Append) .save()`

Flink Doris Connector

The Flink Doris Connector is used to read from and write data to a Doris cluster through Flink. It also integrates FlinkCDC, which allows for more convenient full database synchronization with upstream databases such as MySQL.

Using the Flink Connector, you can perform the following operations:

- **Read data from Doris**: Flink Connector supports parallel reading from BE, improving data retrieval efficiency.
- **Write data to Doris**: After batching in Flink, data is imported into Doris in bulk using Stream Load.
- **Perform dimension table joins with Lookup Join**: Batching and asynchronous queries accelerate dimension table joins.
- **Full database synchronization**: Using Flink CDC, you can synchronize entire databases such as MySQL, Oracle, and PostgreSQL, including automatic table creation and DDL operations.

Version Description

Connector Version	Flink Version	Doris Version	Java Version	Scala Version
1.0.3	1.11,1.12,1.13,1.14	0.15+	8	2.11,2.12
1.1.1	1.14	1.0+	8	2.11,2.12
1.2.1	1.15	1.0+	8	-
1.3.0	1.16	1.0+	8	-
1.4.0	1.15 - 1.17	1.0+	8	-
1.5.2	1.15 - 1.18	1.0+	8	-
1.6.1	1.15 - 1.19	1.0+	8	-
24.0.1	1.15 - 1.20	1.0+	8	-
24.1.0	1.15 - 1.20	1.0+	8	-
25.0.0	1.15 - 1.20	1.0+	8	-
25.1.0	1.15 - 1.20	1.0+	8	-
26.0.0	1.15 - 1.20,2.0 - 2.2	1.0+	8(1.x),17(2.x)	-

Usage

The Flink Doris Connector can be used in two ways: via Jar or Maven.

Jar

You can download the corresponding version of the Flink Doris Connector Jar file here, then copy this file to the `classpath` of your `Flink` setup to use the `Flink-Doris-Connector`. For a `Standalone` mode Flink deployment, place this file under the `lib/` folder. For a Flink cluster running in `Yarn` mode, place the file into the pre-deployment package.

Maven

To use it with Maven, simply add the following dependency to your Pom file:

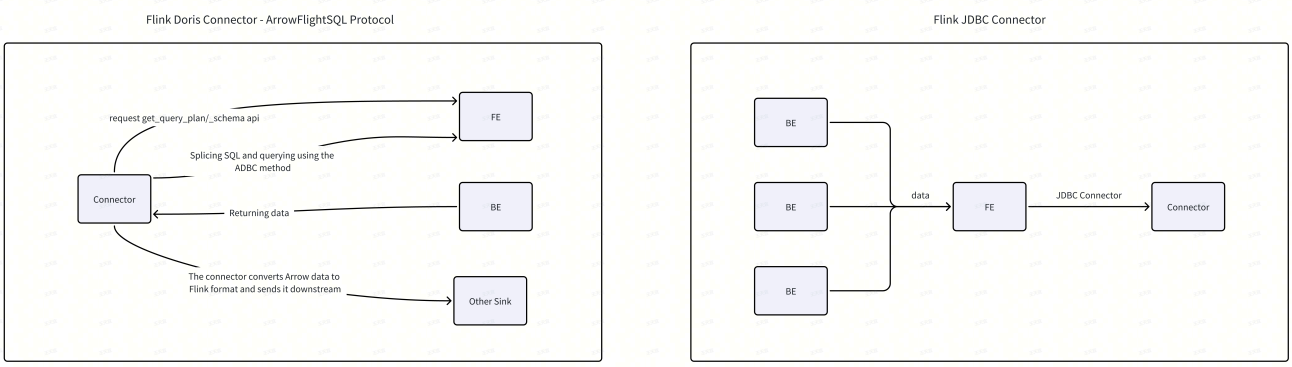
```
<dependency>
  <groupId>org.apache.doris</groupId>
  <artifactId>flink-doris-connector-${flink.version}</artifactId>
  <version>${connector.version}</version>
</dependency>
```

For example:

```
<dependency>
  <groupId>org.apache.doris</groupId>
  <artifactId>flink-doris-connector-1.16</artifactId>
  <version>25.1.0</version>
</dependency>
```

Working Principles

Reading Data from Doris



FlinkConnectorPrinciples-JDBC-Doris

When reading data, Flink Doris Connector offers higher performance compared to Flink JDBC Connector and is recommended for use:

- **Flink JDBC Connector**: Although Doris is compatible with the MySQL protocol, using Flink JDBC Connector for reading and writing to a Doris cluster is not recommended. This approach results in serial read/write operations on a single FE node, creating a bottleneck and affecting performance.
- **Flink Doris Connector**: Starting from Doris 2.1, ADBC is the default protocol for Flink Doris Connector. The reading process follows these steps:

a. Flink Doris Connector first retrieves Tablet ID information from FE based on the query plan.

b. It generates the query statement: `SELECT * FROM tbs TABLET(id1, id2, id3)`.

c. The query is then executed through the ADBC port of FE.

d. Data is returned directly from BE, bypassing FE to eliminate the single-point bottleneck.

Writing Data to Doris

When using Flink Doris Connector for data writing, batch processing is performed in Flink's memory before bulk import via Stream Load. Doris Flink Connector provides two batching modes, with Flink Checkpoint-based streaming writes as the default:

	Streaming Write	Batch Write
Trigger Condition	Relies on Flink Checkpoints and follows Flink's checkpoint cycle to write to Doris	Periodic submission based on connector-defined time or data volume thresholds
Consistency	Exactly-Once	At-Least-Once; Exactly-Once can be ensured with the primary key model
Latency	Limited by the Flink checkpoint interval, generally higher	Independent batch mechanism with flexible adjustment
Fault Tolerance & Recovery	Fully consistent with Flink state recovery	Relies on external deduplication logic (e.g., Doris primary key deduplication)

Quick Start

Preparation

Flink Cluster Deployment

Taking a Standalone cluster as an example:

1. Download the Flink installation package, e.g., Flink 1.18.1;
2. After extraction, place the Flink Doris Connector package in `<FLINK_HOME>/lib`;
3. Navigate to the `<FLINK_HOME>` directory and run `bin/start-cluster.sh` to start the Flink cluster;
4. You can verify if the Flink cluster started successfully using the `jps` command.

Initialize Doris Tables

Run the following statements to create Doris tables:

```
CREATE DATABASE test;

CREATE TABLE test.student (
  `id` INT,
  `name` VARCHAR(256),
  `age` INT
)
UNIQUE KEY(`id`)
```

```
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);

INSERT INTO test.student values(1,"James",18);
INSERT INTO test.student values(2,"Emily",28);

CREATE TABLE test.student_trans (
  `id` INT,
  `name` VARCHAR(256),
  `age` INT
)
UNIQUE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 3"
);
```

Run FlinkSQL Task

Start FlinkSQL Client

```
bin/sql-client.sh
```

Run FlinkSQL

```
CREATE TABLE Student (
    id STRING,
    name STRING,
    age INT
    )
    WITH (
      'connector' = 'doris',
      'fenodes' = '127.0.0.1:8030',
      'table.identifier' = 'test.student',
      'username' = 'root',
      'password' = ''
);

CREATE TABLE StudentTrans (
    id STRING,
    name STRING,
    age INT
    )
    WITH (
      'connector' = 'doris',
      'fenodes' = '127.0.0.1:8030',
      'table.identifier' = 'test.student_trans',
      'username' = 'root',
      'password' = '',
      'sink.label-prefix' = 'doris_label'
);

INSERT INTO StudentTrans SELECT id, concat('prefix_',name), age+1 FROM Student;
```

Query Data

```
mysql> select * from test.student_trans;
+------+--------------+------+
| id   | name         | age  |
+------+--------------+------+
|    1 | prefix_James |   19 |
|    2 | prefix_Emily |   29 |
+------+--------------+------+
2 rows in set (0.02 sec)
```

Scenarios and Operations

Reading Data from Doris

When Flink reads data from Doris, the Doris Source is currently a bounded stream and does not support continuous reading in a CDC manner. Data can be read from Doris using Thrift or ArrowFlightSQL (supported from version 24.0.0 onward). Starting from version 2.1, ArrowFlightSQL is the recommended approach.

- **Thrift**: Data is read by calling the BE's Thrift interface. For detailed steps, refer to Reading Data via Thrift Interface.
- **ArrowFlightSQL**: Based on Doris 2.1, this method allows high-speed reading of large volumes of data using the Arrow Flight SQL protocol. For more information, refer to High-speed Data Transfer via Arrow Flight SQL.

Using FlinkSQL to Read Data

Thrift Method

```
CREATE TABLE student (
    id INT,
    name STRING,
    age INT
    )
    WITH (
      'connector' = 'doris',
      'fenodes' = '127.0.0.1:8030',  -- Fe的host:HttpPort
      'table.identifier' = 'test.student',
      'username' = 'root',
      'password' = ''
);

SELECT * FROM student;
```

ArrowFlightSQL

```
CREATE TABLE student (
    id INT,
    name STRING,
    age INT
```

```
    )
    WITH (
      'connector' = 'doris',
      'fenodes' = '{fe.conf:http_port}',
      'table.identifier' = 'test.student',
      'source.use-flight-sql' = 'true',
      'source.flight-sql-port' = '{fe.conf:arrow_flight_sql_port}',
      'username' = 'root',
      'password' = ''
);

SELECT * FROM student;
```

Using DataStream API to Read Data

When using the DataStream API to read data, you need to include the dependencies in your program's POM file in advance, as described in the "Usage" section.

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
DorisOptions option = DorisOptions.builder()
        .setFenodes("127.0.0.1:8030")
        .setTableIdentifier("test.student")
        .setUsername("root")
        .setPassword("")
        .build();

DorisReadOptions readOptions = DorisReadOptions.builder().build();
DorisSource<List<?>> dorisSource = DorisSource.<List<?>>builder()
        .setDorisOptions(option)
        .setDorisReadOptions(readOptions)
        .setDeserializer(new SimpleListDeserializationSchema())
        .build();

env.fromSource(dorisSource, WatermarkStrategy.noWatermarks(), "doris source").print();
env.execute("Doris Source Test");
```

For the complete code, refer to:DorisSourceDataStream.java

Writing Data to Doris

Flink writes data to Doris using the Stream Load method, supporting both streaming and batch-insertion modes.

> **Difference Between Streaming and Batch-insertion**
>
> Starting from Connector 1.5.0, batch-insertion is supported. Batch-insertion does not rely on Checkpoints; it buffers data in memory and controls the writing timing based on batch parameters. Streaming insertion requires Checkpoints to be enabled, continuously writing upstream data to Doris during the entire Checkpoint period, without keeping data in memory continuously.

Using FlinkSQL to Write Data

For testing, Flink's Datagen is used to simulate the continuously generated upstream data.

```
-- enable checkpoint
SET 'execution.checkpointing.interval' = '30s';

CREATE TABLE student_source (
    id INT,
    name STRING,
    age INT
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1',
  'fields.name.length' = '20',
  'fields.id.min' = '1',
  'fields.id.max' = '100000',
  'fields.age.min' = '3',
  'fields.age.max' = '30'
);

-- doris sink
CREATE TABLE student_sink (
    id INT,
    name STRING,
    age INT
    )
    WITH (
      'connector' = 'doris',
      'fenodes' = '10.16.10.6:28737',
      'table.identifier' = 'test.student',
      'username' = 'root',
      'password' = 'password',
      'sink.label-prefix' = 'doris_label'
      --'sink.enable.batch-mode' = 'true'  Adding this configuration enables batch writing
);

INSERT INTO student_sink SELECT * FROM student_source;
```

Using DataStream API to Write Data

When using the DataStream API to write data, different serialization methods can be used to serialize the upstream data before writing it to the Doris table.

> **The Connector already contains the HttpClient4.5.13 version. If you reference HttpClient separately in your project, you need to ensure that the versions are consistent.**

Standard String Format

When the upstream data is in CSV or JSON format, you can directly use the `SimpleStringSerializer` to serialize the data.

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.enableCheckpointing(30000);
DorisSink.Builder<String> builder = DorisSink.builder();

DorisOptions dorisOptions = DorisOptions.builder()
        .setFenodes("10.16.10.6:28737")
        .setTableIdentifier("test.student")
        .setUsername("root")
        .setPassword("")
        .build();

Properties properties = new Properties();
// When the upstream data is in json format, the following configuration needs to be enabled
properties.setProperty("read_json_by_line", "true");
properties.setProperty("format", "json");

// When writing csv data from the upstream, the following configurations need to be enabled
//properties.setProperty("format", "csv");
//properties.setProperty("column_separator", ",");

DorisExecutionOptions executionOptions = DorisExecutionOptions.builder()
       .setLabelPrefix("label-doris")
       .setDeletable(false)
       //.setBatchMode(true)  Enable batch writing
       .setStreamLoadProp(properties)
       .build();

builder.setDorisReadOptions(DorisReadOptions.builder().build())
       .setDorisExecutionOptions(executionOptions)
       .setSerializer(new SimpleStringSerializer())
       .setDorisOptions(dorisOptions);

List<String> data = new ArrayList<>();
data.add("{\"id\":3,\"name\":\"Michael\",\"age\":28}");
data.add("{\"id\":4,\"name\":\"David\",\"age\":38}");

env.fromCollection(data).sinkTo(builder.build());
env.execute("doris test");
```

For the complete code, refer to:DorisSinkExample.java

RowData Format

RowData is the internal format of Flink. If the upstream data is in RowData format, you need to use the `RowDataSerializer` to serialize the data.

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.enableCheckpointing(10000);
env.setParallelism(1);

DorisSink.Builder<RowData> builder = DorisSink.builder();

Properties properties = new Properties();
properties.setProperty("column_separator", ",");
properties.setProperty("line_delimiter", "\n");
properties.setProperty("format", "csv");
// When writing json data from the upstream, the following configuration needs to be enabled
// properties.setProperty("read_json_by_line", "true");
// properties.setProperty("format", "json");
DorisOptions.Builder dorisBuilder = DorisOptions.builder();
```

```
dorisBuilder
        .setFenodes("10.16.10.6:28737")
        .setTableIdentifier("test.student")
        .setUsername("root")
        .setPassword("");
DorisExecutionOptions.Builder executionBuilder = DorisExecutionOptions.builder();
executionBuilder.setLabelPrefix(UUID.randomUUID().toString()).setDeletable(false).setStreamLoadProp(properties);

// flink rowdata ‘s schema
String[] fields = {"id","name", "age"};
DataType[] types = {DataTypes.INT(), DataTypes.VARCHAR(256), DataTypes.INT()};

builder.setDorisExecutionOptions(executionBuilder.build())
        .setSerializer(
                RowDataSerializer.builder() // serialize according to rowdata
                        .setType(LoadConstants.CSV)
                        .setFieldDelimiter(",")
                        .setFieldNames(fields)
                        .setFieldType(types)
                        .build())
        .setDorisOptions(dorisBuilder.build());

// mock rowdata source
DataStream<RowData> source =
        env.fromElements("")
                .flatMap(
                        new FlatMapFunction<String, RowData>() {
                            @Override
                            public void flatMap(String s, Collector<RowData> out)
                                    throws Exception {
                                GenericRowData genericRowData = new GenericRowData(3);
                                genericRowData.setField(0, 1);
                                genericRowData.setField(1, StringData.fromString("Michael"));
                                genericRowData.setField(2, 18);
                                out.collect(genericRowData);

                                GenericRowData genericRowData2 = new GenericRowData(3);
                                genericRowData2.setField(0, 2);
                                genericRowData2.setField(1, StringData.fromString("David"));
                                genericRowData2.setField(2, 38);
                                out.collect(genericRowData2);
                            }
                        });

source.sinkTo(builder.build());
env.execute("doris test");
```

For the complete code, refer to:DorisSinkExampleRowData.java

Debezium Format

For upstream data in Debezium format, such as data from FlinkCDC or Debezium format in Kafka, you can use the `JsonDebeziumSchemaSerializer` to serialize the data.

```
// enable checkpoint
env.enableCheckpointing(10000);

Properties props = new Properties();
props.setProperty("format", "json");
props.setProperty("read_json_by_line", "true");
DorisOptions dorisOptions = DorisOptions.builder()
```

```
        .setFenodes("127.0.0.1:8030")
        .setTableIdentifier("test.student")
        .setUsername("root")
        .setPassword("").build();

DorisExecutionOptions.Builder  executionBuilder = DorisExecutionOptions.builder();
executionBuilder.setLabelPrefix("label-prefix")
        .setStreamLoadProp(props)
        .setDeletable(true);

DorisSink.Builder<String> builder = DorisSink.builder();
builder.setDorisReadOptions(DorisReadOptions.builder().build())
        .setDorisExecutionOptions(executionBuilder.build())
        .setDorisOptions(dorisOptions)
        .setSerializer(JsonDebeziumSchemaSerializer.builder().setDorisOptions(dorisOptions).build());

env.fromSource(mySqlSource, WatermarkStrategy.noWatermarks(), "MySQL Source")
        .sinkTo(builder.build());
```

For the complete code, refer to:CDCSchemaChangeExample.java

Multi-table Write Format

Currently, DorisSink supports synchronizing multiple tables with a single Sink. You need to pass both the data and the database/table information to the Sink, and serialize it using the `RecordWithMetaSerializer`.

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setParallelism(1);
DorisSink.Builder<RecordWithMeta> builder = DorisSink.builder();
Properties properties = new Properties();
properties.setProperty("column_separator", ",");
properties.setProperty("line_delimiter", "\n");
properties.setProperty("format", "csv");
DorisOptions.Builder dorisBuilder = DorisOptions.builder();
dorisBuilder
        .setFenodes("10.16.10.6:28737")
        .setTableIdentifier("")
        .setUsername("root")
        .setPassword("");

DorisExecutionOptions.Builder executionBuilder = DorisExecutionOptions.builder();

executionBuilder
        .setLabelPrefix("label-doris")
        .setStreamLoadProp(properties)
        .setDeletable(false)
        .setBatchMode(true);

builder.setDorisReadOptions(DorisReadOptions.builder().build())
        .setDorisExecutionOptions(executionBuilder.build())
        .setDorisOptions(dorisBuilder.build())
        .setSerializer(new RecordWithMetaSerializer());

RecordWithMeta record = new RecordWithMeta("test", "student_1", "1,David,18");
RecordWithMeta record1 = new RecordWithMeta("test", "student_2", "1,Jack,28");
env.fromCollection(Arrays.asList(record, record1)).sinkTo(builder.build());
```

For the complete code, refer to:DorisSinkMultiTableExample.java

Lookup Join

Using Lookup Join can optimize dimension table joins in Flink. When using Flink JDBC Connector for dimension table joins, the following issues may arise:

- Flink JDBC Connector uses a synchronous query mode, meaning that after upstream data (e.g., from Kafka) sends a record, it immediately queries the Doris dimension table. This results in high query latency under high-concurrency scenarios.
- Queries executed via JDBC are typically point lookups per record, whereas Doris recommends batch queries for better efficiency.

Using Lookup Join for dimension table joins in Flink Doris Connector provides the following advantages:

- **Batch caching of upstream data**, avoiding the high latency and database load caused by per-record queries.
- **Asynchronous execution of join queries**, improving data throughput and reducing the query load on Doris.

```
CREATE TABLE fact_table (
  `id` BIGINT,
  `name` STRING,
  `city` STRING,
  `process_time` as proctime()
) WITH (
  'connector' = 'kafka',
  ...
);

create table dim_city(
  `city` STRING,
  `level` INT ,
  `province` STRING,
  `country` STRING
) WITH (
  'connector' = 'doris',
  'fenodes' = '127.0.0.1:8030',
  'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
  'table.identifier' = 'dim.dim_city',
  'username' = 'root',
  'password' = '',
  'lookup.cache.max-rows' = '100000',
  'lookup.cache.ttl' = '30Os'
);

SELECT a.id, a.name, a.city, c.province, c.country,c.level
FROM fact_table a
LEFT JOIN dim_city FOR SYSTEM_TIME AS OF a.process_time AS c
ON a.city = c.city
```

Full Database Synchronization

The Flink Doris Connector integrates **Flink CDC** (Flink CDC Documentation), making it easier to synchronize relational databases like MySQL to Doris. This integration also includes automatic table creation, schema

changes, etc. Supported databases for synchronization include: MySQL, Oracle, PostgreSQL, SQLServer, MongoDB, and DB2.

Note

1. When using full database synchronization, you need to add the corresponding Flink CDC dependencies in the `$FLINK_HOME/lib` directory (Fat Jar), such as **flink-sql-connector-mysql-cdc-**$version.jar**,**$ **flink-sql-connector-oracle-cdc-{version}.jar**. FlinkCDC version 3.1 and later is not compatible with previous versions. You can download the dependencies from the following links: FlinkCDC 3.x, FlinkCDC 2.x.
2. For versions after Connector 24.0.0, the required Flink CDC version must be 3.1 or higher. You can download it here. If Flink CDC is used to synchronize MySQL and Oracle, you must also add the relevant JDBC drivers under `$FLINK_HOME/lib`.

MySQL Whole Database Synchronization

After starting the Flink cluster, you can directly run the following command:

```
<FLINK_HOME>bin/flink run \
    -Dexecution.checkpointing.interval=10s \
    -Dparallelism.default=1 \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    lib/flink-doris-connector-1.16-24.0.1.jar \
    mysql-sync-database \
    --database test_db \
    --mysql-conf hostname=127.0.0.1 \
    --mysql-conf port=3306 \
    --mysql-conf username=root \
    --mysql-conf password=123456 \
    --mysql-conf database-name=mysql_db \
    --including-tables "tbl1|test.*" \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password=123456 \
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

Oracle Whole Database Synchronization

```
<FLINK_HOME>bin/flink run \
     -Dexecution.checkpointing.interval=10s \
     -Dparallelism.default=1 \
     -c org.apache.doris.flink.tools.cdc.CdcTools \
     ./lib/flink-doris-connector-1.16-24.0.1.jar \
     oracle-sync-database \
     --database test_db \
     --oracle-conf hostname=127.0.0.1 \
     --oracle-conf port=1521 \
     --oracle-conf username=admin \
     --oracle-conf password="password" \
     --oracle-conf database-name=XE \
     --oracle-conf schema-name=ADMIN \
     --including-tables "tbl1|tbl2" \
```

```
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password=\
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

PostgreSQL Whole Database Synchronization

```
<FLINK_HOME>/bin/flink run \
    -Dexecution.checkpointing.interval=10s \
    -Dparallelism.default=1\
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    ./lib/flink-doris-connector-1.16-24.0.1.jar \
    postgres-sync-database \
    --database db1\
    --postgres-conf hostname=127.0.0.1 \
    --postgres-conf port=5432 \
    --postgres-conf username=postgres \
    --postgres-conf password="123456" \
    --postgres-conf database-name=postgres \
    --postgres-conf schema-name=public \
    --postgres-conf slot.name=test \
    --postgres-conf decoding.plugin.name=pgoutput \
    --including-tables "tbl1|tbl2" \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password=\
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

SQLServer Whole Database Synchronization

```
<FLINK_HOME>/bin/flink run \
    -Dexecution.checkpointing.interval=10s \
    -Dparallelism.default=1 \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    ./lib/flink-doris-connector-1.16-24.0.1.jar \
    sqlserver-sync-database \
    --database db1\
    --sqlserver-conf hostname=127.0.0.1 \
    --sqlserver-conf port=1433 \
    --sqlserver-conf username=sa \
    --sqlserver-conf password="123456" \
    --sqlserver-conf database-name=CDC_DB \
    --sqlserver-conf schema-name=dbo \
    --including-tables "tbl1|tbl2" \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password=\
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

DB2 Whole Database Synchronization

```
<FLINK_HOME>bin/flink run \
    -Dexecution.checkpointing.interval=10s \
    -Dparallelism.default=1 \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    lib/flink-doris-connector-1.16-24.0.1.jar \
    db2-sync-database \
    --database db2_test \
    --db2-conf hostname=127.0.0.1 \
    --db2-conf port=50000 \
    --db2-conf username=db2inst1 \
    --db2-conf password=doris123456 \
    --db2-conf database-name=testdb \
    --db2-conf schema-name=DB2INST1 \
    --including-tables "FULL_TYPES|CUSTOMERS" \
    --single-sink true \
    --use-new-schema-change true \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password=123456 \
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

MongoDB Whole Database Synchronization

```
<FLINK_HOME>/bin/flink run \
    -Dexecution.checkpointing.interval=10s \
    -Dparallelism.default=1 \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    ./lib/flink-doris-connector-1.18-24.0.1.jar \
    mongodb-sync-database \
    --database doris_db \
    --schema-change-mode debezium_structure \
    --mongodb-conf hosts=127.0.0.1:27017 \
    --mongodb-conf username=flinkuser \
    --mongodb-conf password=flinkpwd \
    --mongodb-conf database=test \
    --mongodb-conf scan.startup.mode=initial \
    --mongodb-conf schema.sample-percent=0.2 \
    --including-tables "tbl1|tbl2" \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password= \
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --sink-conf sink.enable-2pc=false \
    --table-conf replication_num=1
```

AWS Aurora MySQL Whole Database Synchronization

```
<FLINK_HOME>bin/flink run \
    -Dexecution.checkpointing.interval=10s \
    -Dparallelism.default=1 \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    lib/flink-doris-connector-1.18-25.0.0.jar \
    mysql-sync-database \
```

```
    --database testwd \
    --mysql-conf hostname=xxx.us-east-1.rds.amazonaws.com \
    --mysql-conf port=3306 \
    --mysql-conf username=admin \
    --mysql-conf password=123456 \
    --mysql-conf database-name=test \
    --mysql-conf server-time-zone=UTC \
    --including-tables "student" \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password= \
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

AWS RDS MySQL Whole Database Synchronization

```
<FLINK_HOME>bin/flink run \
    -Dexecution.checkpointing.interval=10s \
    -Dparallelism.default=1 \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    lib/flink-doris-connector-1.18-25.0.0.jar \
    mysql-sync-database \
    --database testwd \
    --mysql-conf hostname=xxx.ap-southeast-1.rds.amazonaws.com \
    --mysql-conf port=3306 \
    --mysql-conf username=admin \
    --mysql-conf password=123456 \
    --mysql-conf database-name=test \
    --mysql-conf server-time-zone=UTC \
    --including-tables "student" \
    --sink-conf fenodes=127.0.0.1:8030 \
    --sink-conf username=root \
    --sink-conf password= \
    --sink-conf jdbc-url=jdbc:mysql://127.0.0.1:9030 \
    --sink-conf sink.label-prefix=label \
    --table-conf replication_num=1
```

Usage Instructions

Parameter Configuration

General Configuration Items

Key	Default Value	Required	Comment
fenodes	–	Y	Doris FE http addresses. Multiple addresses are supported and should be separated by commas.
benodes	–	N	Doris BE http addresses. Multiple addresses are supported and should be separated by commas.
jdbc-url	–	N	JDBC connection information, such as jdbc:mysql://127.0.0.1:9030.
table.identifier	–	Y	Doris table name, such as db.tbl.
username	–	Y	Username for accessing Doris.

Key	Default Value	Required	Comment
password	–	Y	Password for accessing Doris.
auto-redirect	TRUE	N	Whether to redirect StreamLoad requests. After enabling, StreamLoad will write through FE and will no longer explicitly obtain BE information.
doris.request.retries	3	N	The number of retries for sending requests to Doris.
doris.request.connect.timeout	30s	N	The connection timeout for sending requests to Doris.
doris.request.read.timeout	30s	N	The read timeout for sending requests to Doris.

Source Configuration

Key	Default Value	Required	Comment
doris.request.query.timeout	21600s	N	The timeout for querying Doris. The default value is 6 hours.
doris.request.tablet.size	1	N	The number of Doris Tablets corresponding to one Partition. The smaller this value is set, the more Partitions will be generated, which can increase the parallelism on the Flink side. However, it will also put more pressure on Doris.
doris.batch.size	4064	N	The maximum number of rows read from BE at one time. Increasing this value can reduce the number of connections established between Flink and Doris, thereby reducing the additional time overhead caused by network latency.
doris.exec.mem.limit	8192mb	N	The memory limit for a single query. The default is 8GB, in bytes.
source.use-flight-sql	FALSE	N	Whether to use Arrow Flight SQL for reading.
source.flight-sql-port	-	N	The arrow_flight_sql_port of FE when using Arrow Flight SQL for reading.

DataStream-Specific Configuration

Key	Default Value	Required	Comment
doris.read.field	–	N	The list of column names for reading Doris tables. Multiple columns should be separated by commas.
doris.filter.query	–	N	The expression for filtering read data. This expression is passed to Doris. Doris uses this expression to complete source data filtering. For example, age=18.

Sink Configuration

Key	Default Value	Required	Comment
sink.label-prefix	–	Y	The label prefix used for Stream load import. In the 2pc scenario, it is required to be globally unique to ensure the EOS semantics of Flink.

Key	Default Value	Required	Comment
sink.properties.*	–	N	Import parameters for Stream Load. For example, 'sink.properties.column_separator' = ',' defines the column separator, and 'sink.properties.escape_delimiters' = 'true' means that special characters as delimiters, like , will be converted to binary 0x01. For JSON format import, 'sink.properties.format' = 'json', 'sink.properties.read_json_by_line' = 'true'. For detailed parameters, refer to here. For Group Commit mode, for example, 'sink.properties.group_commit' = 'sync_mode' sets the group commit to synchronous mode. The Flink connector has supported import configuration group commit since version 1.6.2. For detailed usage and limitations, refer to group commit.
sink.enable-delete	TRUE	N	Whether to enable deletion. This option requires the Doris table to have the batch deletion feature enabled (enabled by default in Doris 0.15+ versions), and only supports the Unique model.
sink.enable-2pc	TRUE	N	Whether to enable two-phase commit (2pc). The default is true, ensuring Exactly-Once semantics. For details about two-phase commit, refer to here.
sink.buffer-size	1MB	N	The size of the write data cache buffer, in bytes. It is not recommended to modify it, and the default configuration can be used.
sink.buffer-count	3	N	The number of write data cache buffers. It is not recommended to modify it, and the default configuration can be used.
sink.max-retries	3	N	The maximum number of retries after a Commit failure. The default is 3 times.
sink.enable.batch-mode	FALSE	N	Whether to use the batch mode to write to Doris. After enabling, the writing timing does not rely on Checkpoint, and it is controlled by parameters such as sink.buffer-flush.max-rows, sink.buffer-flush.max-bytes, and sink.buffer-flush.interval. Meanwhile, after enabling, Exactly-once semantics will not be guaranteed, but idempotency can be achieved with the help of the Uniq model.
sink.flush.queue-size	2	N	The size of the cache queue in batch mode.
sink.buffer-flush.max-rows	500000	N	The maximum number of rows written in a single batch in batch mode.
sink.buffer-flush.max-bytes	100MB	N	The maximum number of bytes written in a single batch in batch mode.
sink.buffer-flush.interval	10s	N	The interval for asynchronously flushing the cache in batch mode.
sink.ignore.update-before	TRUE	N	Whether to ignore the update-before event. The default is to ignore it.

Lookup Join Configuration

Key	Default Value	Required	Comment
lookup.cache.max-rows	-1	N	The maximum number of rows in the lookup cache. The default value is -1, which means the cache is not enabled.
lookup.cache.ttl	10s	N	The maximum time for the lookup cache. The default is 10 seconds.
lookup.max-retries	1	N	The number of retries after a lookup query fails.

Key	Default Value	Required	Comment
lookup.jdbc.async	FALSE	N	Whether to enable asynchronous lookup. The default is false.
lookup.jdbc.read.batch.size	128	N	The maximum batch size for each query in asynchronous lookup.
lookup.jdbc.read.batch.queue-size	256	N	The size of the intermediate buffer queue during asynchronous lookup.
lookup.jdbc.read.thread-size	3	N	The number of jdbc threads for lookup in each task.

Full Database Synchronization Configuration

Syntax

```
<FLINK_HOME>bin/flink run \
    -c org.apache.doris.flink.tools.cdc.CdcTools \
    lib/flink-doris-connector-1.16-1.6.1.jar \
    <mysql-sync-database|oracle-sync-database|postgres-sync-database|sqlserver-sync-database|mongodb-sync-database> \
    --database <doris-database-name> \
    [--job-name <flink-job-name>] \
    [--table-prefix <doris-table-prefix>] \
    [--table-suffix <doris-table-suffix>] \
    [--including-tables <mysql-table-name|name-regular-expr>] \
    [--excluding-tables <mysql-table-name|name-regular-expr>] \
    --mysql-conf <mysql-cdc-source-conf> [--mysql-conf <mysql-cdc-source-conf> ...] \
    --oracle-conf <oracle-cdc-source-conf> [--oracle-conf <oracle-cdc-source-conf> ...] \
    --postgres-conf <postgres-cdc-source-conf> [--postgres-conf <postgres-cdc-source-conf> ...] \
    --sqlserver-conf <sqlserver-cdc-source-conf> [--sqlserver-conf <sqlserver-cdc-source-conf> ...] \
    --sink-conf <doris-sink-conf> [--table-conf <doris-sink-conf> ...] \
    [--table-conf <doris-table-conf> [--table-conf <doris-table-conf> ...]]
```

Configuration

Key	Comment
–job-name	The name of the Flink task, which is optional.
–database	The name of the database synchronized to Doris.
–table-prefix	The prefix name of the Doris table, for example, –table-prefix ods_.
–table-suffix	The suffix name of the Doris table, similar to the prefix.
–including-tables	The MySQL tables that need to be synchronized. Multiple tables can be separated by |, and regular expressions are supported. For example, –including-tables table1.
–excluding-tables	The tables that do not need to be synchronized. The usage is the same as that of –including-tables.
–mysql-conf	The configuration of the MySQL CDCSource, for example, –mysql-conf hostname=127.0.0.1. You can view all the configurations of MySQL-CDC here. Among them, hostname, username, password, and database-name are required. When the synchronized database and table contain non-primary key tables, scan.incremental.snapshot.chunk.key-column must be set, and only one non-null type field can be selected. For example: scan.incremental.snapshot.chunk.key-column=database.table:column,database.table1:column…, and columns of different databases and tables are separated by commas.

Key	Comment
–oracle-conf	The configuration of the Oracle CDCSource, for example, –oracle-conf hostname=127.0.0.1. You can view all the configurations of Oracle-CDC here. Among them, hostname, username, password, database-name, and schema-name are required.
–postgres-conf	The configuration of the Postgres CDCSource, for example, –postgres-conf hostname=127.0.0.1. You can view all the configurations of Postgres-CDC here. Among them, hostname, username, password, database-name, schema-name, and slot.name are required.
–sqlserver-conf	The configuration of the SQLServer CDCSource, for example, –sqlserver-conf hostname=127.0.0.1. You can view all the configurations of SQLServer-CDC here. Among them, hostname, username, password, database-name, and schema-name are required.
–db2-conf	The configuration of the SQLServer CDCSource, for example, –db2-conf hostname=127.0.0.1. You can view all the configurations of DB2-CDC here. Among them, hostname, username, password, database-name, and schema-name are required.
–mongodb-conf	The configuration of the MongoDB CDCSource, for example, –mongodb-conf hosts=127.0.0.1:27017. You can view all the configurations of Mongo-CDC here. Among them, hosts, username, password, and database are required. –mongodb-conf schema.sample-percent is the configuration for automatically sampling MongoDB data to create tables in Doris, and the default value is 0.2.
–sink-conf	All the configurations of the Doris Sink can be viewed here.
–table-conf	The configuration items of the Doris table, that is, the content included in properties (except for table-buckets, which is not a properties attribute). For example, –table-conf replication_num=1, and –table-conf table-buckets="tbl1:10,tbl2:20,a.*:30,b.*:40,.*:50" means specifying the number of buckets for different tables in the order of regular expressions. If there is no match, the BUCKETS AUTO method will be used to create tables.
–schema-change-mode	The modes for parsing schema change, including debezium_structure and sql_parser. The debezium_structure mode is used by default. The debezium_structure mode parses the data structure used when the upstream CDC synchronizes data and judges DDL change operations by parsing this structure. The sql_parser mode parses the DDL statements when the upstream CDC synchronizes data to judge DDL change operations, so this parsing mode is more accurate. Usage example: –schema-change-mode debezium_structure. This function will be available in versions after 24.0.0.
–single-sink	Whether to use a single Sink to synchronize all tables. After enabling, it can also automatically identify newly created tables upstream and create tables automatically.
–multi-to-one-origin	The configuration of the source tables when multiple upstream tables are written to the same table, for example: –multi-to-one-origin "a_.*|b_.*", refer to #208
–multi-to-one-target	Used in combination with multi-to-one-origin, the configuration of the target table, for example: –multi-to-one-target "a|b"
–create-table-only	Whether to only synchronize the structure of the table.

Doris to Flink Data Type Mapping

Doris Type	Flink Type
NULL_TYPE	NULL
BOOLEAN	BOOLEAN
TINYINT	TINYINT

Doris Type	Flink Type
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DATE	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
CHAR	STRING
LARGEINT	STRING
VARCHAR	STRING
STRING	STRING
DECIMALV2	DECIMAL
ARRAY	ARRAY
MAP	STRING
JSON	STRING
VARIANT	STRING
IPV4	STRING
IPV6	STRING

Flink to Doris Data Type Mapping

Flink Type	Doris Type
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL	DECIMAL
CHAR	CHAR
VARCHAR	VARCHAR/STRING
STRING	STRING

Flink Type	Doris Type
DATE	DATE
TIMESTAMP	DATETIME
TIMESTAMP_LTZ	DATETIME
ARRAY	ARRAY
MAP	MAP/JSON
ROW	STRUCT/JSON

Monitoring Metrics

Flink provides multiple Metrics for monitoring the indicators of the Flink cluster. The following are the newly added monitoring metrics for the Flink Doris Connector.

Name	Metric Type	Description
totalFlushLoadBytes	Counter	The total number of bytes that have been flushed and imported.
flushTotalNumberRows	Counter	The total number of rows that have been imported and processed.
totalFlushLoadedRows	Counter	The total number of rows that have been successfully imported.
totalFlushTimeMs	Counter	The total time taken for successful imports to complete.
totalFlushSucceededNumber	Counter	The number of times that imports have been successfully completed.
totalFlushFailedNumber	Counter	The number of times that imports have failed.
totalFlushFilteredRows	Counter	The total number of rows with unqualified data quality.
totalFlushUnselectedRows	Counter	The total number of rows filtered by the where condition.
beginTxnTimeMs	Histogram	The time taken to request the Fe to start a transaction, in milliseconds.
putDataTimeMs	Histogram	The time taken to request the Fe to obtain the import data execution plan.
readDataTimeMs	Histogram	The time taken to read data.
writeDataTimeMs	Histogram	The time taken to execute the write data operation.
commitAndPublishTimeMs	Histogram	The time taken to request the Fe to commit and publish the transaction.
loadTimeMs	Histogram	The time taken for the import to complete.

Best Practices

FlinkSQL Quickly Connects to MySQL Data via CDC

```
-- enable checkpoint
SET 'execution.checkpointing.interval' = '10s';

CREATE TABLE cdc_mysql_source (
  id int
  ,name VARCHAR
  ,PRIMARY KEY (id) NOT ENFORCED
```

```
) WITH (
 'connector' = 'mysql-cdc',
 'hostname' = '127.0.0.1',
 'port' = '3306',
 'username' = 'root',
 'password' = 'password',
 'database-name' = 'database',
 'table-name' = 'table'
);

-- Supports synchronizing insert/update/delete events
CREATE TABLE doris_sink (
id INT,
name STRING
)
WITH (
  'connector' = 'doris',
  'fenodes' = '127.0.0.1:8030',
  'table.identifier' = 'database.table',
  'username' = 'root',
  'password' = '',
  'sink.properties.format' = 'json',
  'sink.properties.read_json_by_line' = 'true',
  'sink.enable-delete' = 'true',  -- Synchronize delete events
  'sink.label-prefix' = 'doris_label'
);

insert into doris_sink select id,name from cdc_mysql_source;
```

Flink Performs Partial Column Updates

```
CREATE TABLE doris_sink (
    id INT,
    name STRING,
    bank STRING,
    age int
)
WITH (
  'connector' = 'doris',
  'fenodes' = '127.0.0.1:8030',
  'table.identifier' = 'database.table',
  'username' = 'root',
  'password' = '',
  'sink.properties.format' = 'json',
  'sink.properties.read_json_by_line' = 'true',
  'sink.properties.columns' = 'id,name,bank,age', -- Columns that need to be updated
  'sink.properties.partial_columns' = 'true' -- Enable partial column updates
);
```

Flink Imports Bitmap Data

```
CREATE TABLE bitmap_sink (
dt int,
page string,
user_id int
)
WITH (
  'connector' = 'doris',
  'fenodes' = '127.0.0.1:8030',
  'table.identifier' = 'test.bitmap_test',
```

```
  'username' = 'root',
  'password' = '',
  'sink.label-prefix' = 'doris_label',
  'sink.properties.columns' = 'dt,page,user_id,user_id=to_bitmap(user_id)'
)
```

FlinkCDC Updates Key Columns

Generally, in a business database, a number is often used as the primary key of a table. For example, for the Student table, the number (id) is used as the primary key. However, as the business develops, the number corresponding to the data may change. In this scenario, when using Flink CDC + Doris Connector to synchronize data, the data of the primary key column in Doris can be automatically updated.

Principle

The underlying collection tool of Flink CDC is Debezium. Debezium internally uses the op field to identify corresponding operations. The values of the op field are c, u, d, and r, corresponding to create, update, delete, and read respectively. For the update of the primary key column, Flink CDC will send DELETE and INSERT events downstream, and the data of the primary key column in Doris will be automatically updated after the data is synchronized to Doris.

Usage

The Flink program can refer to the above CDC synchronization examples. After successfully submitting the task, execute the statement to update the primary key column on the MySQL side (for example, update student set id = ‘1002’ where id = ‘1001’), and then the data in Doris can be modified.

Flink Deletes Data According to Specified Columns

Generally, messages in Kafka use specific fields to mark the operation type, such as {“op_type”:“delete”,data:{…}}. For this kind of data, it is hoped to delete the data with op_type=delete.

The DorisSink will, by default, distinguish the types of events according to RowKind. Usually, in the case of CDC, the event type can be directly obtained, and the hidden column `__DORIS_DELETE_SIGN__` can be assigned a value to achieve the purpose of deletion. However, for Kafka, it is necessary to judge according to the business logic and explicitly pass in the value of the hidden column.

```
-- For example, the upstream data:{"op_type":"delete",data:{"id":1,"name":"zhangsan"}}
CREATE TABLE KAFKA_SOURCE(
  data STRING,
  op_type STRING
) WITH (
  'connector' = 'kafka',
  ...
);

CREATE TABLE DORIS_SINK(
  id INT,
  name STRING,
  __DORIS_DELETE_SIGN__ INT
) WITH (
  'connector' = 'doris',
  'fenodes' = '127.0.0.1:8030',
  'table.identifier' = 'db.table',
```

```
  'username' = 'root',
  'password' = '',
  'sink.enable-delete' = 'false',        -- false means not to obtain the event type from RowKind
  'sink.properties.columns' = 'id, name, __DORIS_DELETE_SIGN__'  -- Explicitly specify the import columns of streamload
);

INSERT INTO DORIS_SINK
SELECT json_value(data,'$.id') as id,
json_value(data,'$.name') as name,
if(op_type='delete',1,0) as __DORIS_DELETE_SIGN__
from KAFKA_SOURCE;
```

Flink CDC Synchronize DDL Statements

Generally, when synchronizing upstream data sources such as MySQL, when adding or deleting fields in the upstream, you need to synchronize the Schema Change operation in Doris.

For this scenario, you usually need to write a program for the DataStream API and use the JsonDebeziumSchemaSerializer serializer provided by DorisSink to automatically perform SchemaChange. For details, please refer to CDCSchemaChangeExample.java

In the whole database synchronization tool provided by the Connector, no additional configuration is required, and the upstream DDL will be automatically synchronized and the SchemaChange operation will be performed in Doris.

Frequently Asked Questions (FAQ)

1. **errCode = 2, detailMessage = Label [label_0_1] has already been used, relate to txn [19650]**

   In the Exactly-Once scenario, the Flink Job must be restarted from the latest Checkpoint/Savepoint, otherwise the above error will be reported. When Exactly-Once is not required, this problem can also be solved by disabling 2PC submission (sink.enable-2pc=false) or changing to a different sink.label-prefix.

2. **errCode = 2, detailMessage = transaction [19650] not found**

   This occurs during the Commit stage. The transaction ID recorded in the checkpoint has expired on the FE side. When committing again at this time, the above error will occur. At this point, it's impossible to start from the checkpoint. Subsequently, you can extend the expiration time by modifying the `streaming_label_keep_max_second` configuration in `fe.conf`. The default expiration time is 12 hours. After doris version 2.0, it will also be limited by the `label_num_threshold` configuration in `fe.conf` (default 2000), which can be increased or changed to -1 (-1 means only limited by time).

3. **errCode = 2, detailMessage = current running txns on db 10006 is 100, larger than limit 100**

   This is because the concurrent imports into the same database exceed 100. It can be solved by adjusting the parameter `max_running_txn_num_per_db` in `fe.conf`. For specific details, please refer to max_running_txn_num_per_db.

   Meanwhile, frequently modifying the label and restarting a task may also lead to this error. In the 2pc scenario (for Duplicate/Aggregate models), the label of each task needs to be unique. And when restarting from a checkpoint, the Flink task will actively abort the transactions that have been

pre-committed successfully but not yet committed. Frequent label modifications and restarts will result in a large number of pre-committed successful transactions that cannot be aborted and thus occupy transactions. In the Unique model, 2pc can also be disabled to achieve idempotent writes.

4. **tablet writer write failed, tablet_id=190958, txn_id=3505530, err=-235**

   This usually occurs before Connector version 1.1.0 and is caused by too high a writing frequency, which leads to an excessive number of versions. You can reduce the frequency of Streamload by setting the `sink.batch.size` and `sink.batch.interval` parameters. After Connector version 1.1.0, the default writing timing is controlled by Checkpoint, and you can reduce the writing frequency by increasing the Checkpoint interval.

5. **How to skip dirty data when Flink is importing?**

   When Flink imports data, if there is dirty data, such as issues with field formats or lengths, it will cause StreamLoad to report errors. At this time, Flink will keep retrying. If you need to skip such data, you can disable the strict mode of StreamLoad (by setting `strict_mode=false` and `max_filter_ratio=1`) or filter the data before the Sink operator.

6. **How to configure when the network between Flink machines and BE machines is not connected?**

   When Flink initiates writing to Doris, Doris will redirect the write operation to BE. At this time, the returned address is the internal network IP of BE, which is the IP seen through the `show backends` command. If Flink and Doris have no network connectivity at this time, an error will be reported. In this case, you can configure the external network IP of BE in `benodes`.

7. **stream load error: HTTP/1.1 307 Temporary Redirect**

   Flink will first request FE, and after receiving 307, it will request BE after redirection. When FE is in FullGC/high pressure/network delay, HttpClient will send data without waiting for a response within a certain period of time (3 seconds) by default. Since the request body is InputStream by default, when a 307 response is received, the data cannot be replayed and an error will be reported directly. There are three ways to solve this problem: 1. Upgrade to Connector25.1.0 or above to increase the default time; 2. Modify auto-redirect=false to directly initiate a request to BE (not applicable to some cloud scenarios); 3. The unique key model can enable batch mode.

Doris Kafka Connector

Kafka Connect is a scalable and reliable tool for data transmission between Apache Kafka and other systems. Connectors can be defined Move large amounts of data in and out of Kafka.

The Doris community provides the doris-kafka-connector plug-in, which can write data in the Kafka topic to Doris.

Version Description

Connector Version	Kafka Version	Doris Version	Java Version
1.0.0	2.4+	2.0+	8
1.1.0	2.4+	2.0+	8
24.0.0	2.4+	2.0+	8
25.0.0	2.4+	2.0+	8

Usage

Download

doris-kafka-connector

maven dependencies

```
<dependency>
  <groupId>org.apache.doris</groupId>
  <artifactId>doris-kafka-connector</artifactId>
  <version>25.0.0</version>
</dependency>
```

Standalone mode startup

Create the plugins directory under $KAFKA_HOME and put the downloaded doris-kafka-connector jar package into it
Configure config/connect-standalone.properties

```
# Modify broker address
bootstrap.servers=127.0.0.1:9092

# Modify to the created plugins directory
# Note: Please fill in the direct path to Kafka here. For example: plugin.path=/opt/kafka/plugins
plugin.path=$KAFKA_HOME/plugins

# It is recommended to increase the max.poll.interval.ms time of Kafka to more than 30 minutes, the default is 5 minutes
# Avoid Stream Load import data consumption timeout and consumers being kicked out of the consumer group
```

```
max.poll.interval.ms=1800000
consumer.max.poll.interval.ms=1800000
```

Configure doris-connector-sink.properties

Create doris-connector-sink.properties in the config directory and configure the following content:

```
name=test-doris-sink
connector.class=org.apache.doris.kafka.connector.DorisSinkConnector
topics=topic_test
doris.topic2table.map=topic_test:test_kafka_tbl
doris.urls=10.10.10.1
doris.http.port=8030
doris.query.port=9030
doris.user=root
doris.password=
doris.database=test_db
buffer.count.records=10000
buffer.flush.time=120
buffer.size.bytes=5000000
enable.combine.flush=true
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=false
```

Start Standalone

```
$KAFKA_HOME/bin/connect-standalone.sh -daemon $KAFKA_HOME/config/connect-standalone.properties $KAFKA_HOME/config/doris-connector-sink.properties
```

> **Note: It is generally not recommended to use standalone mode in a production environment.**

Distributed mode startup

Create the plugins directory under $KAFKA_HOME and put the downloaded doris-kafka-connector jar package into it

Configure config/connect-distributed.properties

```
# Modify kafka server address
bootstrap.servers=127.0.0.1:9092

# Modify group.id, the same cluster needs to be consistent
group.id=connect-cluster

# Modify to the created plugins directory
# Note: Please fill in the direct path to Kafka here. For example: plugin.path=/opt/kafka/plugins
plugin.path=$KAFKA_HOME/plugins

# It is recommended to increase the max.poll.interval.ms time of Kafka to more than 30 minutes, the default is 5 minutes
# Avoid Stream Load import data consumption timeout and consumers being kicked out of the consumer group
max.poll.interval.ms=1800000
consumer.max.poll.interval.ms=1800000
```

Start Distributed

```
$KAFKA_HOME/bin/connect-distributed.sh -daemon $KAFKA_HOME/config/connect-distributed.properties
```

Add Connector

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"test-doris-sink-cluster",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
    "topics":"topic_test",
    "doris.topic2table.map": "topic_test:test_kafka_tbl",
    "doris.urls":"10.10.10.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test_db",
    "enable.combine.flush": "true",
    "buffer.count.records":"10000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"5000000",
    "key.converter":"org.apache.kafka.connect.storage.StringConverter",
    "value.converter":"org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "false"
  }
}'
```

Operation Connector

```
# View connector status
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/status -X GET
# Delete connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster -X DELETE
# Pause connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/pause -X PUT
# Restart connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/resume -X PUT
# Restart tasks within the connector
curl -i http://127.0.0.1:8083/connectors/test-doris-sink-cluster/tasks/0/restart -X POST
```

Refer to: Connect REST Interface

> **Note that when kafka-connect is started for the first time, three topics `config.storage.topic` `offset.storage.topic` and `status.storage.topic` will be created in the kafka cluster to record the shared connector configuration of kafka-connect. Offset data and status updates. How to Use Kafka Connect - Get Started**

Access an SSL-certified Kafka cluster

Accessing an SSL-certified Kafka cluster through kafka-connect requires the user to provide a certificate file (client.truststore.jks) used to authenticate the Kafka Broker public key. You can add the following configuration in the `connect-distributed.properties` file:

```
# Connect worker
security.protocol=SSL
ssl.truststore.location=/var/ssl/private/client.truststore.jks
ssl.truststore.password=test1234

# Embedded consumer for sink connectors
consumer.security.protocol=SSL
consumer.ssl.truststore.location=/var/ssl/private/client.truststore.jks
consumer.ssl.truststore.password=test1234
```

For instructions on configuring a Kafka cluster connected to SSL authentication through kafka-connect, please refer to: Configure Kafka Connect

Dead letter queue

By default, any errors encountered during or during the conversion will cause the connector to fail. Each connector configuration can also tolerate such errors by skipping them, optionally writing the details of each error and failed operation as well as the records in question (with varying levels of detail) to a dead-letter queue for logging.

```
errors.tolerance=all
errors.deadletterqueue.topic.name=test_error_topic
errors.deadletterqueue.context.headers.enable=true
errors.deadletterqueue.topic.replication.factor=1
```

Configuration items

Key	Enum	Default Value	Required	Description
name	-	-		YConnect application name, must be unique within the Kafka Connect environment
connector.class	-	-		Yorg.apache.doris.kafka.connector.DorisSinkConnector

Key	Enum	Default Value	Required	Description
topics	-	-	Y	List of subscribed topics, separated by commas. like: topic1, topic2
doris.urls	-	-	Y	Doris FE connection address. If there are multiple, separate them with commas. like: 10.20.30.1,10.20.30.2,10.20.30.3
doris.http.port	-	-	Y	Doris HTTP protocol port
doris.query.port	-	-	Y	Doris MySQL protocol port

Key	Enum	Default Value	Required	Description
doris.user	-	-	Y	Doris username
doris.password	-	-	Y	Doris password
doris.database	-	-	Y	The database to write to. It can be empty when there are multiple libraries. At the same time, the specific library name needs to be configured in topic2table.map.

Key	Enum	Default Value	Required	Description
doris.topic2table.map	-	-	Y	The corresponding relationship between topic and table table, for example: topic1:tb1,topic2:tb2 If left blank, the topic will be used as the default table name for writing. The format of multiple libraries is topic1:db1.tbl1,topic2:db2.tbl2
buffer.count.records	-	50000	N	Number of records buffered in memory before flush triggering. Default 50000 records
buffer.flush.time	-	120	N	Buffer refresh interval, in seconds, default 120 seconds

Key	Enum	Default Value	Required	Description
buffer.size.bytes	-	104857600(100MB)	N	Cumulative size of records buffered in memory before flush triggering, default 100MB
enable.combine.flush	true, false	false	N	Whether to merge data from all partitions together and write them. The default value is false. When enabled, only at_least_once semantics are guaranteed.
jmx	-	true	N	To obtain connector internal monitoring indicators through JMX, please refer to: Doris-Connector-JMX
label.prefix	-	${name}	N	Stream load label prefix when importing data. Defaults to the Connector application name.

Key	Enum	Default Value	Required	Description
auto.redirect	-	true	No	Whether to redirect StreamLoad requests. After being turned on, StreamLoad will redirect to the BE where data needs to be written through FE, and the BE information will no longer be displayed.
sink.properties.*	-	`'sink.properties.format':'json', 'sink.properties.read_json_by_line':'true'`	No	Import parameters for Stream Load. For example: define column separator `'sink.properties.column_separator':','` Detailed parameter reference here **Enable Group Commit**, for example, enable group commit in sync_mode mode: `"sink.properties.group_commit":"sync_mode"`. Group Commit can be configured with three modes: `off_mode`, `sync_mode`, and `async_mode`. For specific usage, please refer to: Group-Commit **Enable partial column update**, for example, enable update of partial columns of specified col2: `"sink.properties.partial_columns":"true"`, `"sink.properties.columns": " col2",`
delivery.guarantee	`at_least_once`, `exactly_once`	at_least_once	No	How to ensure data consistency when consuming Kafka data is imported into Doris. Supports `at_least_once` `exactly_once`, default is `at_least_once`. Doris needs to be upgraded to 2.1.0 or above to ensure data `exactly_once`

Key	Enum	Default Value	Required	Description
converter.mode	`normal`, `debezium_ingestion`	normal	N	Type conversion mode of upstream data when using Connector to consume Kafka data. `normal` means consuming data in Kafka normally without any type conversion. `debezium_ingestion` means that when Kafka upstream data is collected through CDC (Changelog Data Capture) tools such as Debezium, the upstream data needs to undergo special type conversion to support it.
debezium.schema.evolution	`none`, `basic`	none	N	Use Debezium to collect upstream database systems (such as MySQL), and when structural changes occur, the added fields can be synchronized to Doris. `none` means that when the structure of the upstream database system changes, the changed structure will not be synchronized to Doris. `basic` means synchronizing the data change operation of the upstream database. Since changing the column structure is a dangerous operation (it may lead to accidentally deleting columns of the Doris table structure), currently only the operation of adding columns synchronously upstream is supported. When a column is renamed, the old column remains unchanged, and the Connector will add a new column in the target table and sink the renamed new data into the new column.
enable.delete	-	false	N	Whether to delete records synchronously, default false

Key	Enum	Default Value	Required	Description
database.time_zone	-	UTC	No	When `converter.mode` is not `normal` mode, it provides a way to specify time zone conversion for date data types (such as datetime, date, timestamp, etc.). The default is UTC time zone.
avro.topic2schema.filepath	-	-	No	By reading the locally provided Avro Schema file, the Avro file content in the Topic is parsed to achieve decoupling from the Schema registration center provided by Confluent. This configuration needs to be used with the `key.converter` or `value.converter` prefix. For example, the local Avro Schema file for configuring avro-user and avro-product Topic is as follows: `"value.converter.avro.topic2schema.filepath":"avro-user:file:///opt/avro_user.avsc, avro-product:file:///opt/avro_product.avsc"` For specific usage, please refer to: #32

Key	Enum	Default Value	Required	Description
record.tablename.field	-	-	N	Configure this parameter, data from one kafka topic can flow to multiple doris tables. For configuration details, refer to: #58
max.retries	-	10	N	The maximum number of times to retry on errors before failing the task.
retry.interval.ms	-	6000	N	The time in milliseconds to wait following an error before attempting a retry.

Key	Enum	Default Value	Required	Description
behavior.on.null.values	ignore, fail	ignore	N	Defined how to handle records with null values.

For other Kafka Connect Sink common configuration items, please refer to: connect_configuring

Type mapping

Doris-kafka-connector uses logical or primitive type mapping to resolve the column's data type.
Primitive types refer to simple data types represented using Kafka connect's `Schema`. Logical data types usually use the `Struct` structure to represent complex types, or date and time types.

Kafka Primitive Type	Doris Type
INT8	TINYINT
INT16	SMALLINT
INT32	INT
INT64	BIGINT
FLOAT32	FLOAT
FLOAT64	DOUBLE
BOOLEAN	BOOLEAN
STRING	STRING
BYTES	STRING

Kafka Logical Type	Doris Type
org.apache.kafka.connect.data.Decimal	DECIMAL
org.apache.kafka.connect.data.Date	DATE
org.apache.kafka.connect.data.Time	STRING
org.apache.kafka.connect.data.Timestamp	DATETIME

Debezium Logical Type	Doris Type
io.debezium.time.Date	DATE
io.debezium.time.Time	String
io.debezium.time.MicroTime	DATETIME
io.debezium.time.NanoTime	DATETIME
io.debezium.time.ZonedTime	DATETIME
io.debezium.time.Timestamp	DATETIME
io.debezium.time.MicroTimestamp	DATETIME
io.debezium.time.NanoTimestamp	DATETIME
io.debezium.time.ZonedTimestamp	DATETIME
io.debezium.data.VariableScaleDecimal	DOUBLE

Best Practices

Load plain JSON data

1. Import data sample
   In Kafka, there is the following sample data

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-data-topic --from-beginning
{"user_id":1,"name":"Emily","age":25}
{"user_id":2,"name":"Benjamin","age":35}
{"user_id":3,"name":"Olivia","age":28}
{"user_id":4,"name":"Alexander","age":60}
{"user_id":5,"name":"Ava","age":17}
{"user_id":6,"name":"William","age":69}
{"user_id":7,"name":"Sophia","age":32}
{"user_id":8,"name":"James","age":64}
{"user_id":9,"name":"Emma","age":37}
{"user_id":10,"name":"Liam","age":64}
```

2. Create the table that needs to be imported
   In Doris, create the imported table, the specific syntax is as follows `sql CREATE TABLE test_db.test_kafka_connector_tbl( user_id BIGINT NOT NULL COMMENT "user id", name VARCHAR(20) COMMENT "name", age INT COMMENT "age" ) DUPLICATE KEY(user_id) DISTRIBUTED BY HASH(user_id) BUCKETS 12;`

3. Create an import task
   On the machine where Kafka-connect is deployed, submit the following import task through the curl

command `shell curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{ "name":"test-doris-sink-cluster", "config":{ "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector", "tasks.max":"10", "topics":"test-data-topic", "doris.topic2table.map": "test-data-topic:test_kafka_connector_tbl", "doris.urls":"10.10.10.1", "doris.user":"root", "doris.password":"", "doris.http.port":"8030", "doris.query.port":"9030", "doris.database":"test_db", "buffer.count.records":"10000", "buffer.flush.time":"120", "buffer.size.bytes":"5000000", "enable.combine.flush": "true", "key.converter":"org.apache.kafka.connect.storage.StringConverter", "value.converter": "org.apache.kafka.connect.json.JsonConverter", "value.converter.schemas.enable": "false" } }'`

Load data collected by Debezium components

1. The MySQL database has the following table

```
CREATE TABLE test.test_user (
user_id int NOT NULL ,
name varchar(20),
age int,
PRIMARY KEY (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

insert into test.test_user values(1,'zhangsan',20);
insert into test.test_user values(2,'lisi',21);
insert into test.test_user values(3,'wangwu',22);
```

2. Create the imported table in Doris

```
CREATE TABLE test_db.test_user(
user_id            BIGINT       NOT NULL COMMENT "user id",
name               VARCHAR(20)           COMMENT "name",
age                INT                   COMMENT "age"
)
UNIQUE KEY(user_id)
DISTRIBUTED BY HASH(user_id) BUCKETS 12;
```

3. Deploy the Debezium connector for MySQL component, refer to: Debezium connector for MySQL
4. Create doris-kafka-connector import task
   Assume that the MySQL table data collected through Debezium is in the `mysql_debezium.test.test_user` Topic

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
"name":"test-debezium-doris-sink",
"config":{
"connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
"tasks.max":"10",
"topics":"mysql_debezium.test.test_user",
"doris.topic2table.map": "mysql_debezium.test.test_user:test_user",
"doris.urls":"10.10.10.1",
"doris.user":"root",
"doris.password":"",
"doris.http.port":"8030",
"doris.query.port":"9030",
"doris.database":"test_db",
"buffer.count.records":"10000",
"buffer.flush.time":"30",
```

```
   "buffer.size.bytes":"5000000",
   "enable.combine.flush": "true",
   "converter.mode":"debezium_ingestion",
   "enable.delete":"true",
   "key.converter":"org.apache.kafka.connect.json.JsonConverter",
   "value.converter":"org.apache.kafka.connect.json.JsonConverter"
   }
   }'
```

Load Avro serialized data

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"doris-avro-test",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
    "topics":"avro_topic",
    "tasks.max":"10",
    "doris.topic2table.map": "avro_topic:avro_tab",
    "doris.urls":"127.0.0.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test",
    "buffer.count.records":"100000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"10000000",
    "enable.combine.flush": "true",
    "key.converter":"io.confluent.connect.avro.AvroConverter",
    "key.converter.schema.registry.url":"http://127.0.0.1:8081",
    "value.converter":"io.confluent.connect.avro.AvroConverter",
    "value.converter.schema.registry.url":"http://127.0.0.1:8081"
  }
}'
```

Load Protobuf serialized data

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name":"doris-protobuf-test",
  "config":{
    "connector.class":"org.apache.doris.kafka.connector.DorisSinkConnector",
    "topics":"proto_topic",
    "tasks.max":"10",
    "doris.topic2table.map": "proto_topic:proto_tab",
    "doris.urls":"127.0.0.1",
    "doris.user":"root",
    "doris.password":"",
    "doris.http.port":"8030",
    "doris.query.port":"9030",
    "doris.database":"test",
    "buffer.count.records":"100000",
    "buffer.flush.time":"120",
    "buffer.size.bytes":"10000000",
    "enable.combine.flush": "true",
    "key.converter":"io.confluent.connect.protobuf.ProtobufConverter",
    "key.converter.schema.registry.url":"http://127.0.0.1:8081",
    "value.converter":"io.confluent.connect.protobuf.ProtobufConverter",
    "value.converter.schema.registry.url":"http://127.0.0.1:8081"
```

```
  }
}'
```

Loading Data with Kafka Connect Single Message Transforms

For example, consider data in the following format:

```
{
  "registertime": 1513885135404,
  "userid": "User_9",
  "regionid": "Region_3",
  "gender": "MALE"
}
```

To add a hard-coded column to Kafka messages, InsertField can be used. Additionally, TimestampConverter can be used to convert Bigint type timestamps to time strings.

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type: application/json" -X POST -d '{
  "name": "insert_field_tranform",
  "config": {
    "connector.class": "org.apache.doris.kafka.connector.DorisSinkConnector",
    "tasks.max": "1",
    "topics": "users",
    "doris.topic2table.map": "users:kf_users",
    "buffer.count.records": "10000",
    "buffer.flush.time": "10",
    "buffer.size.bytes": "5000000",
    "doris.urls": "127.0.0.1:8030",
    "doris.user": "root",
    "doris.password": "123456",
    "doris.http.port": "8030",
    "doris.query.port": "9030",
    "doris.database": "testdb",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter.schemas.enable": "false",
    "transforms": "InsertField,TimestampConverter",
    // Insert Static Field
    "transforms.InsertField.type": "org.apache.kafka.connect.transforms.InsertField$Value",
    "transforms.InsertField.static.field": "repo",
    "transforms.InsertField.static.value": "Apache Doris",
    // Convert Timestamp Format
    "transforms.TimestampConverter.type": "org.apache.kafka.connect.transforms.TimestampConverter$Value",
    "transforms.TimestampConverter.field": "registertime",
    "transforms.TimestampConverter.format": "yyyy-MM-dd HH:mm:ss.SSS",
    "transforms.TimestampConverter.target.type": "string"
  }
}'
```

After InsertField and TimestampConverter transformations, the data becomes:

```
{
  "userid": "User_9",
  "regionid": "Region_3",
  "gender": "MALE",
  "repo": "Apache Doris",// Static field added
   "registertime": "2017-12-21 03:38:55.404"  // Unix timestamp converted to string
}
```

For more examples of Kafka Connect Single Message Transforms (SMT), please refer to the SMT documentation.

FAQ

1. The following error occurs when reading Json type data:

```
Caused by: org.apache.kafka.connect.errors.DataException: JsonConverter with schemas.enable requires "schema" and "payload" fields
and may not contain additional fields. If you are trying to deserialize plain JSON data, set schemas.enable=false in your converter
configuration.
    at org.apache.kafka.connect.json.JsonConverter.toConnectData(JsonConverter.java:337)
    at org.apache.kafka.connect.storage.Converter.toConnectData(Converter.java:91)
    at org.apache.kafka.connect.runtime.WorkerSinkTask.lambda$convertAndTransformRecord$4(WorkerSinkTask.java:536)
    at org.apache.kafka.connect.runtime.errors.RetryWithToleranceOperator.execAndRetry(RetryWithToleranceOperator.java:180)
    at org.apache.kafka.connect.runtime.errors.RetryWithToleranceOperator.execAndHandleError(RetryWithToleranceOperator.java:214)
```

reason: This is because using the `org.apache.kafka.connect.json.JsonConverter` converter requires matching the “schema” and “payload” fields.

Two solutions, choose one: 1. Replace `org.apache.kafka.connect.json.JsonConverter` with `org.apache.kafka.connect.storage.StringConverter` 2. If the startup mode is **Standalone** mode, change `value.converter.schemas.enable` or `key.converter.schemas.enable` in config/connect-standalone.properties to false; If the startup mode is **Distributed** mode, change `value.converter.schemas.enable` or `key.converter.schemas.enable` in config/connect-distributed.properties to false

2. The consumption times out and the consumer is kicked out of the consumption group:

```
org.apache.kafka.clients.consumer.CommitFailedException: Offset commit cannot be completed since the consumer is not part of an
active group for auto partition assignment; it is likely that the consumer was kicked out of the group.
        at org.apache.kafka.clients.consumer.internals.ConsumerCoordinator.sendOffsetCommitRequest(ConsumerCoordinator.java:1318)
        at org.apache.kafka.clients.consumer.internals.ConsumerCoordinator.doCommitOffsetsAsync(ConsumerCoordinator.java:1127)
        at org.apache.kafka.clients.consumer.internals.ConsumerCoordinator.commitOffsetsAsync(ConsumerCoordinator.java:1093)
        at org.apache.kafka.clients.consumer.KafkaConsumer.commitAsync(KafkaConsumer.java:1590)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.doCommitAsync(WorkerSinkTask.java:361)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.doCommit(WorkerSinkTask.java:376)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.commitOffsets(WorkerSinkTask.java:467)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.commitOffsets(WorkerSinkTask.java:381)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.iteration(WorkerSinkTask.java:221)
        at org.apache.kafka.connect.runtime.WorkerSinkTask.execute(WorkerSinkTask.java:206)
        at org.apache.kafka.connect.runtime.WorkerTask.doRun(WorkerTask.java:204)
        at org.apache.kafka.connect.runtime.WorkerTask.run(WorkerTask.java:259)
        at org.apache.kafka.connect.runtime.isolation.Plugins.lambda$withClassLoader$1(Plugins.java:181)
        at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:539)
        at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
        at java.base/java.lang.Thread.run(Thread.java:833)
```

Solution:

Increase `max.poll.interval.ms` in Kafka according to the scenario. The default value is `300000` - If it is started in Standalone mode, add the `max.poll.interval.ms` and `consumer.max.poll.interval.ms` parameters in the configuration file of config/connect-standalone.properties, and configure the parameter values. - If it is started in Distributed mode, add the `max.poll.interval.ms` and `consumer.max.poll.interval.ms` parameters in the configuration file of config/connect-distributed.properties, and configure the parameter values.

After adjusting the parameters, restart kafka-connect

3. Doris-kafka-connector reports an error when upgrading version from 1.0.0 or 1.1.0 to 24.0.0

```
org.apache.kafka.common.config.ConfigException: Topic 'connect-status' supplied via the 'status.storage.topic' property is required
to have 'cleanup.policy=compact' to guarantee consistency and durability of connector and task statuses, but found the topic
currently has 'cleanup.policy=delete'. Continuing would likely result in eventually losing connector and task statuses and problems
restarting this Connect cluster in the future. Change the 'status.storage.topic' property in the Connect worker configurations to
use a topic with 'cleanup.policy=compact'.
    at org.apache.kafka.connect.util.TopicAdmin.verifyTopicCleanupPolicyOnlyCompact(TopicAdmin.java:581)
    at org.apache.kafka.connect.storage.KafkaTopicBasedBackingStore.lambda$topicInitializer$0(KafkaTopicBasedBackingStore.java:47)
    at org.apache.kafka.connect.util.KafkaBasedLog.start(KafkaBasedLog.java:247)
    at org.apache.kafka.connect.util.KafkaBasedLog.start(KafkaBasedLog.java:231)
    at org.apache.kafka.connect.storage.KafkaStatusBackingStore.start(KafkaStatusBackingStore.java:228)
    at org.apache.kafka.connect.runtime.AbstractHerder.startServices(AbstractHerder.java:164)
    at org.apache.kafka.connect.runtime.distributed.DistributedHerder.run
```

Solution: Adjust the clearing strategy of `connect-configs` `connect-status` Topic to compact

```
$KAFKA_HOME/bin/kafka-configs.sh --alter --entity-type topics --entity-name connect-configs --add-config cleanup.policy=compact --
bootstrap-server 127.0.0.1:9092
$KAFKA_HOME/bin/kafka-configs.sh --alter --entity-type topics --entity-name connect-status --add-config cleanup.policy=compact --
bootstrap-server 127.0.0.1:9092
```

4. Table schema change failed in `debezium_ingestion` converter mode

```
[2025-01-07 14:26:20,474] WARN [doris-normal_test_sink-connector|task-0] Table 'test_sink' cannot be altered because schema
evolution is disabled. (org.apache.doris.kafka.connector.converter.RecordService:183)
[2025-01-07 14:26:20,475] ERROR [doris-normal_test_sink-connector|task-0] WorkerSinkTask{id=doris-normal_test_sink-connector-0} Task
threw an uncaught and unrecoverable exception. Task is being killed and will not recover until manually restarted. Error: Cannot
alter table org.apache.doris.kafka.connector.model.TableDescriptor@67cd8027 because schema evolution is disabled
(org.apache.kafka.connect.runtime.WorkerSinkTask:612)
org.apache.doris.kafka.connector.exception.SchemaChangeException: Cannot alter table
org.apache.doris.kafka.connector.model.TableDescriptor@67cd8027 because schema evolution is disabled
    at org.apache.doris.kafka.connector.converter.RecordService.alterTableIfNeeded(RecordService.java:186)
    at org.apache.doris.kafka.connector.converter.RecordService.checkAndApplyTableChangesIfNeeded(RecordService.java:150)
    at org.apache.doris.kafka.connector.converter.RecordService.processStructRecord(RecordService.java:100)
    at org.apache.doris.kafka.connector.converter.RecordService.getProcessedRecord(RecordService.java:305)
    at org.apache.doris.kafka.connector.writer.DorisWriter.putBuffer(DorisWriter.java:155)
    at org.apache.doris.kafka.connector.writer.DorisWriter.insertRecord(DorisWriter.java:124)
    at org.apache.doris.kafka.connector.writer.StreamLoadWriter.insert(StreamLoadWriter.java:151)
    at org.apache.doris.kafka.connector.service.DorisDefaultSinkService.insert(DorisDefaultSinkService.java:154)
    at org.apache.doris.kafka.connector.service.DorisDefaultSinkService.insert(DorisDefaultSinkService.java:135)
    at org.apache.doris.kafka.connector.DorisSinkTask.put(DorisSinkTask.java:97)
    at org.apache.kafka.connect.runtime.WorkerSinkTask.deliverMessages(WorkerSinkTask.java:583)
    at org.apache.kafka.connect.runtime.WorkerSinkTask.poll(WorkerSinkTask.java:336)
    at org.apache.kafka.connect.runtime.WorkerSinkTask.iteration(WorkerSinkTask.java:237)
    at org.apache.kafka.connect.runtime.WorkerSinkTask.execute(WorkerSinkTask.java:206)
    at org.apache.kafka.connect.runtime.WorkerTask.doRun(WorkerTask.java:202)
    at org.apache.kafka.connect.runtime.WorkerTask.run(WorkerTask.java:257)
    at org.apache.kafka.connect.runtime.isolation.Plugins.lambda$withClassLoader$1(Plugins.java:177)
    at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:515)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
```

Solution:

In `debezium_ingestion` converter mode, table schema changes are turned off by default. You need to configure `debezium.schema.evolution` to `basic` to enable table schema changes.
It should be noted that enabling table structure changes does not accurately keep this changed column as the only column in the Doris table (see `debezium.schema.evolution` parameter description for details). If you need to keep only unique columns in the upstream and downstream, it is best to manually add the changed columns to the Doris table, and then restart the Connector task. The Connector will continue to consume the unconsumed `offset` to maintain data consistency.

Doris Kubernetes Operator

Kubernetes Operator (referred to as Doris Operator) was born to meet the user's demand for efficient deployment and operation of Doris on the Kubernetes platform. It integrates the complex management capabilities of native Kubernetes resources, and integrates the distributed collaboration between Doris components, on-demand customization of user cluster forms, and other experiences, providing users with a more concise, efficient and easy-to-use containerized deployment solution. It aims to achieve efficient management and control of Doris on Kubernetes, helping users reduce operation and maintenance management and learning costs while providing powerful functions and flexible configuration capabilities.

Doris Operator implements the configuration, management and scheduling of Doris on the Kubernetes platform based on Kubernetes CustomResourceDefinitions (CRD). Doris Operator can automatically create Pods and other resources to start services according to the user-defined desired state. Through the automatic registration mechanism, all started services can be integrated into a complete Doris cluster. This implementation significantly reduces the complexity and learning cost of processing configuration information, node discovery and registration, access communication and health checks in the Doris cluster, which are essential operations in the production environment.

Doris Operator Architecture

The design of Doris Operator is based on the principle of a two-layer scheduler. The first-layer scheduling of each component uses native StatefulSet and Service resources to directly manage the corresponding Pod service, which makes it fully compatible with open source Kubernetes clusters, including public clouds, private clouds, and self-built Kubernetes platforms.

Based on the deployment definition provided by Doris Operator, users can customize the Doris deployment state and send it to the Kubernetes cluster through the kubectl management command of Kubernetes. Doris Operator converts the deployment of each service into StatefulSet and its affiliated resources (such as Service) according to the customized state, and then schedules the desired Pods through StatefulSet. It simplifies unnecessary configuration in the StatefulSet specification by abstracting the final state of the Doris cluster, thereby reducing the user's learning cost.

Key capabilities

- **Final state deployment**:

  Kubernetes uses the final state operation and maintenance mode to manage services, and Doris Operator defines a resource type that can describe the Doris cluster - DorisCluster. Users can refer to relevant documents and usage examples to easily configure the required cluster. Users can send the configuration to the Kubernetes cluster through the Kubernetes command line tool kubectl. Doris Operator automatically builds the required cluster and updates the cluster status to the corresponding resources in real time. This process ensures efficient management and monitoring of the cluster and greatly simplifies operation and maintenance operations.

- **Easy to expand**:

Doris Operator supports concurrent real-time horizontal expansion in a cloud disk-based environment. All component services of Doris are deployed and managed through Kubernetes' StatefulSet. When deploying or expanding, Pods are created using StatefulSet's Parallel mode, so that in theory all replicas can be started within the time it takes to start a node. The startup of each replica does not interfere with each other, and when a service fails to start, the startup of other services will not be affected. Doris Operator uses concurrent mode to start services and has a built-in distributed architecture, which greatly simplifies the process of service expansion. Users only need to set the number of replicas to easily complete the expansion, completely freeing up the complexity of operation and maintenance operations.

- **Unnoticeable changes**:

  In a distributed environment, service restarts may cause temporary instability of services. Especially for services such as databases that have extremely high requirements for stability, how to ensure the stability of services during the restart process is a very important topic. Doris uses the following three mechanisms on Kubernetes to ensure the stability of the service restart process, thereby achieving an imperceptible experience for the business during the restart and upgrade process.

  1. Graceful exit
  2. Rolling restart
  3. Actively stop query allocation

- **Host system configuration**:

  In some scenarios, it is necessary to configure the host system parameters to achieve the ideal performance of Apache Doris. In the containerized scenario, the uncertainty of host deployment and the difficulty of parameter modification bring challenges to users. To solve this problem, Doris Operator uses Kubernetes's initialization container to make the host parameters configurable. Doris Operator allows users to configure commands executed on the host and make them effective by initializing containers. To improve availability, Doris Operator abstracts the configuration method of Kubernetes initialization containers, making the setting of host commands simpler and more intuitive.

- **Persistent configuration**:

  Doris Operator uses the Kubernetes StorageClass mode to provide storage configuration for each service. It allows users to customize the mount directory. When customizing the startup configuration, if the storage directory is modified, the directory can be set as a persistent location in the custom resource, so that the service uses the specified directory in the container to store data.

- **Runtime debugging**:

  One of the biggest challenges for Trouble Shooting with containerized services is how to debug at runtime. While pursuing availability and ease of use, Doris Operator also provides more convenient conditions for problem location. In the basic image of Doris, a variety of tools for problem location are pre-set. When you need to view the status in real time, you can enter the container through the exec command provided by kubectl and use the built-in tools for troubleshooting. When the service cannot be started for unknown reasons, Doris Operator provides a Debug running mode. When a Pod is set to Debug startup mode, the container will automatically enter the running state. At this time, you can enter the container through the `exec` command, manually start the service and locate the problem. For details, please refer to this document

Compatibility

Doris Operator is developed in accordance with standard K8s specifications and is compatible with all standard K8s platforms, including those provided by mainstream cloud vendors, self-built K8s platforms based on standards, and user-built platforms.

Cloud vendor compatibility

Fully compatible with the containerized service platforms of mainstream cloud vendors. For environment preparation and usage suggestions for Doris Operator, please refer to the following documents:

- Alibaba Cloud
- AWS

Installation and management

Prerequisites

Before deployment, you need to check the host system. Refer to Operating System Check

Deploy Doris Operator

Before deploying Doris Operator on Kubernetes, you need to install Doris Operator CRD.

- For detailed installation documents, please refer to: Doris Operator Installation

Deploy Doris cluster

- For cluster configuration documents, please refer to: Doris Operator Cluster Configuration
- For installation documents, please refer to: Doris Cluster Installation

Cluster operation and maintenance

- For cluster operation and maintenance documents, please refer to: Doris Operator Cluster Operation
- For cluster access documents, please refer to: Doris Operator Cluster Access

Recommendations on Alibaba Cloud

Alibaba ACK

Alibaba Cloud Container Service ACK is a managed containerized service after purchasing an ECS instance, so you can obtain full access control permissions to adjust related system parameters. Use the instance image: Alibaba Cloud Linux 3. The current system parameters fully meet the requirements for running Doris. Those that do not meet the requirements can also be corrected in the container through the K8s privileged mode to ensure stable operation.
Alibaba Cloud ACK cluster, deployed using Doris Operator, most environmental requirements can be met by the ECS default configuration. If not met, Doris Operator can correct it by itself. Users can also manually correct it, as follows:

Already exists cluster

If the Container Service cluster has already been created, you can modify it by referring to this document: Cluster Environment OS Checking
Focus on the BE startup parameter requirements:
1. Disable and close swap: `swapon --show` will not be output if it is not enabled 2. Check the maximum number of open file handles in the system `ulimit -n` 3. Check and modify the number of virtual memory areas `sysctl vm.max_map_count` 4. Whether transparent huge pages are closed `cat /sys/kernel/mm/transparent_hugepage/enabled` contains never
The default values of the corresponding parameters are as follows:

```
[root@iZj6c12a1czxk5oer9rbp8Z ~]# swapon --show
[root@iZj6c12a1czxk5oer9rbp8Z ~]# ulimit -n
65535
[root@iZj6c12a1czxk5oer9rbp8Z ~]# sysctl vm.max_map_count
vm.max_map_count = 262144
[root@iZj6c12a1czxk5oer9rbp8Z ~]# cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

Create a new cluster

If the cluster has not been purchased and created, you can click “Create Cluster” in the Alibaba Cloud Container Service ACK console to purchase it. You can adjust the configuration as needed. The above parameters can be added to the system adjustment script in “Instance Pre-customized Data” in the “Node Pool Configuration” step of creating a cluster. After the cluster is started, restart the node to complete the configuration. The reference script is as follows:

```
#!/bin/bash
chmod +x /etc/rc.d/rc.local
echo "sudo systemctl stop firewalld.service" >> /etc/rc.d/rc.local
echo "sudo systemctl disable firewalld.service" >> /etc/rc.d/rc.local
echo "sysctl -w vm.max_map_count=2000000" >> /etc/rc.d/rc.local
echo "swapoff -a" >> /etc/rc.d/rc.local
current_limit=$(ulimit -n)
desired_limit=1000000
```

```
config_file="/etc/security/limits.conf"
 if [ "$current_limit" -ne "$desired_limit" ]; then
    echo "* soft nofile 1000000" >> "$config_file"
    echo "* hard nofile 1000000" >> "$config_file"
fi
```

Alibaba ACS

The ACS service is a cloud computing service that uses K8s as the user interface to provide container computing resources, providing elastic computing resources that are billed on demand. Unlike the above ACK, you do not need to pay attention to the specific use of ECS. The following points should be noted when using ACS:

Image repository

When using ACS, it is recommended to use the supporting Alibaba Container Registry(ACR). The personal and enterprise versions are enabled on demand.

After configuring the ACR and image transfer environment, you need to migrate the official image provided by Doris to the corresponding ACR.

If you use a private ACR to enable authentication, you can refer to the following steps:

1. You need to set a `secret` of type `docker-registry` in advance to configure the authentication information for accessing the image warehouse.

```
kubectl create secret docker-registry image-hub-secret --docker-server={your-server} --docker-username={your-username} --docker-password={your-pwd}
```

2. Configure the secret using the above steps on DCR:

```
spec:
  feSpec:
    replicas: 1
    image: crpi-4q6quaxa0ta96k7h-vpc.cn-hongkong.personal.cr.aliyuncs.com/selectdb-test/doris.fe-ubuntu:3.0.3
    imagePullSecrets:
    - name: image-hub-secret
  beSpec:
    replicas: 3
    image: crpi-4q6quaxa0ta96k7h-vpc.cn-hongkong.personal.cr.aliyuncs.com/selectdb-test/doris.be-ubuntu:3.0.3
    imagePullSecrets:
    - name: image-hub-secret
    systemInitialization:
      initImage: crpi-4q6quaxa0ta96k7h-vpc.cn-hongkong.personal.cr.aliyuncs.com/selectdb-test/alpine:latest
```

Be systemInitialization

Currently, Alibaba Cloud is gradually pushing the ability to enable privileged mode on fully managed ACS services (some regions may not be enabled yet, you can submit a work order to apply for the ability to be enabled).
The Doris BE node startup requires some special environment parameters, such as Modify the number of virtual memory areas `sysctl -w vm.max_map_count=2000000`

Setting this parameter inside the container requires modifying the host configuration, so regular K8s clusters need to enable privileged mode in the pod. Operator adds `InitContainer` to the BE pod through `systemInitialization` to perform such operations.

> **Tip**
>
> **If the current cluster cannot use privileged mode, the BE node cannot be started**. You can choose ACK container service + host to deploy the cluster.

Service

Since the ACS service is a cloud computing service that uses K8s as the user interface to provide container computing resources, it provides computing resources. Its nodes are virtual computing resources, and users do not need to pay attention to them. They are charged according to the amount of resources used, and can be expanded infinitely. That is, there is no physical concept of conventional nodes:

```
$ kubectl get nodes
NAME                           STATUS   ROLES   AGE   VERSION
virtual-kubelet-cn-hongkong-d  Ready    agent   27h   v1.31.1-aliyun.1
```

Therefore, when deploying the Doris cluster, serviceType disables the NodePort mode and allows the use of ClusterIP and LB modes.

- ClusterIP mode:

  ClusterIP modethe default network mode of Operator. For specific usage and access methods, please refer to this document

- Load balancing mode:

  can be configured as follows:

  - Configure LB access through the DCR service annotations provided by Operator. The steps are as follows:
    1. A CLB or NLB instance has been created through the load balancing console, and the instance is in the same region as the ACK cluster. If you haven't created one yet, see Create and manage a CLB instance and Create and manage an NLB instance。
    2. Through DCR configuration, the access annotations of the above LB are in the following format:

    ```
    feSpec:
      replicas: 3
      image: crpi-4q6quaxa0ta96k7h-vpc.cn-hongkong.personal.cr.aliyuncs.com/selectdb-test/doris.fe-ubuntu:3.0.3
      service:
        type: LoadBalancer
        annotations:
          service.beta.kubernetes.io/alibaba-cloud-loadbalancer-address-type: "intranet"
    ```

  - Host the LB service through the ACS console and generate a statefulset service bound to the corresponding resource control of FE or BE

The steps are as follows:

1. serviceType is ClusterIP (default policy)
2. You can create a load balancing service through the Alibaba Cloud console interface: Container Compute Service ACS -> Cluster List -> Cluster -> Service, and use the `Create` button.
3. Select the newly created LB in the interface for creating `service`, which will be bound to `service` and will also be deregistered when the `service` is deregistered. However, this `service` is not controlled by Doris Operator.

Recommendations on AWS

AWS EKS

Create a new cluster

The containers running in the EKS cluster are hosted on EC2 instances, and the EC2 instances need to be configured at the system level according to the requirements of Doris. When creating a cluster, the user needs to confirm the EKS mode, auto mode or non-auto mode. It is recommended not to use the autonomous mode here, because the computing resources in the autonomous mode are allocated and recycled through the built-in node pool. At each resource application or release, the existing resources will be reintegrated. For stateful services such as statefulset, especially those that take a long time to start and services with strict distributed collaboration requirements such as Doris, it will cause turbulence in all services in the shared node pool. The direct phenomenon is that it may cause all nodes in the entire Doris cluster to drift (this is more terrifying than restarting. This process will not be a rolling restart, but when the previously stable service is on the node, the node is forced to be released, and K8s schedules these pods to the new node) There are great security risks to the production environment.
- As mentioned above, the autonomous mode is suitable for stateless service operation and maintenance deployment. The non-autonomous mode is recommended for installing the Doris cluster - Recommended operating system image: Amazon Linux 2

Already exists cluster

On an existing cluster (non-auto mode), you can run the Doris cluster through Doris Operator, unless the cluster is restricted to use the privileged mode of K8s. It is recommended that the existing cluster configure a new node group to deploy and maintain Doris cluster resources separately, which involves the system settings for Doris BE operation and may adjust the system parameters of the host machine.

Assess DockerHub

If you need to access the DockerHub public image repository on EKS, you need to add network plug-ins such as `Amazon VPC CNI`, `CoreDNS`, `kube-proxy` to the cluster, and when configuring VPC for the cluster, select a subnet that can access the public environment.

K8s Privileged

Under EKS, EC2 instances completely belong to the current EKS user, and there is no situation where different user clusters affect each other in the resource pool and disable the K8s privileged mode.

- If your EKS allows privileged mode (allowed by default), you don’t need to care about system parameters. Doris Operator will adjust system parameters for Doris operation by default.
- If privileged mode is not allowed, you need to adjust the following system parameters on the host:
    - Modify the number of virtual memory areas: `sysctl -w vm.max_map_count=2000000` to adjust the maximum number of virtual memory mappings. View it through `sysctl vm.max_map_count`.

- Turn off transparent huge pages: Transparent huge pages may have an adverse effect on performance, so you need to turn it off. Judge by whether cat /sys/kernel/mm/transparent_hugepage/enabled contains never.
- Set the maximum number of open file handles: adjust the maximum number of file handles by modifying `/etc/security/limits.conf`. View it through `ulimit -n`.
- Disable swap: `swapoff -a` is used to disable all swap partitions and files. Verify with `swapon --show`, no output if not enabled.

Storage

Doris Operator must use persistent configuration in the production environment to save the node storage. EBS is recommended.

There are the following points to note:

- In the cluster configuration installation or management interface, add the EBS storage plug-in. If you use the EKS autonomous mode (not recommended), it is recommended to install EFS, and the storage plug-in needs to have the corresponding role permissions
- Ensure that the IAM role of the EKS node has the following permissions:
  - AmazonEC2FullAccess
  - AmazonEKSWorkerNodePolicy
  - AmazonEKS_CNI_Policy
  - AmazonSSMManagedInstanceCore

Compute resource pool configuration

- AWS Resource Groups(recommended)
  You can create a node group in the cluster creation interface, or add a node group after the cluster is initialized. Use EC2 > Launch Template > Create Launch Template to set the node group launch template for the node pool. Use the template to inject scripts to automatically adjust the system environment configuration of the EC2 instance to ensure that the node automatically configures the required system parameters when it starts. By configuring the node template, you can also achieve the ability to automatically configure the system parameters of the newly added nodes when using EKS automatic elastic expansion and contraction.
  Example startup script:

```
#!/bin/bash
chmod +x /etc/rc.d/rc.local
echo "sudo systemctl stop firewalld.service" >> /etc/rc.d/rc.local
echo "sudo systemctl disable firewalld.service" >> /etc/rc.d/rc.local
echo "sysctl -w vm.max_map_count=2000000" >> /etc/rc.d/rc.local
echo "swapoff -a" >> /etc/rc.d/rc.local
current_limit=$(ulimit -n)
desired_limit=1000000
config_file="/etc/security/limits.conf"
if [ "$current_limit" -ne "$desired_limit" ]; then
  echo "* soft nofile 1000000" >> "$config_file"
  echo "* hard nofile 1000000" >> "$config_file"
fi
```

In addition, when creating a node group, if you want to access it through the command line, you need to configure remote node access permissions.

- Default node pools(not recommended)

  Enable the resource pool used in EKS autonomous mode. When creating a node pool, you can choose a custom EC2 instance type and adjust the instance's CPU, memory and other resources. When configuring the node pool, you can add a startup script for the EC2 instance to adjust the system parameters. However, this type of resource pool requires autonomous mode and reduces the freedom to manage the cluster. For details on the specific modification operations, please refer to: Cluster Environment OS Checking

Doris Streamloader

Overview

Doris Streamloader is a client tool designed for loading data into Apache Doris. In comparison to single-threaded load using `curl`, it reduces the load latency of large datasets by its concurrent loading capabilities. It comes with the following features:

- **Parallel loading**: multi-threaded load for the Stream Load method. You can set the parallelism level using the `workers` parameter.
- **Multi-file load:** simultaneously load of multiple files and directories with one shot. It supports recursive file fetching and allows you to specify file names with wildcard characters.
- **Path traversal support:** support path traversal when the source files are in directories
- **Resilience and continuity:** in case of partial load failures, it can resume data loading from the point of failure.
- **Automatic retry mechanism:** in case of loading failures, it can automatically retry a default number of times. If the loading remains unsuccessful, it will print the command for manual retry.

Installation

Source Code: https://github.com/apache/doris-streamloader/ Binary File: https://doris.apache.org/download

> **The obtained result is the executable binary.**

How to use

```
doris-streamloader --source_file={FILE_LIST} --url={FE_OR_BE_SERVER_URL}:{PORT} --header={STREAMLOAD_HEADER} --db={TARGET_DATABASE}
--table={TARGET_TABLE}
```

1. `FILE_LIST` **support:**

- Single file

  E.g. Load a single file

  ```
  doris-streamloader --source_file="dir" --url="http://localhost:8330" --header="column_separator:|?columns:col1,col2" --
  db="testdb" --table="testtbl"
  ```

- Single directory

  E.g. Load a single directory

```
doris-streamloader --source_file="dir" --url="http://localhost:8330" --header="column_separator:|?columns:col1,col2" --
db="testdb" --table="testtbl"
```

- File names with wildcard characters (enclosed in quotes)

  E.g. Load file0.csv, file1.csv, file2.csv

```
doris-streamloader --source_file="file*" --url="http://localhost:8330" --header="column_separator:|?columns:col1,col2" --
db="testdb" --table="testtbl"
```

- A list of files separated by commas

  E.g. Load file0.csv, file1.csv, file2.csv

```
 doris-streamloader --source_file="file0.csv,file1.csv,file2.csv" --url="http://localhost:8330" --
header="column_separator:|?columns:col1,col2" --db="testdb" --table="testtbl"
```

- A list of directories separated by commas

  E.g. Load dir1, dir2, dir3

```
 doris-streamloader --source_file="dir1,dir2,dir3" --url="http://localhost:8330" --header="column_separator:|?
columns:col1,col2" --db="testdb" --table="testtbl"
```

2. `STREAMLOAD_HEADER` **supports all streamload headers separated with '?' if there is more than one**

Example:

```
doris-streamloader --source_file="data.csv" --url="http://localhost:8330" --header="column_separator:|?columns:col1,col2" --
db="testdb" --table="testtbl"
```

The parameters above are required, and the following parameters are optional:

Parameter	Description	Default Value	Suggestions
–u	Username of the database	root	
–p	Password	empty string	
–compress	Whether to compress data upon HTTP transmission	false	Remain as default. Compression and decompression can increase pressure on Doris Streamloader side and the CPU resources on Doris BE side, so it is advised to only enable this when network bandwidth is constrained.
–timeout	Timeout of the HTTP request sent to Doris (seconds)	60*60*10	Remain as default
–batch	Granularity of batch reading and sending of files (rows)	4096	Remain as default

Parameter	Description	Default Value	Suggestions
–batch_byte	Granularity of batch reading and sending of files (byte)	943718400 (900MB)	Remain as default
–workers	Concurrency level of data loading	0	“0” means the auto mode, in which the streamload speed is based on the data size and disk throughput. You can dial this up for a high-performance cluster, but it is advised to keep it below 10. If you observe excessive memory usage (via the memtracker in log), you can dial this down.
–disk_throughput	Disk throughput (MB/s)	800	Usually remain as default. This parameter is a basis of the automatic inference of workers. You can adjust this based on your needs to get a more appropriate value of workers.
–streamload_throughput	Streamload throughput (MB/s)	100	Usually remain as default. The default value is derived from the streamload throughput and predicted performance provided by the daily performance testing environment. To get a more appropriate value of workers, you can configure this based on your measured streamload throughput: (LoadBytes*1000)/(LoadTimeMs* 1024*1024)
–max_byte_per_task	Maximum data size for each load task. For a dataset exceeding this size, the remaining part will be split into a new load task.	107374182400 (100G)	This is recommended to be large in order to reduce the number of load versions. However, if you encounter a “body exceed max size” and try to avoid adjusting the streaming_load_max_mb parameter (which requires restarting the backend), or if you encounter a “-238 TOO MANY SEGMENT” error, you can temporarily dial this down.
–check_utf8		true	Remain as default
–debug	Print debug log	false	Remain as default

Parameter	Description	Default Value	Suggestions
–auto_retry	The list of failed workers and tasks for auto retry	empty string	This is only used when there is an load failure. The serial numbers of the failed workers and tasks will be shown and all you need is to copy and execute the the entire command. For example, if –auto_retry="1,1;2,1", that means the failed tasks include the first task in the first worker and the first task in the second worker.
–auto_retry_times	Times of auto retries	3	Remain as default. If you don't need retries, you can set this to 0.
–auto_retry_interval	Interval of auto retries	60	Remain as default. If the load failure is caused by a Doris downtime, it is recommended to set this parameter based on the restart interval of Doris.
–log_filename	Path for log storage	""	Logs are printed to the console by default. To print them to a log file, you can set the path, such as –log_filename="/var/log".

Result description

A result will be returned no matter the data loading succeeds or fails.

Parameter	Description
Status	Loading succeeded or failed
TotalRows	Total number of rows
FailLoadRows	Number of rows failed to be loaded
LoadedRows	Number of rows loaded
FilteredRows	Number of rows filtered
UnselectedRows	Number of rows unselected
LoadBytes	Number of bytes loaded
LoadTimeMs	Actual loading time
LoadFiles	List of loaded files

Examples:

- If the loading succeeds, you will see a result like:

```
Load Result: {
        "Status": "Success",
```

```
        "TotalRows": 120,
        "FailLoadRows": 0,
        "LoadedRows": 120,
        "FilteredRows": 0,
        "UnselectedRows": 0,
        "LoadBytes": 40632,
        "LoadTimeMs": 971,
        "LoadFiles": [
                "basic.csv",
                "basic_data1.csv",
                "basic_data2.csv",
                "dir1/basic_data.csv",
                "dir1/basic_data.csv.1",
                "dir1/basic_data1.csv"
        ]
}
```

- If the loading fails (or partially fails), you will see a retry message:

```
load has some error and auto retry failed, you can retry by :
./doris-streamloader --source_file /mnt/disk1/laihui/doris/tools/tpch-tools/bin/tpch-data/lineitem.tbl.1  --
url="http://127.0.0.1:8239" --header="column_separator:|?columns: l_orderkey, l_partkey, l_suppkey, l_linenumber,
l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag,l_linestatus,
l_shipdate,l_commitdate,l_receiptdate,l_shipinstruct,l_shipmode,l_comment,temp" --db="db" --table="lineitem1" -u root -p ""
--compress=false --timeout=36000 --workers=3 --batch=4096 --batch_byte=943718400 --max_byte_per_task=1073741824 --
check_utf8=true --report_duration=1 --auto_retry="2,1;1,1;0,1" --auto_retry_times=0 --auto_retry_interval=60
```

You can copy and execute the command. The failure message will also be provided:

```
Load Result: {
      "Status": "Failed",
      "TotalRows": 1,
      "FailLoadRows": 1,
      "LoadedRows": 0,
      "FilteredRows": 0,
      "UnselectedRows": 0,
      "LoadBytes": 0,
      "LoadTimeMs": 104,
      "LoadFiles": [
              "/mnt/disk1/laihui/doris/tools/tpch-tools/bin/tpch-data/lineitem.tbl.1"
      ]
}
```

Best practice

Parameter suggestions

1. Required parameters:
   `--source_file=FILE_LIST --url=FE_OR_BE_SERVER_URL_WITH_PORT --header=STREAMLOAD_HEADER --db=TARGET_DATABASE --table=TARGET_TABLE` If you need to load multiple files, you should configure all of them at a time in `source_file`.

2. The default value of `workers` is the number of CPU cores. When that is large, for example, 96 cores, the value of `workers` should be dialed down. **The recommended value for most cases is 8.**

3. `max_byte_per_task` is recommended to be large in order to reduce the number of load versions. However, if you encounter a “body exceed max size” and try to avoid adjusting the streaming_load_max_mb parameter (which requires restarting the backend), or if you encounter a `-238 TOO MANY SEGMENT` error, you can temporarily dial this down. **For most cases, this can remain as default.**

Two parameters that impacts the number of versions:

- `workers`: The more `workers`, the higher concurrency level, and thus the more versions. The recommended value for most cases is 8.
- `max_byte_per_task`: The larger `max_byte_per_task` , the larger data size in one single version, and thus the less versions. However, if this is excessively high, it could easily cause an `-238 TOO MANY SEGMENT` error. For most cases, this can remain as default.

Recommended commands

In most cases, you only need to set the required parameters and `workers`.

```
./doris-streamloader --source_file="demo.csv,demoFile*.csv,demoDir" --url="http://127.0.0.1:8030" --header="column_separator:," --db="demo" --table="test_load" --u="root" --workers=8
```

FAQ

- Before resumable loading was available, to fix any partial failures in loading would require deleting the current table and starting over. In this case, Doris Streamloader would retry automatically. If the retry fails, a retry command will be printed so you can copy and execute it.
- The default maximum data loading size for Doris Streamloader is limited by BE config `streaming_load_max_mb` (default: 100GB). If you don’t want to restart BE, you can also dial down `max_byte_per_task`.

  To show current `streaming_load_max_mb`:

  ```
  curl "http://127.0.0.1:8040/api/show_config"
  ```

- If you encounter an `-238 TOO MANY SEGMENT` error, you can dial down `max_byte_per_task`.

Apache Superset

Apache Superset is an open-source data mining platform that supports rich data source connections, various visualization methods, and fine-grained user access control. Its main features include self-service analysis, customizable dashboards, visualization (export) of analysis results, user/role access control, and an integrated SQL editor for SQL editing and querying.

Apache Superset version 3.1 provides an official connection method, officially supporting querying and visualization of both internal and external data from Apache Doris. Apache Doris version 2.0.4 and above are recommended.

Through this connection method, Superset can integrate Apache Doris databases and tables as data sources. To enable this feature, please follow the setup guide below:

- Prerequisite setup
- Configuring the Apache Doris data source in Apache Superset
- Building visualizations in Apache Superset
- Connection and usage tips

Installing Superset and the Doris Python client

1. Install Python 3, version 3.1.11 is recommended.
2. Install Apache Superset version 3.1 or later. See Installing Superset from PyPI repository for details.
3. Install the Apache Doris Python client on the Apache Superset server. You can refer to the following commands:

```
pip install pydoris
```

Verify Installation Results:

```
-> pip list | grep pydoris
pydoris                    1.1.0
```

After confirming the environment is correct, you can now configure a Doris data source in Superset and start building data visualizations!

Configuring a Doris Data Source in Superset

Now that you have installed the **Pydoris** and **Apache Superset** drivers, let's see how to define a data source in Superset that connects to the tpch database in Doris.

1. To connect to Apache Doris via Pydoris, you need to configure the SQLAlchemy URI connection string:

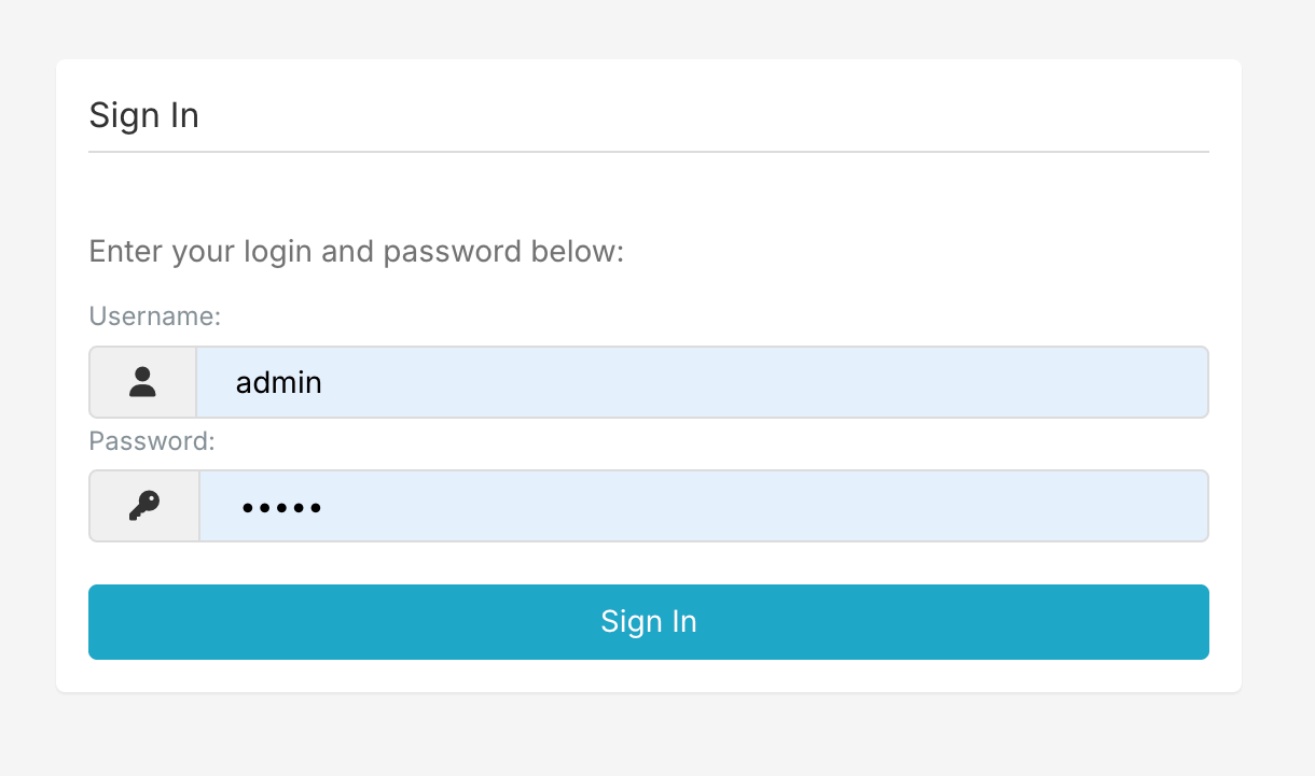
Complete the configuration in this format:

```
doris://<User>:<Password>@<Host>:<Port>/<Catalog>.<Database>
```

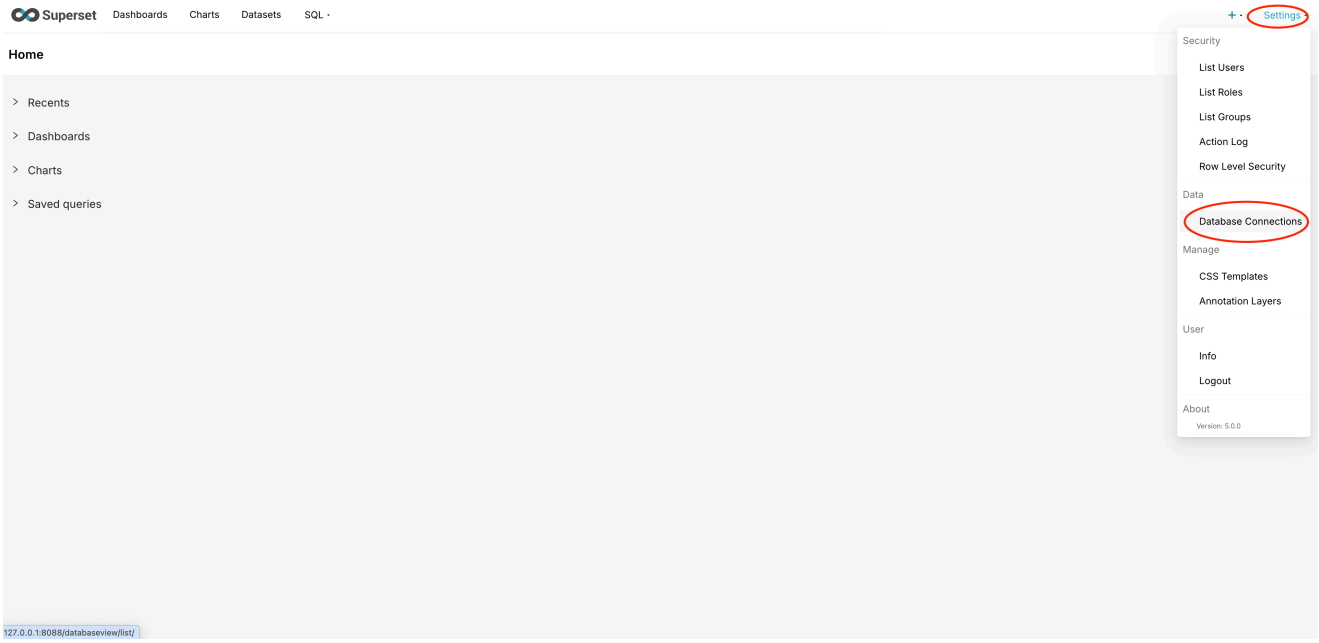
The URI parameters are explained below:

Parameter	Meaning	Example
User	Username	testuser
Password	Password	xxxxxx
Host	Database host	127.0.1.28
Port	Database query port	9030
Catalog	Doris Catalog, used when querying external tables and data lakes; internal tables are internal	internal
Database	Database name	tpch

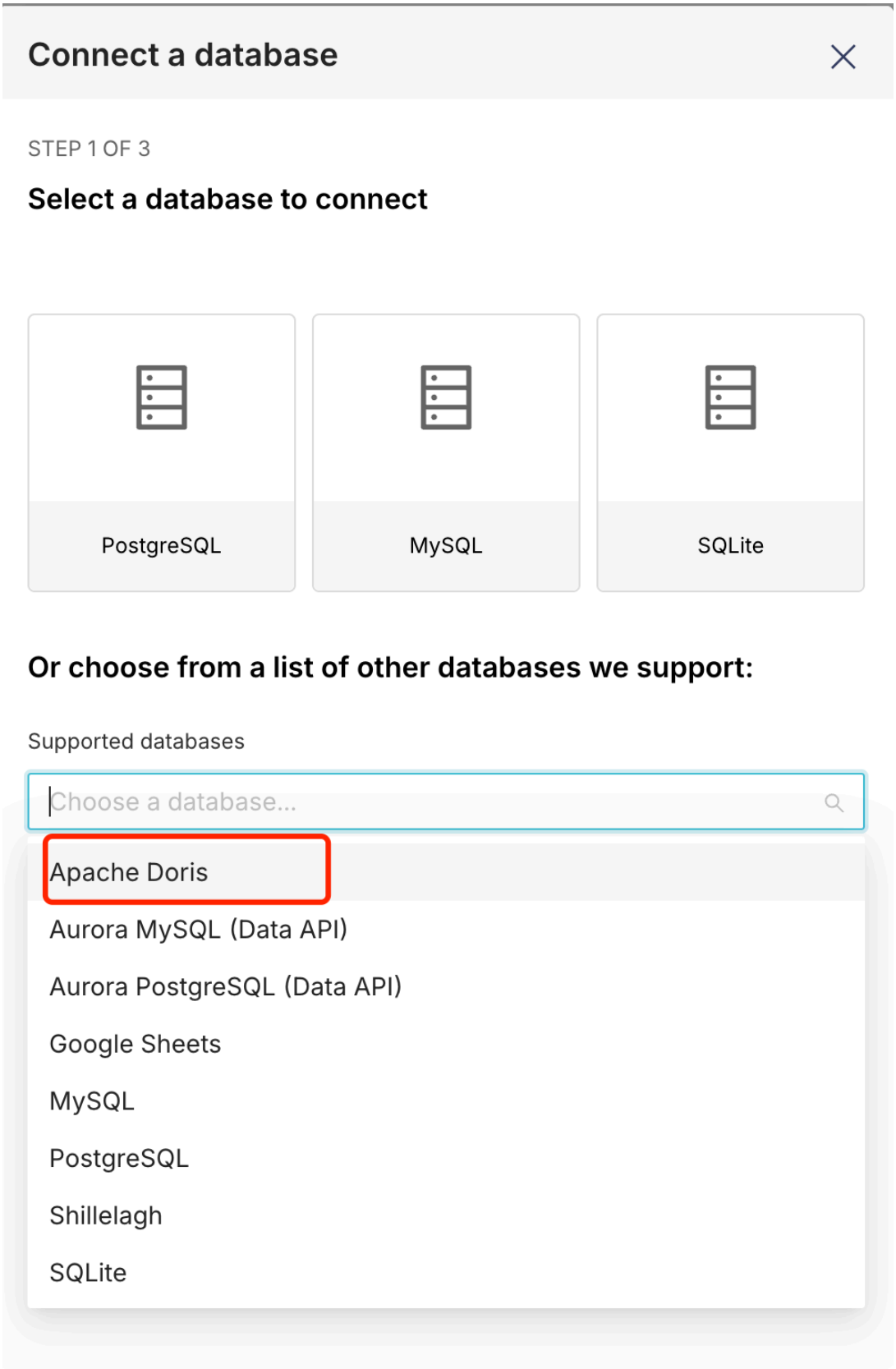
2. Accessing the Superset.



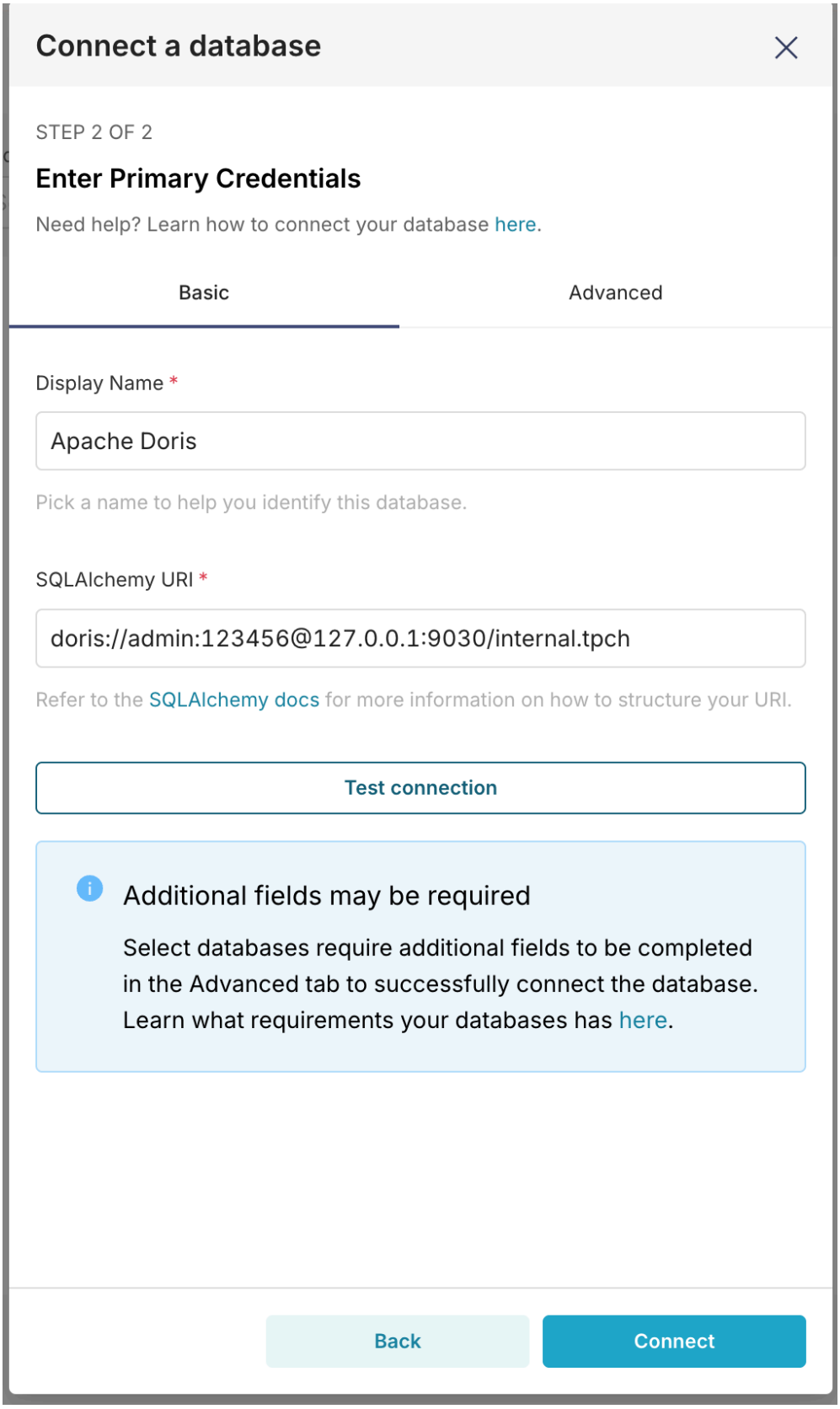
3. After logging in, click Settings -> Database Connectors in the upper right corner.



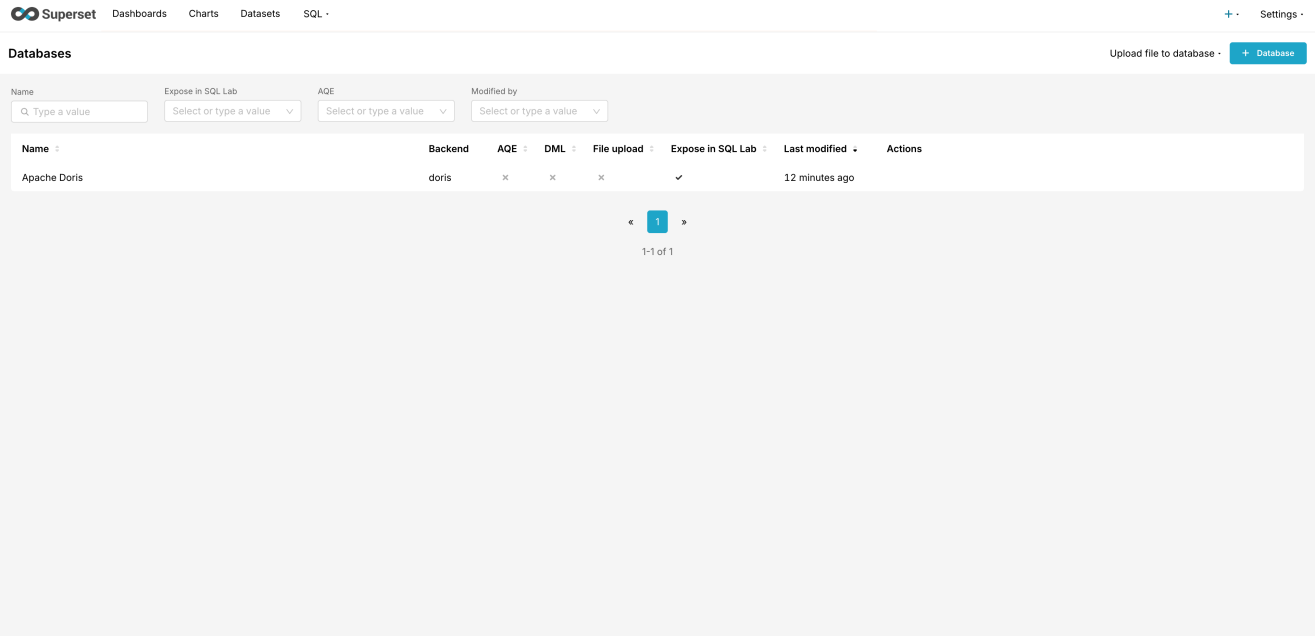
4. Click Add Database. In the Connect a database pop-up window, select Apache Doris.



5. Enter the SQLAlchemy URI in the connection information. After verifying the connection is correct, click Connect.



6. Adding a data source is now complete.



Next, we can build some visualizations in Superset!

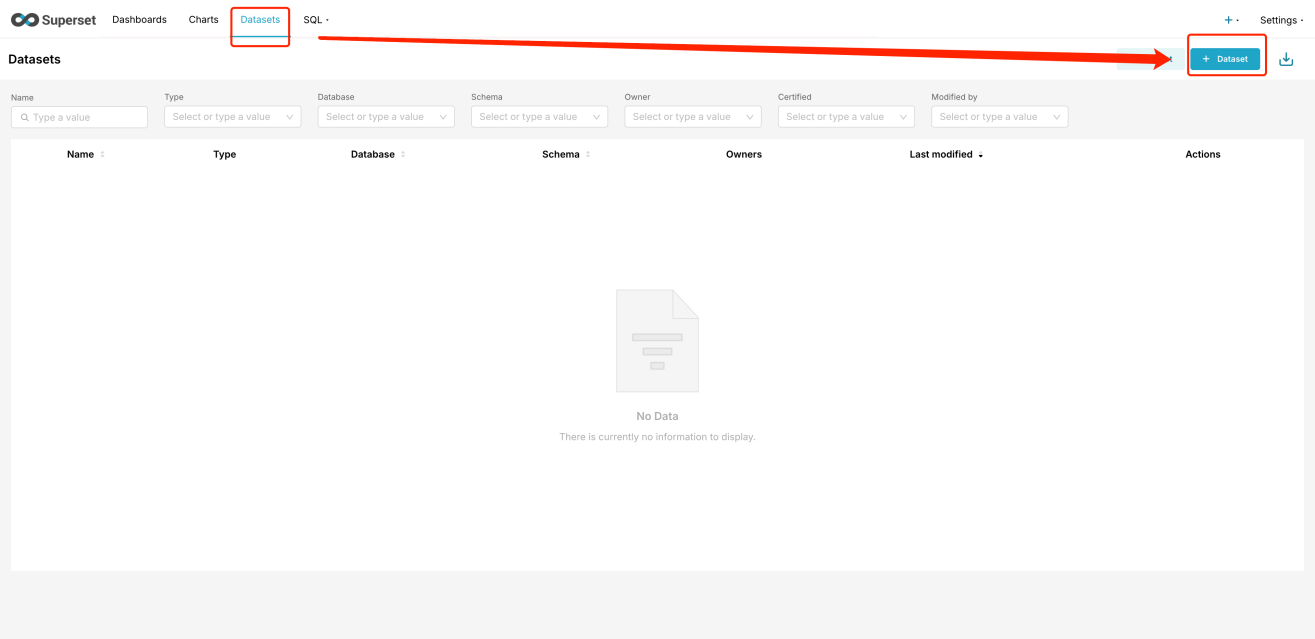
Building Visualizations in Superset

We choose TPC-H data as our data source. Refer to this document for instructions on building a Doris TPC-H data source.

Now that we have configured the Doris data source in Superset, let's visualize the data…

Suppose we need to analyze the time-varying curves of order amounts for different freight methods for cost analysis.

1. Click Datasets to add a Dataset

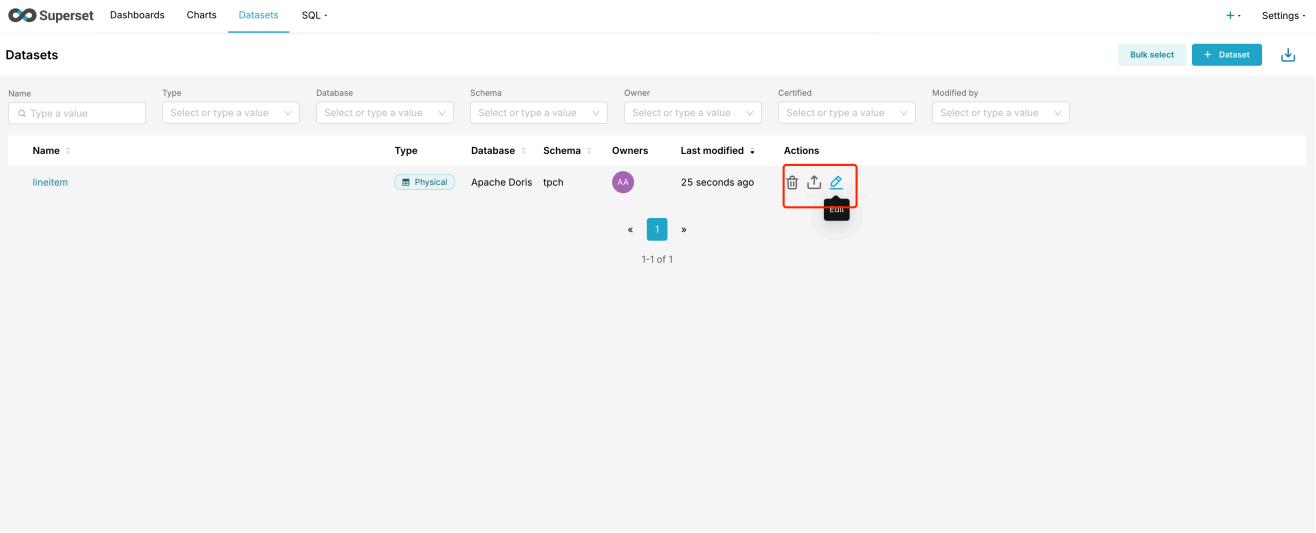


2. Select the following in sequence, then click Create dataset and create chart in the lower right corner:

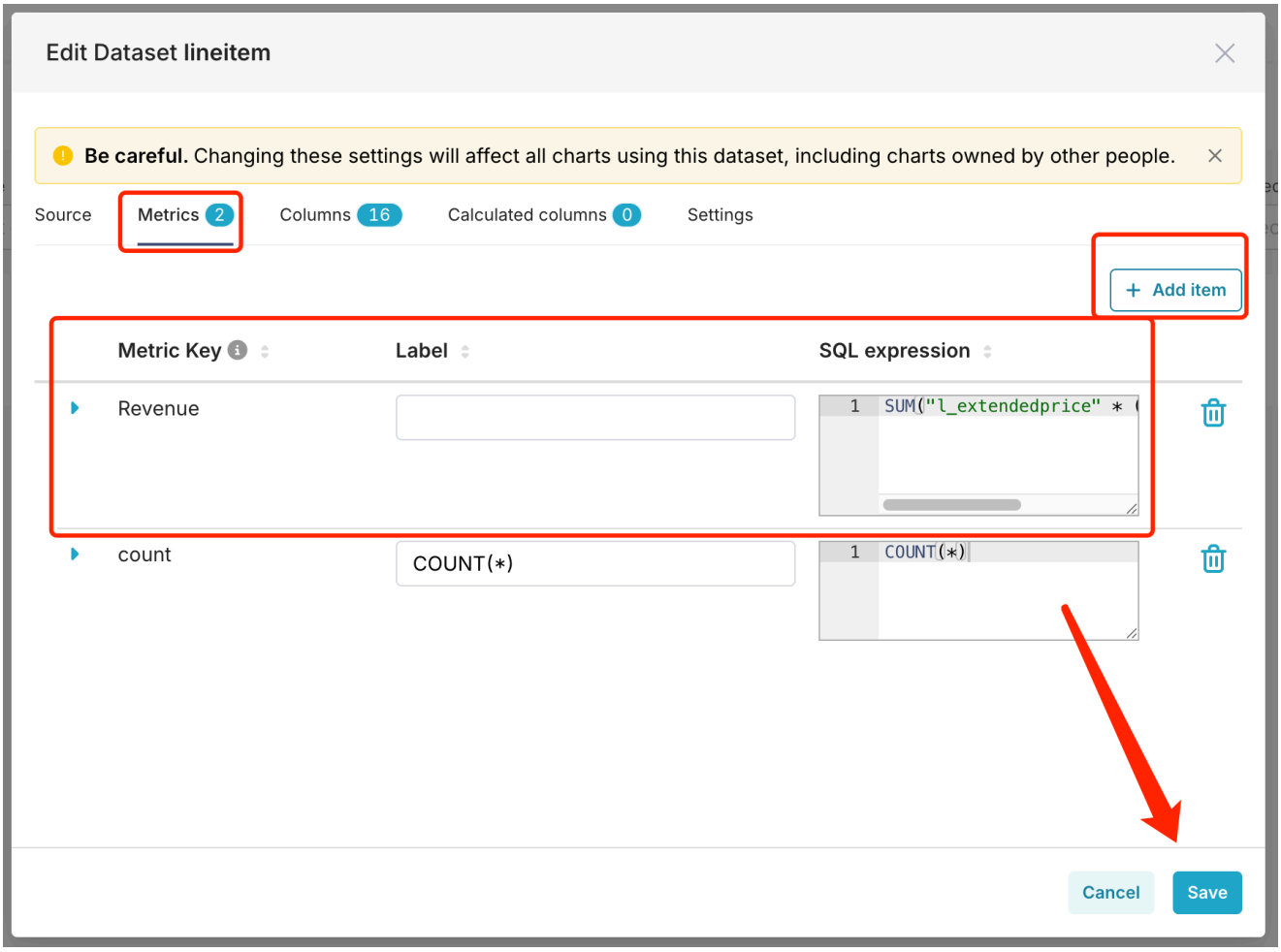
- Database：Doris
- Schema： tpch
- Table：lineitem



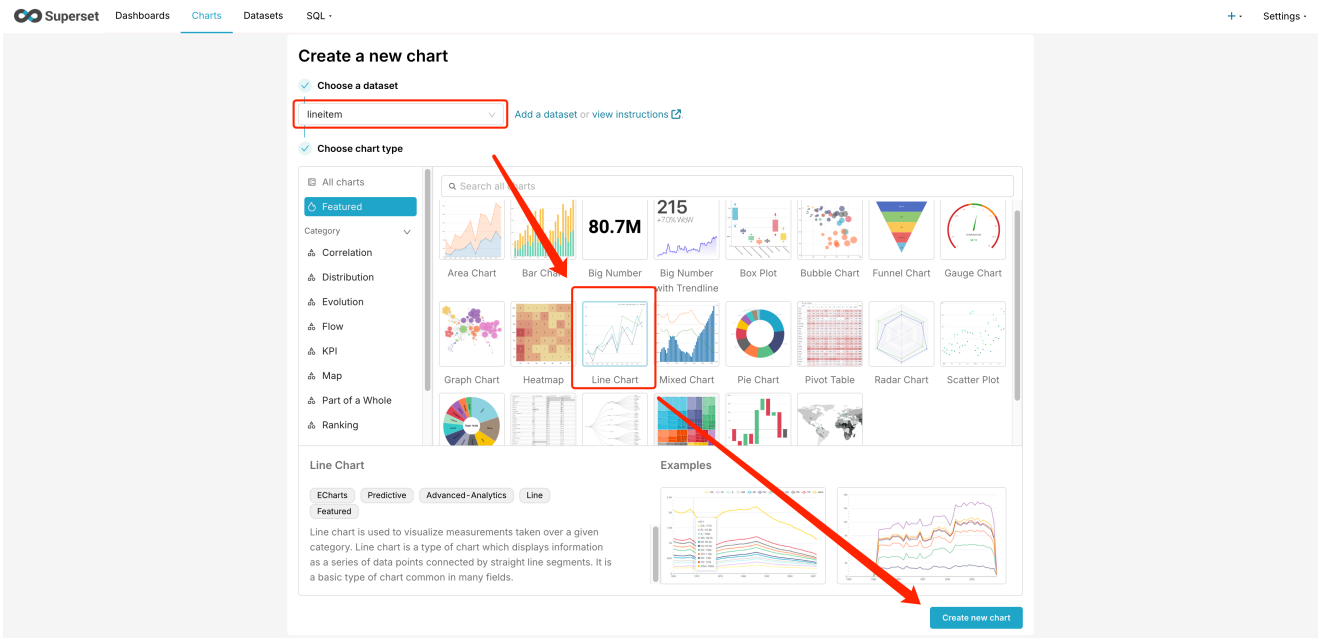
3. Edit the lineitem Dataset



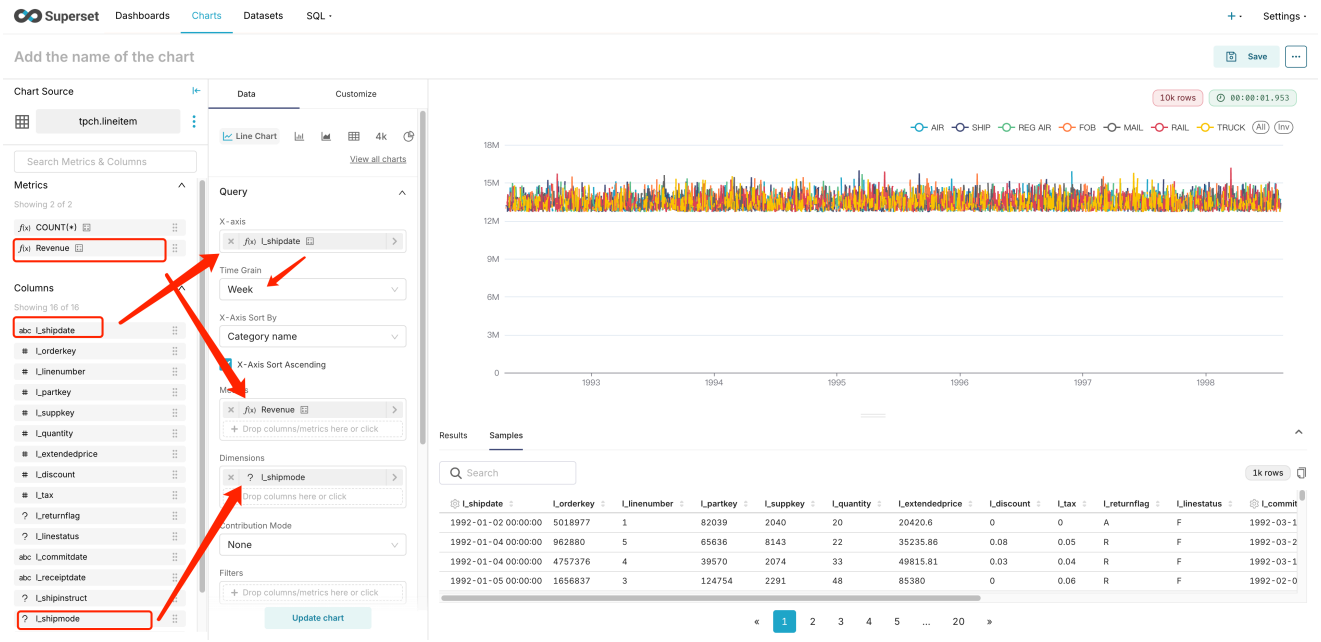
4. Click Metrics -> Add item to add a calculated metric.
   - Metric Key : Revenue
   - SQL expression : `SUM(l_extendedprice* (1 -l_discount))`



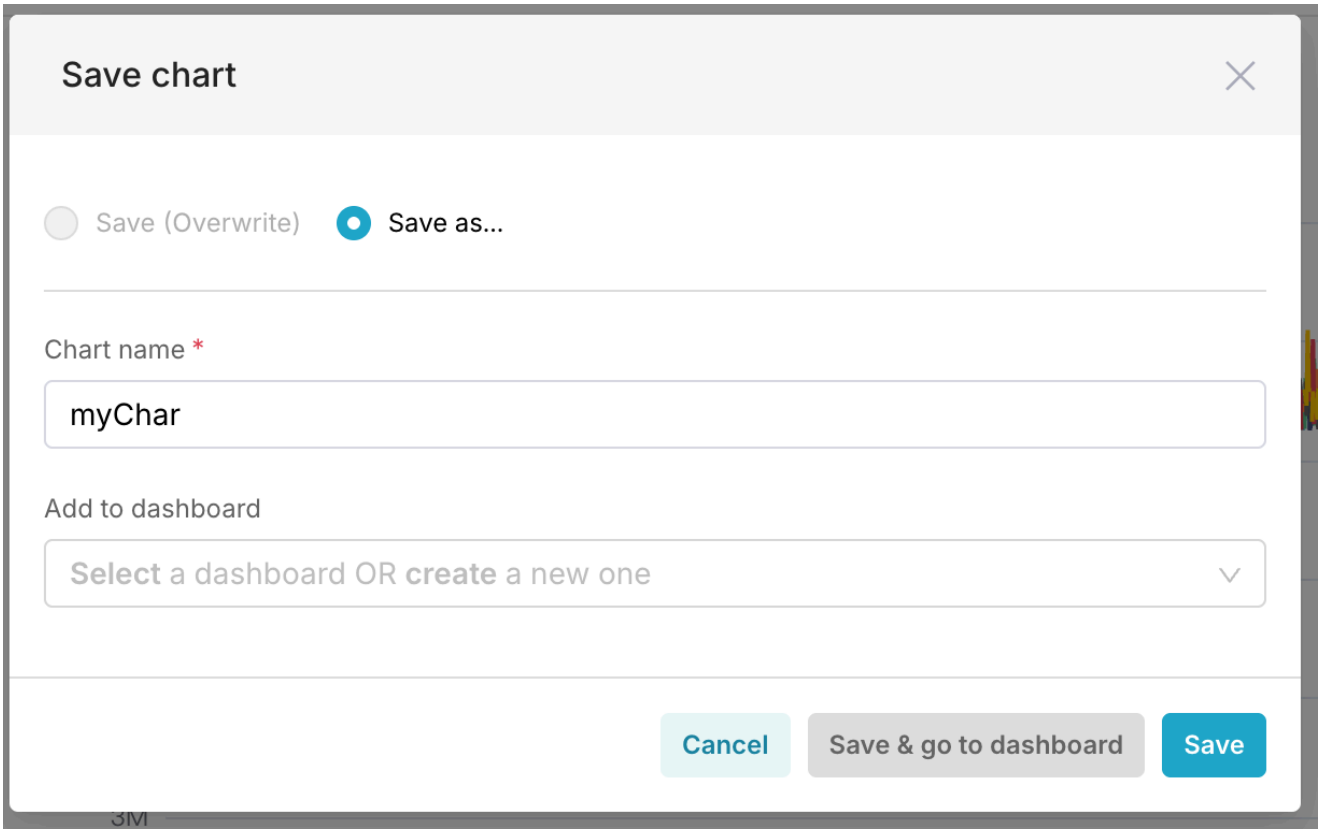
5. Go to Chart -> Add Chart, select lineitem for the dateset, and select Line Chart for the chart type.



6. Drag l_shipdate to the X-axis and set the time granularity. Simultaneously, drag the Revenum custom metric and the data column l_shipmode to Meters and Dimensions respectively.



7. Click Update chart to view the dashboard content. Click Save to save the dashboard.



At this point, Superset has been successfully connected to Apache Doris, and data analysis and visualization dashboard creation have been implemented.

Connection and Usage Tips

- Pre-install pydoris in the Superset environment to select Apache Doris when creating the database.
- Create Doris database tables reasonably according to actual needs, partitioning and bucketing by time, which can effectively reduce predicate filtering and most data transmission.

- It is recommended to use a VPC private connection to avoid the security risks introduced by public network access.
- Refine Doris user account roles and access permissions to avoid excessive delegation of permissions.

FineBI

FineBI Introduction

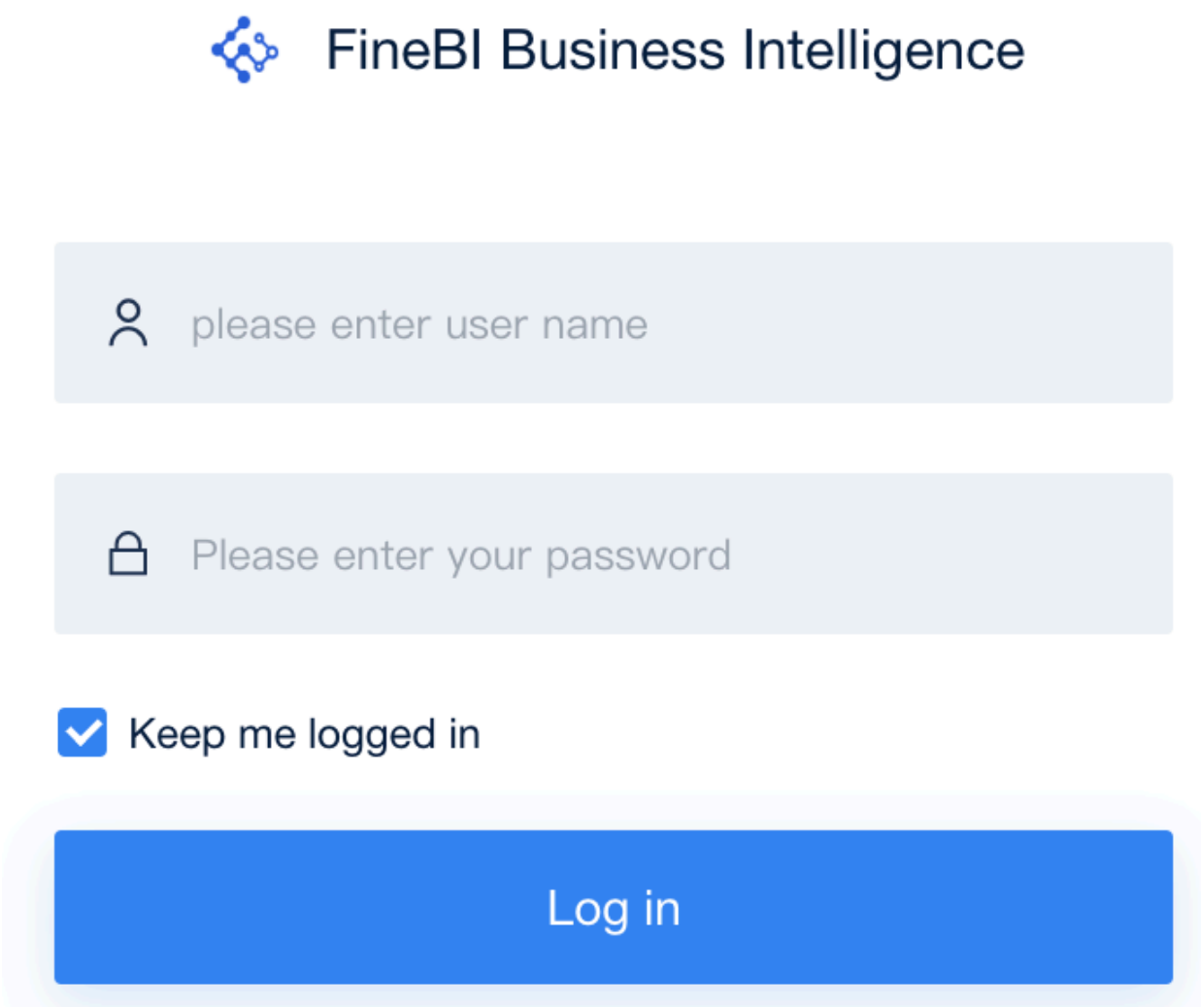
As a business intelligence product, FineBI has a system architecture of data processing, real-time analysis, multidimensional analysis Dashboard and other functions. FineBI supports rich data source connection and analysis and management of tables with multiple views. FineBI can successfully support the modeling and visualization of internal and external data of Apache Doris.

Precondition

Install FineBI 5.0 or later, Download link: https://intl.finebi.com/

Login and Connection

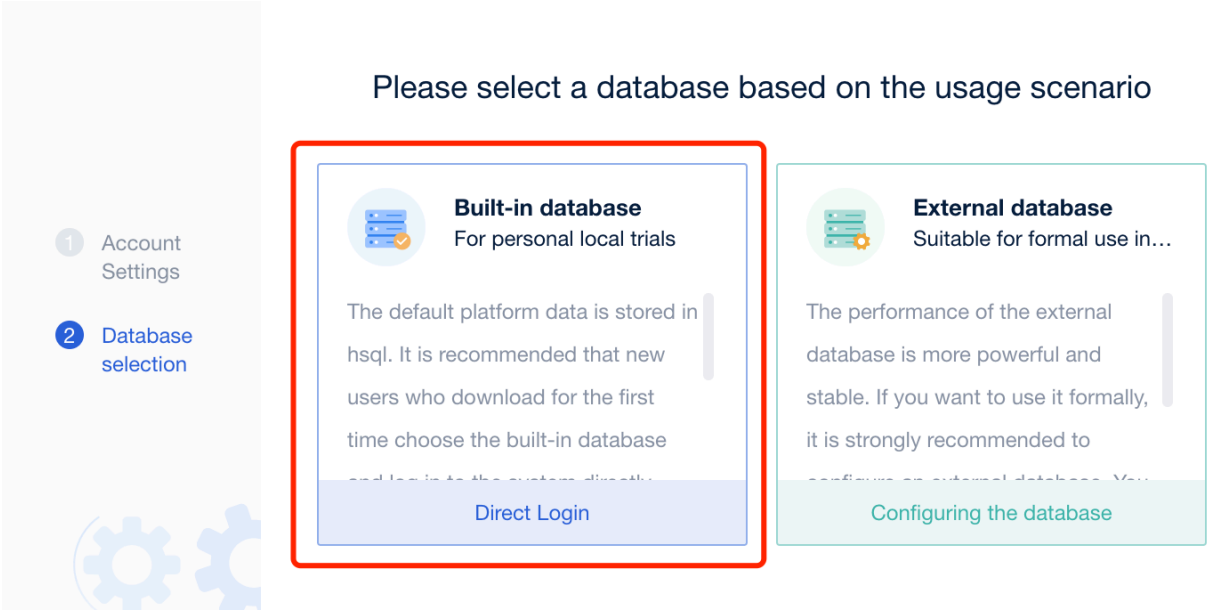
1. Create account and log in FineBI



login page

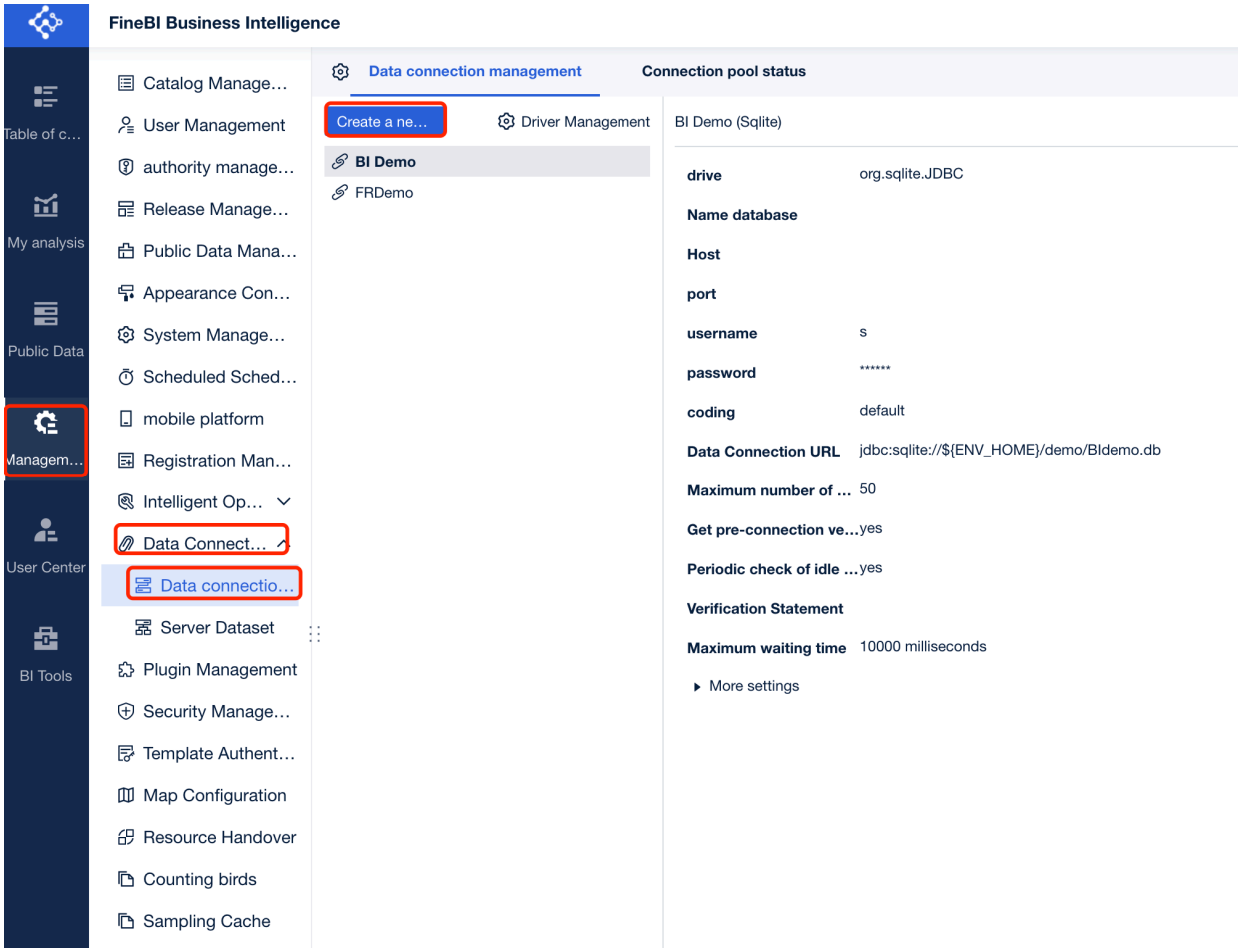
2. Select the Built-in database, If you need to select an external database configuration, the documentation is available: https://help.fanruan.com/finebi-en/doc-view-4437.html

   :::info Note It is recommended to select the built-in database as the information repository of FineBI. The database type selected here is not the target database for querying and analyzing data, but the database for storing and maintaining FineBI model, dashboard and other information. FineBI needs to add, delete, modify and check it. :::



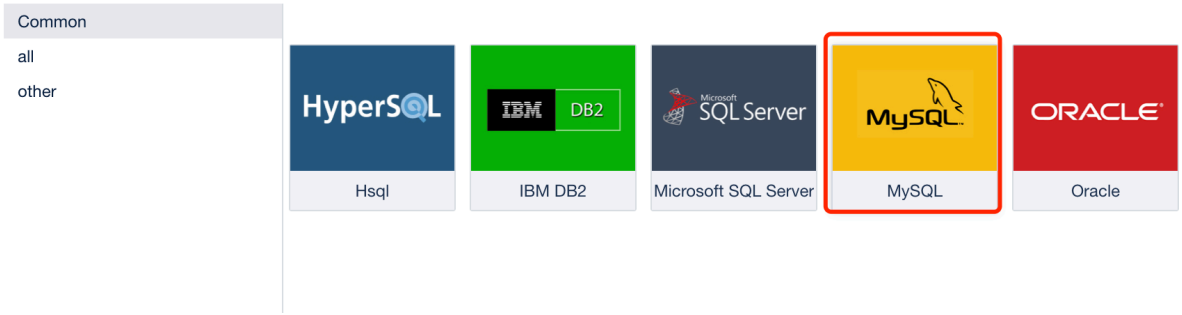
select database

3. Click the Management System button and select the database connection management in Data Connections to create a new database connection.



data connection

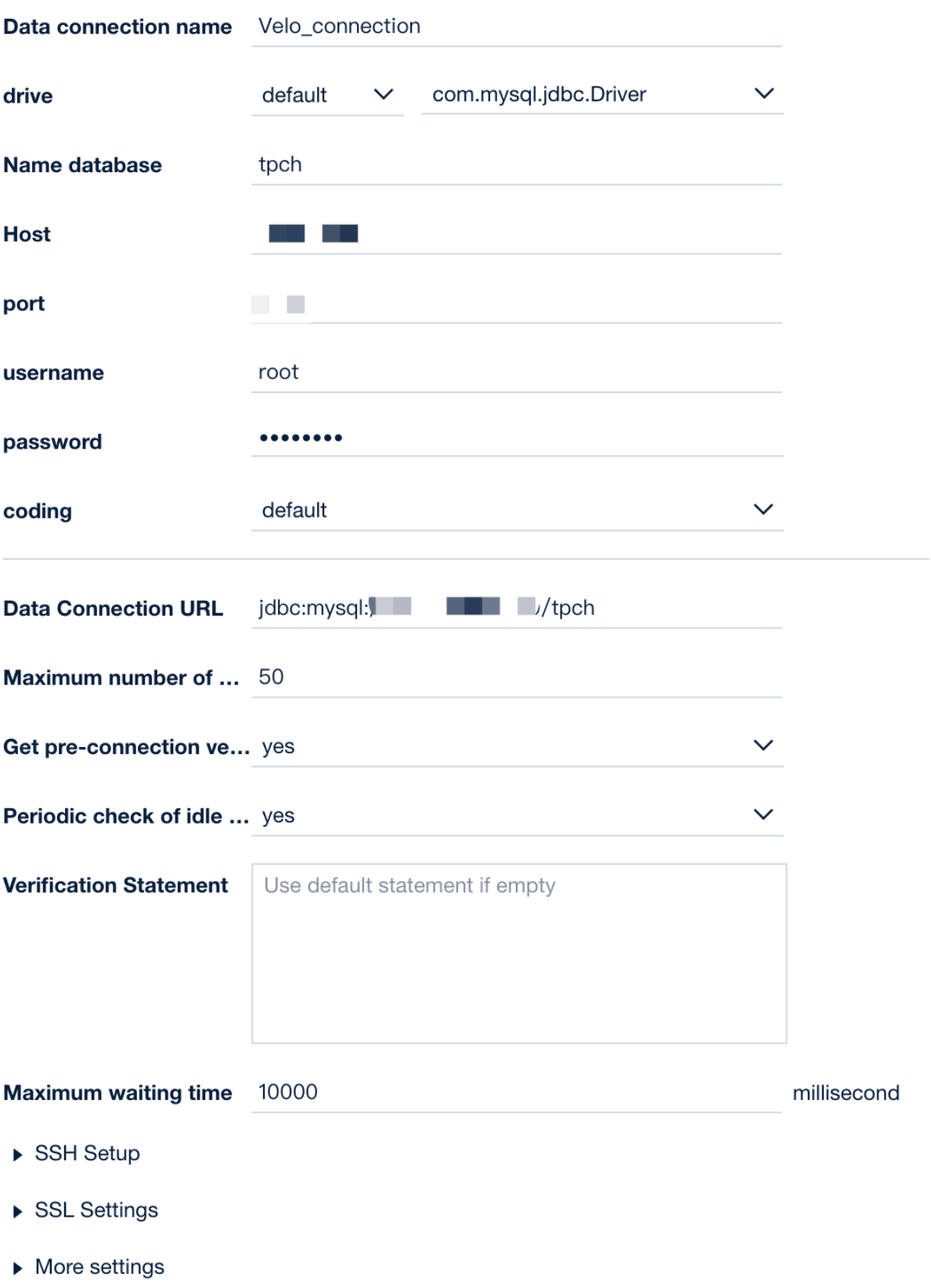
4. On the new database connection page, select MySQL database



select connection

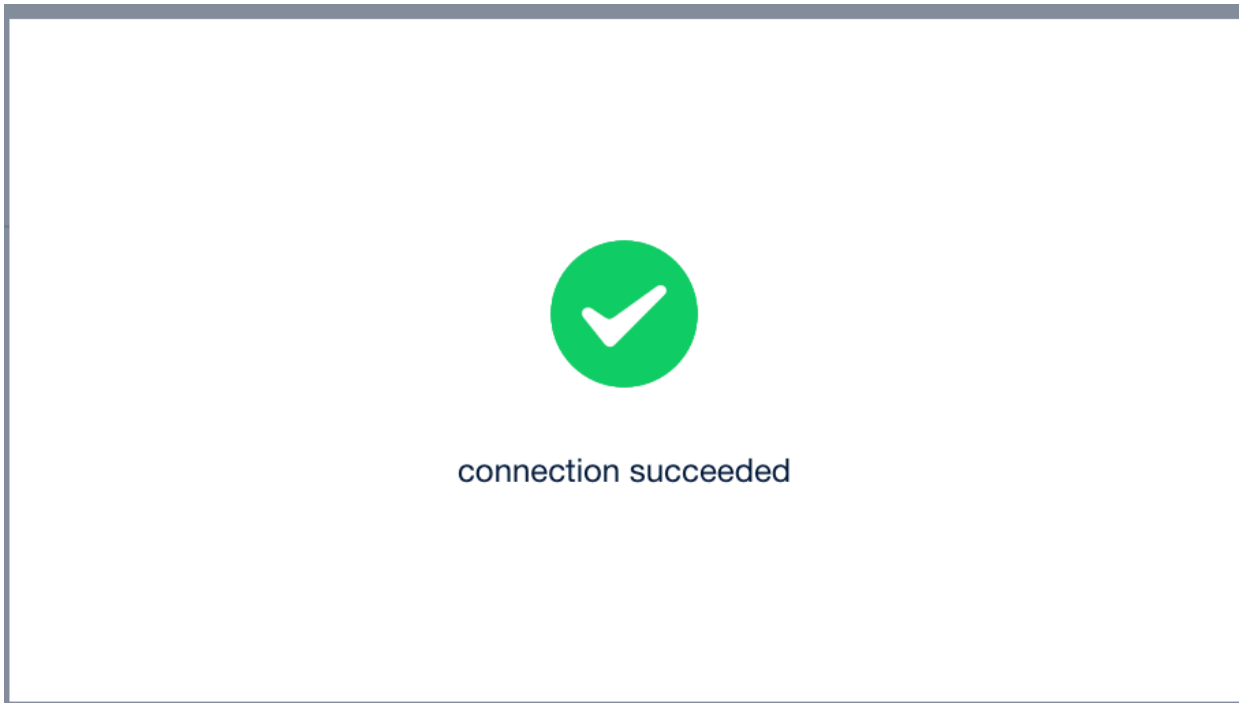
5. Fill in the link information of the Doris database
    - Parameters are described as follows:
        - Username: The username for logging into Doris.
        - Password: Password of the current user.
        - Host: IP address of the FE host in the Doris cluster.
        - Port: FE query port of the Doris cluster.

- Coding: Encoding format of the Doris cluster.
- Name Database: Target database in Doris cluster.



connection information

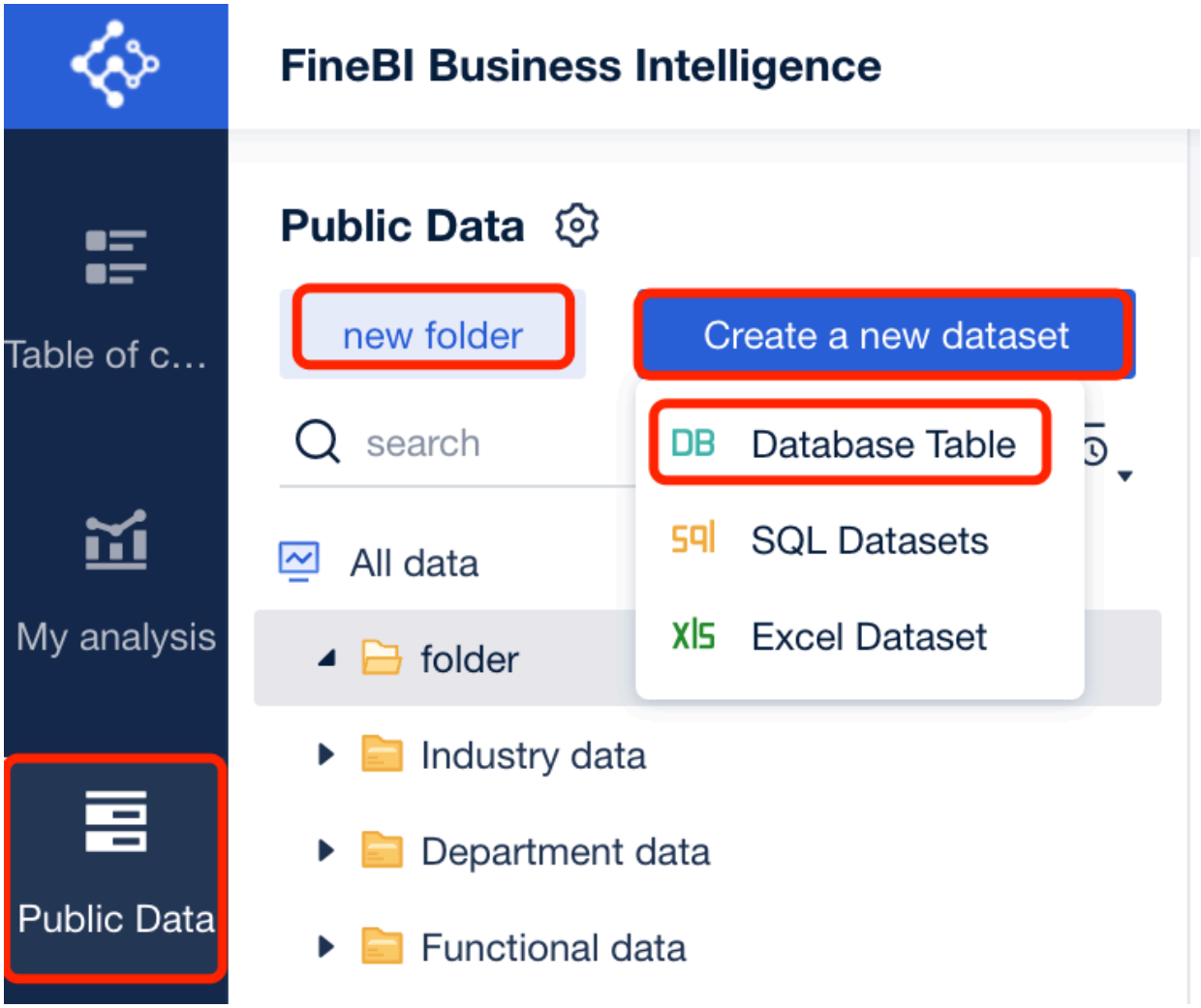
6. Click the test link. Connection succeeded is displayed when the connection information is correct



connection test

Create model

1. In the “Public Data” section, click to create a new dataset. Then click the database table



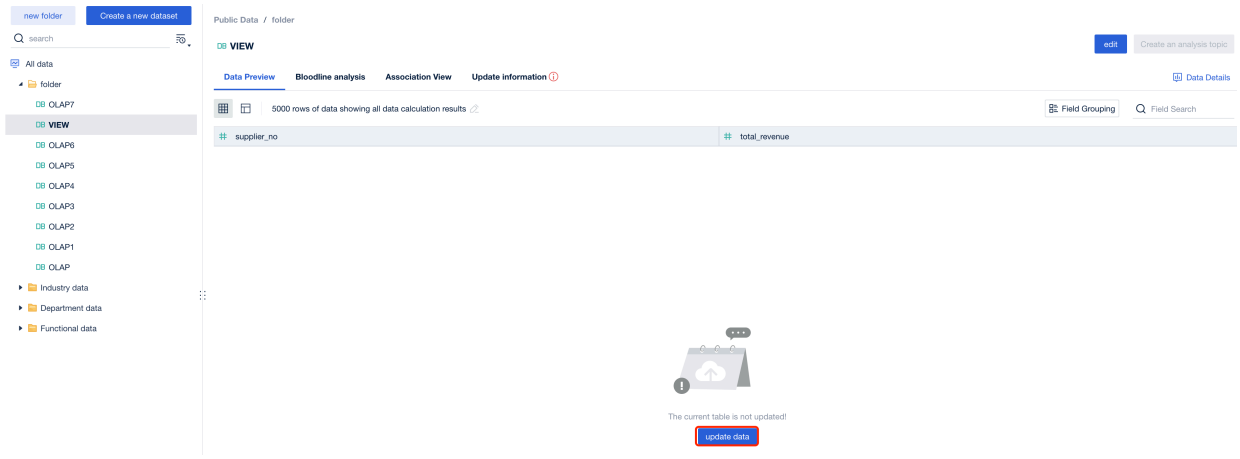
new dataset

2. You need to import tables in the existing database connection



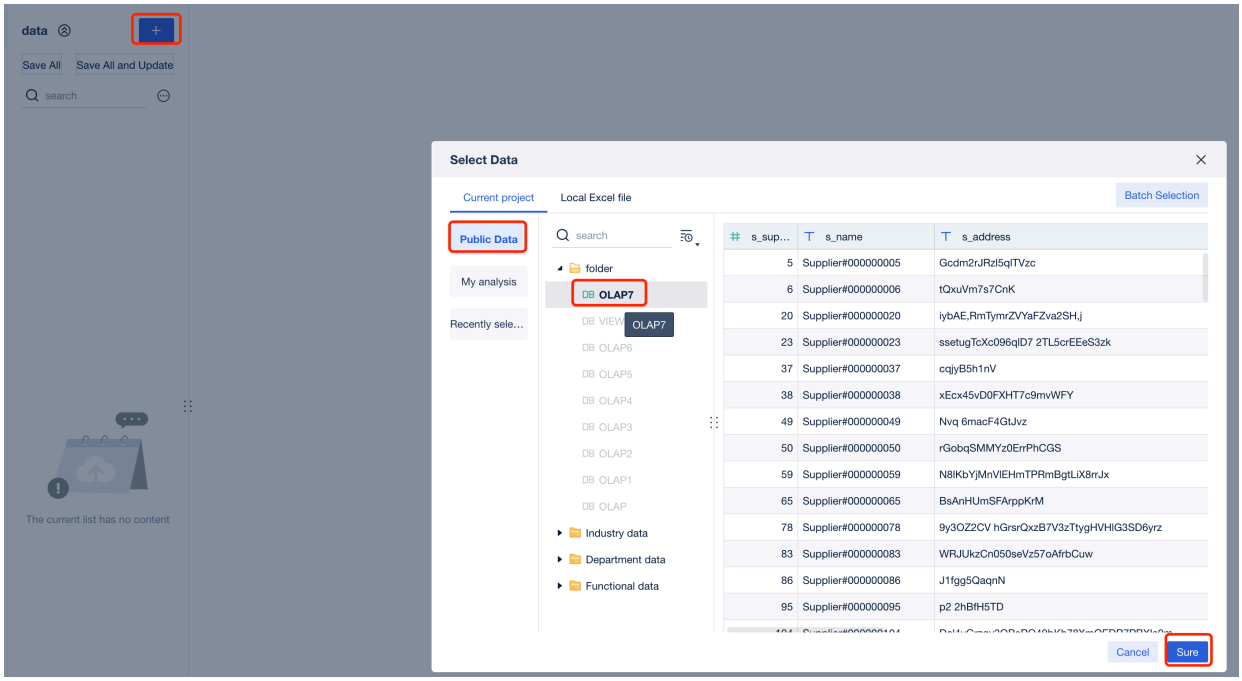
select table

3. You need to refresh each imported table after importing the table. You can analyze the table in the topic only after refreshing the table



refresh table

4. Add the imported public data to the edited topic, and then conduct compass analysis and configuration according to business logic.



data analysis

Metabase

Metabase is an open-source business intelligence tool that provides simple and easy-to-use data analysis and visualization capabilities, supports rich data source connections, and enables the rapid construction of interactive dashboards. Its key features include a user-friendly interface, ease of use, support for self-service analysis, visualization dashboard creation, data drill-down exploration, and an integrated SQL query editor for SQL queries and data export.

The Metabase Apache Doris Driver allows Metabase to connect to Apache Doris databases, enabling querying and visualization of both internal and external Doris data.

This driver allows Metabase to integrate Apache Doris databases and tables as data sources. To enable this feature, follow the setup guide below:

- Installing and configuring the driver
- Configuring the Apache Doris data source in Metabase
- Building visualizations in Metabase
- Connection and usage tips

Installing Metabase and the Doris Driver

Prerequisites

1. Download and install Metabase version 0.48.0 or later. See the Metabase Installation Documentation for details.
2. Prepare the Apache Doris cluster.

Install the Doris Driver

First, you need to download the latest metabase-doris-driver.

Then install the driver. The installation method depends on your Metabase deployment method:

Metabase Standard Deployment

1. Download the Driver

2. Create the Metabase plugin directory (if it doesn't exist):

```
mkdir -p $path_metabase/plugins
```

3. Copy the JAR file to the plugin directory:

```
cp doris.metabase-driver.jar $path_metabase/plugins
```

4. Restart the Metabase service

Metabase Docker Deployment

If Metabase is started using Docker, it is recommended to start it by mounting `doris.metabase-driver.jar`. The plugin path inside the Docker container is `/plugins/`.

1. Download the Driver

2. Start Metabase using the following command:

```
docker run -d -p 3000:3000 --name metabase  -v $host_path/doris.metabase-driver.jar:/plugins/doris.metabase-driver.jar
metabase/metabase
```

Configuring a Doris Data Source in Metabase

Now that you have **Metabase** and **metabase-doris-driver** installed, let's see how to define a data source in Metabase that connects to the tpch database in Doris.

Connection Parameter Description

The following parameters need to be configured when connecting to Apache Doris:

Parameters	Meaning	Example
Display Name	Data source display name	Doris-TPCH
Host	Doris FE node address	127.0.0.1
Port	Doris Query Port (MySQL protocol port)	9030
Catalog name	Catalog name (optional, defaults to internal)	internal
Database name	Database name (required)	tpch
Username	Username	root
Password	Password	your_password

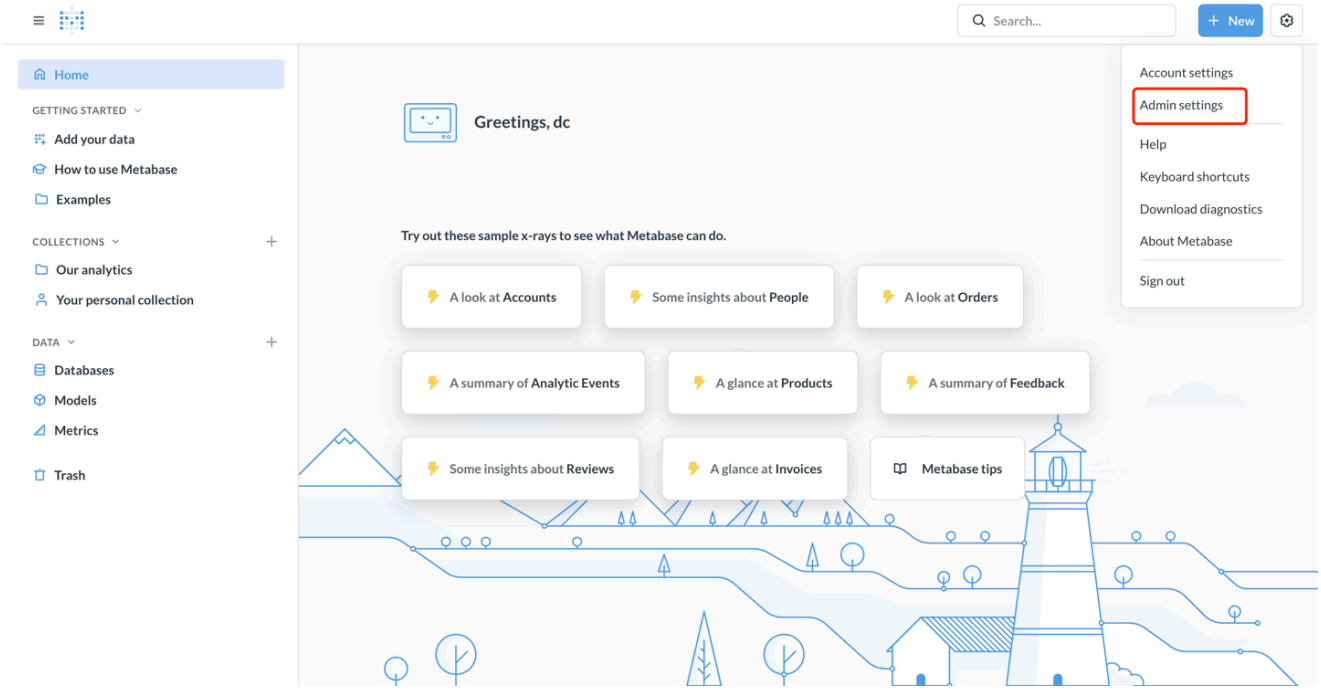
Database Name Format Explanation:

- **Internal Tables**: Enter the database name directly, such as `tpch`. The system will automatically use the `internal` catalog.
- **External Tables/Data Lake**: Enter the Catalog configuration. If only linking internal tables, this item is not required.

Configuration Steps

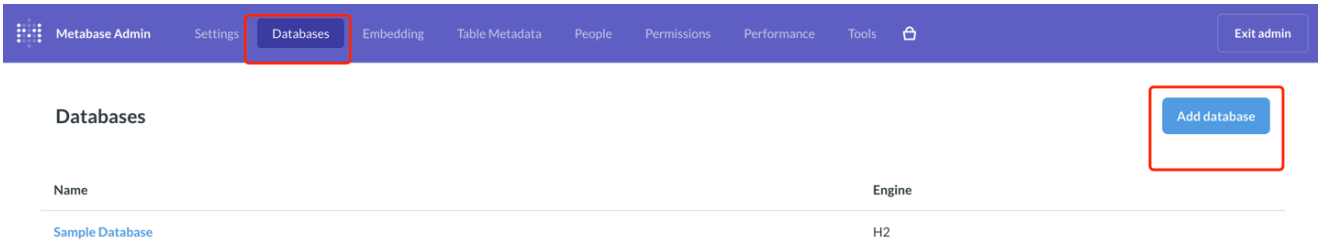
1. Start Metabase and log in.

2. Click the gear icon in the upper right corner and select **Admin Settings**.



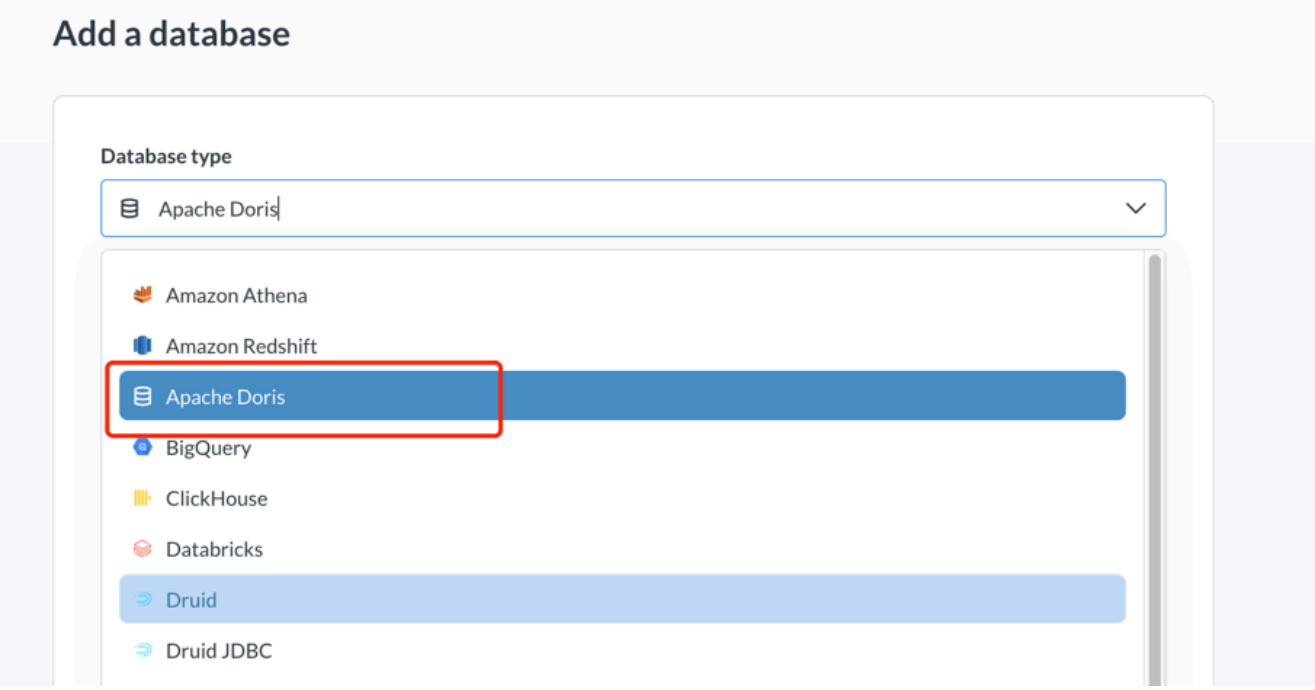
Metabase Admin Settings

3. In the left-hand menu, select **Databases**, and click the **Add database** button in the upper right corner.



Add database

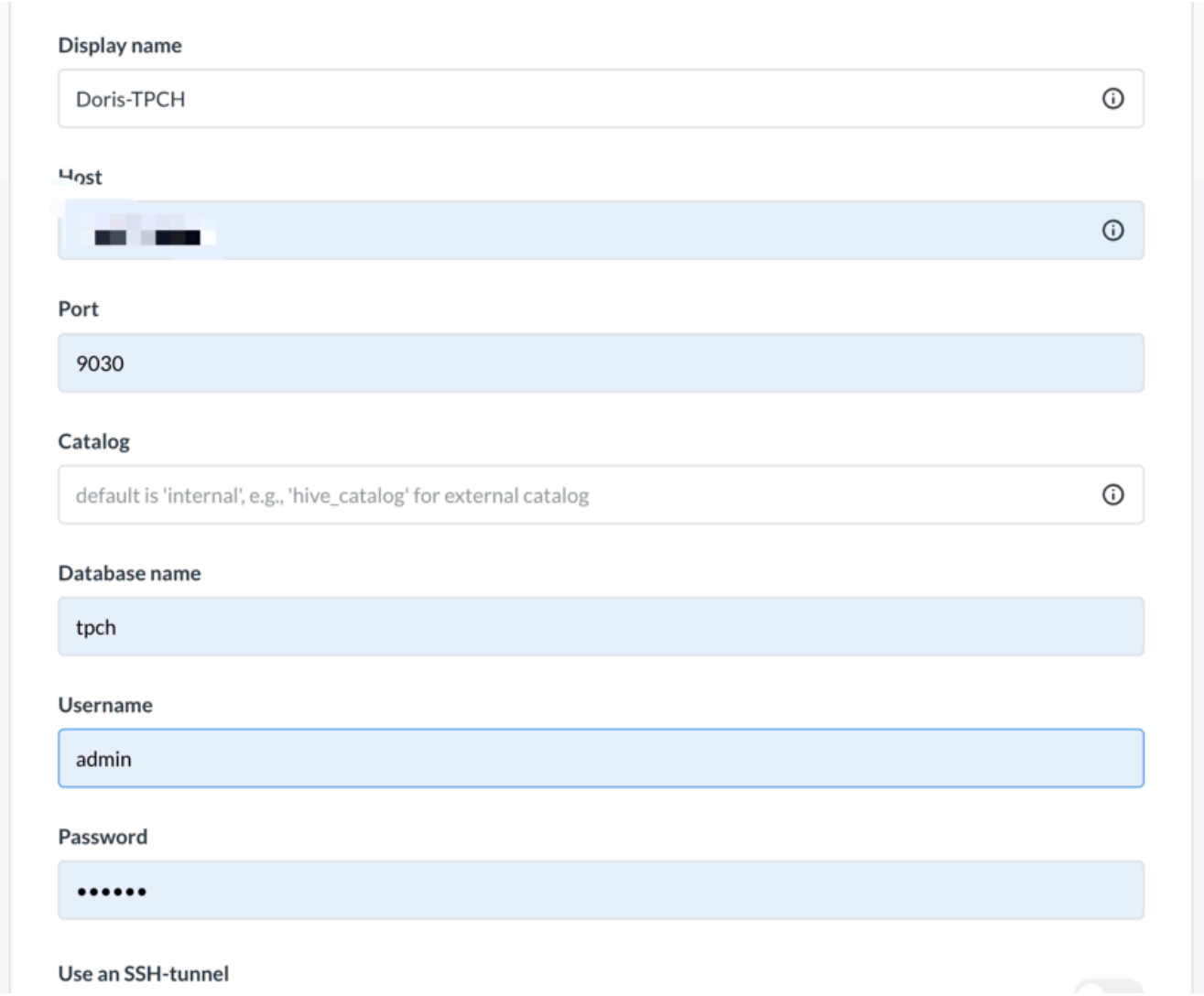
4. In the **Database type** dropdown menu, select **Apache Doris**.



Select Apache Doris

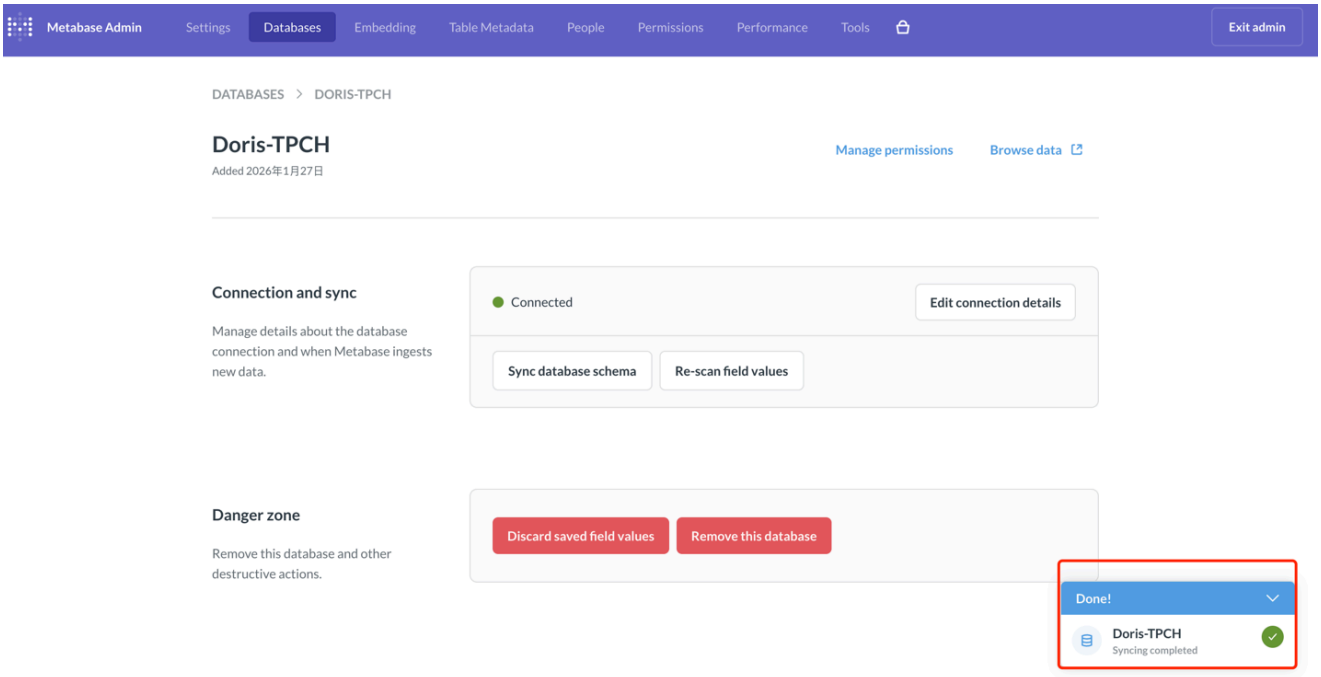
5. Fill in the connection information:

- **Display name**: Doris-TPCH
- **Host**: 127.0.0.1
- **Port**: 9030
- **Database name**: tpch
- **Username**: admin
- **Password**: ******



Fill in connection information

6. Click **Save** to save the configuration.
7. Metabase will automatically test the connection and synchronize database metadata. If the connection is successful, a success message will be displayed.



Connection successful

At this point, the data source configuration is complete! Next, we can build visualizations in Metabase.

Building Visualizations in Metabase

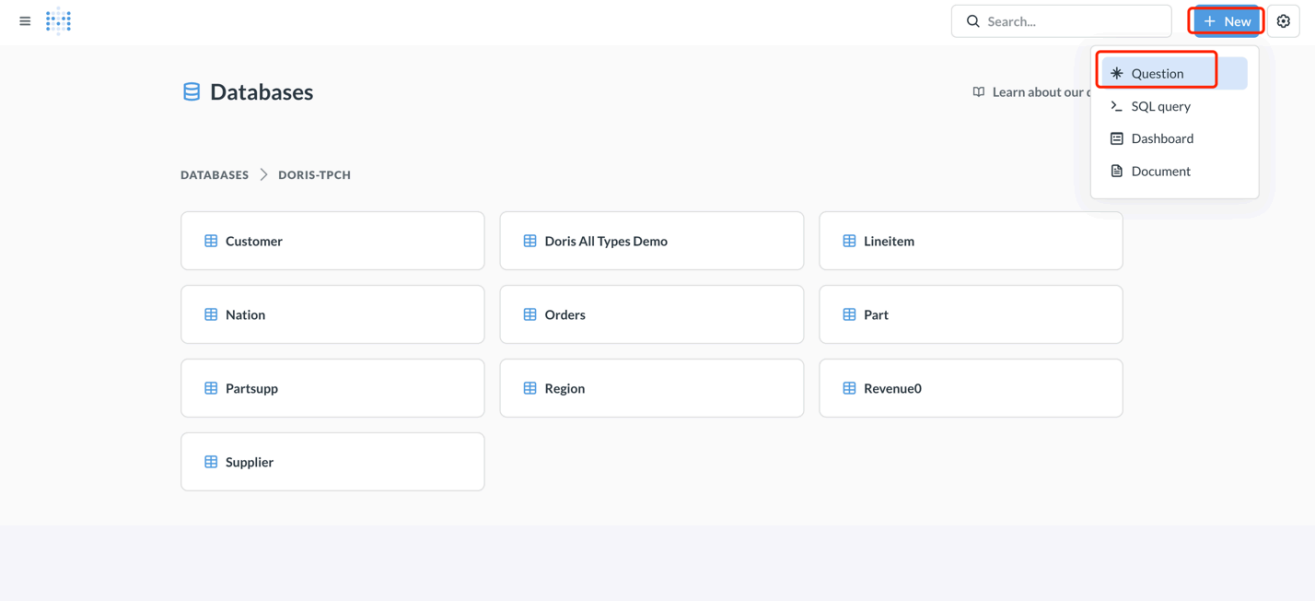
We choose TPC-H data as the data source. Refer to this document for instructions on building the Doris TPC-H data source.

Now that we have configured the Doris data source in Metabase, let's visualize the data…

Suppose we need to analyze the order amount growth curve over time for different freight methods for cost analysis.

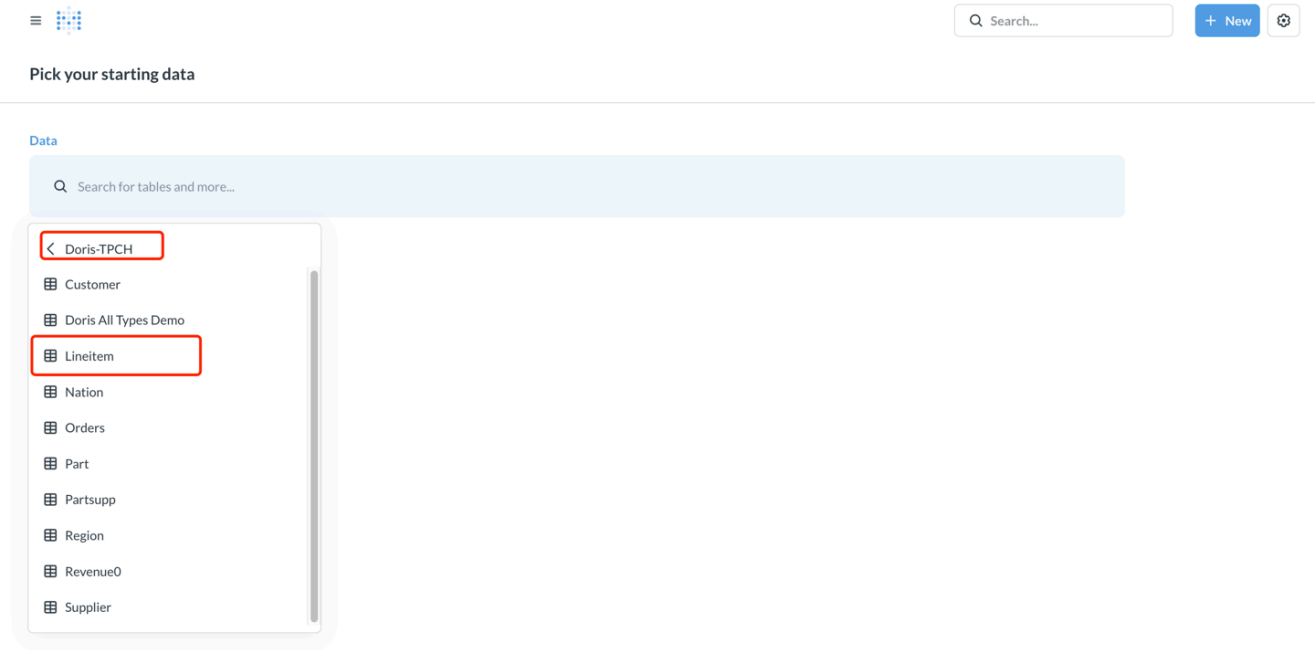
Creating a Question

1. Click the **New +** button in the upper right corner of the homepage and select **Question**.



Create a new question

2. Select the data source:
    - **Database**: Doris TPCH
    - **Table**: lineitem

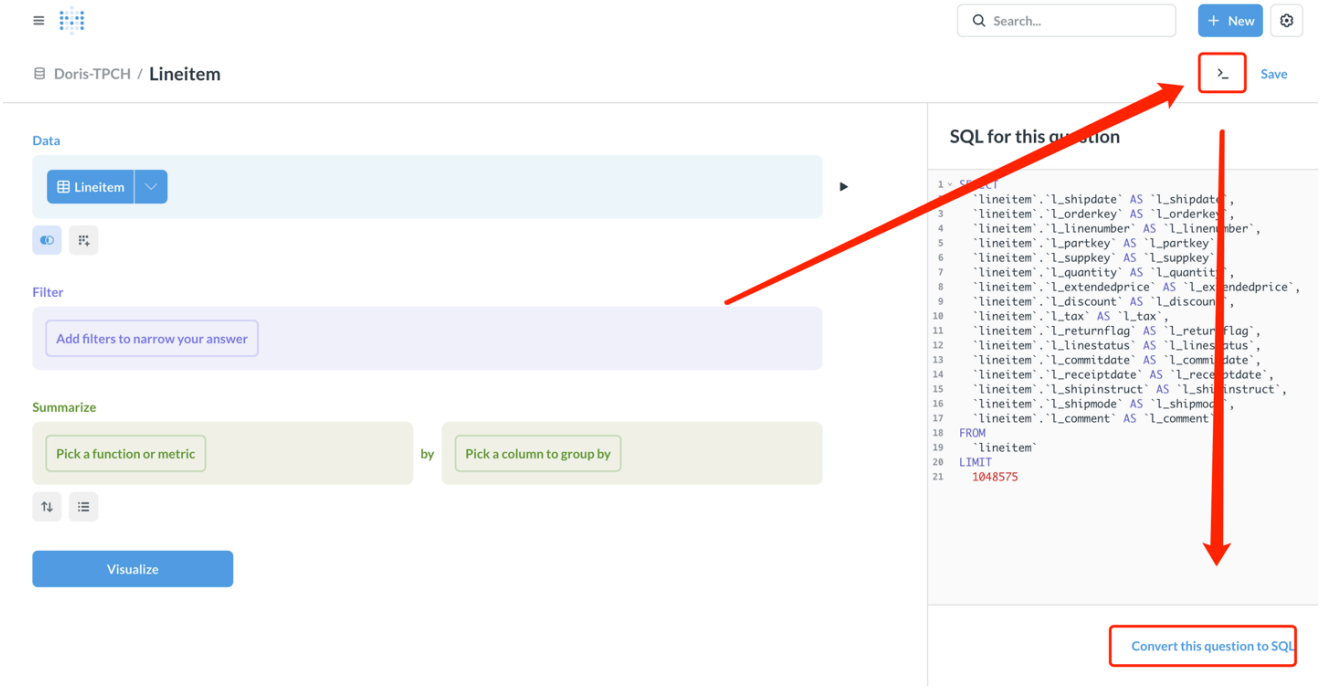


Select the table

Building Custom Metrics Using SQL

To calculate revenue, we need to use a custom SQL expression:

1. Click the **view sql** switch in the upper right corner, then click **convert this question to SQL** to edit the SQL.

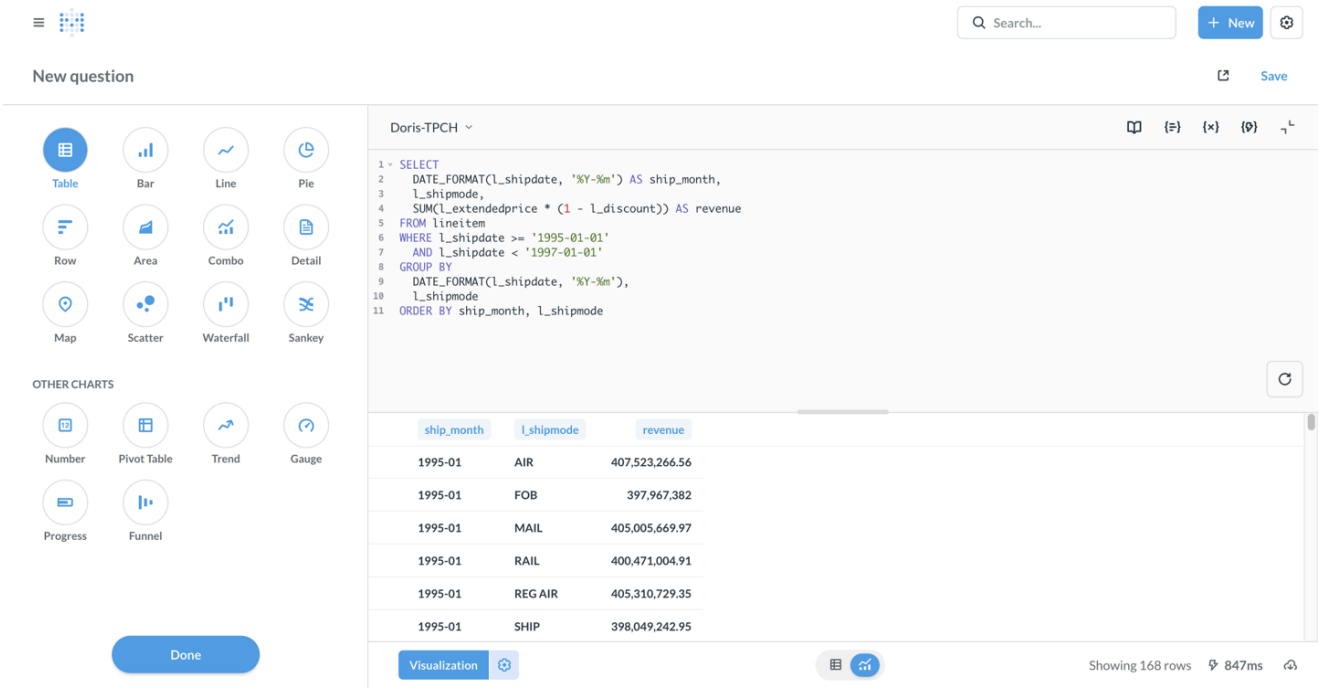


Switch to SQL mode

2. Enter the following SQL query:

```
SELECT
  DATE_FORMAT(l_shipdate, '%Y-%m') AS ship_month,
  l_shipmode,
  SUM(l_extendedprice * (1 - l_discount)) AS revenue
FROM lineitem
WHERE l_shipdate >= '1995-01-01'
  AND l_shipdate < '1997-01-01'
GROUP BY
  DATE_FORMAT(l_shipdate, '%Y-%m'),
  l_shipmode
ORDER BY ship_month, l_shipmode
```

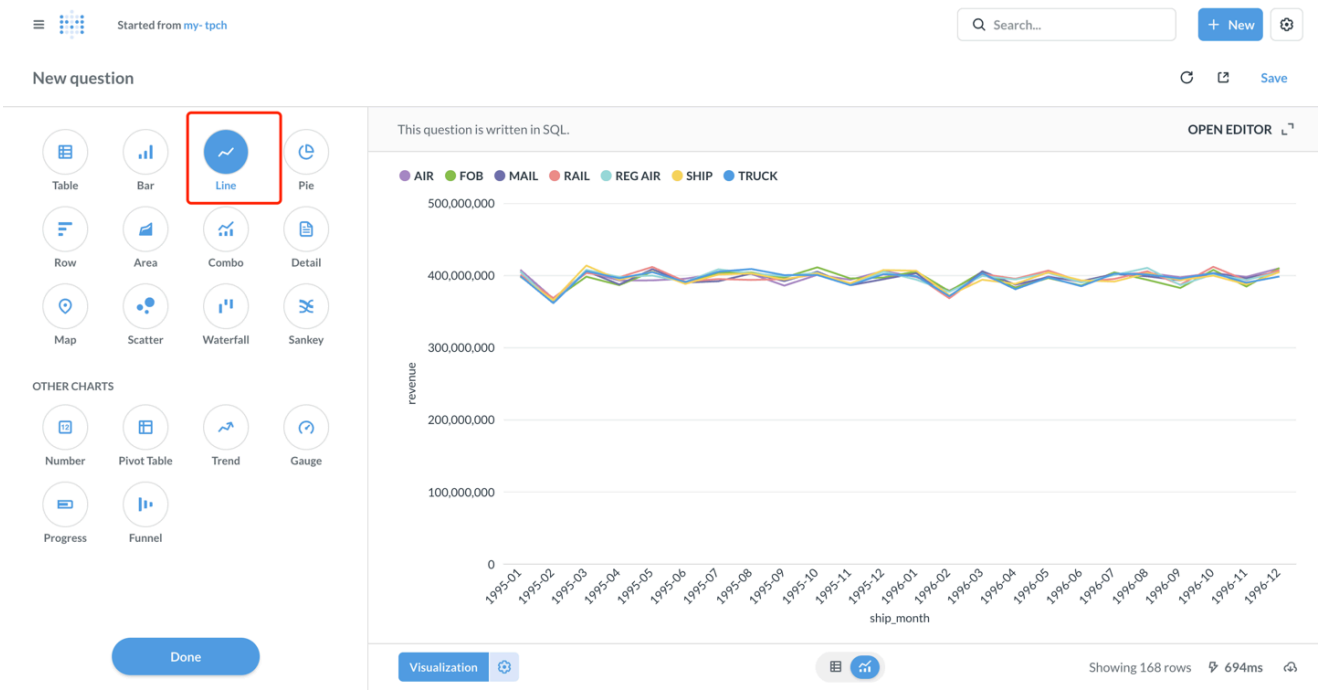
3. Click the **Visualize** button in the lower right corner to view the results.



View Results

Configure Visualization Charts

1. The default display is a table. Click the **Visualization** button in the lower left corner and select the **Line** chart type.

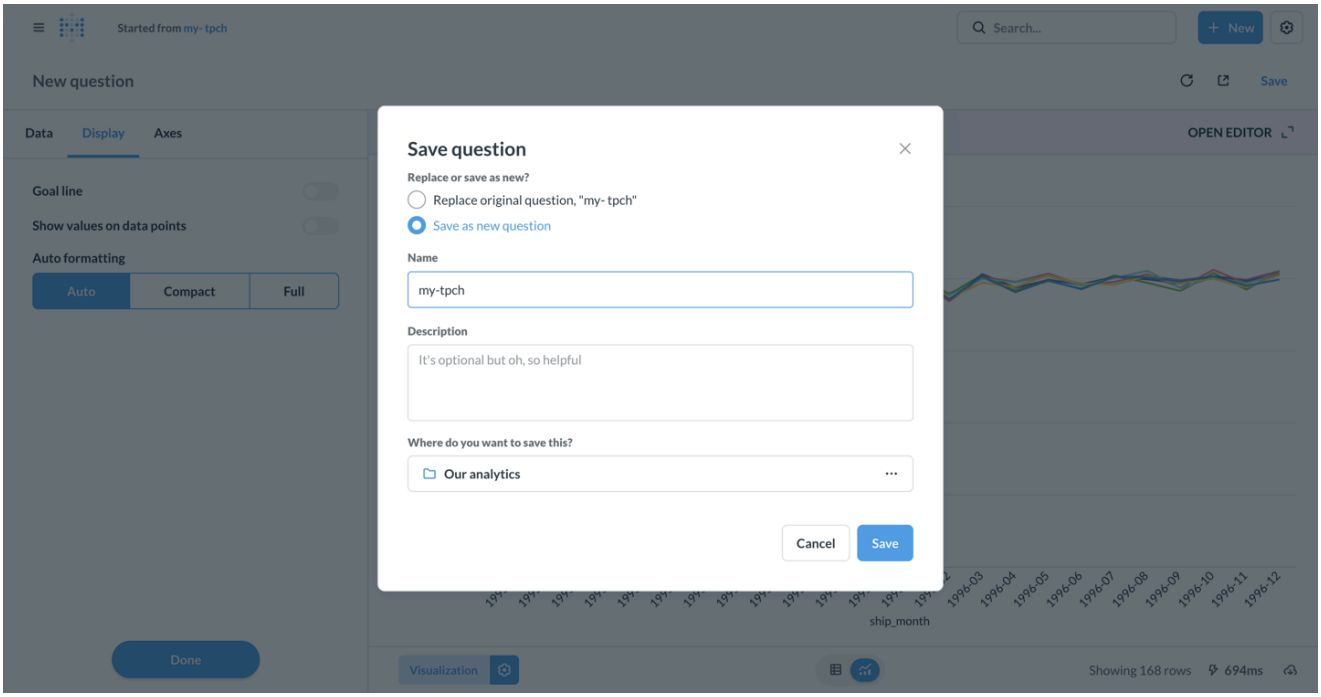


Select Line Chart

2. Configure chart parameters as needed (metabase automatically configures as follows):

   - **X-axis**: ship_month (shipping month)
   - **Y-axis**: revenue (revenue)
   - **Series**: l_shipmode (shipping mode)

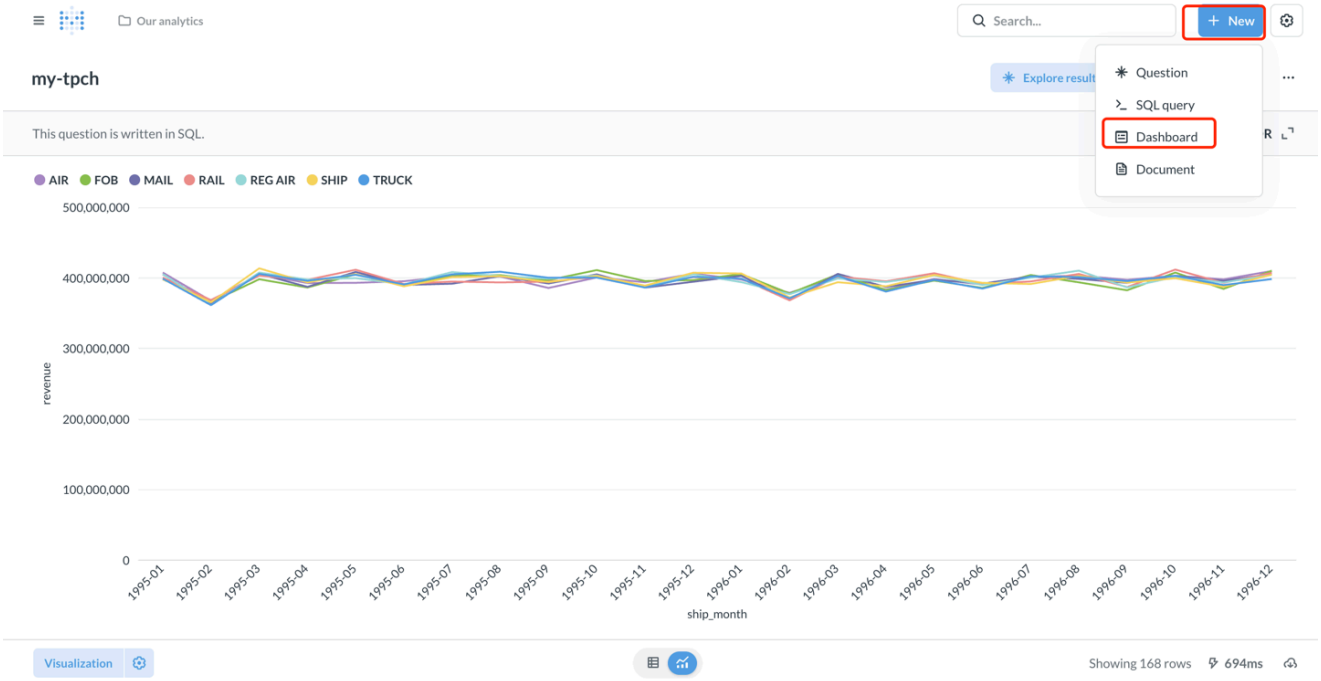
3. Customize chart style:
    - Click the **Settings** icon to adjust colors, labels, legend position, etc.
    - In the **Display** tab, you can set axis titles, value formats, etc.
4. After configuring the chart, click **Save** in the upper right corner.
5. Enter the issue name: **my-tpch**, and select the collection to save to.



Naming the issue

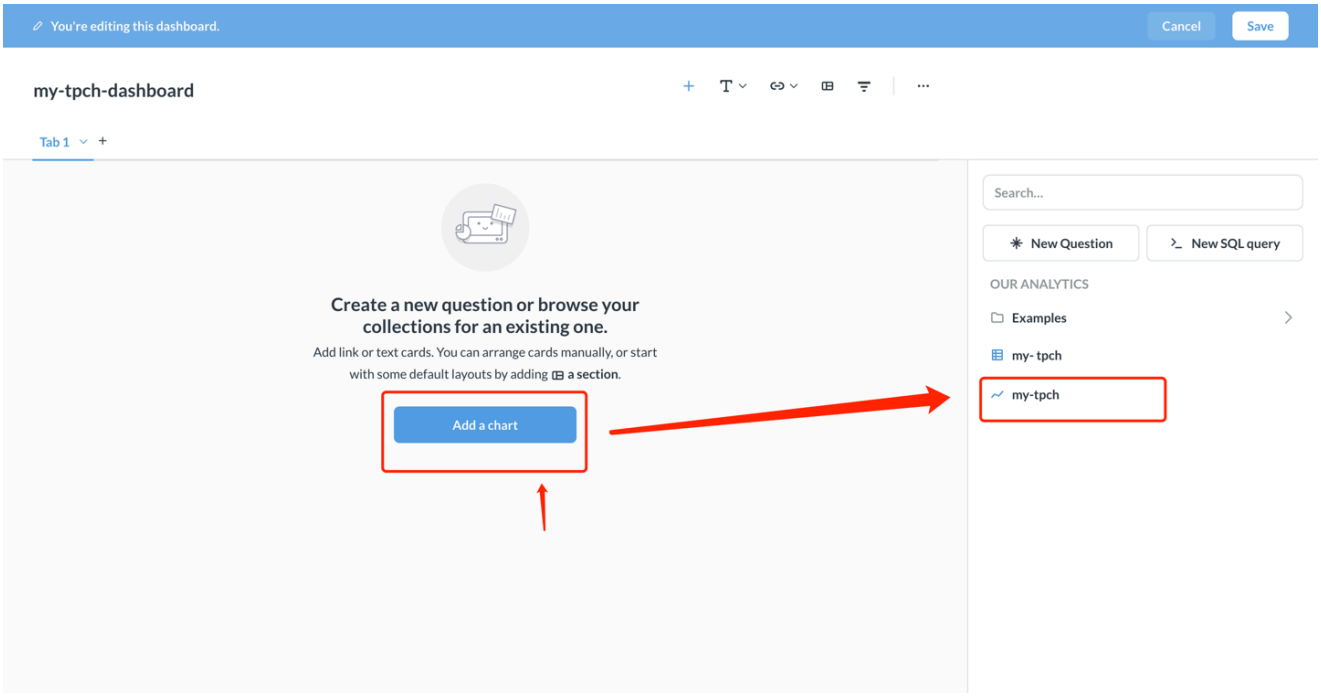
Creating a Dashboard

1. Click **+ New** → **Dashboard** to create a new dashboard. Enter the dashboard name: **my-tpch**



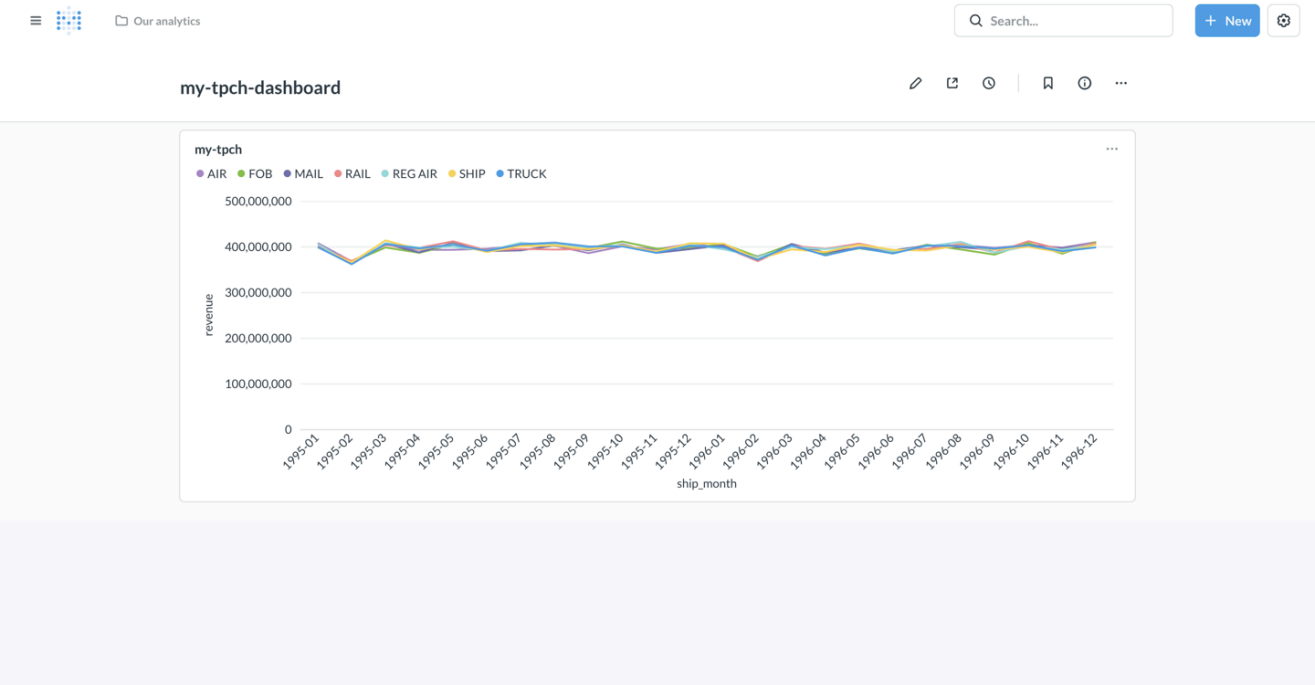
Creating a Dashboard

2. Click **Add a chart** to add the saved question to the dashboard.



Adding a Question

3. Adjust the chart position and size, and click **Save** in the upper right corner to save the dashboard.



Saving the Dashboard

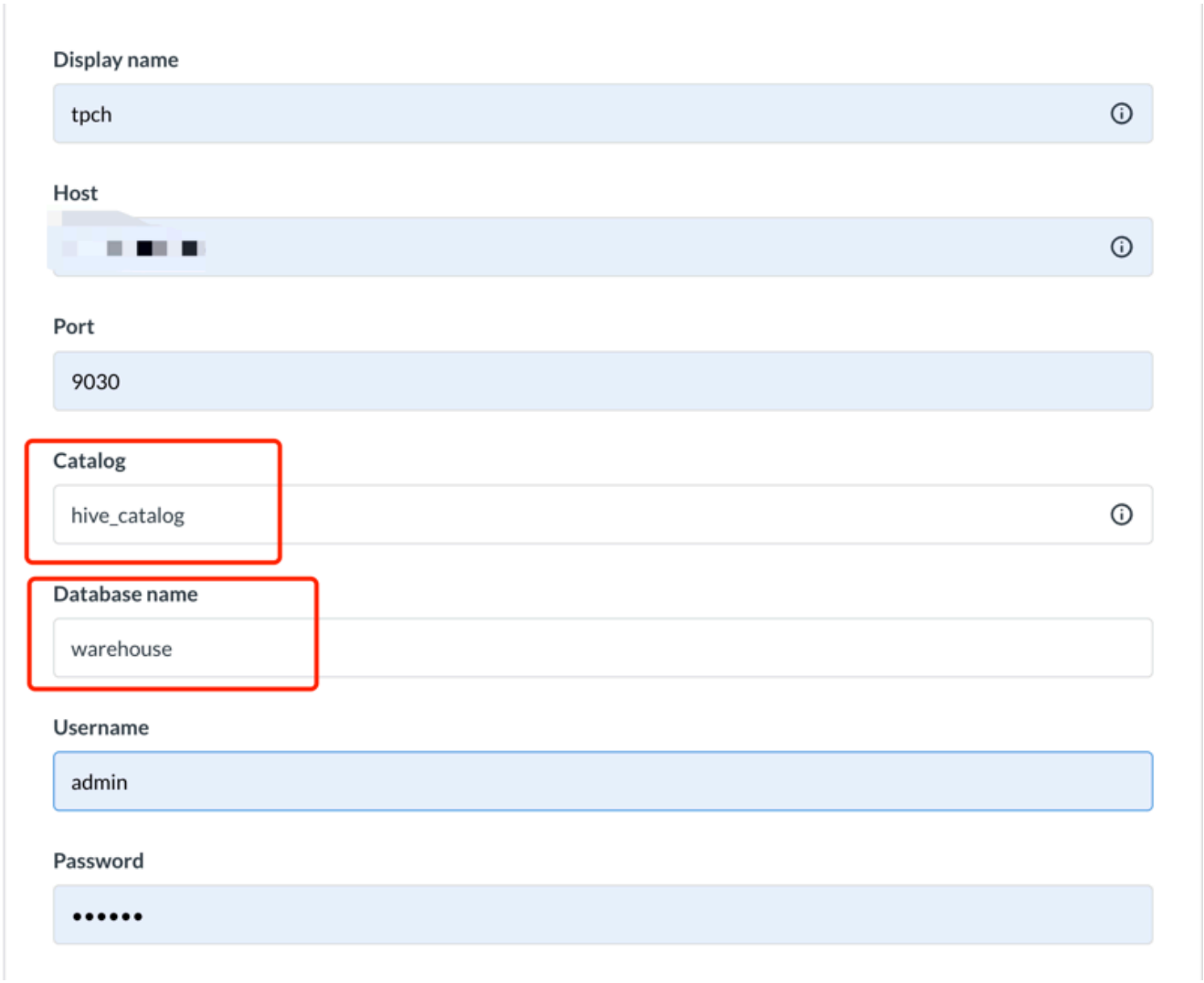
At this point, Metabase has been successfully connected to Apache Doris, and data analysis and visualization dashboard creation have been implemented!

Advanced Features

Accessing External Data Using Catalogs

Doris supports multi-catalog functionality, allowing queries to external data sources and cross-data source queries. When using it in Metabase:

1. Configure `Catalog` in the Links configuration interface, and configure the external database under that catalog in `Database`, for example:
   `catalog: hive_catalog`, `database: warehouse` - Access the warehouse database named hive_catalog



Configuring catalog

2. Or explicitly specify the Catalog in an SQL query:

```
SELECT * FROM hive.warehouse.orders LIMIT 100;
```

Using Parameterized Queries

Metabase supports using variables in SQL queries, making it easy to create interactive dashboards:

```
SELECT
  l_shipmode,
  SUM(l_extendedprice * (1 - l_discount)) AS revenue
FROM lineitem
WHERE l_shipdate BETWEEN {{start_date}} AND {{end_date}}
  AND l_shipmode = {{ship_mode}}
GROUP BY l_shipmode
```

After saving, data can be dynamically filtered in the dashboard using dropdown menus or date pickers.

Performance Optimization Recommendations

1. **Use Partition Clipping**: Add a partition column filter condition to the WHERE clause.

```
WHERE date >= '2024-01-01' AND date < '2024-02-01'
```

2. **Utilize Materialized Views:** For complex aggregation queries, creating materialized views in Doris can accelerate the query process.
3. **Control Result Set Size:** Use LIMIT to limit the number of rows returned, avoiding loading too much data at once.
4. **Query Caching:** Metabase automatically caches query results; setting appropriate cache times can improve performance.

Connection and Usage Tips

- **Driver Installation:** Ensure `doris.metabase-driver.jar` is placed in the Metabase `plugins` directory and restart Metabase.
- **Time Zone Settings:** If you encounter time zone issues, add `serverTimezone=Asia/Shanghai` to the JDBC connection string.
- **Partitioned Table Optimization:** Creating appropriate Doris partitioned tables, dividing them by time and bucketing them, can effectively reduce the amount of data scanned in queries.
- **Network Connection:** It is recommended to use a VPC private connection to avoid security risks introduced by public network access.
- **Access Control:** Fine-tune Doris user account roles and access permissions, following the principle of least privilege.
- **Metadata Synchronization:** When Doris… When the table structure changes, manually synchronize it by clicking “Sync database schema now” on the Metabase management page.
- **Performance Monitoring**: For slow queries, you can use `SHOW QUERY PROFILE` in Doris to analyze performance bottlenecks.

Data Type Display Anomalies

- Ensure you are using the latest version of the Doris Driver.
- Doris largeint types need to be explicitly converted in SQL.

```
SELECT CAST(large_int_col AS STRING) FROM table
```

Power BI

Microsoft Power BI can query from Apache Doris or load data into memory.

You can use Power BI Desktop, the Windows desktop application for creating dashboards and visualizations.

This tutorial will guide you through the following process:

- Install the MySQL ODBC driver
- Install the Doris Power BI connector into Power BI Desktop
- Query data from Doris to visualize it in Power BI Desktop

Prerequisites

Power BI installation

This tutorial assumes that you have installed Microsoft Power BI Desktop on a Windows computer. You can download and install Power BI Desktop here.

We recommend updating to the latest version of Power BI.

Connection information

Collect your Apache Doris connection details

You will need the following details to connect to your Apache Doris instance:

Parameter	Description	Example
Doris Data Source	Database connection string, host + port	127.0.1.28:9030
Database	Database name	test_db
Data Connectivity Mode	Data connectivity mode, includes Import and DirectQuery	DirectQuery

Parameter	Description	Example
SQL Statement	SQL statement that must include the Database, only for Import mode	select * from database.table
User Name	User name	admin
Password	Password	xxxxxx

Power BI Desktop

To start querying data in Power BI Desktop, complete the following steps:

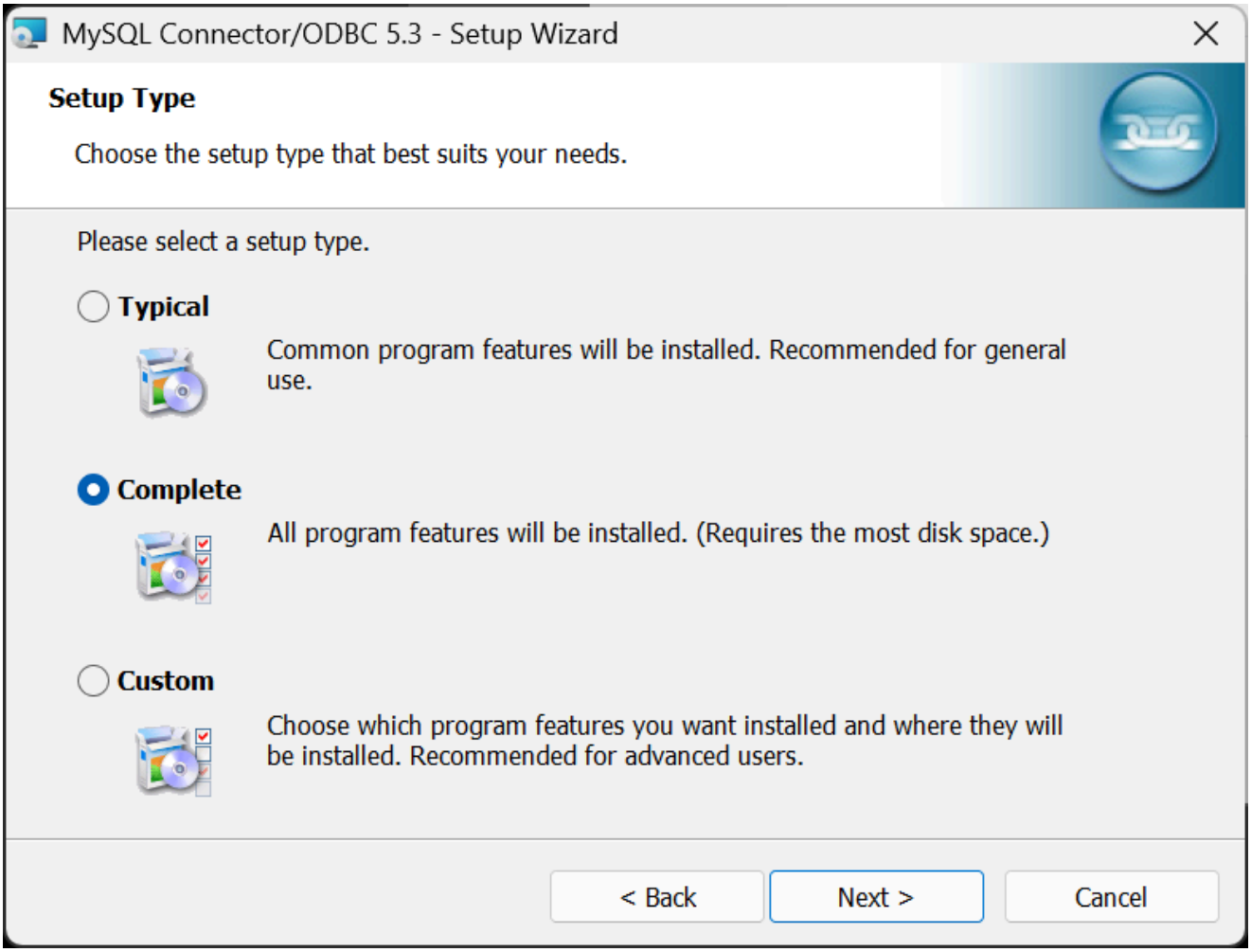
1. Install the MySQL ODBC driver
2. Find the Doris connector
3. Connect to Doris
4. Query and visualize data

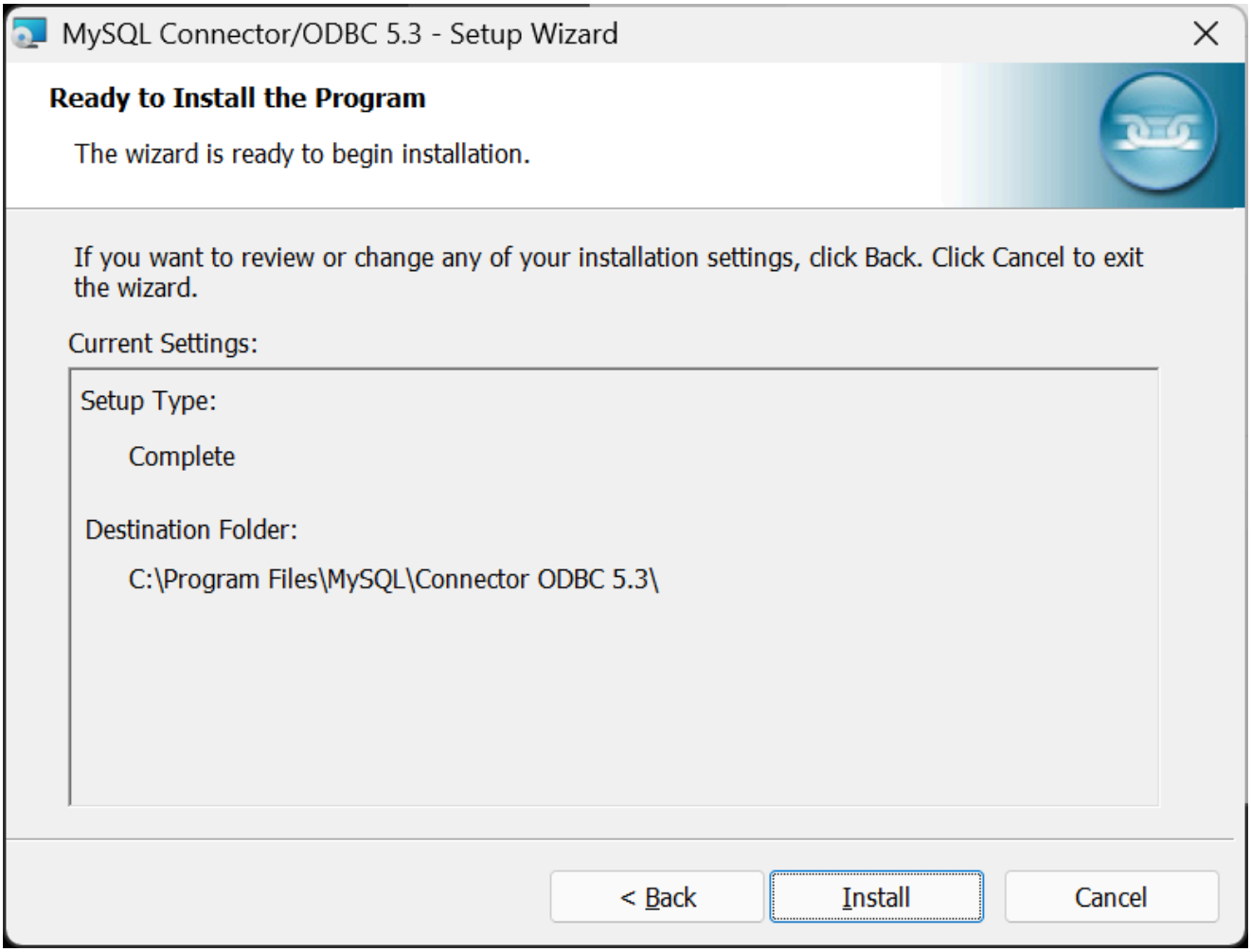
Install the ODBC driver

Download and install MySQL ODBC, and configure it (version 5.3).

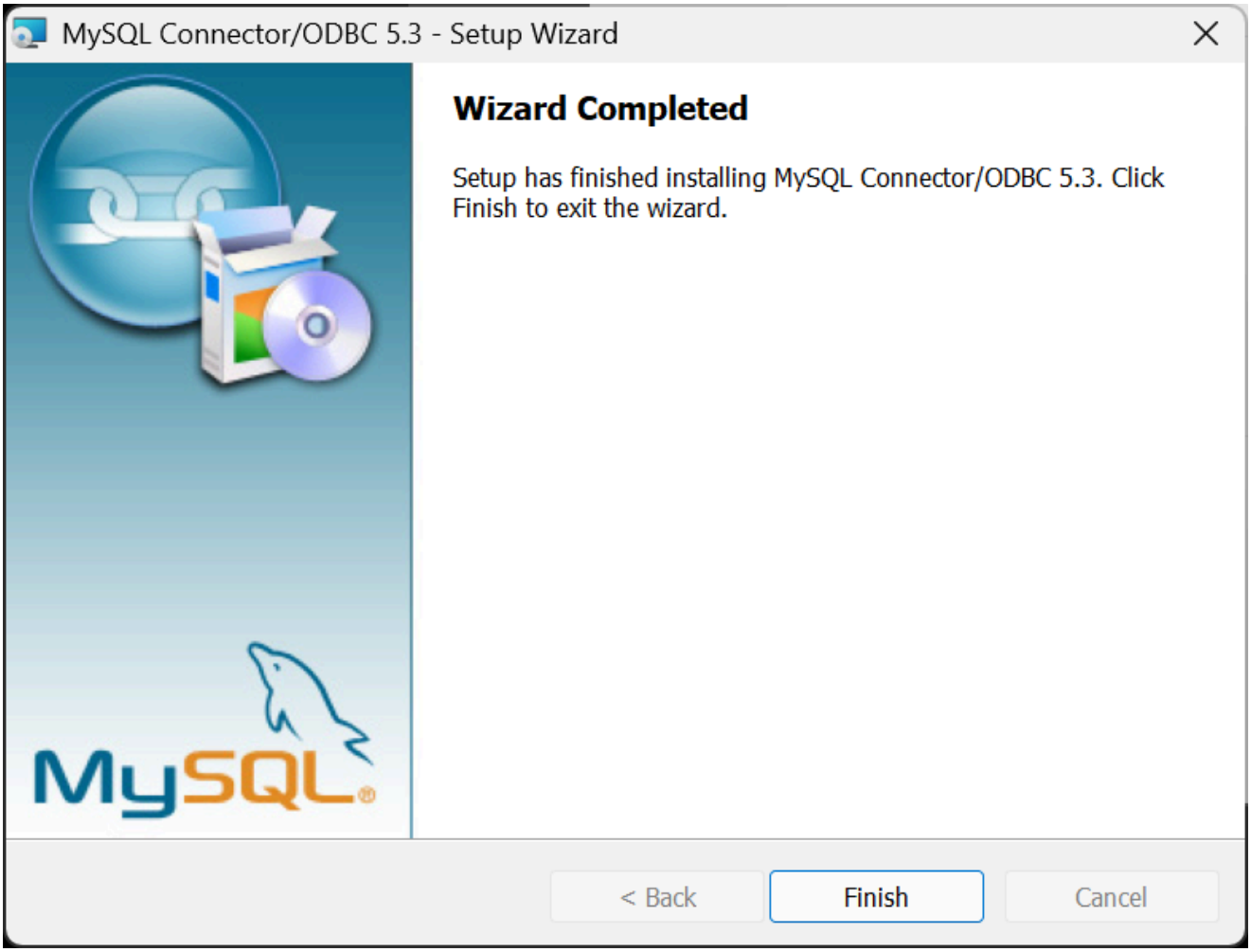
Run the provided `.msi` installer and follow the wizard.


MySQL Connector/ODBC 5.3 - Setup Wizard
Welcome to the Setup Wizard for MySQL Connector/ODBC 5.3
The Setup Wizard will install MySQL Connector/ODBC 5.3 release 5.3.13 on your computer. To continue, click Next.
MySQL
< Back
Next >
Cancel


MySQL Connector/ODBC 5.3 - Setup Wizard
Setup Type
Choose the setup type that best suits your needs.
Please select a setup type.
Typical
Common program features will be installed. Recommended for general use.
Complete
All program features will be installed. (Requires the most disk space.)
Custom
Choose which program features you want installed and where they will be installed. Recommended for advanced users.
< Back
Next >
Cancel



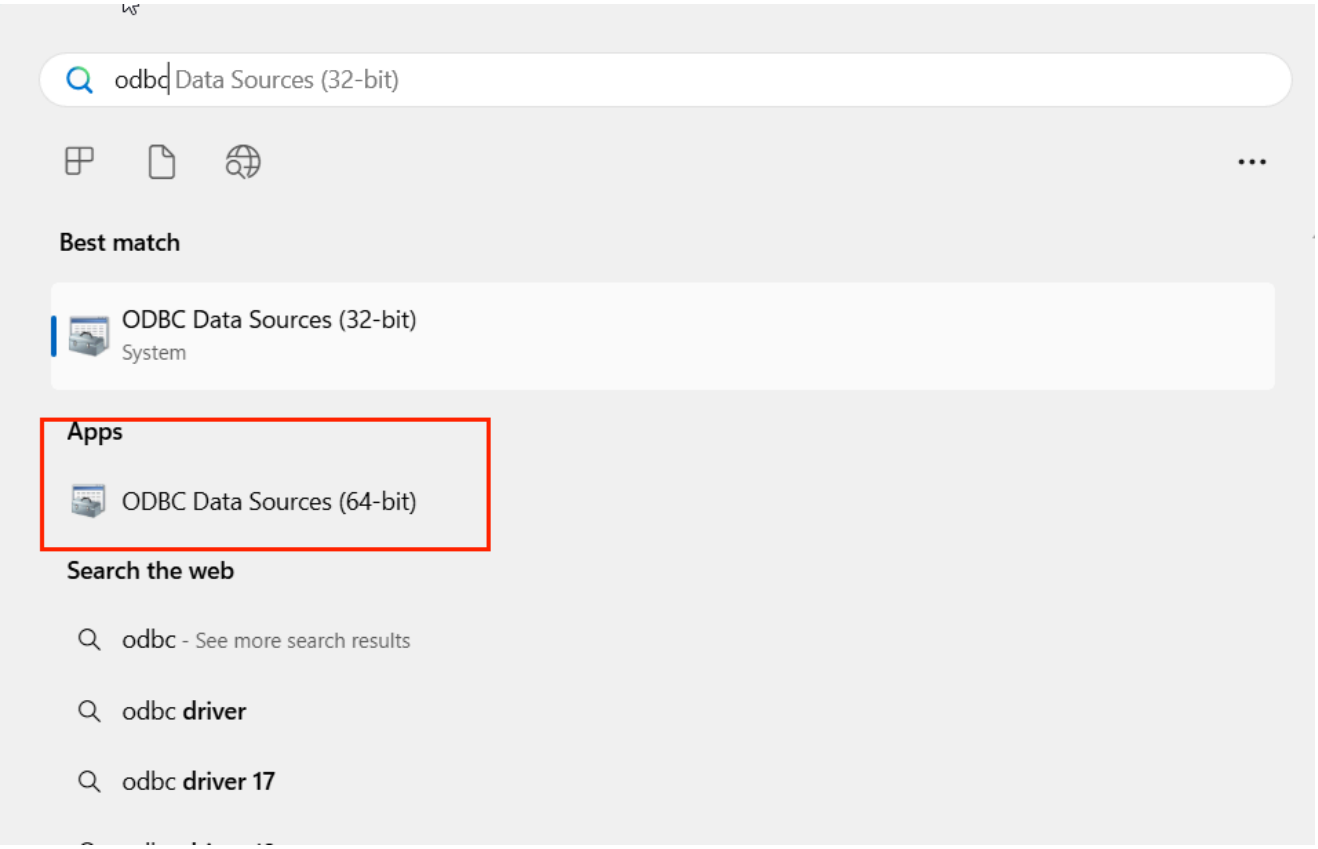
Installation completed



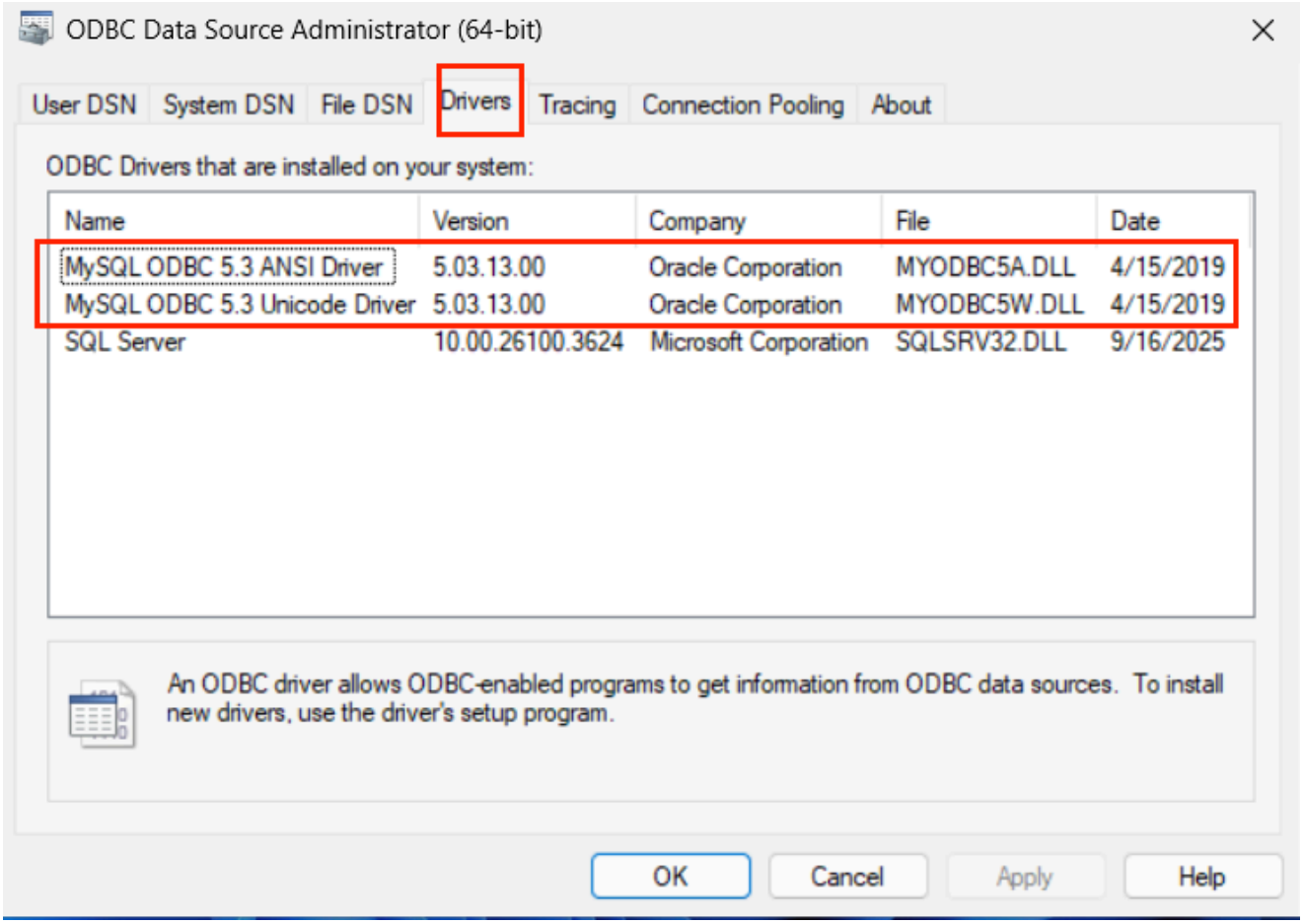
Verify the ODBC driver

After the driver installation is complete, you can verify that it was successful as follows:

In the Start menu, type ODBC and select “ODBC Data Sources **(64-bit)**”.



Verify that the MySQL driver is listed.

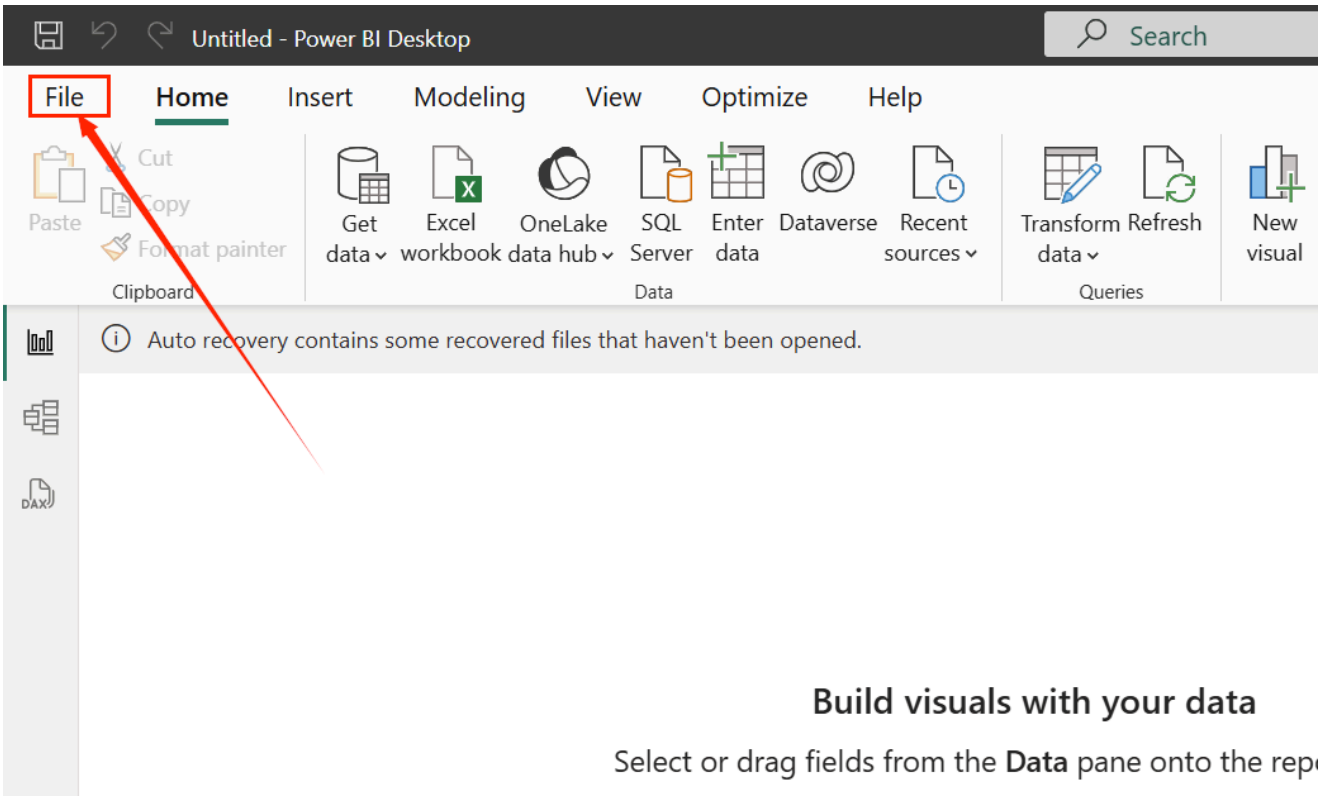


Install the Doris connector

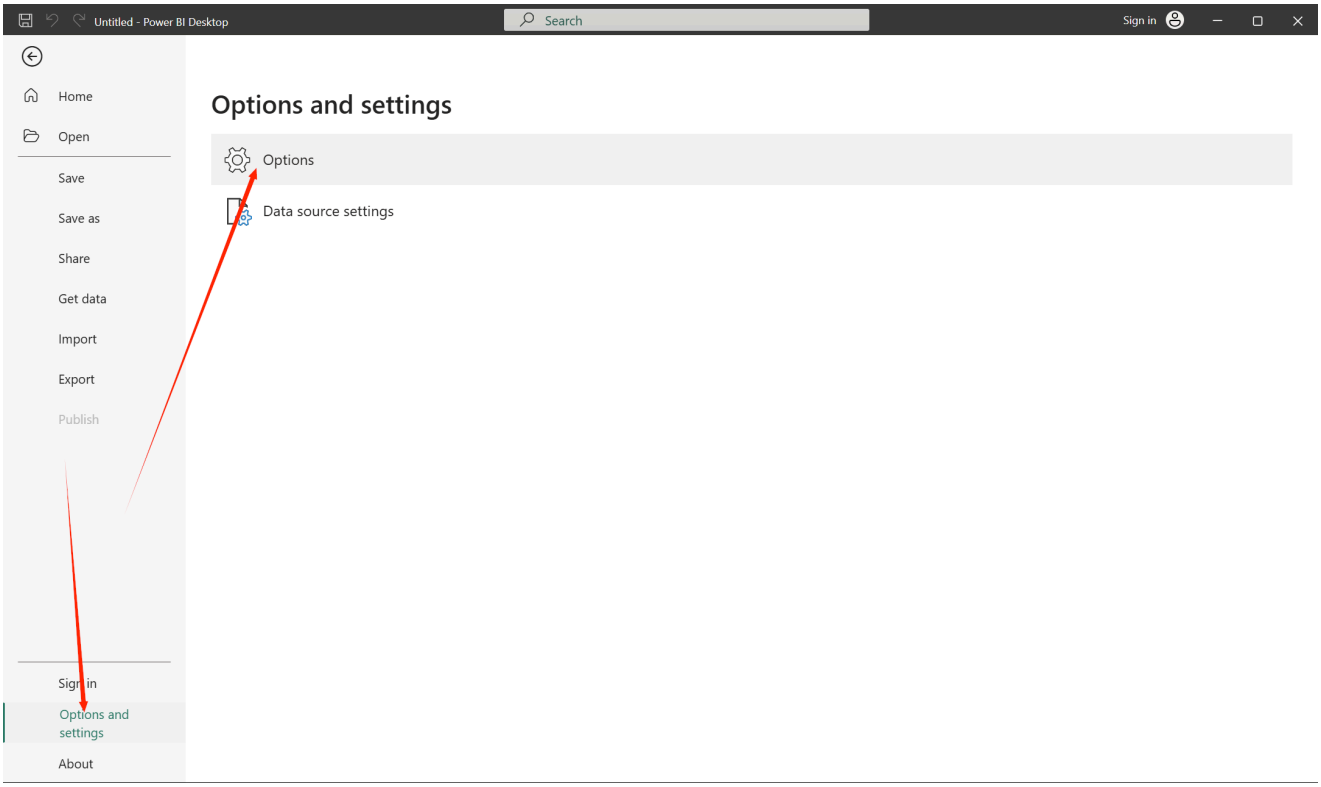
The certification channel for Power BI custom connectors is currently closed, so the Doris custom connector is uncertified. For uncertified connectors, configure it as follows (https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-connector-extensibility#custom-connectors):

1. Assuming `power_bi_path` is the directory of Power BI Desktop in the Windows operating system, the default is usually: `power_bi_path = C:\Program Files\Power BI Desktop`. Refer to this path `%power_bi_path%\Custom Connectors folder` and place the Doris.mez custom connector file (if the path does not exist, create it manually as needed).
2. In Power BI Desktop, choose `File` > `Options and settings` > `Options` > `Security`. Under `Data Extensions`, check `(Not Recommended) Allow any extension to load without validation or warning` to bypass the restriction on uncertified connectors.

First, choose `File`

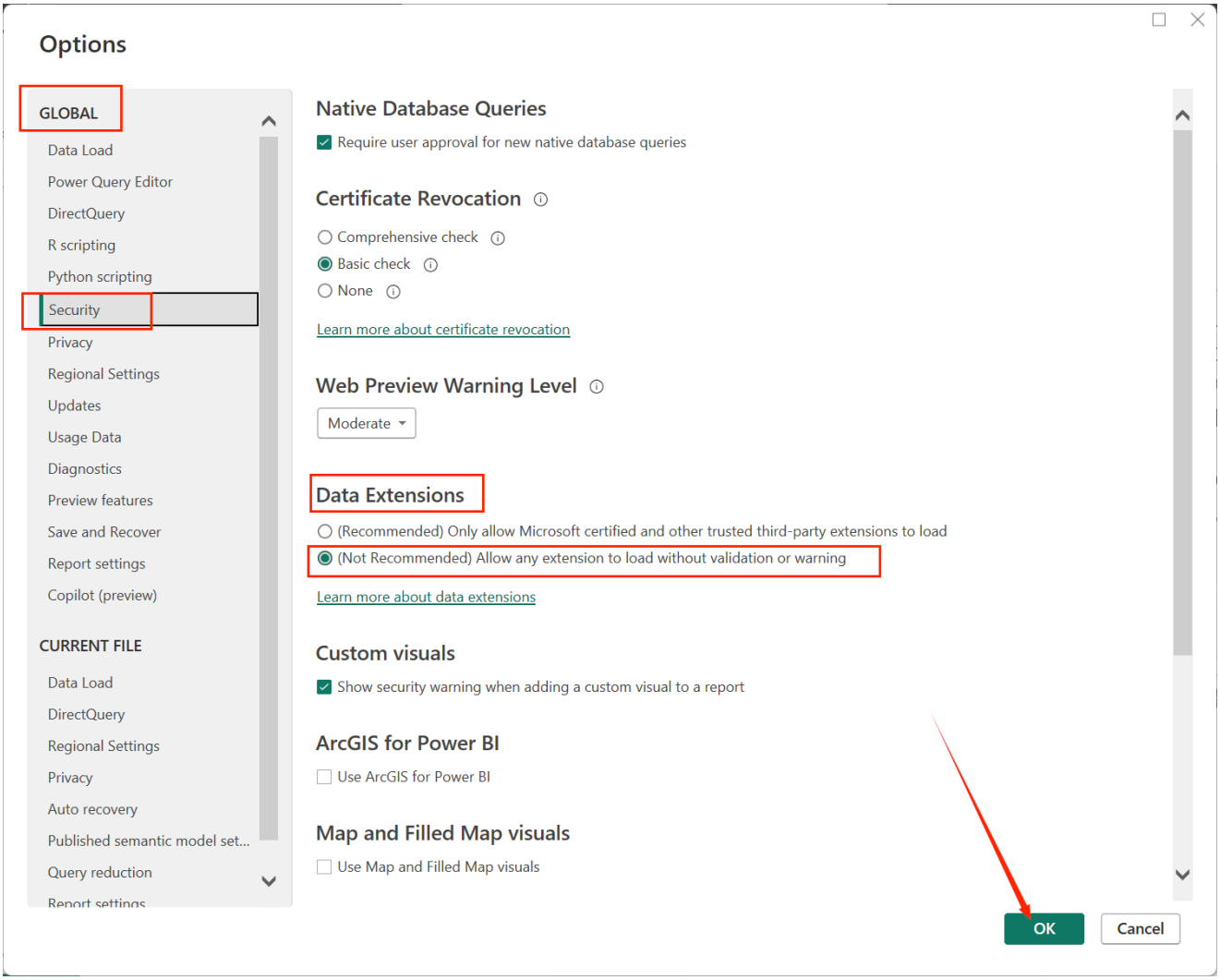


Then choose `Options and settings` > `Options`



In the `Options` dialog, go to `GLOBAL` > `Security`. Under `Data Extensions`,

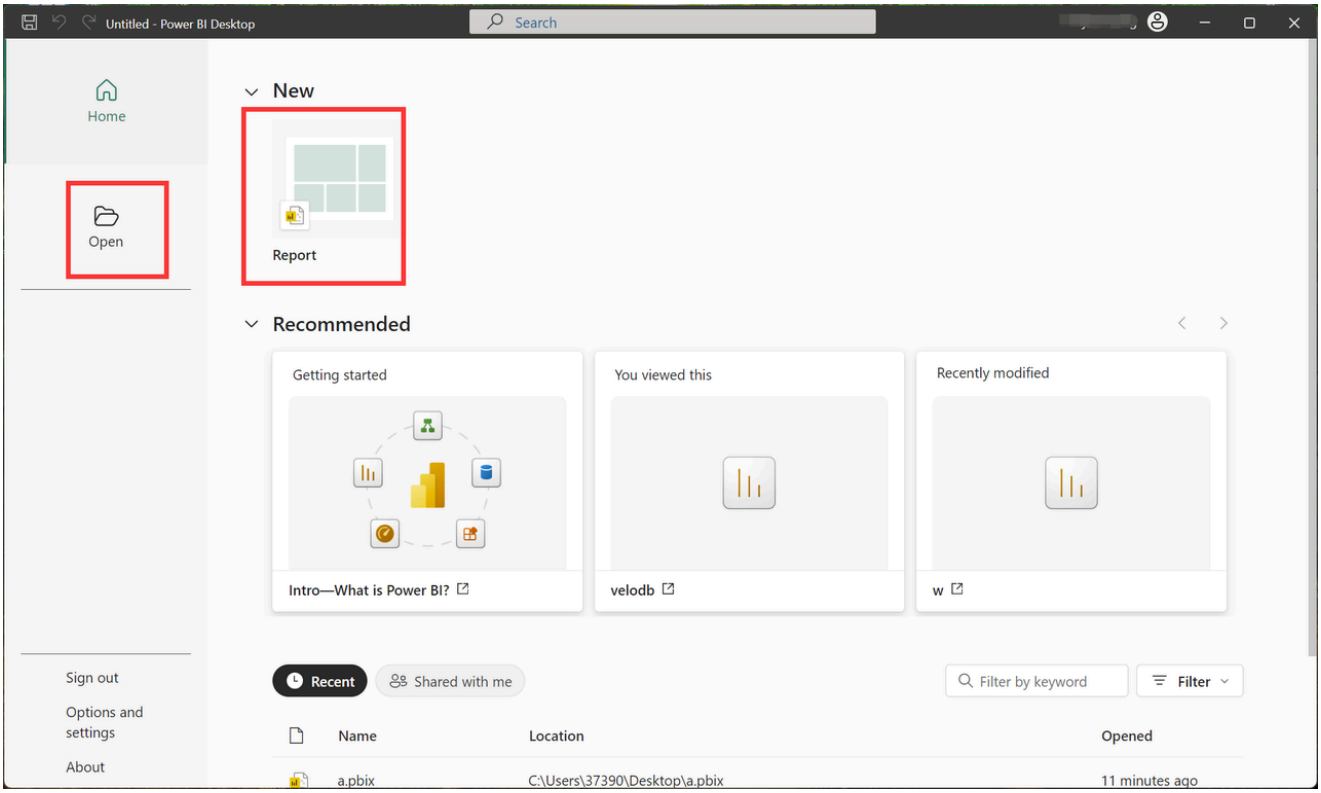
check `(Not Recommended) Allow any extension to load without validation or warning`.



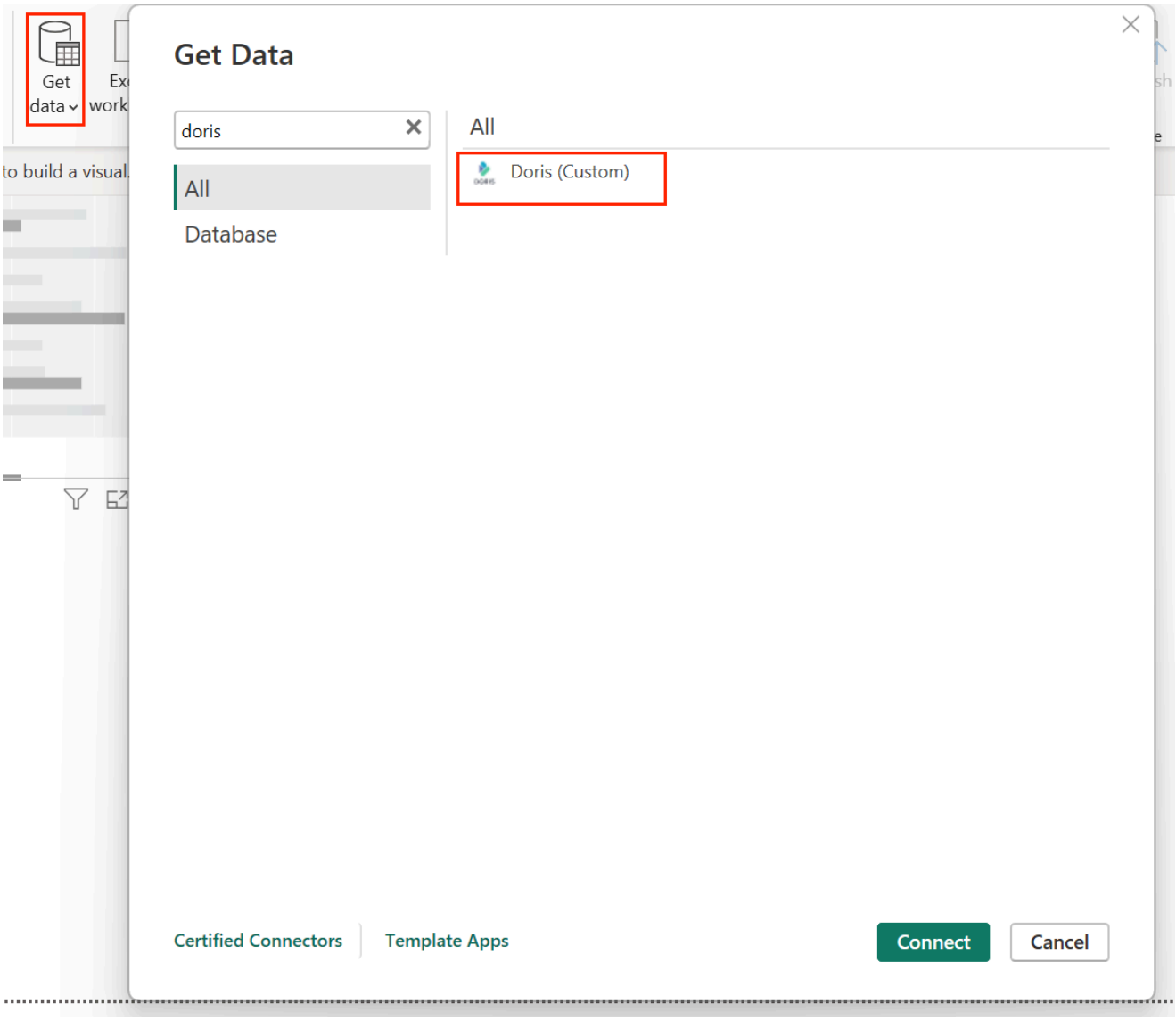
Click `OK`, then restart Power BI Desktop.

Find the Doris connector

1. Launch Power BI Desktop
2. On the Power BI Desktop start screen, click “New report”. If you already have a local report, you can open the existing report.



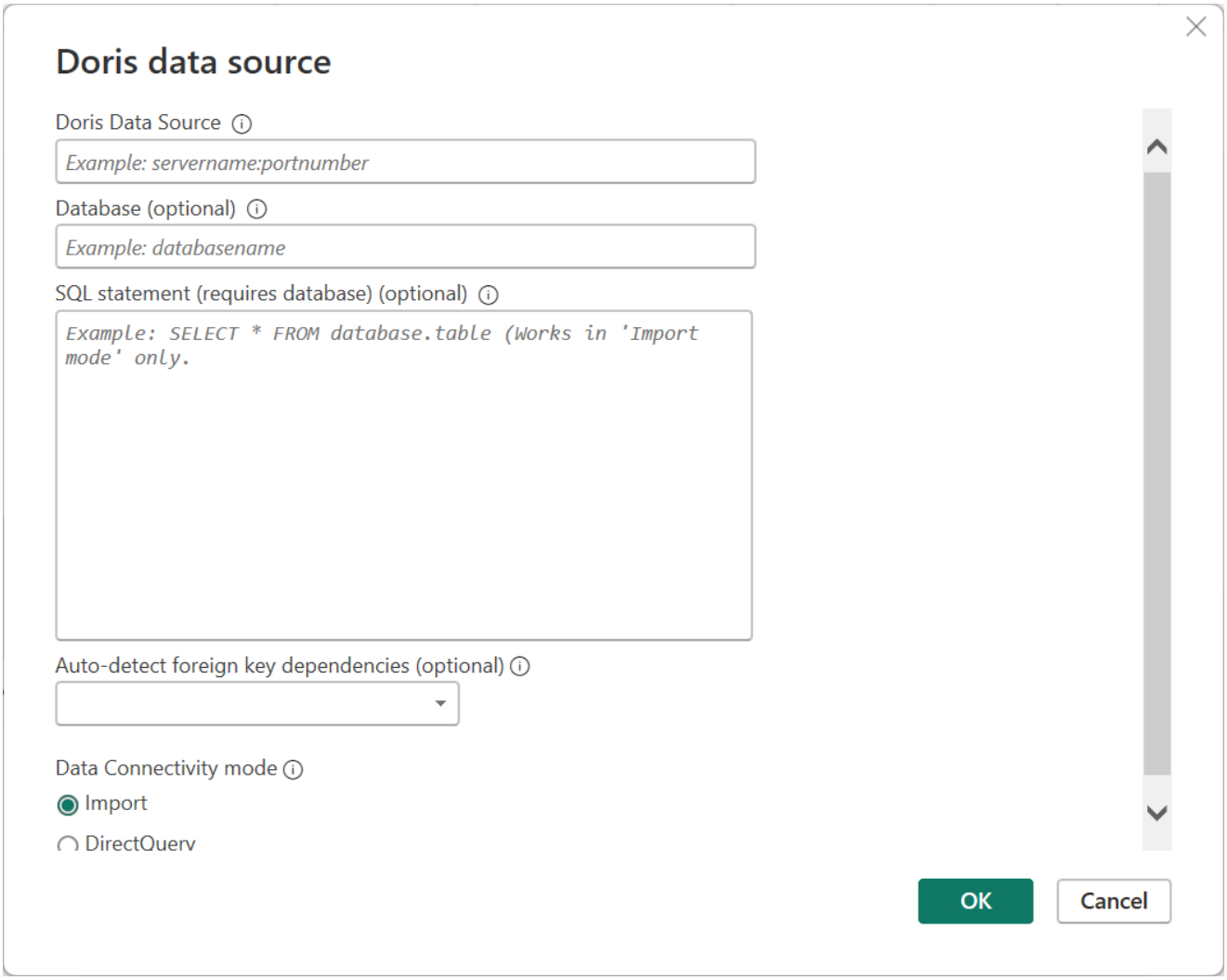
3. Click “Get Data” and select the Doris database in the pop-up window.



Connect to Doris

Select the connector and enter your Doris instance credentials:

- Doris Data Source (required) - Your instance domain/address or host:port.
- Database (required) - Your database name.
- SQL statement - A pre-executed SQL statement (only available in 'Import' mode)
- Data connectivity mode - DirectQuery/Import

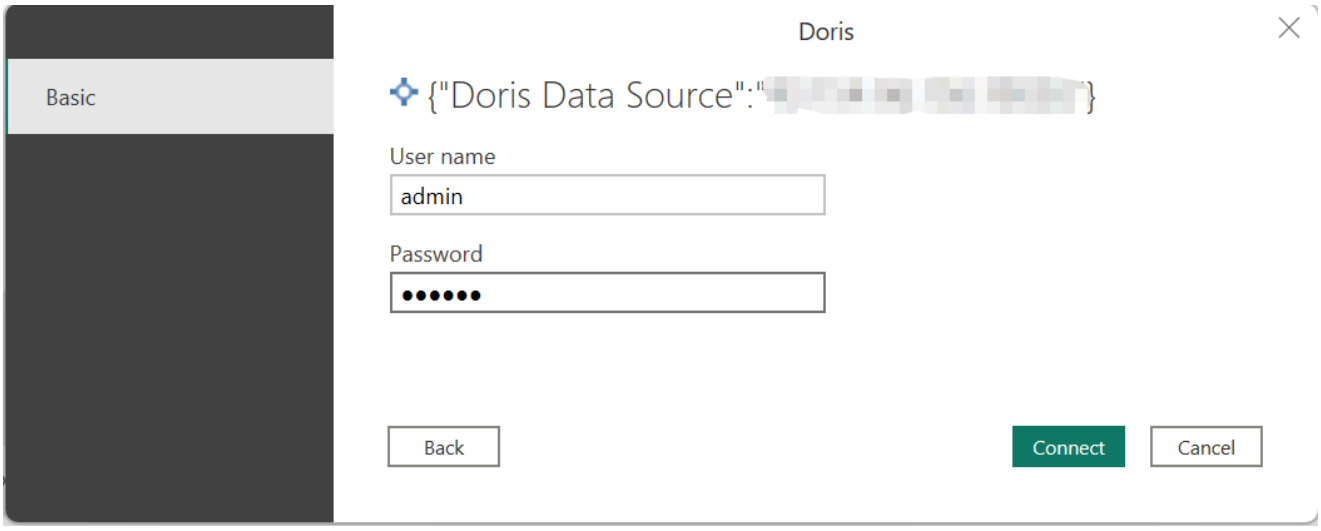


Note

We recommend choosing DirectQuery to query Doris directly.

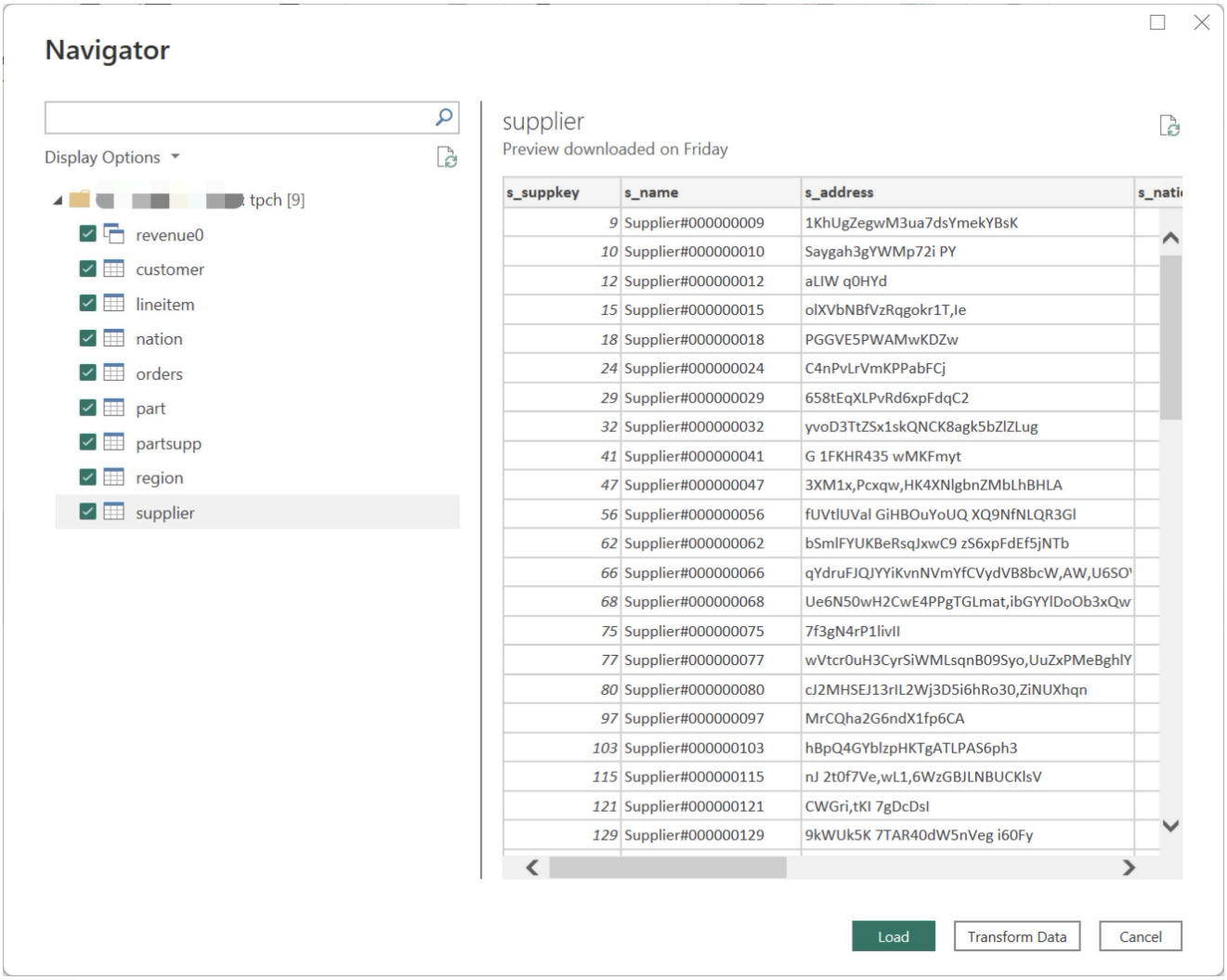
If you have use cases with a small amount of data, you can choose Import mode, and the entire dataset will be loaded into Power BI.

- Specify the user name and password

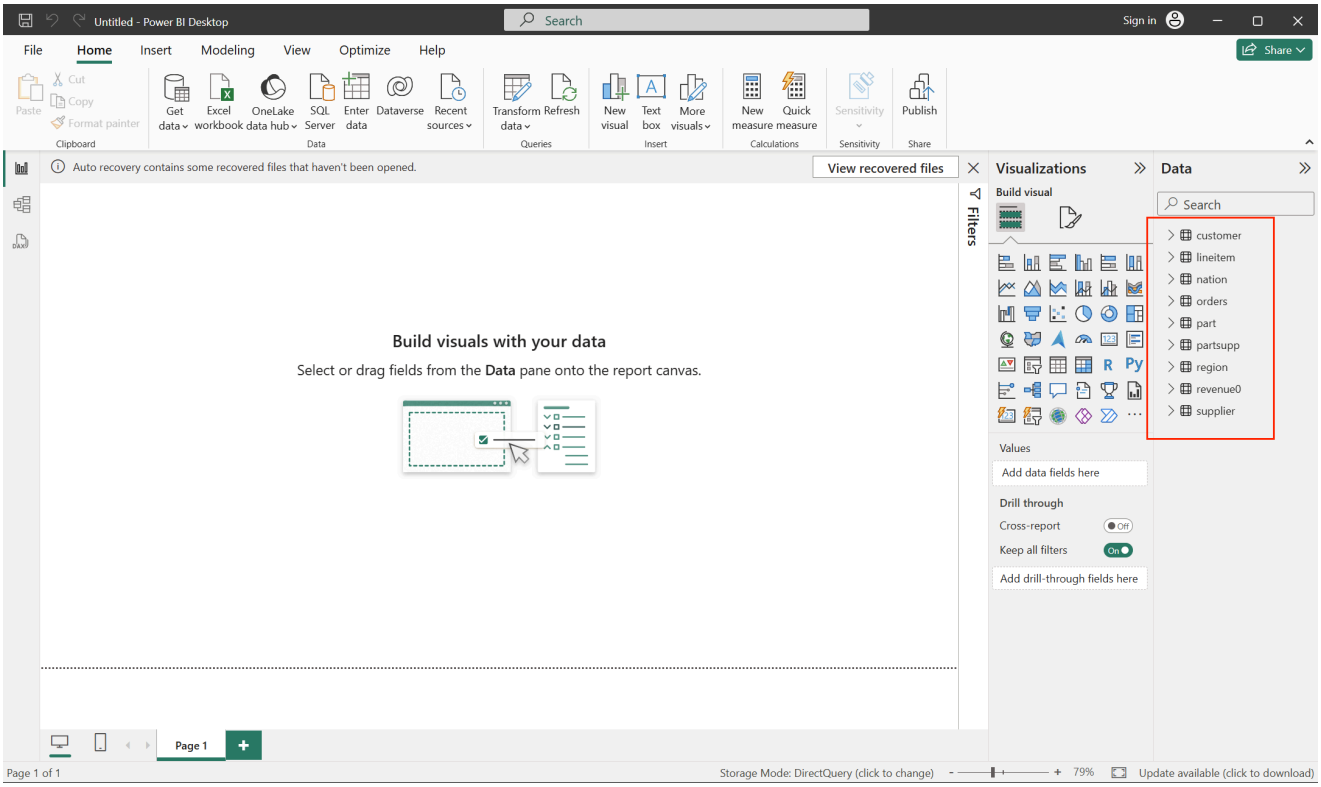


Query and visualize data

Finally, you should see the database and tables in the navigator view. Select the desired table and click “Load” to load the table structure and preview the data from Apache Doris.



After the import is complete, your Doris data should be accessible in Power BI as usual, Configure the required statistical compass.

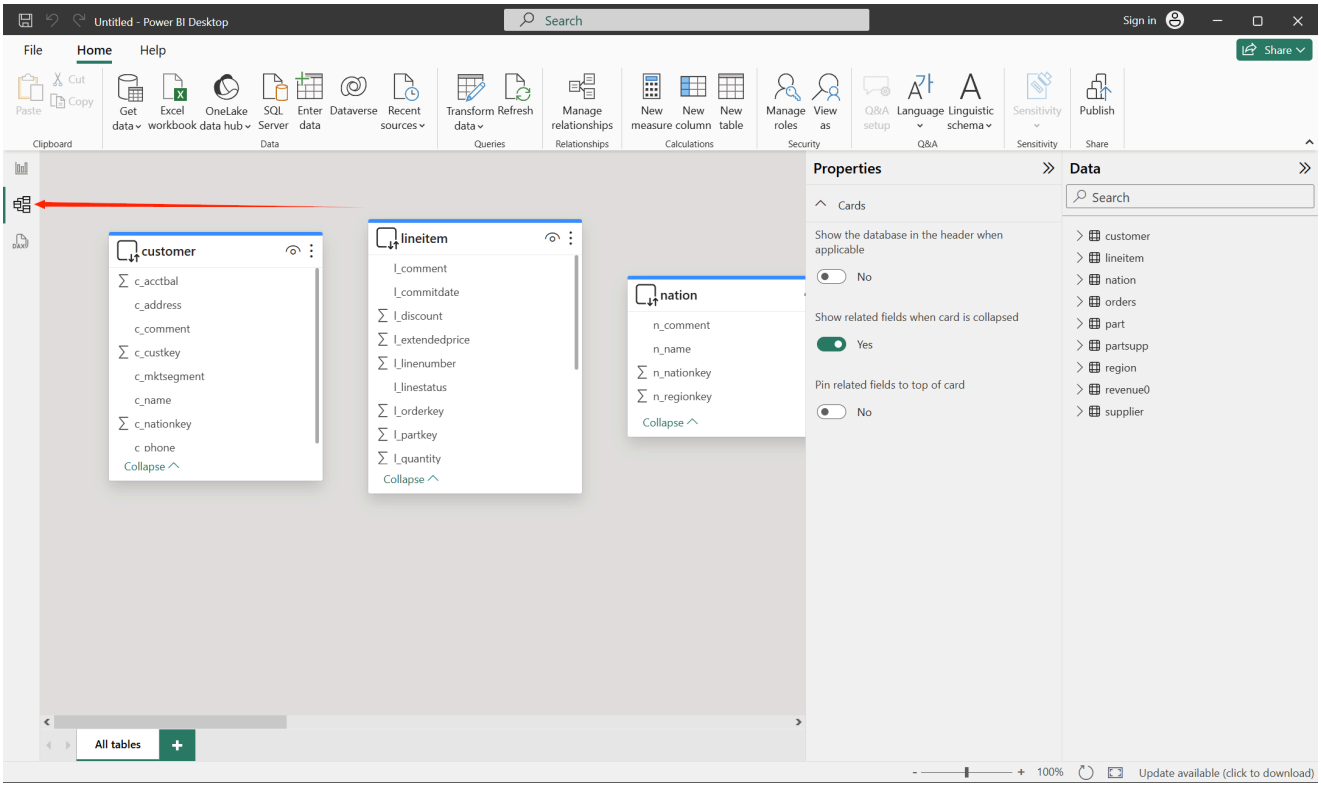


Building Visualizations in Power BI

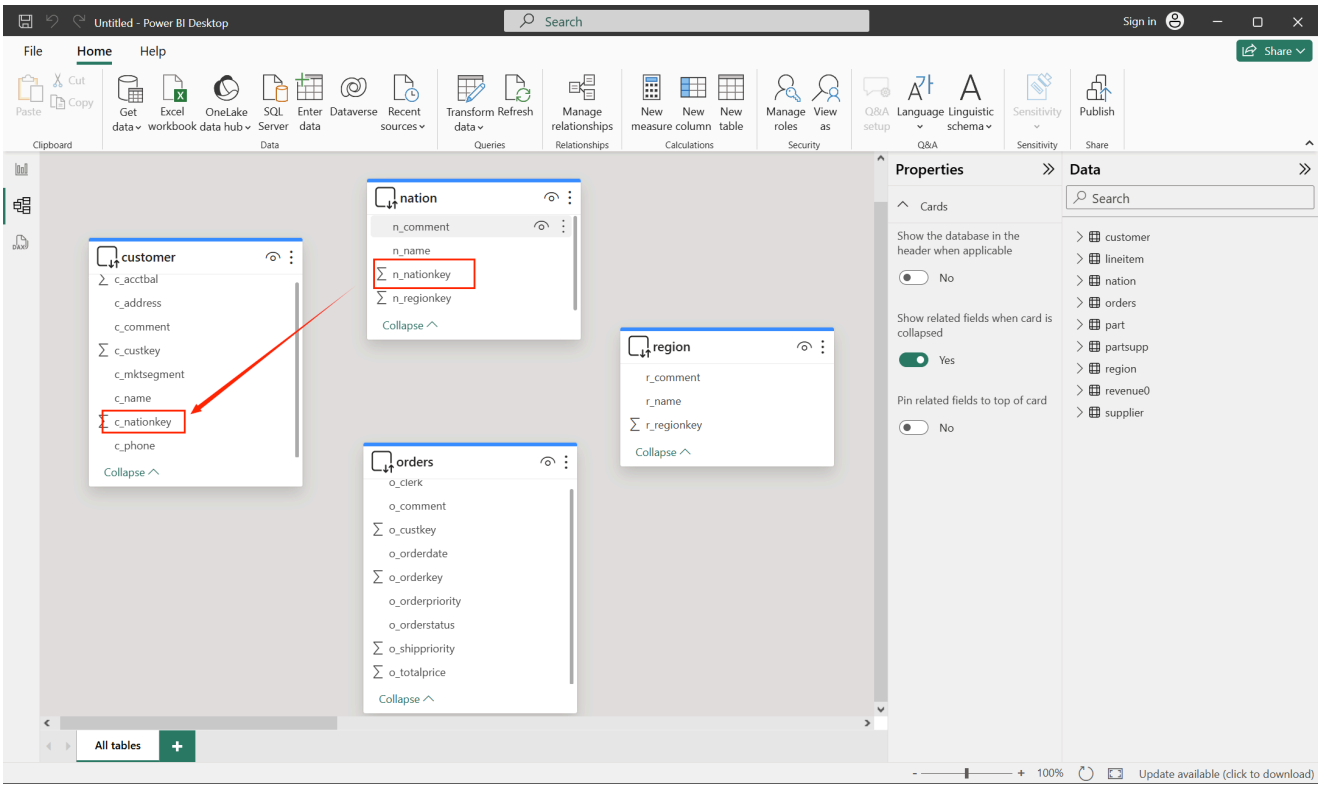
We've chosen TPC-H data as our data source. For instructions on building a Doris TPC-H data source, refer to this document. Now that we've configured the Doris data source in Power BI, let's visualize the data…

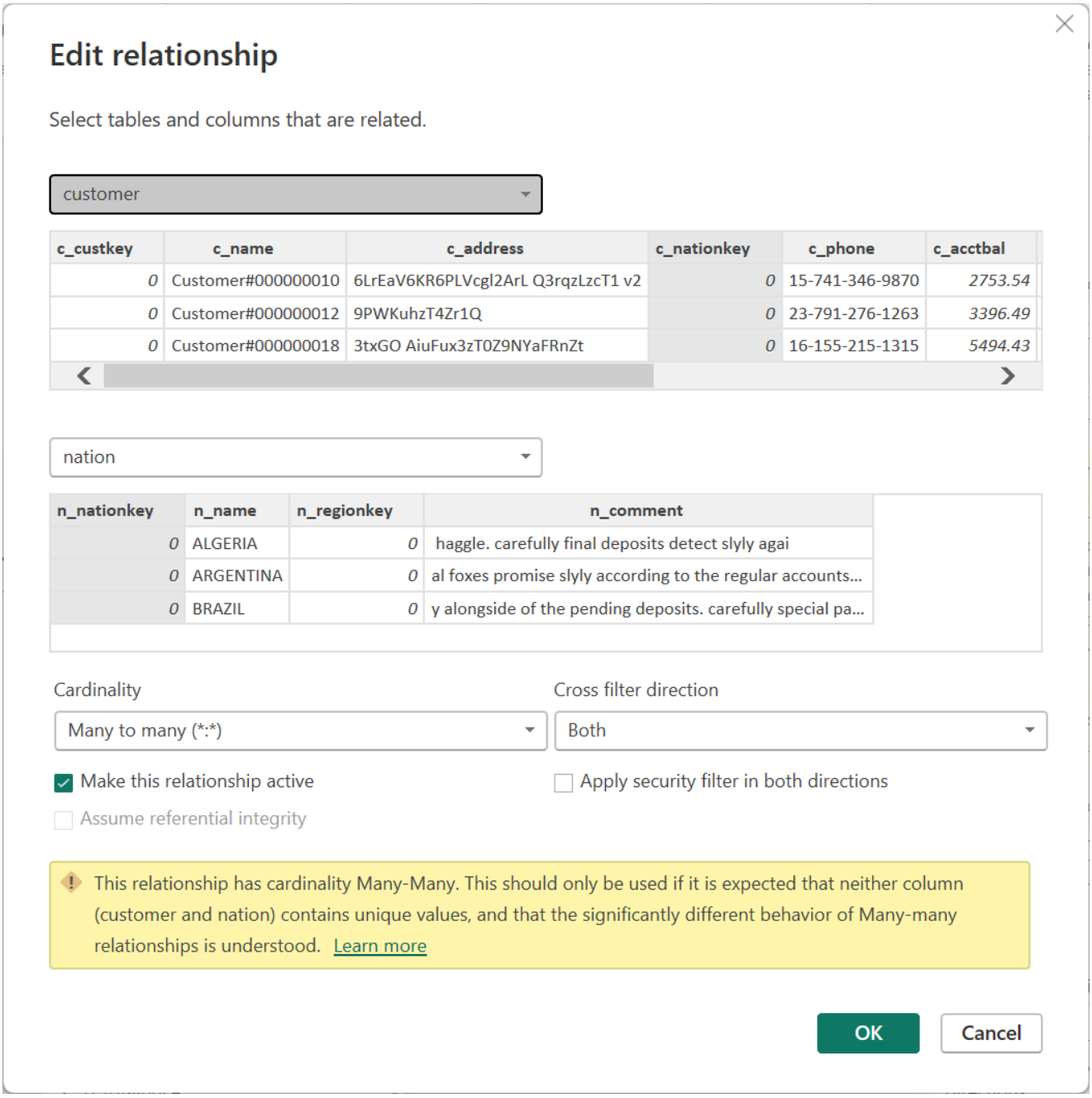
Suppose we need to know the order revenue statistics for each region, then we will build a dashboard based on this requirement.

1. First, create the table model relationships. Click Model view.



2. Drag and drop to place these four tables on the same screen as needed, and then drag and drop the related fields.

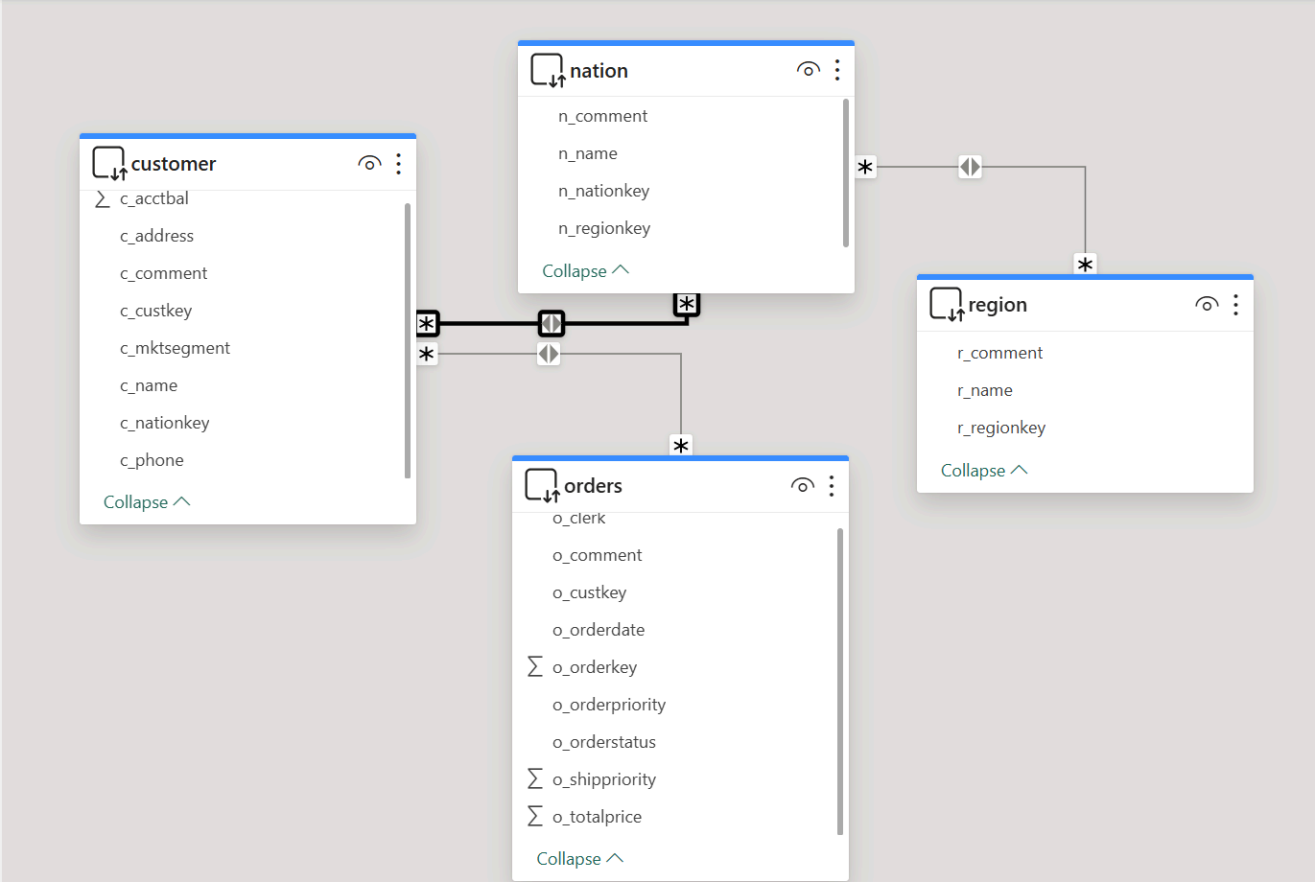




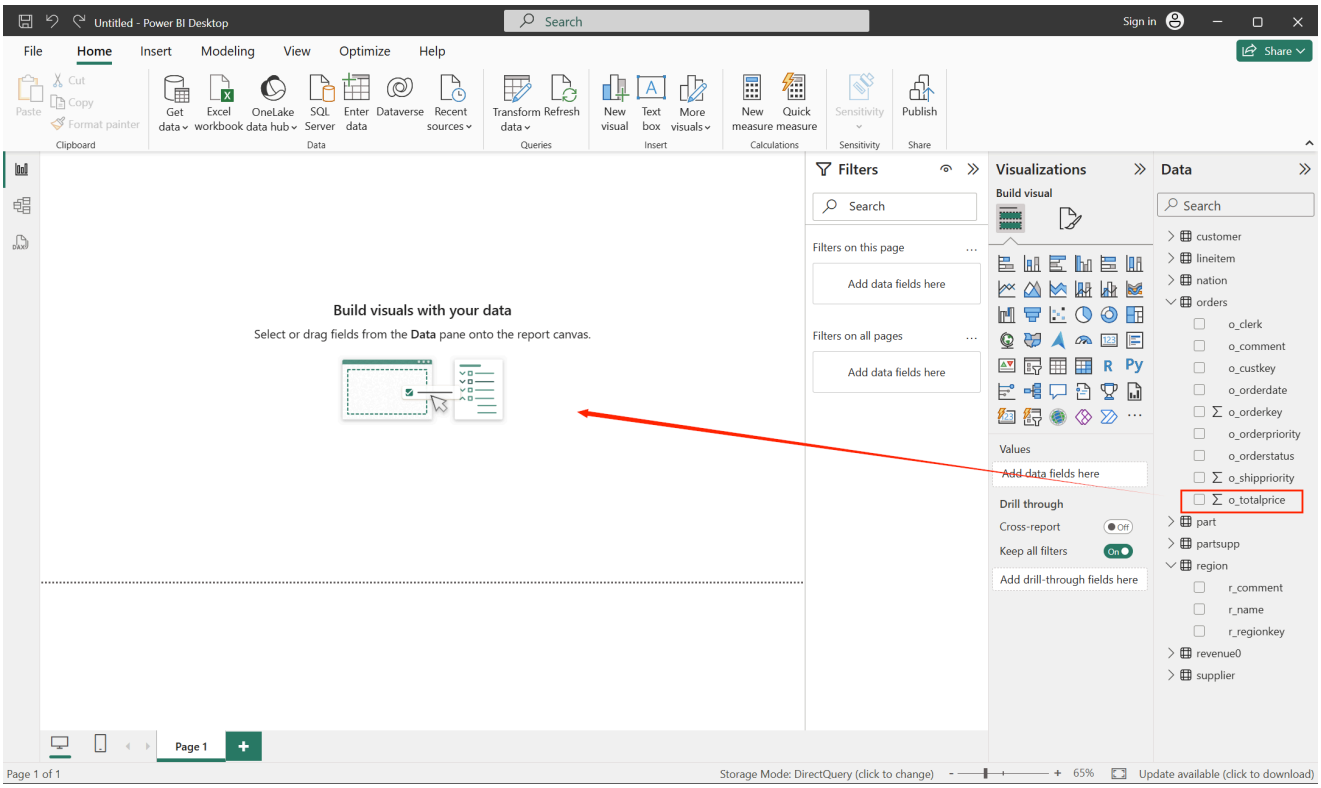
The relationships between the four tables are as follows:

- **customer**：c_nationkey – **nation** : n_nationkey
- **customer**：c_custkey – **orders** : o_custkey
- **nation** : n_regionkey – **region** : r_regionkey

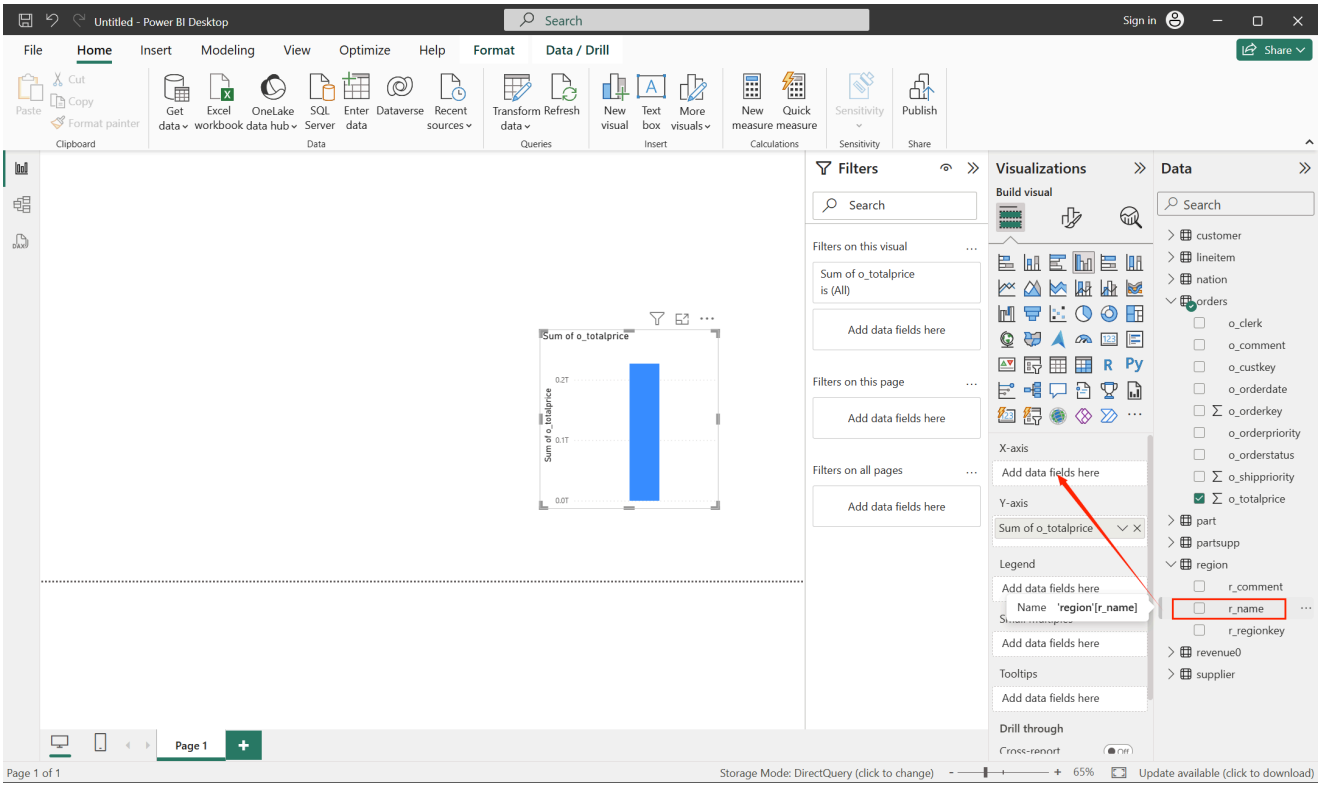
3. The results after association are as follows:



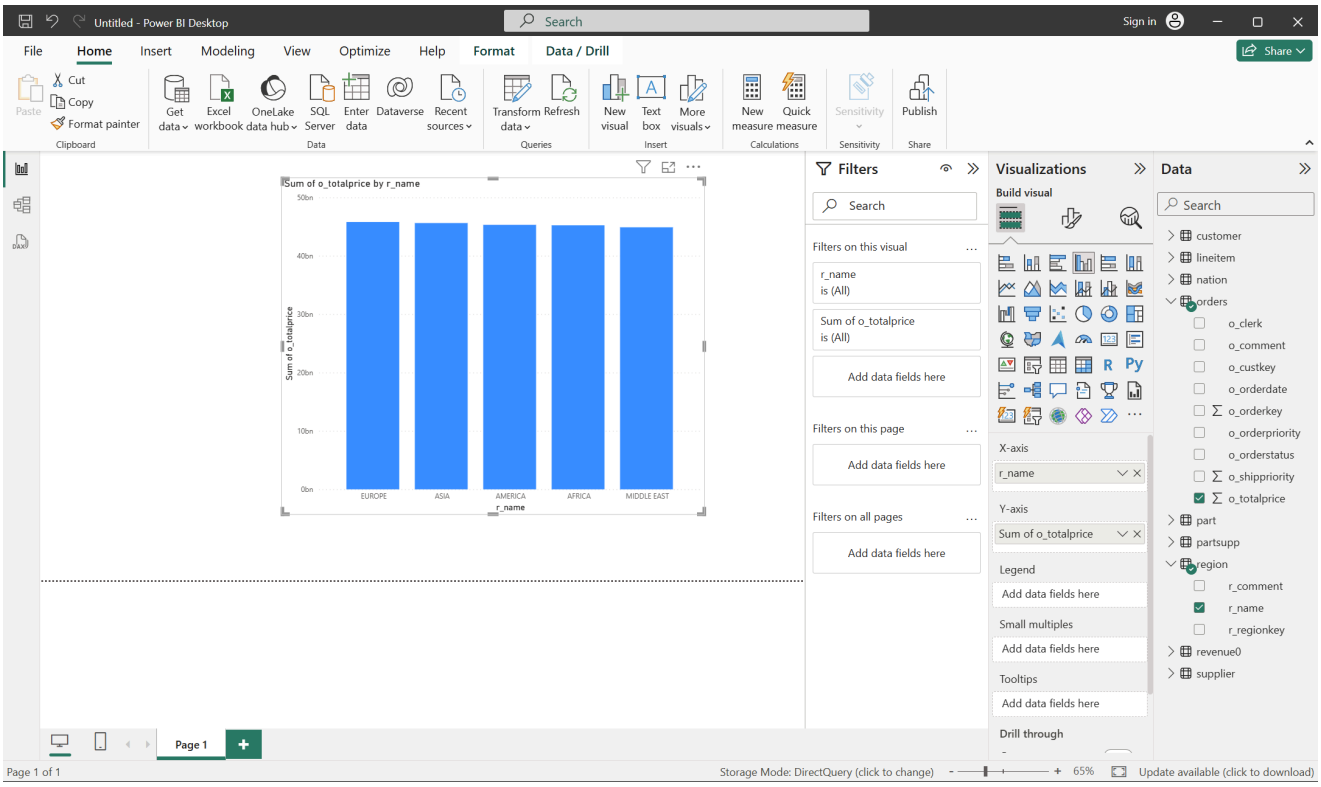
4. Return to the Report view workbench and build the dashboard.
5. Drag the `o_totalprice` field from the `orders` table to the dashboard.



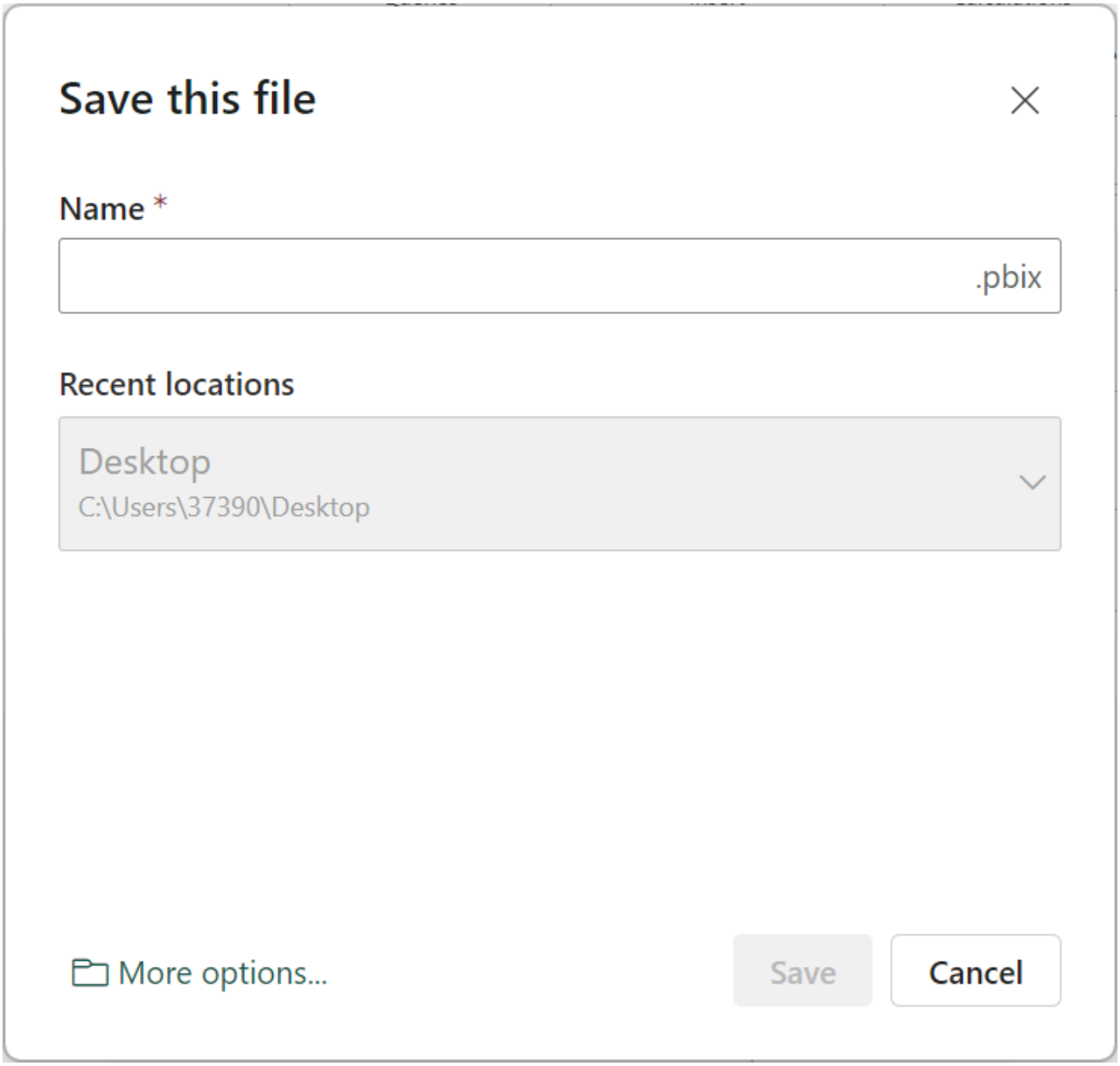
6. Drag the `r_name` field from the `region` table to column X.



7. You should now have the expected dashboard content.



8. Click the save button in the upper left corner of the workbench to save the created statistical compass to your local machine.



At this point, you have successfully connected Power BI to Apache Doris and implemented data analysis and dashboard creation.

Tableau

For enabling Apache Doris access on Tableau, Tableau's official MySQL connector can meet your needs. This connector uses the MySQL JDBC driver to access data.

Through the MySQL connector, Tableau can integrate Apache Doris databases and tables as data sources. To enable this feature, follow the setup guide below:

- Required setup before use
- Configuring the Apache Doris data source in Tableau
- Building visualizations in Tableau
- Connection and usage tips

Installing Tableau and the JDBC driver

1. Download and install Tableau desktop.
2. Obtain MySQL JDBC (version 8.3.0).
3. JDBC Driver Placement Path
    - macOS: JDBC driver JAR file placement path: `~/Library/Tableau/Drivers`
    - Windows: Assuming `tableau_path` is the Tableau installation directory on the Windows operating system, generally the default is: `tableau_path = C:\Program Files\Tableau`, then the JDBC driver JAR file placement path is: `%tableau_path%\Drivers\`

Next, you can configure a Doris data source in Tableau and start building data visualizations!

Configuring a Doris Data Source in Tableau

Now that you have installed and set up the **JDBC and Connector** drivers, let's see how to define a data source in Tableau that connects to the tpch database in Doris.

1. Gathering Your Connection Details

To connect to Apache Doris via JDBC, you need the following information:

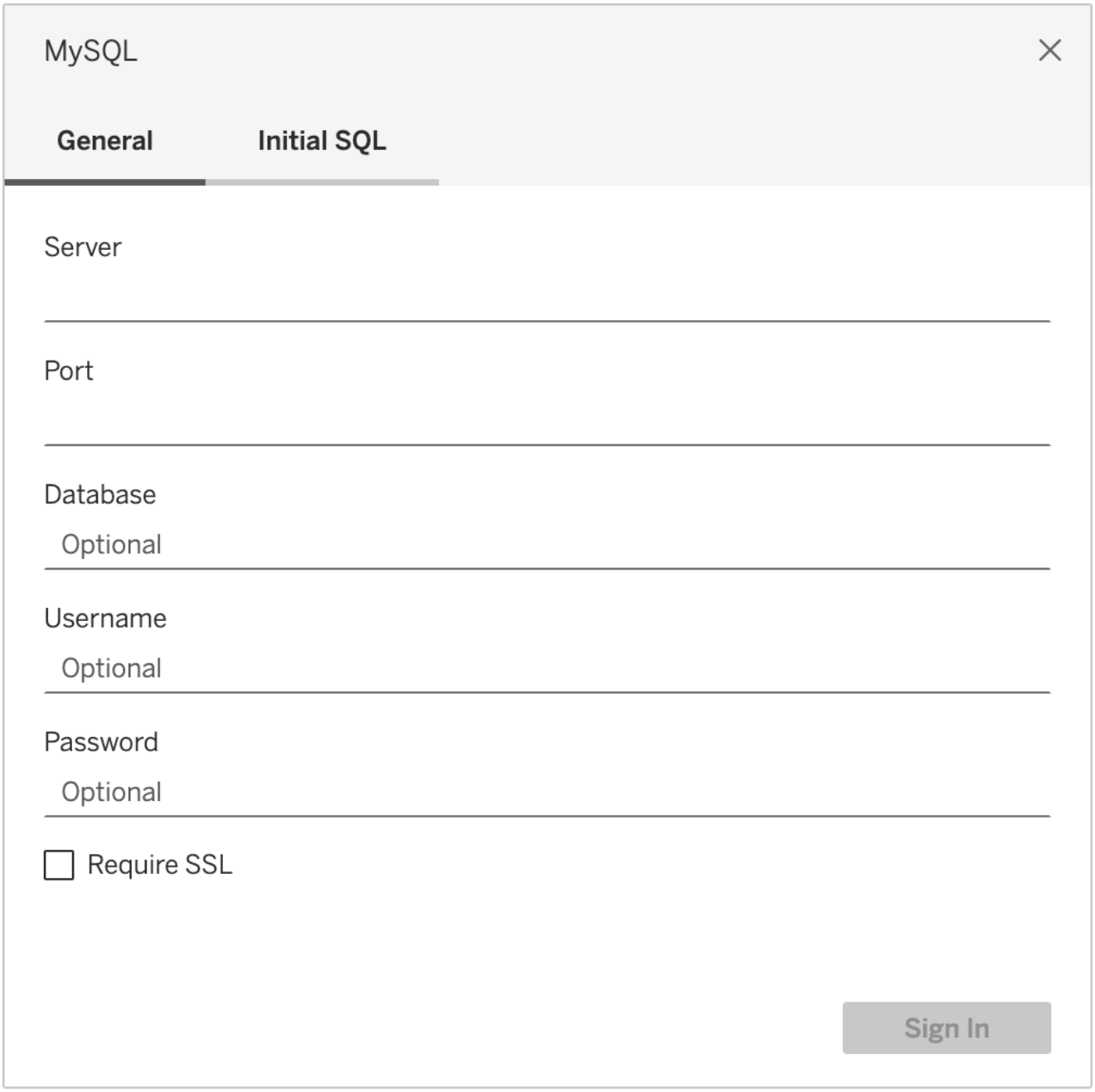
Parameter	Meaning	Example
Server	Database host	127.0.1.28
Port	Database MySQL port	9030
Database	Database name	tpch
Username	Username	testuser
Password	Password	Leave blank
Init SQL Statement	Initial SQL statement	`select * from database.table`

2. Start Tableau. (If you are already running it, please restart it.)

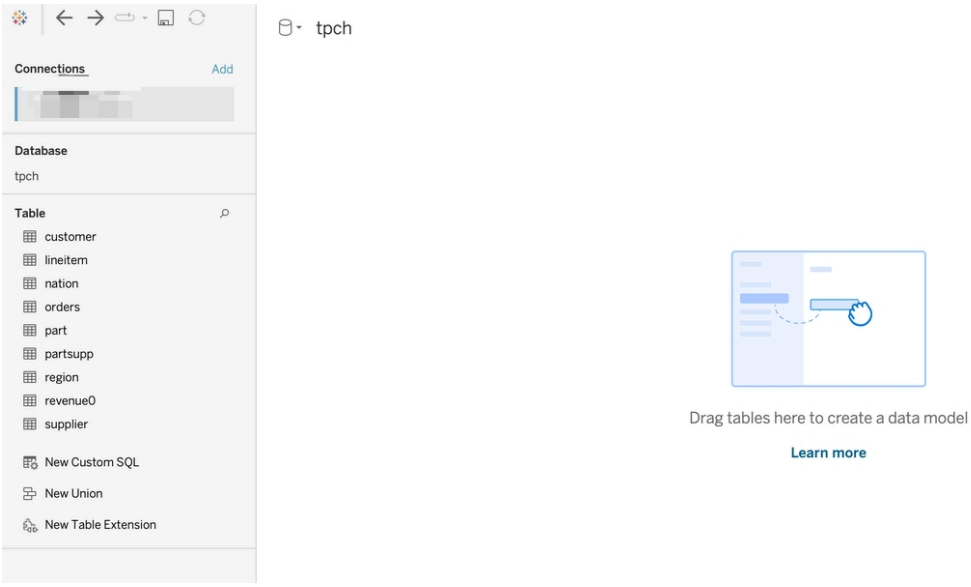
3. From the left-hand menu, click **More** under the **To a Server** section. Search for **mysql** in the list of available connectors.



4. Clicking **MySQL** will bring up the following dialog box:



5. Follow the prompts in the dialog box to enter the corresponding connection information.
6. After completing the above input boxes, click the **Sign In** button. You should then see a new Tableau workbook.



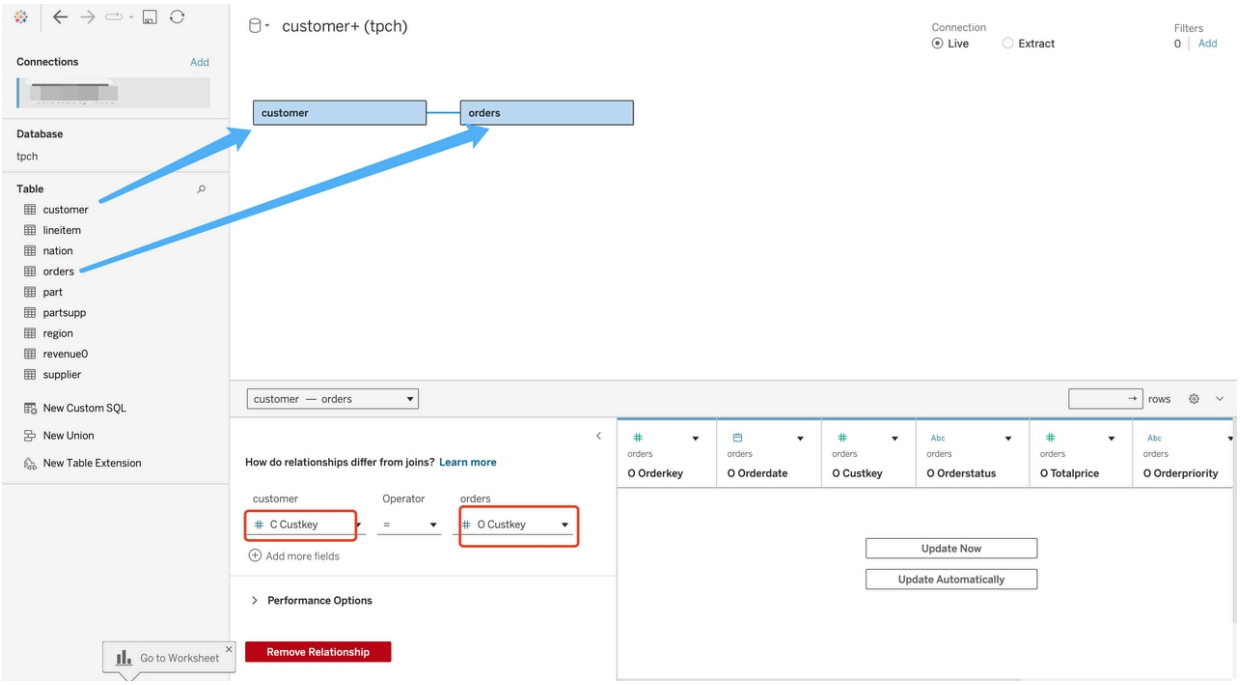
Next, we can build some visualizations in Tableau!

Building Visualizations in Tableau

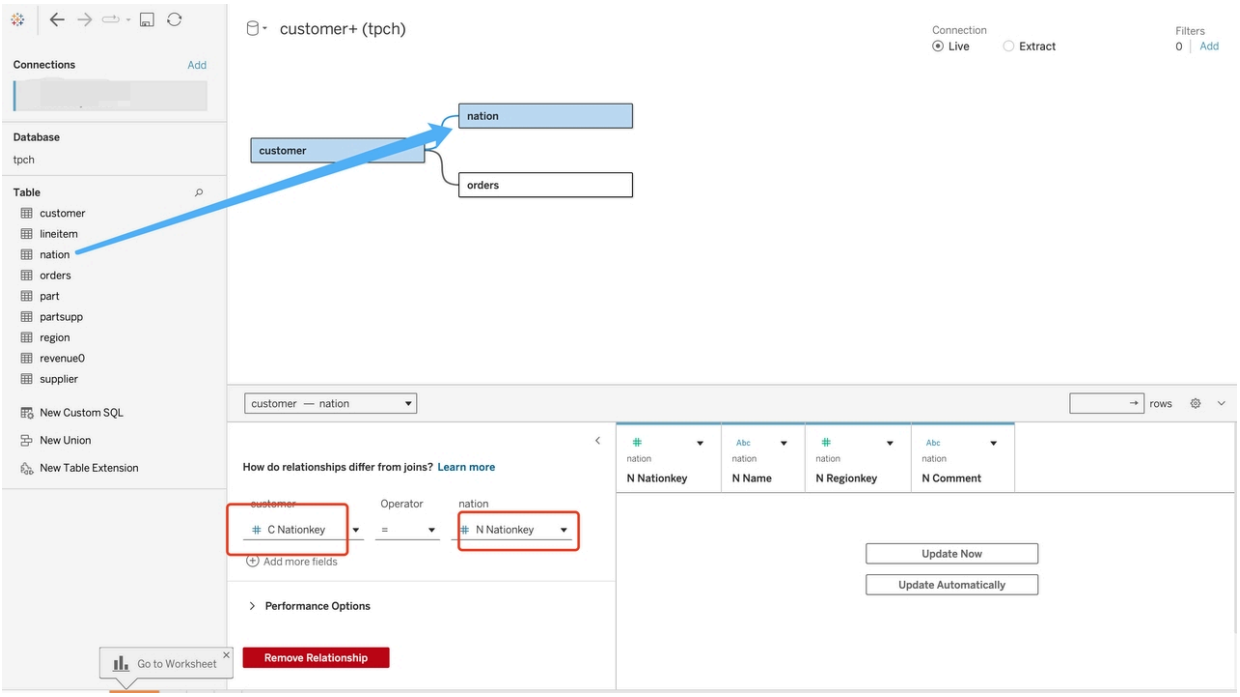
We've chosen TPC-H data as our data source. For instructions on building a Doris TPC-H data source, refer to this document.

Now that we've configured the Doris data source in Tableau, let's visualize the data.

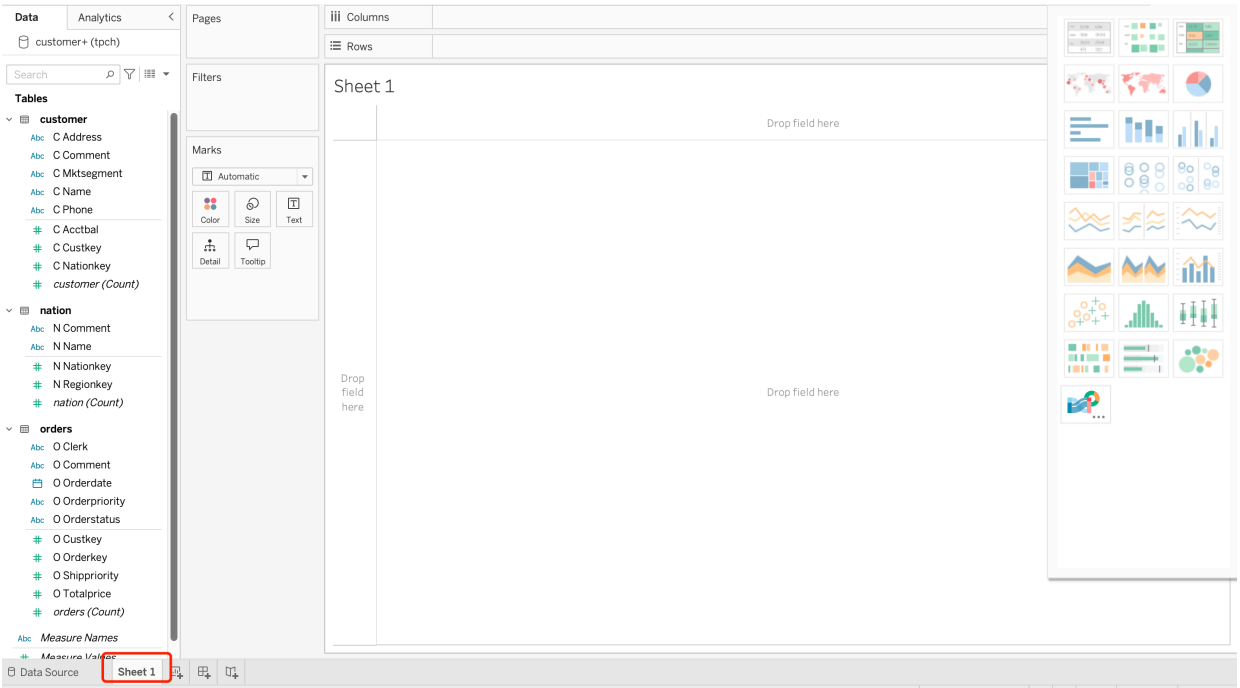
1. Drag the `customer` and `orders` tables into the workbook. Then, select the `Custkey` field for the table association below.



2. Drag the `nation` table into the workbook and associate it with the `customer` table by selecting the `Nationkey` field.

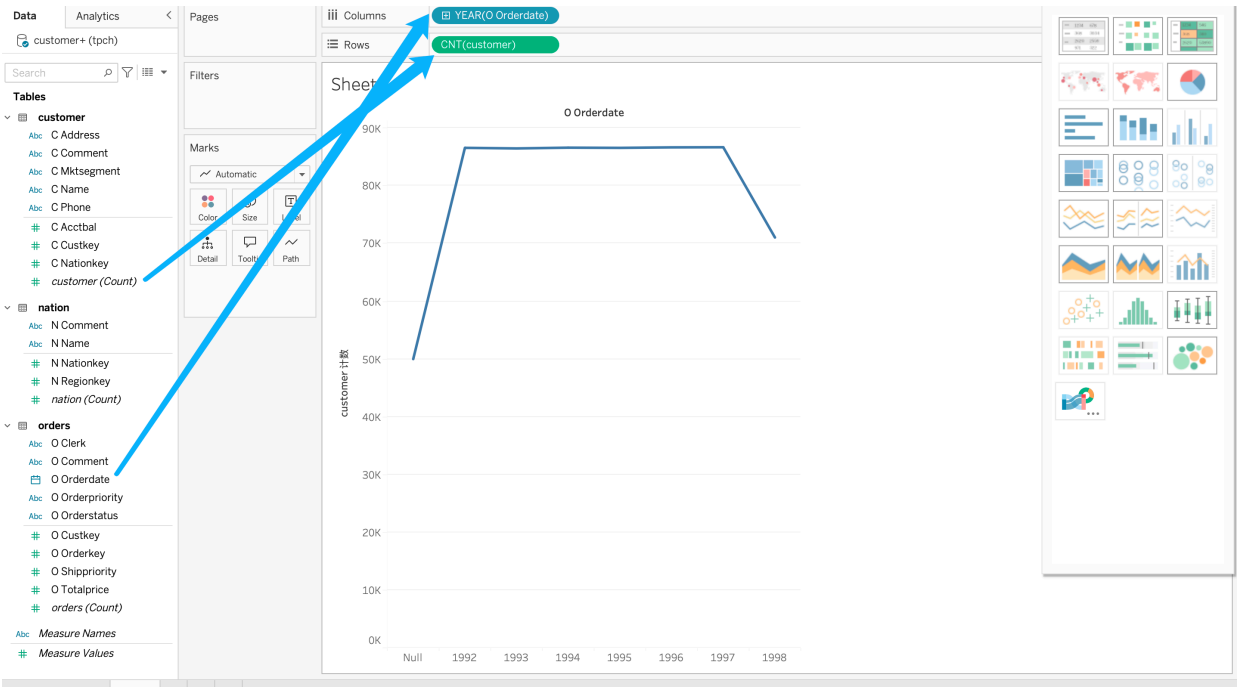


3. Now you have linked the `customer`, `orders`, and `nation` tables as data sources, so you can use this relationship to work on data-related issues. Select the `Sheet 1` tab at the bottom of the workbook to access the workbench.



4. Suppose you want to know the total number of users each year. Drag OrderDate from orders to the Columns area (a horizontal field), and then drag customer(count) from customer to Rows. Tableau will

generate the following line chart:



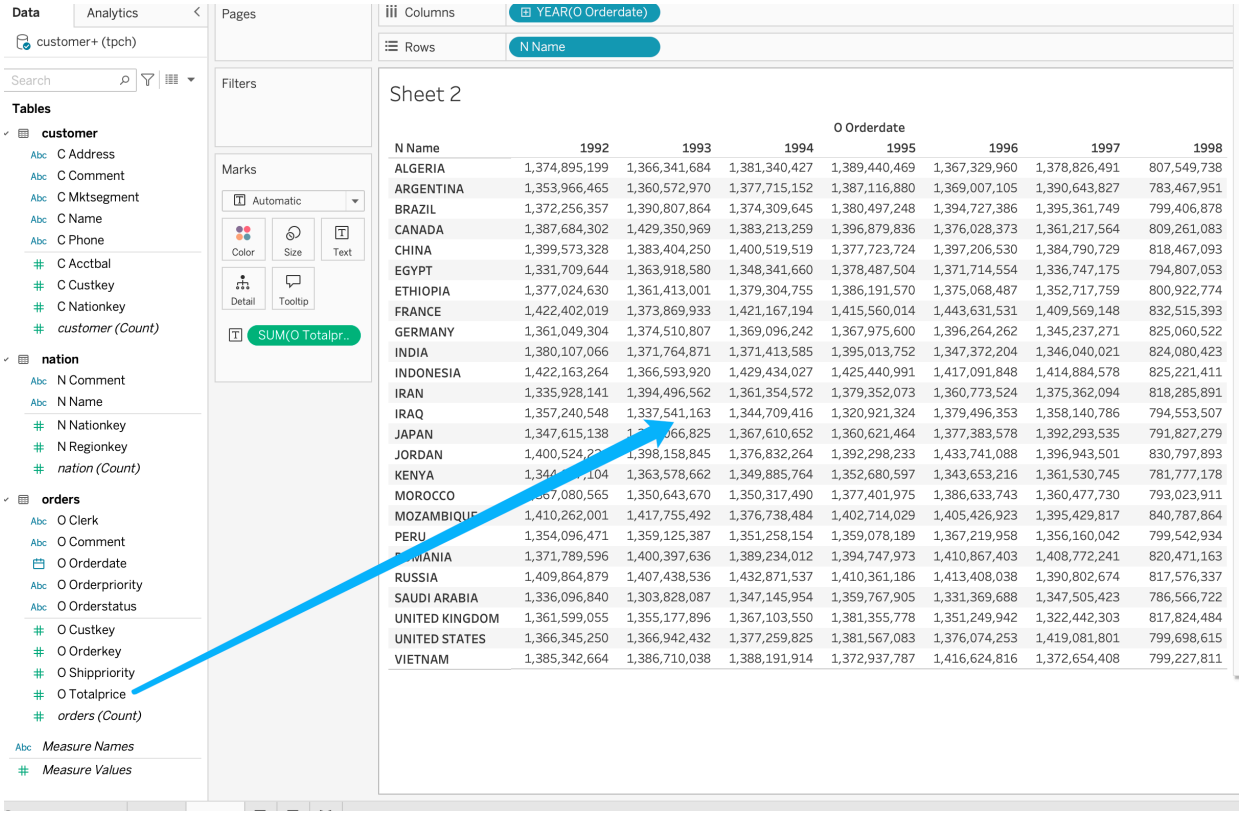
A simple line chart is now complete. However, this dataset is not real data and was automatically generated by the tpch script and default rules; it is intended for testing usability.

5. Suppose you want to know the average order amount (USD) by region (country) and year:
    - Click the `New Worksheet` tab to create a new table
    - Drag Name from the nation table into `Rows`
    - Drag OrderDate from the orders table into `Columns`

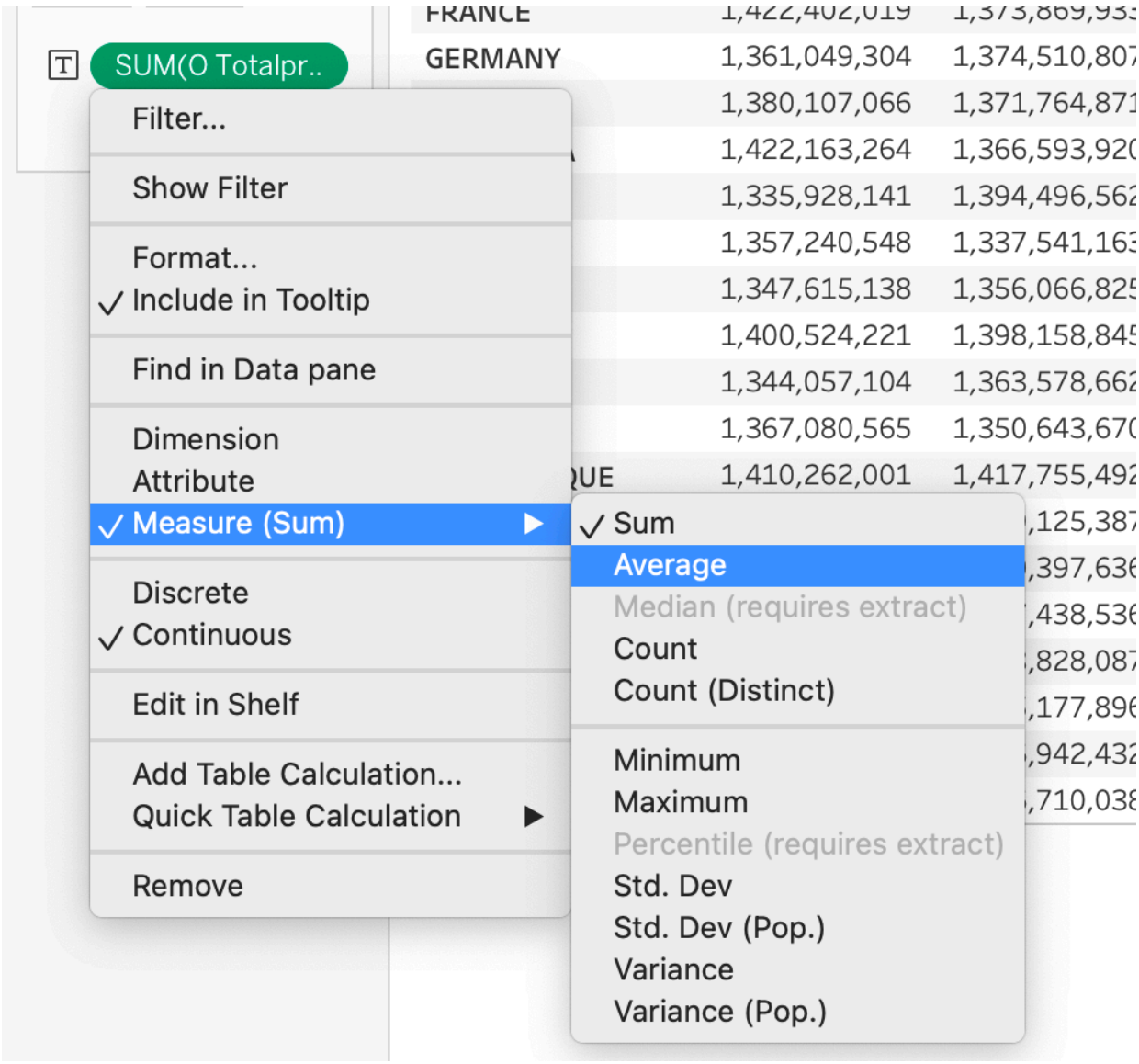
You should see the following:

6. Note: The `Abc` value is just a populated value because you haven't defined the aggregation logic to this icon, so you need to drag the measure onto the table. Drag Totalprice from the orders table to the middle

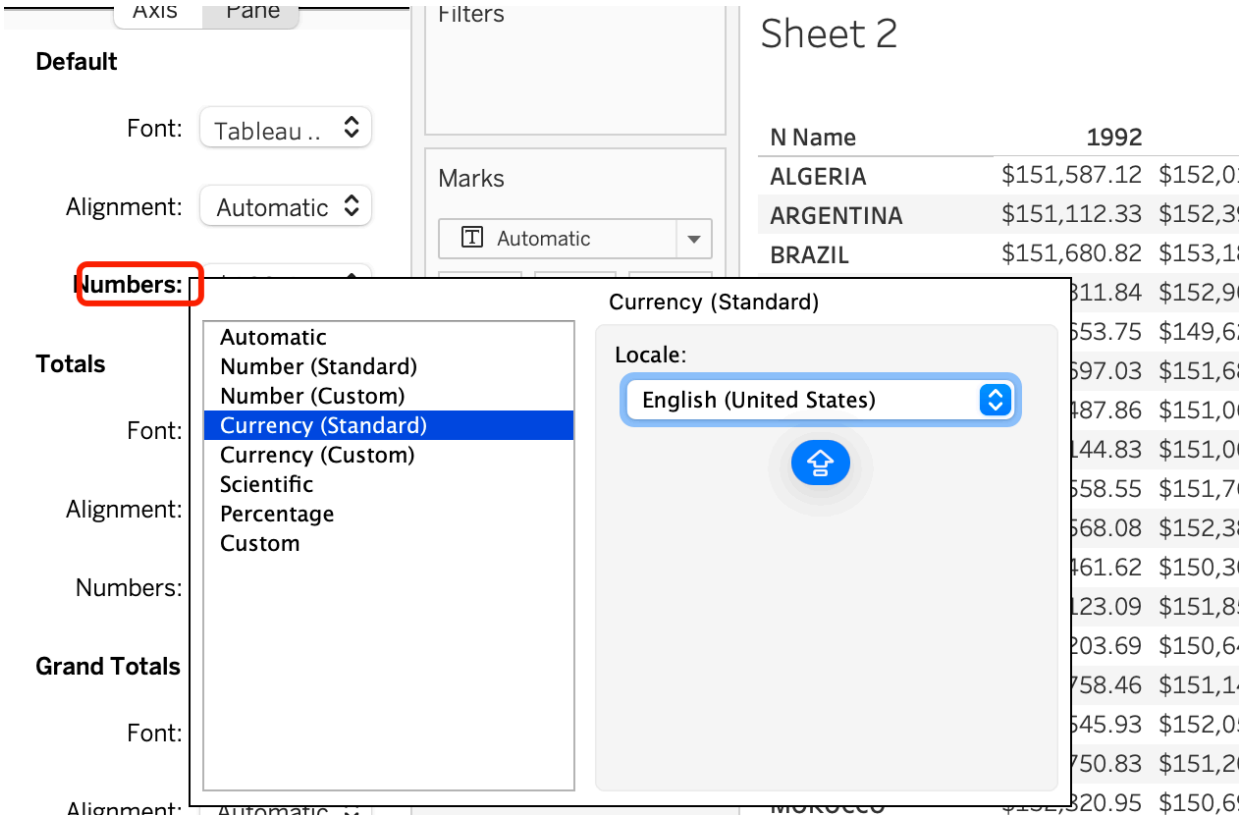
of the table. Note that the default calculation is a SUM on Totalprices:



7. Click SUM and change Measure to Average.



8. From the same drop-down menu, select `Format` and change `Numbers` to `Currency (Standard)`



9. You will get a table that meets your expectations.

N Name	1992	1993	1994	1995	1996	1997	1998
ALGERIA	$151,587.12	$152,018.43	$152,146.76	$151,619.43	$152,043.81	$152,137.98	$153,380.77
ARGENTINA	$151,112.33	$152,393.93	$151,247.68	$151,996.15	$151,288.22	$152,432.73	$149,175.16
BRAZIL	$151,680.82	$153,189.54	$150,956.68	$150,973.01	$150,374.92	$151,983.63	$150,632.54
CANADA	$152,811.84	$152,904.47	$150,365.61	$151,063.03	$149,861.51	$150,427.40	$150,588.22
CHINA	$150,653.75	$149,621.92	$150,723.15	$149,997.14	$151,837.27	$150,618.96	$152,272.95
EGYPT	$150,697.03	$151,681.34	$151,057.77	$151,782.37	$151,336.56	$149,708.50	$150,503.13
ETHIOPIA	$151,487.86	$151,066.69	$151,989.50	$150,836.95	$152,328.40	$150,519.39	$150,408.03
FRANCE	$152,144.83	$151,007.91	$150,213.21	$152,473.07	$150,425.29	$151,664.42	$150,900.02
GERMANY	$150,558.55	$151,762.26	$151,936.11	$150,990.68	$152,347.44	$150,490.80	$151,331.72
INDIA	$150,568.08	$152,384.46	$150,324.85	$152,945.26	$150,493.94	$148,898.23	$151,988.27
INDONESIA	$150,461.62	$150,307.29	$152,586.89	$153,289.71	$150,947.15	$151,583.95	$151,583.65
IRAN	$151,123.09	$151,856.32	$150,028.06	$150,633.62	$151,956.84	$151,371.57	$152,097.75
IRAQ	$150,203.69	$150,640.97	$150,347.65	$149,917.30	$152,649.81	$152,121.50	$150,855.04
JAPAN	$151,758.46	$151,144.32	$151,990.51	$150,361.53	$150,813.92	$152,563.39	$150,910.48
JORDAN	$152,545.93	$152,056.43	$150,539.28	$152,030.82	$153,538.35	$151,331.76	$151,412.05
KENYA	$151,750.83	$151,206.33	$150,371.59	$150,465.03	$150,616.88	$151,685.69	$150,312.86
MOROCCO	$152,320.95	$150,691.03	$151,279.13	$150,618.04	$152,427.59	$149,832.35	$150,393.31
MOZAMBIQUE	$151,218.31	$152,463.22	$151,373.12	$150,296.16	$150,894.02	$149,339.66	$151,329.71
PERU	$150,992.02	$152,385.40	$152,117.32	$151,581.33	$151,744.72	$150,533.92	$151,917.71
ROMANIA	$149,481.27	$150,937.45	$151,052.95	$150,247.55	$150,508.58	$151,366.95	$152,051.74
RUSSIA	$151,500.63	$151,810.87	$152,644.25	$150,182.22	$150,234.70	$149,919.44	$149,876.51
SAUDI ARABIA	$151,039.66	$150,418.56	$151,042.26	$150,267.20	$150,658.56	$151,031.77	$150,914.57
UNITED KINGDOM	$150,154.28	$150,659.02	$151,278.47	$150,490.88	$150,573.87	$149,259.85	$153,093.31
UNITED STATES	$150,511.70	$151,495.34	$152,351.75	$151,040.46	$152,338.56	$152,360.08	$151,286.15
VIETNAM	$151,089.83	$151,718.82	$150,448.89	$151,254.58	$152,325.25	$149,363.92	$152,408.05

At this point, Tableau has been successfully connected to Apache Doris, enabling data analysis and dashboard creation.

Connection and Usage Tips

Performance Optimization

- Create Doris database tables appropriately based on actual needs, partitioning and bucketing by time to effectively reduce predicate filtering and most data transfer.
- Appropriate data pre-aggregation can be achieved by creating materialized views on the Doris side.
- Set a reasonable refresh schedule to balance refresh computational resource consumption and dashboard data timeliness.

Security Configuration

- It is recommended to use a VPC private connection to avoid security risks introduced by public network access.
- Configure security groups to restrict access.
- Enable SSL/TLS connections and other access methods.
- Fine-tune Doris user account roles and access permissions to avoid excessive delegation of permissions.

QuickSight

QuickSight can connect to Apache Doris via the official MySQL data source in Directly query or Import mode

Prerequisites

- Apache Doris version must be no less than 3.1.2
- Network connectivity (VPC, security group configuration) needs to be configured according to the Doris deployment environment to ensure that AWS servers can access your Doris cluster.
- Run the following SQL on the MySQL client that connects to Doris to adjust the declared MySQL compatibility version:

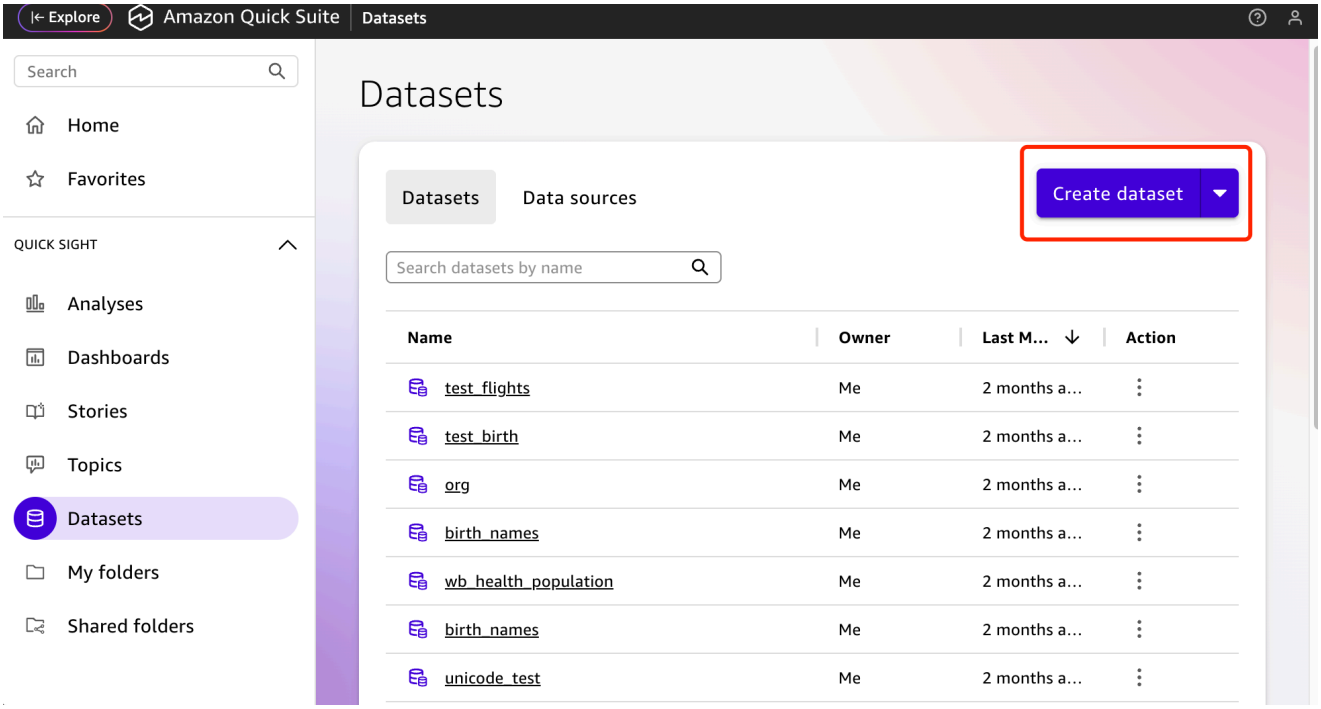
  ```
  SET GLOBAL version = '8.3.99';
  ```

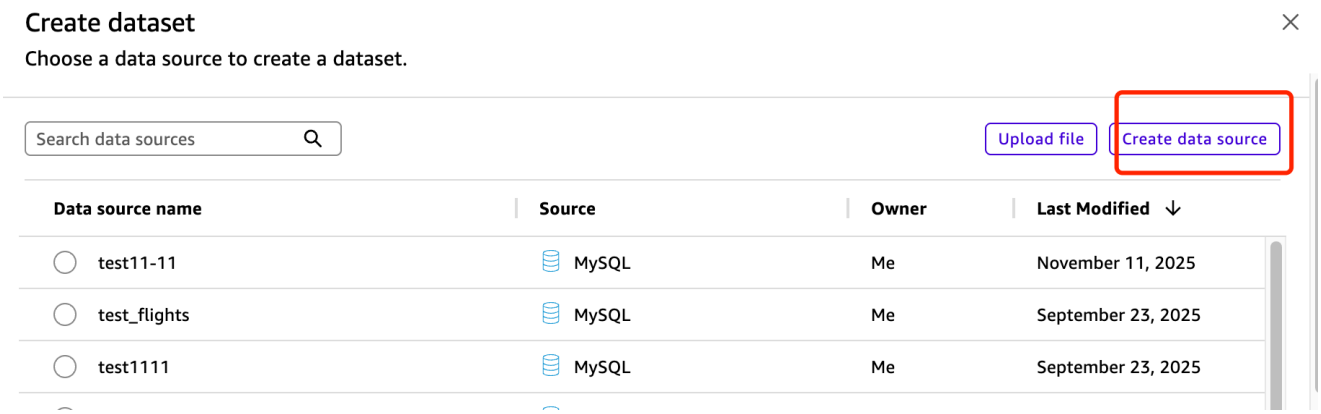
  Verification result:

  ```
  mysql> show variables like "version";
  +---------------+--------+---------------+---------+
  | Variable_name | Value  | Default_Value | Changed |
  +---------------+--------+---------------+---------+
  | version       | 8.3.99 | 5.7.99        | 1       |
  +---------------+--------+---------------+---------+
  1 row in set (0.01 sec)
  ```

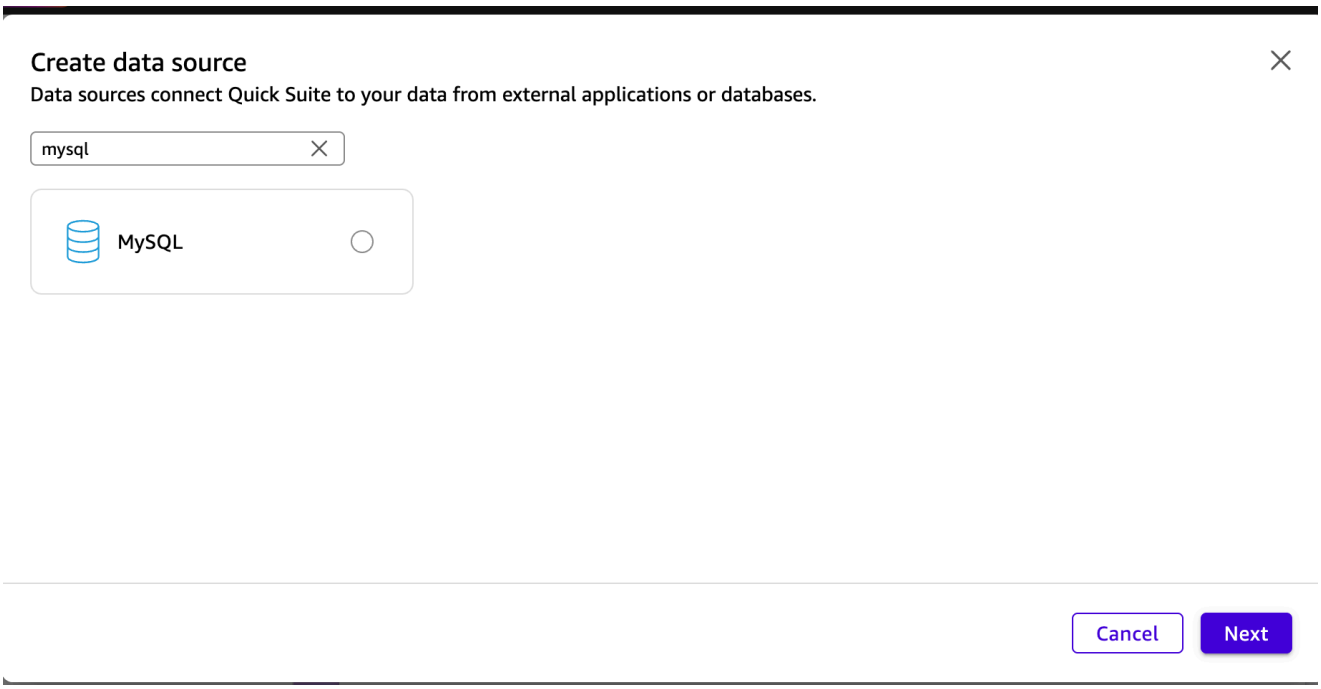
Connect QuickSight to Apache Doris

First, visit https://quicksight.aws.amazon.com, navigate to Datasets, and click “New dataset”:

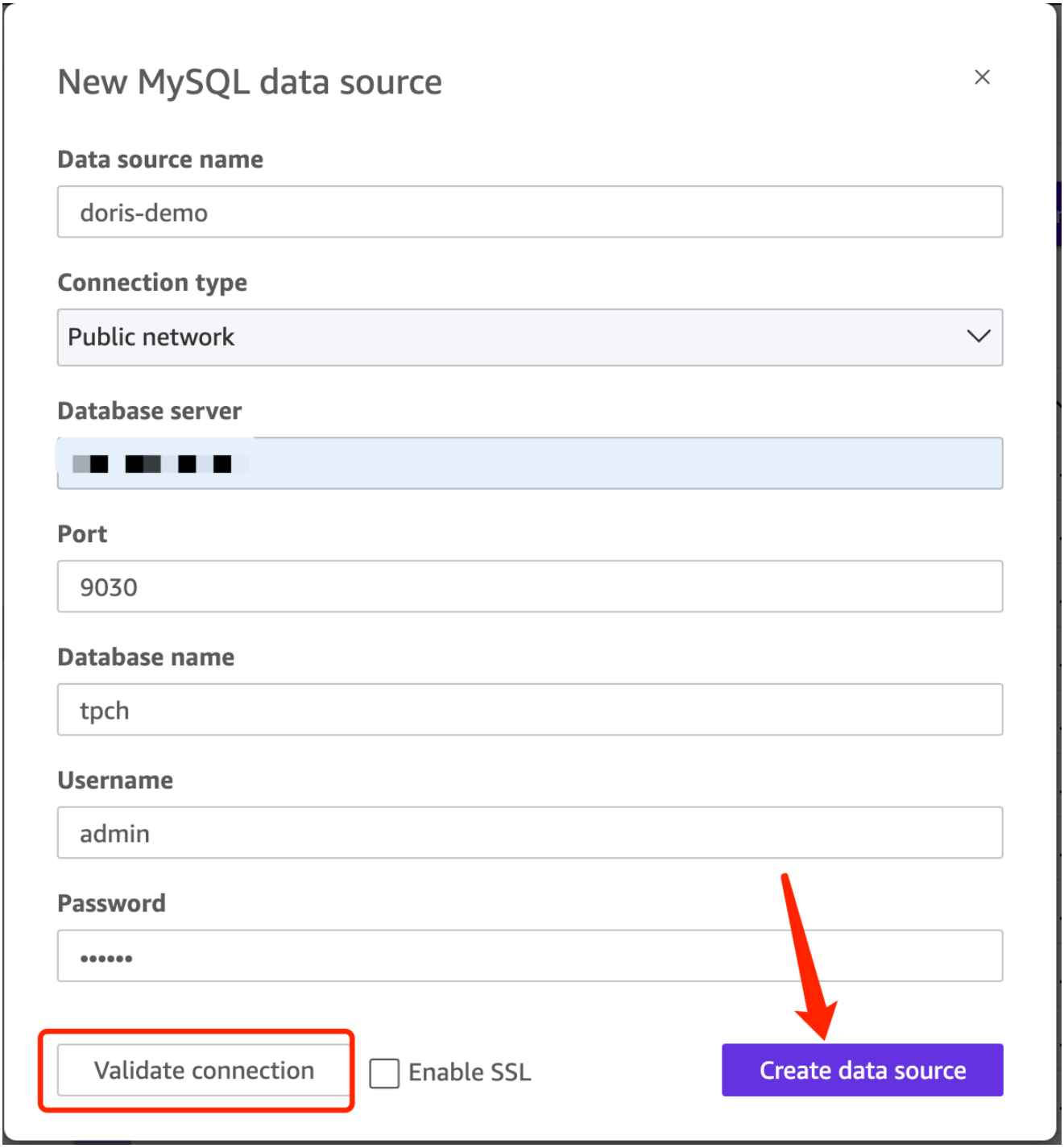




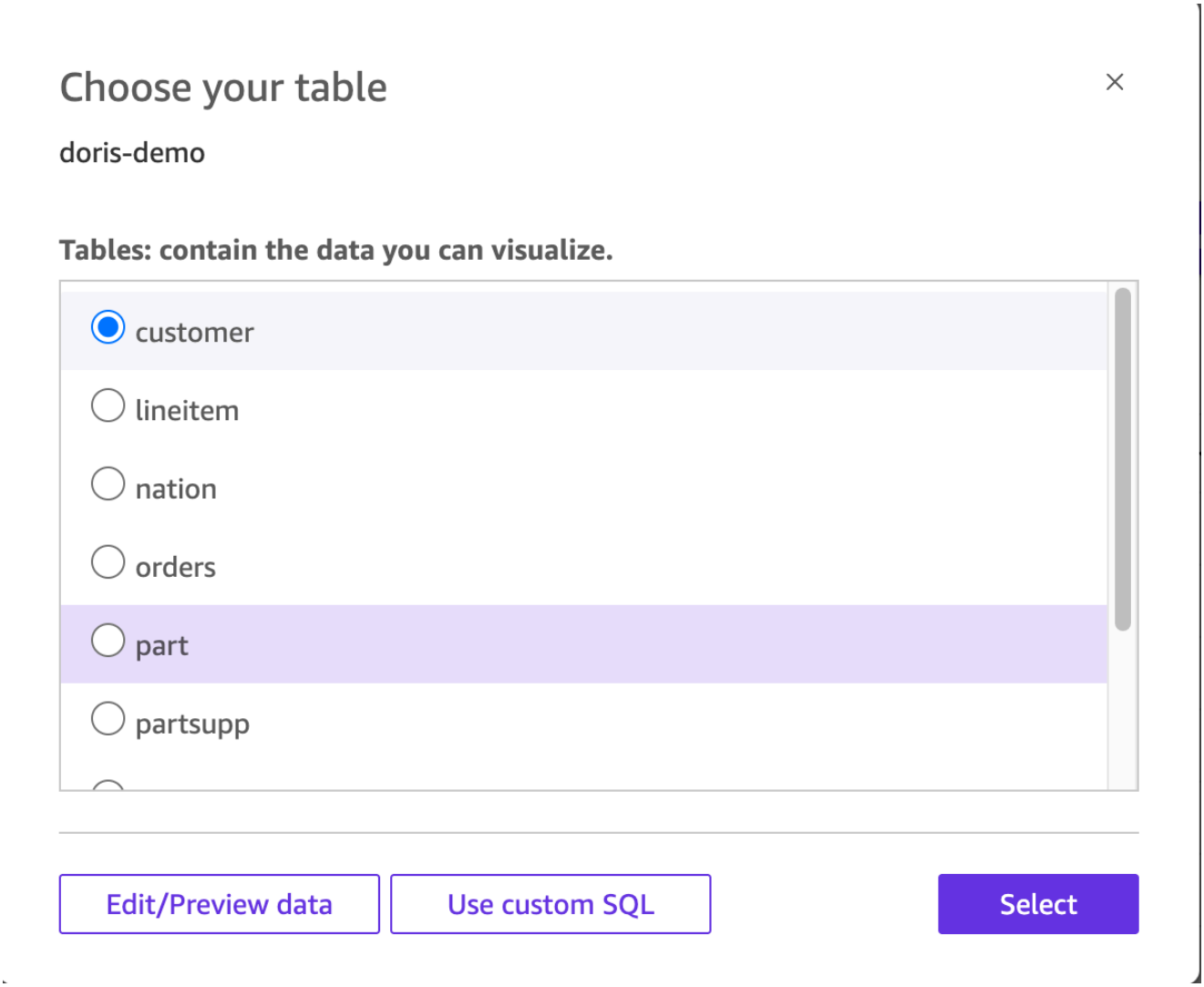
Search for the official MySQL connector bundled with QuickSight:



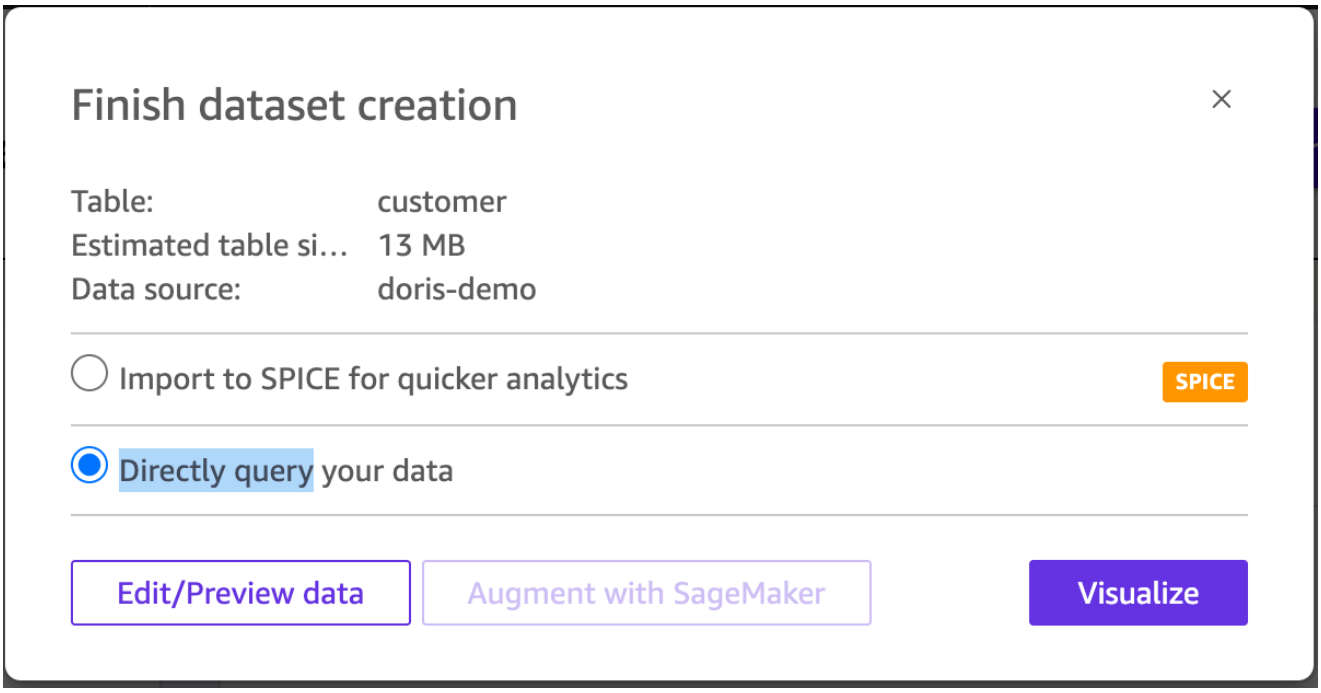
Specify your connection details. Note that the MySQL interface port defaults to 9030, which may vary depending on your FE `query_port` configuration.



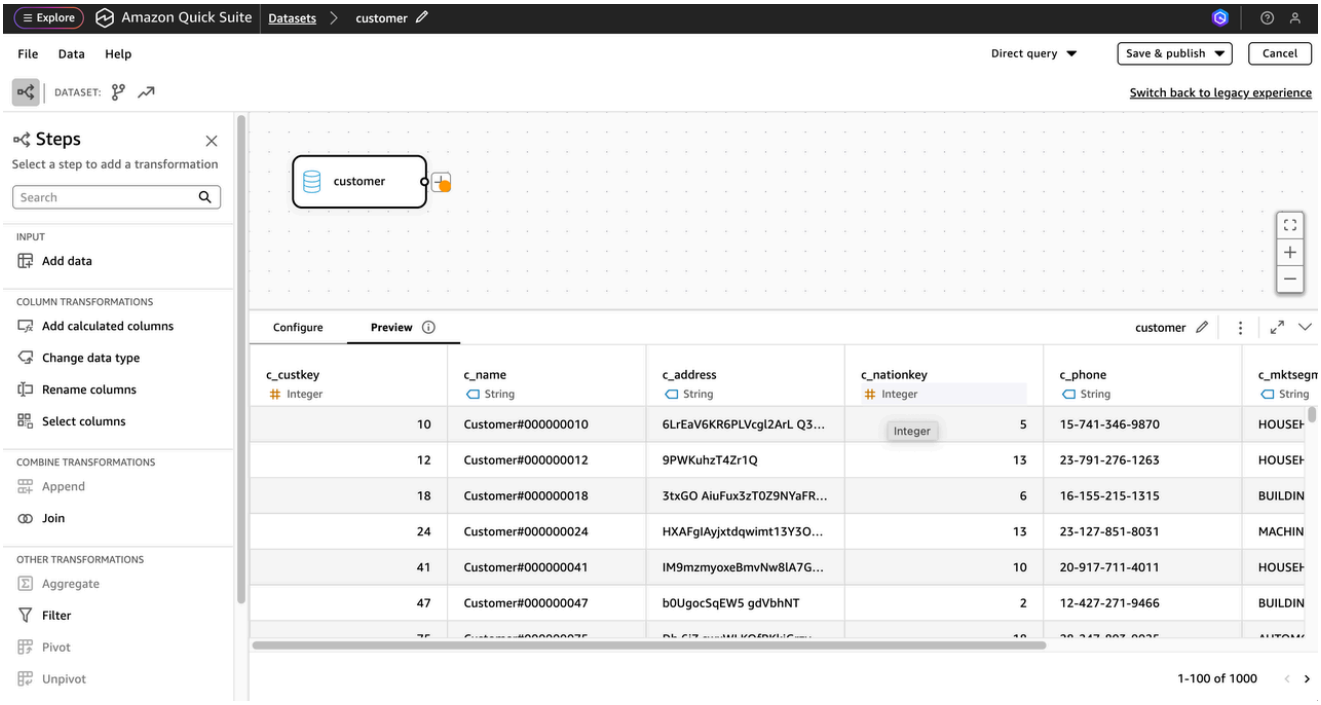
Now, you can select a table from the list:



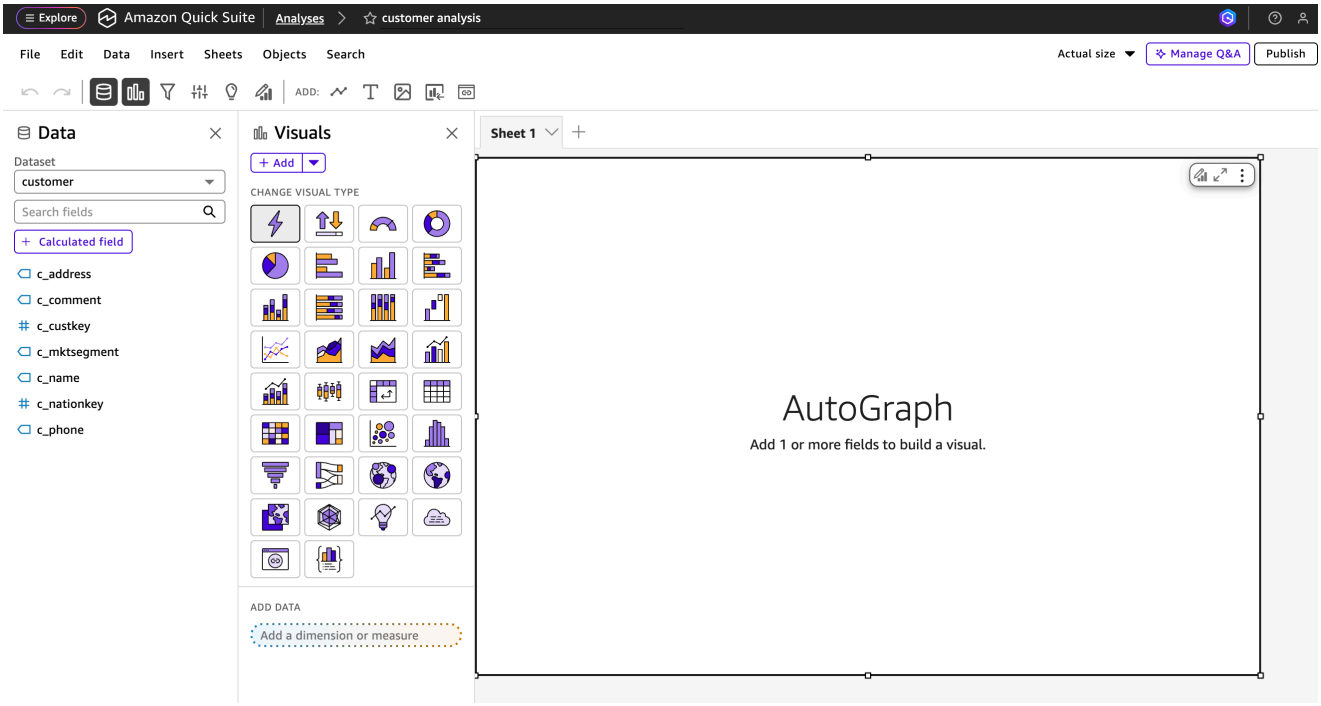
It is recommended to choose the “Directly query” mode:



Additionally, by clicking “Edit/Preview data”, you should be able to view the internal table structure or adjust the custom SQL, and you can adjust the dataset here:



Now, you can proceed to publish the dataset and create new visualizations!



Building Visualizations in QuickSight

We’ve chosen TPC-H data as our data source. For instructions on building a Doris TPC-H data source, refer to this document.

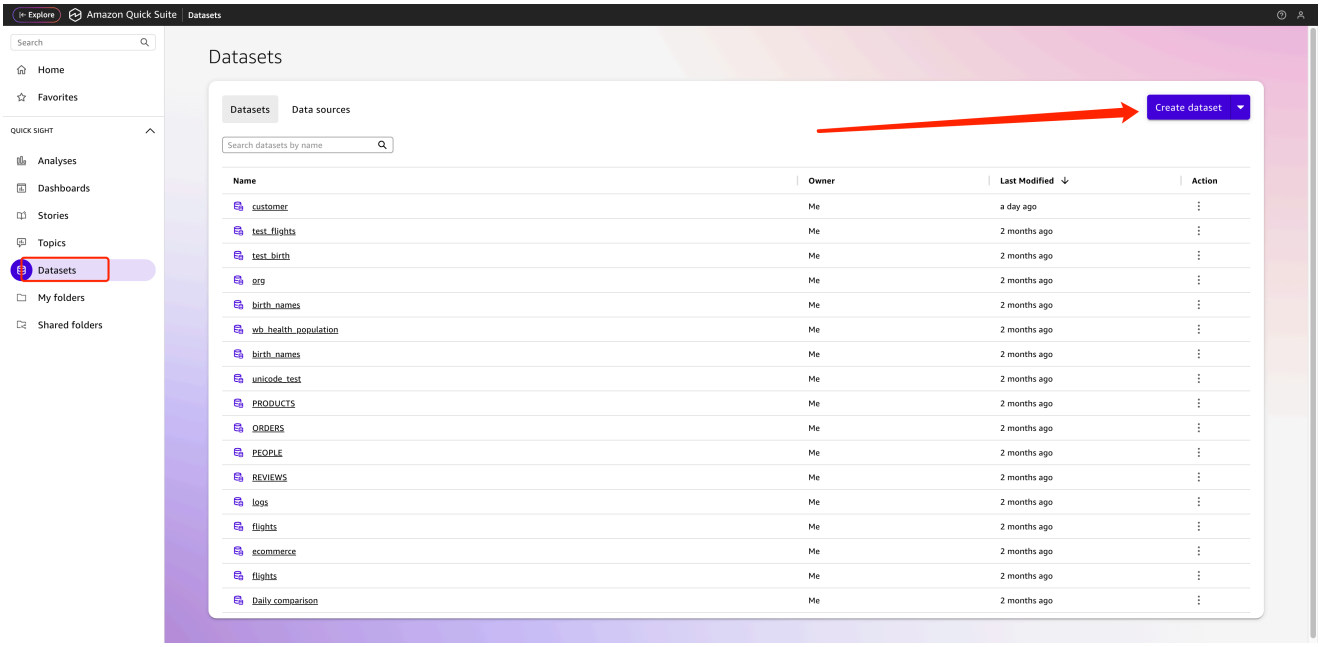
Now that we’ve configured the Doris data source in QuickSight, let’s visualize the data…

Due to Doris's excellent performance in multi-table join scenarios, we chose to design a dashboard based on this scenario. Let's say we need to know order statistics for different countries at different statuses. We will then build the dashboard according to this requirement.

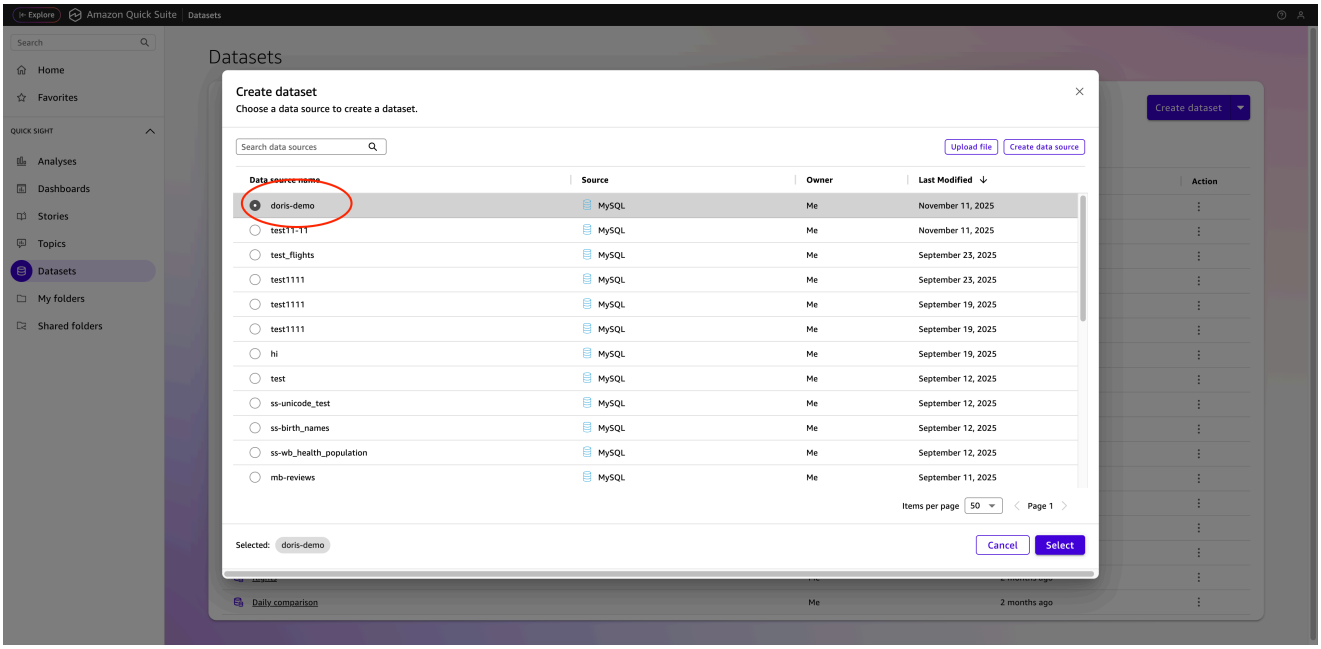
1. Add the following table as a Dataset to the Data source created using the steps above.

- customer
- nation
- orders

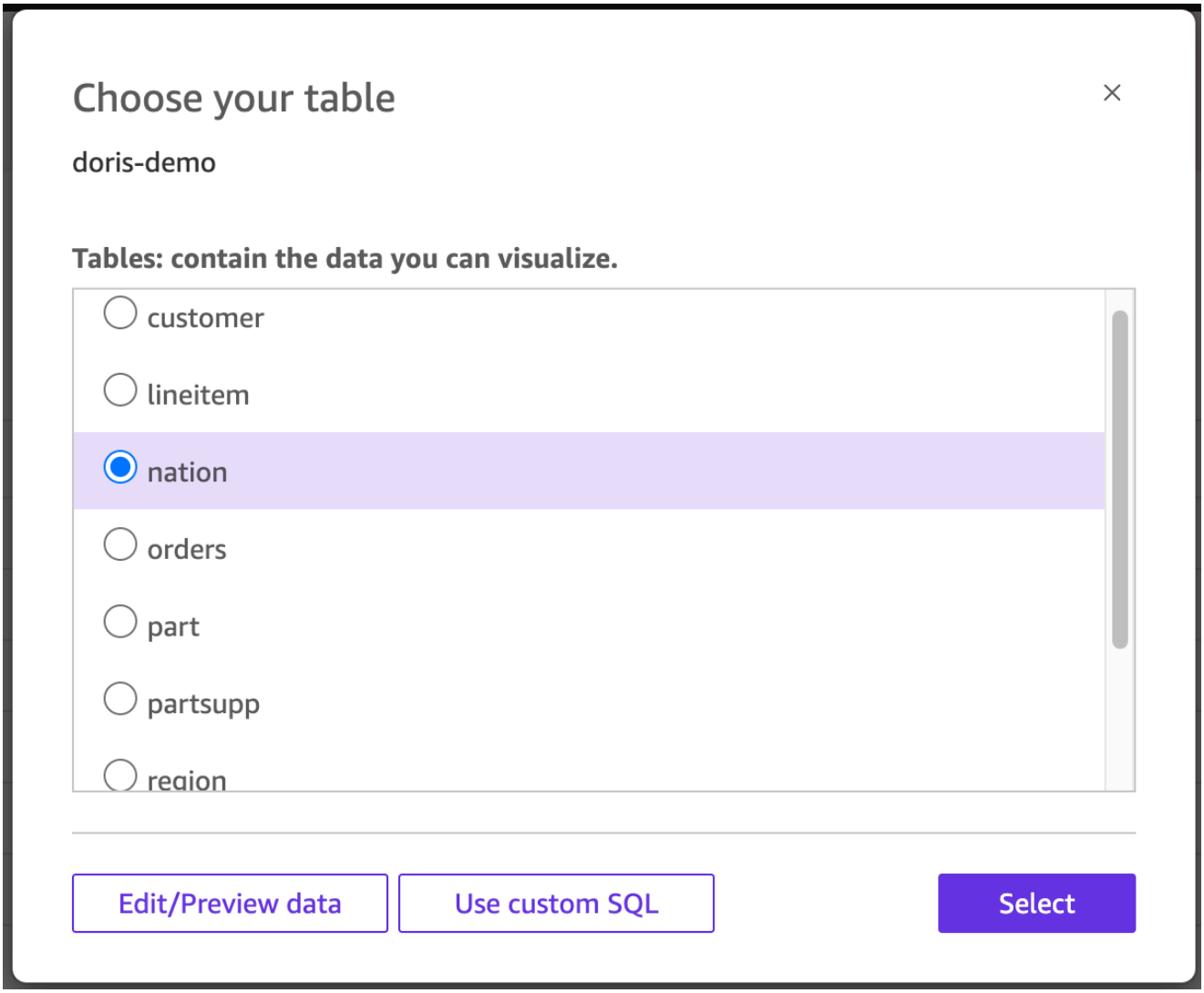
2. Click 'Create Dataset'



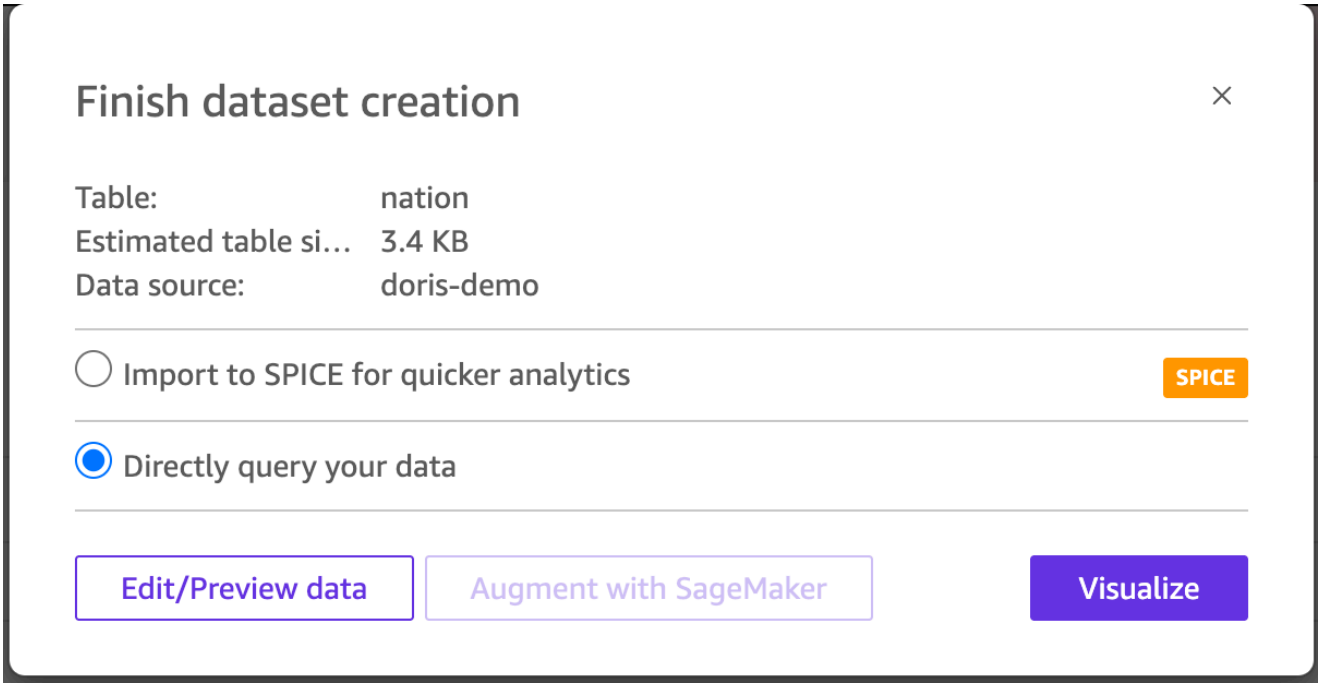
3. Select the data source created in the above steps



4. Select the required tables

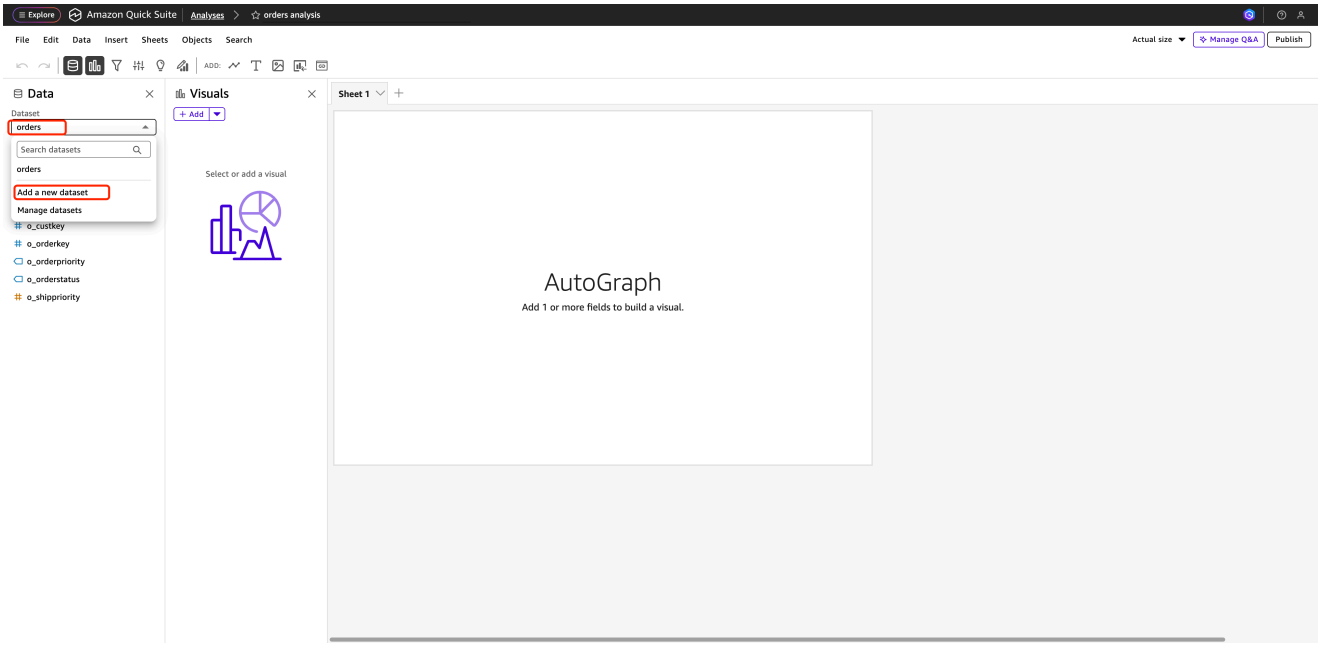


Select Directly Query mode

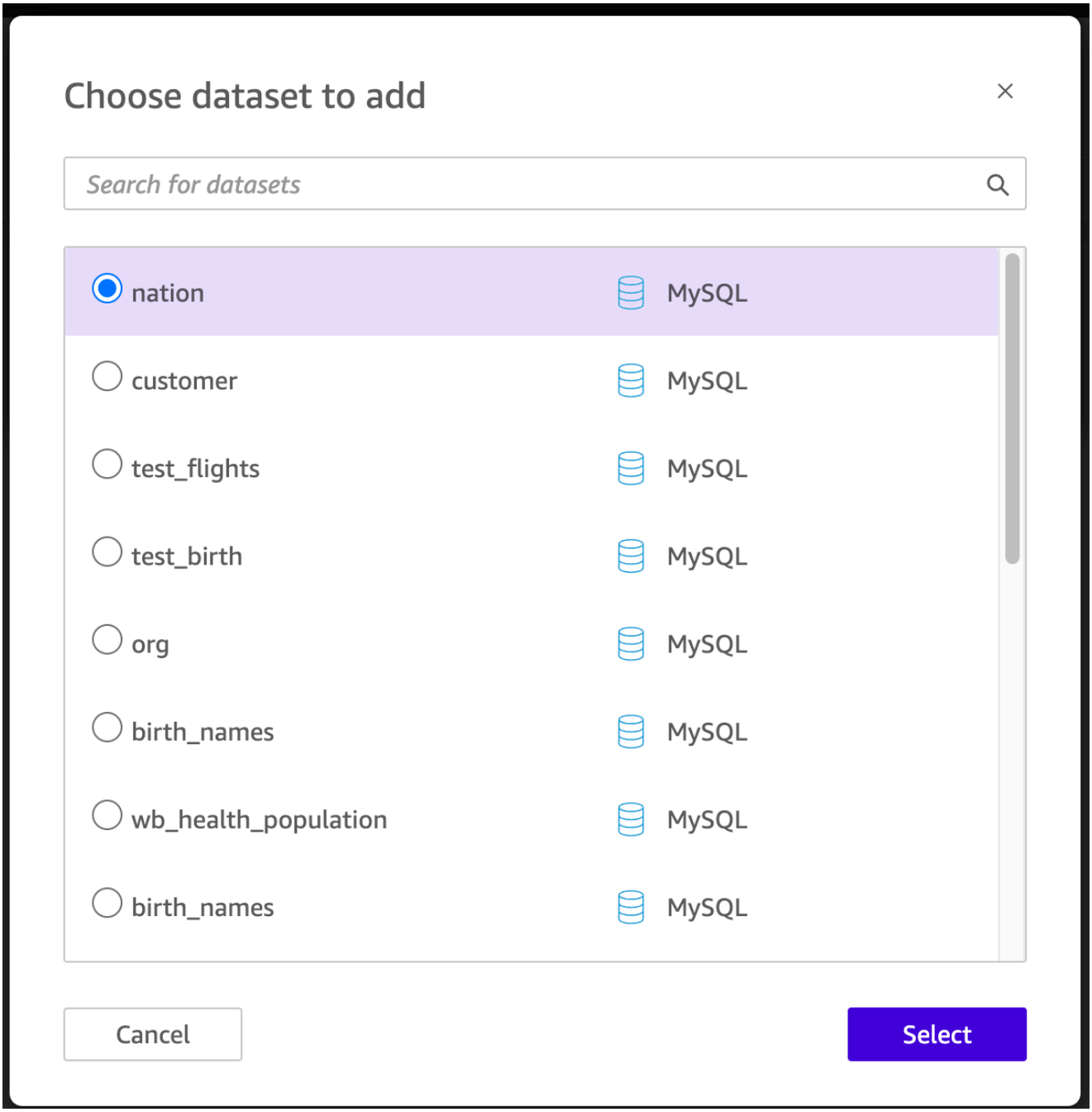


Click ‘Visualize’ to create the data source. Follow these steps to create data sources for other tables as well.

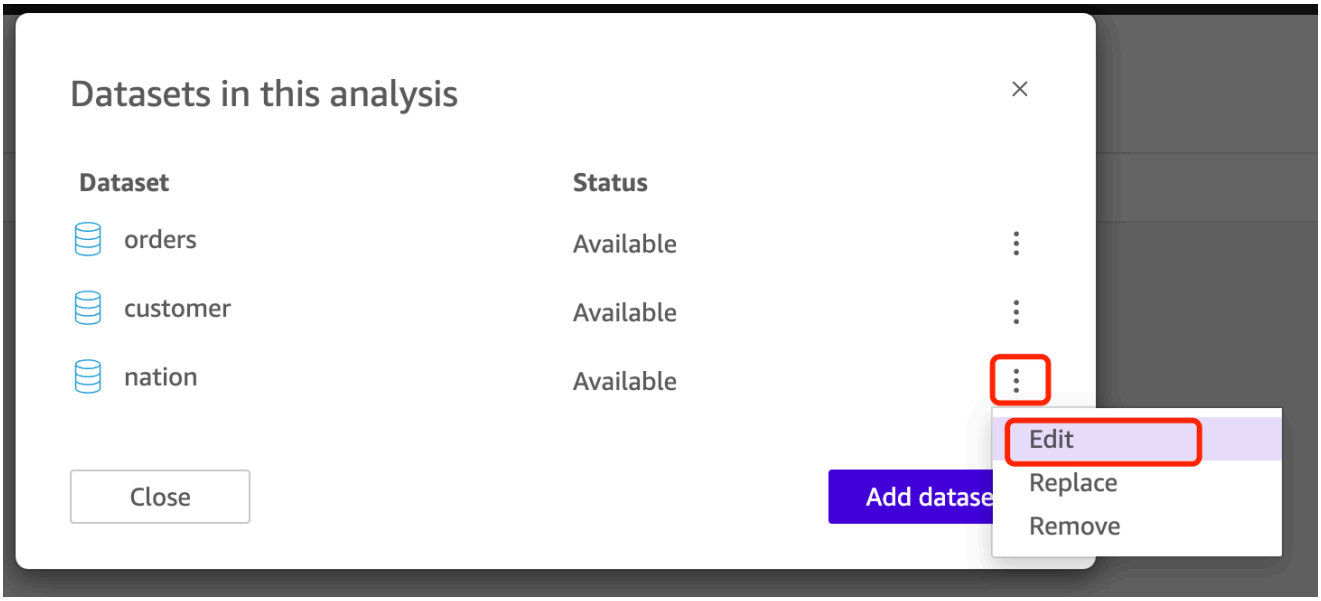
5. Enter the dashboard creation workbench, click the current Dataset dropdown menu, and select Add New Dataset.



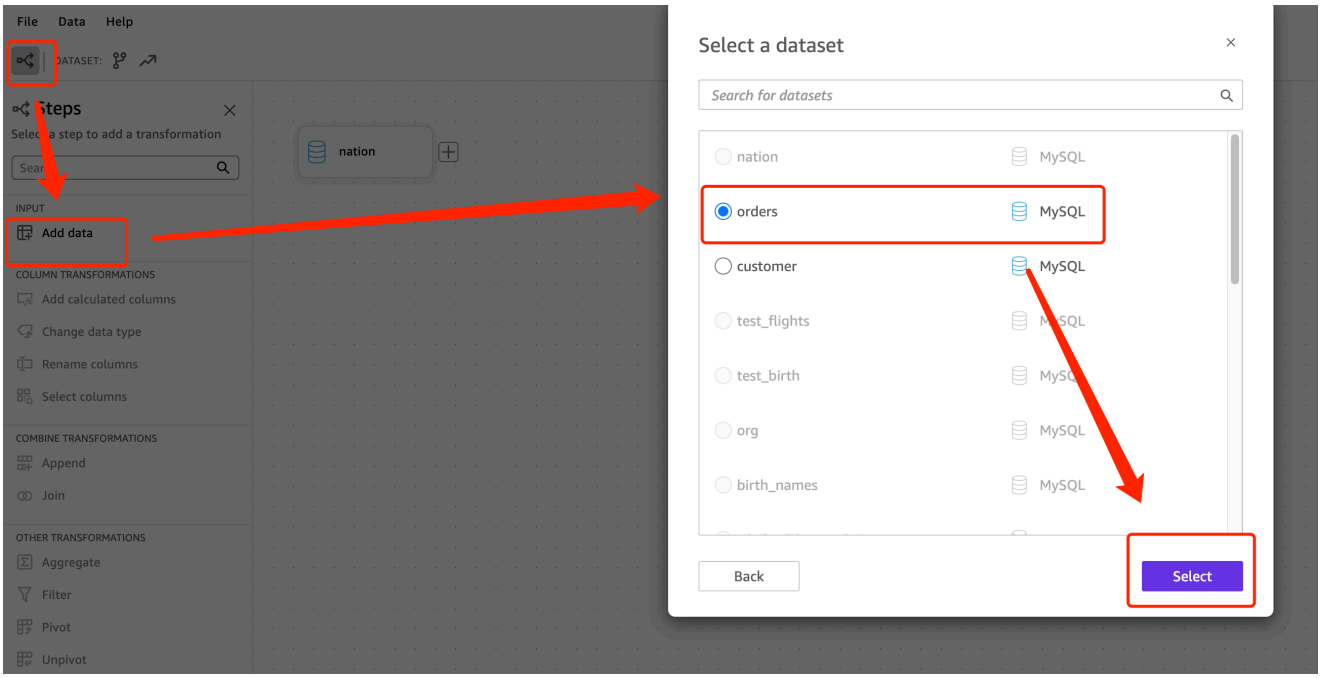
6. Select all datasets in sequence, click Select, and add them to the dashboard.



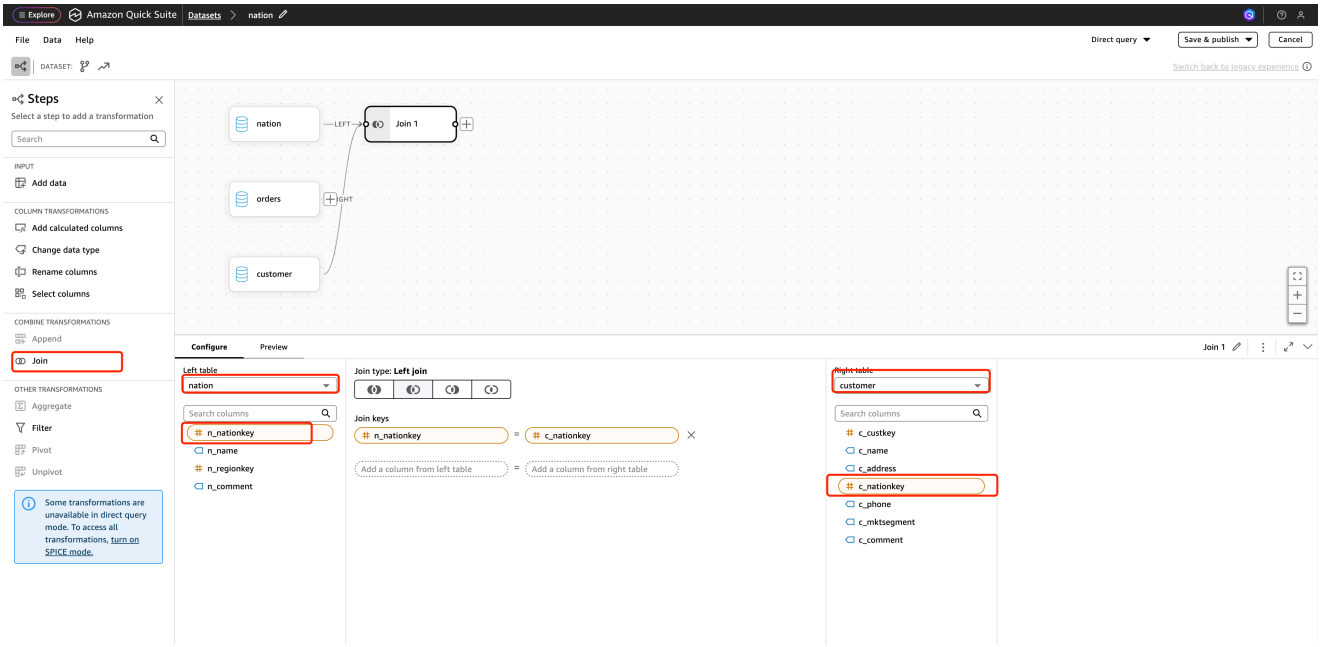
7. After completion, click the nation's operation interface to enter the dataset editing interface. We will now perform column joins on the dataset.



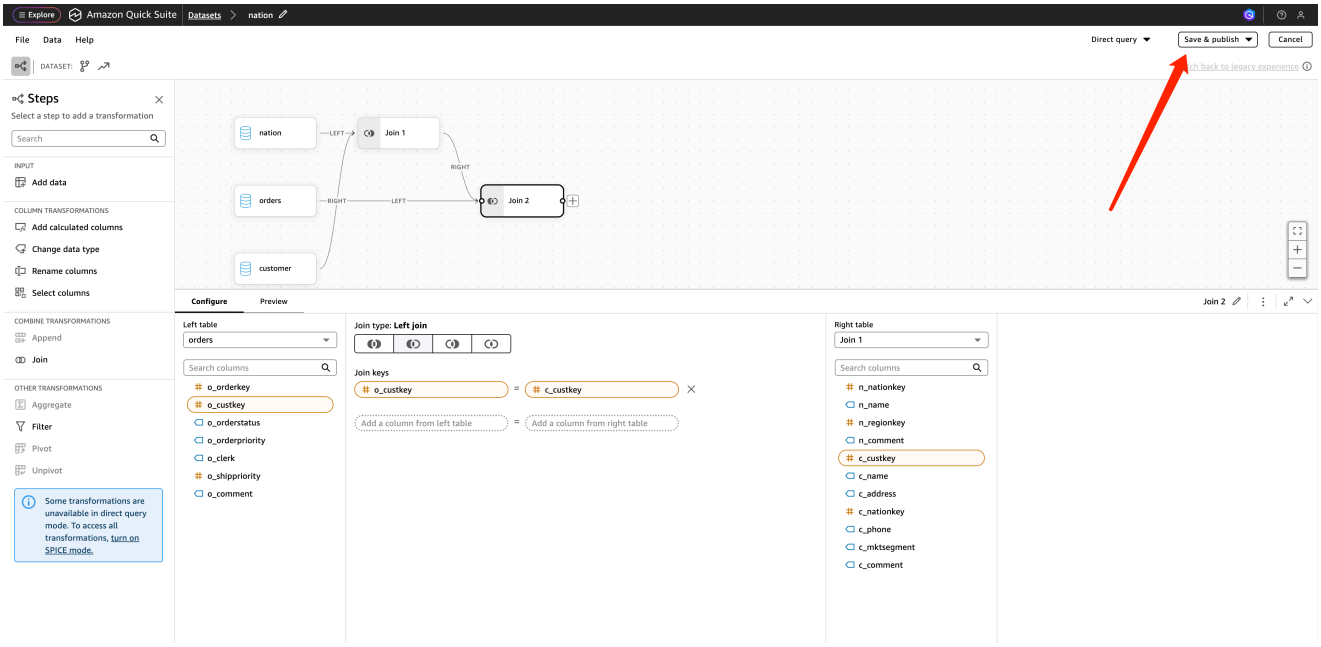
8. As shown in the image, click Add data to add a data source.



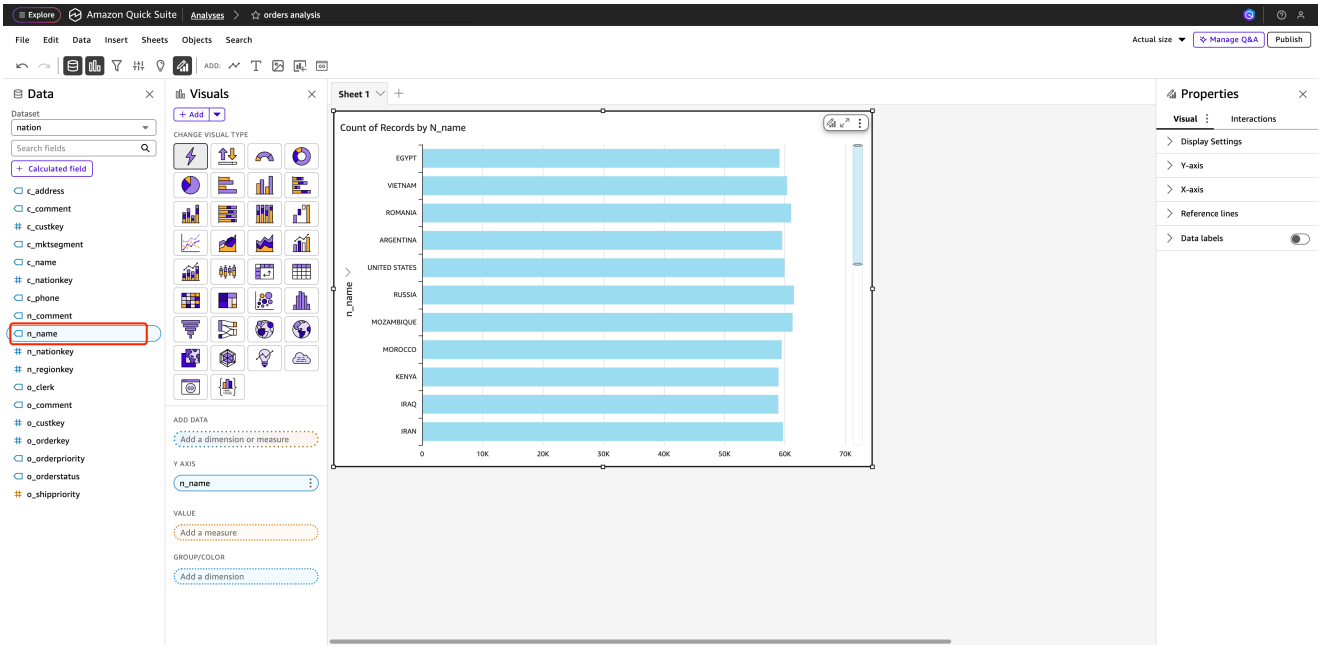
9. After adding the three tables, perform joins. The join relationship is as follows:
    - **customer**：c_nationkey – **nation** : n_nationkey
    - **customer**：c_custkey – **orders** : o_custkey



10. After the join is complete, click Save & Publish in the upper right corner to publish.



11. Return to the Analyses interface where you just added the three data sources, click n_name to display the total number of orders by country name.



12. Click VALUE, select o_orderkey, click GROUP/COLOR, and select o_orderstatus to obtain the demand dashboard.



13. Click Publish in the upper right corner to complete the dashboard publication.

At this point, QuickSight has been successfully connected to Apache Doris, and data analysis and visualization dashboard creation have been implemented.

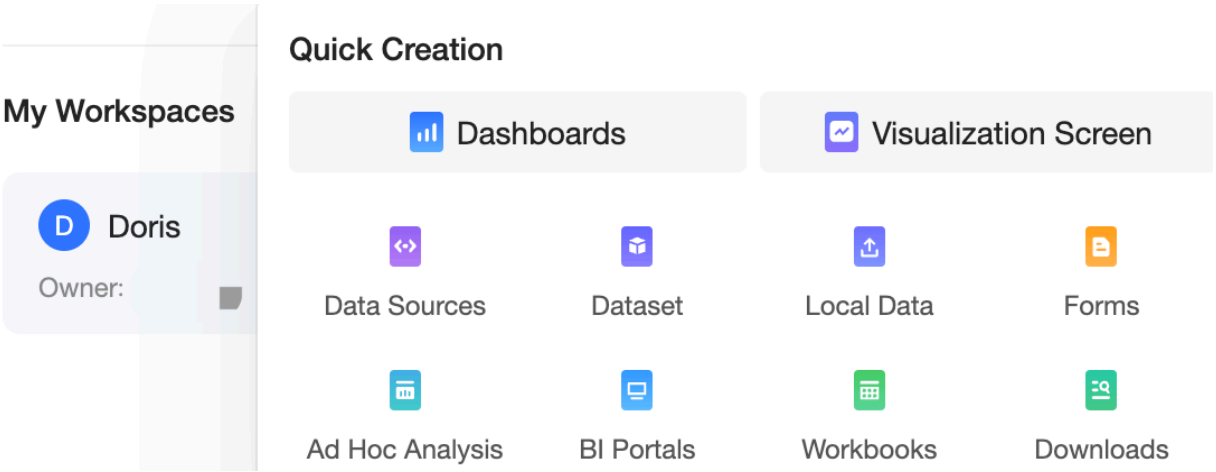
Quick BI

Introduction

Quick BI is a data warehouse-based business intelligence tool that helps enterprises set up impressive visual analyses quickly. Quick BI supports a variety of data sources, including databases like MySQL, Oracle, SQL Server, and Apache Doris, as well as file formats such as Excel, CSV, and JSON. It offers a wealth of visualization components, such as tables, charts, and maps, allowing users to easily achieve data visualization through simple drag-and-drop operations.

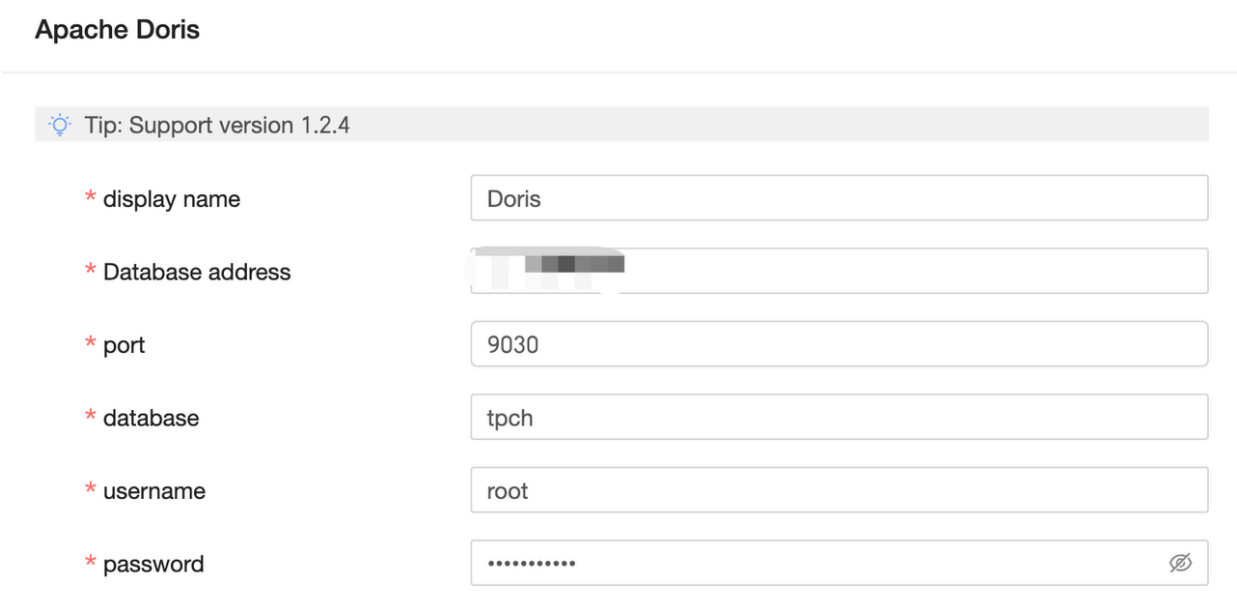
Data connection and application

1. Login Quick BI and create a workspace.
2. Click Data Source under the current workspace.



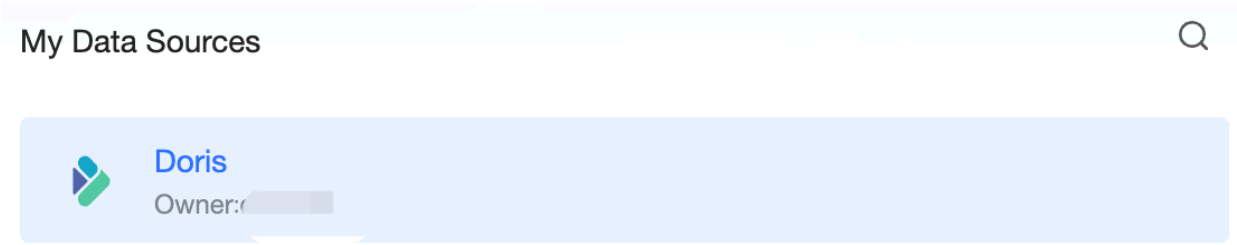
create workspace

3. Select Apache Doris in the already created data source and fill in the corresponding Apache Doris connection information.



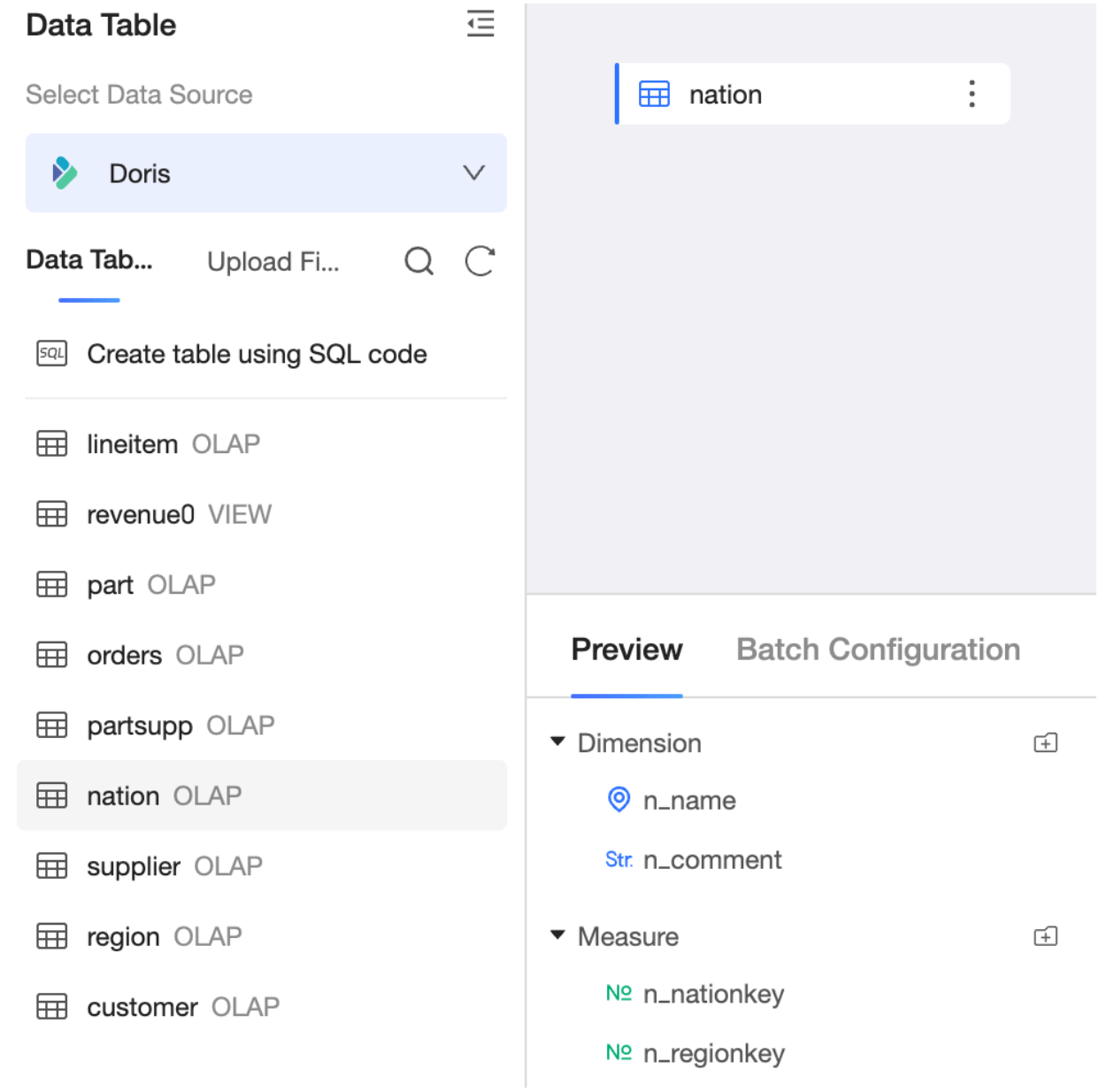
Doris information

4. Once the connection is successful, you can see the data source we created in the data source list.



data source

5. Create a dataset in the data source we created, using the TPC-H dataset as an example. After the dataset is created, you can set the corresponding report.


Data Table
Select Data Source
Doris
Data Tab...
Upload Fi...
Create table using SQL code
lineitem OLAP
revenue0 VIEW
part OLAP
orders OLAP
partsupp OLAP
nation OLAP
supplier OLAP
region OLAP
customer OLAP
nation
Preview
Batch Configuration
Dimension
n_name
n_comment
Measure
n_nationkey
n_regionkey

Doris table

Smartbi

Smartbi Introduction

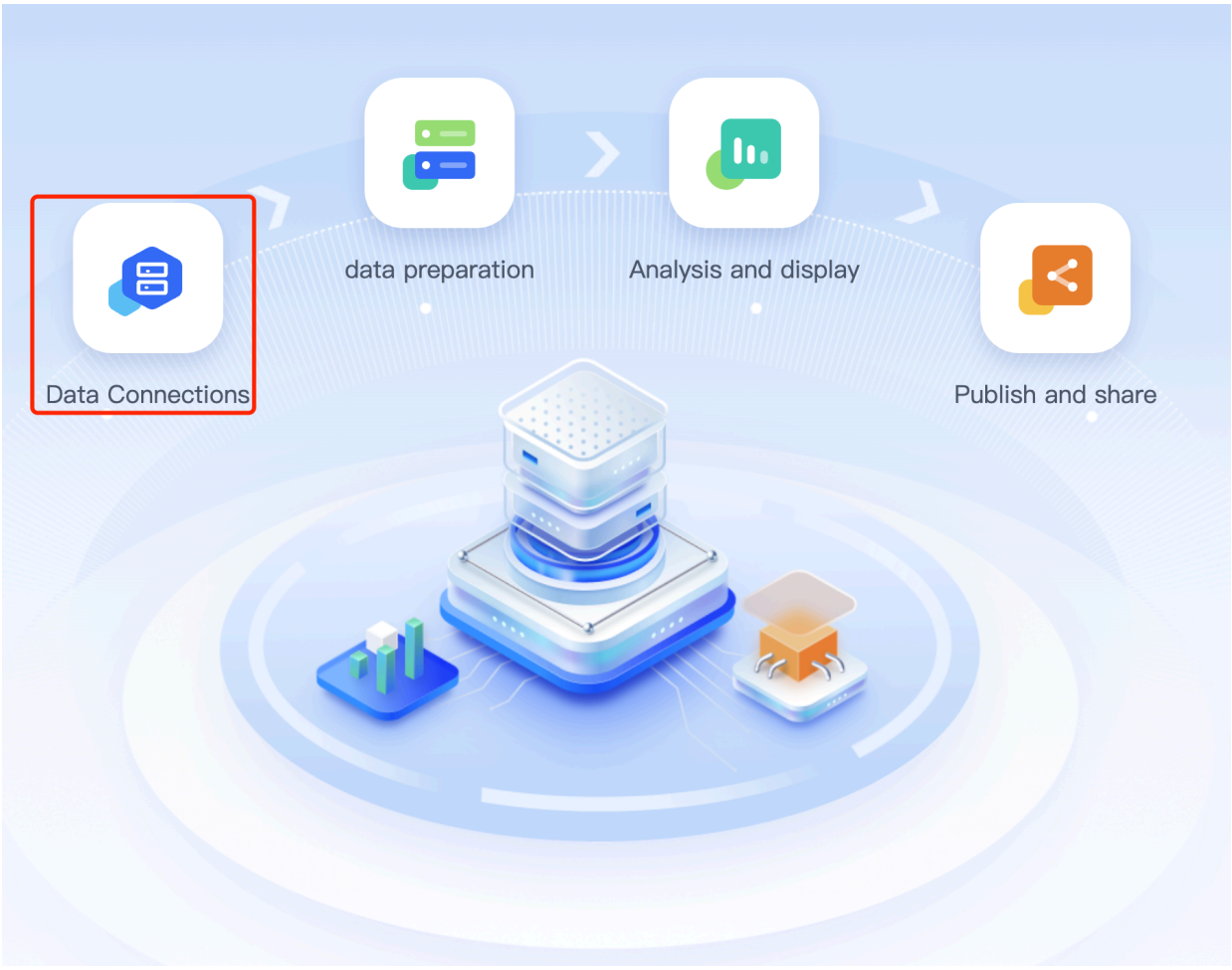
Smartbi is a collection of software services and application connectors that can connect to a variety of data sources, including Oracle, SQL Server, MySQL, and Doris, enabling users to integrate and cleanse their data easily. With Smartbi's data modeling capabilities, users can not only create complex relational models but also write data analysis expressions and establish data relationships, laying the foundation for advanced data analysis and visualization. SmartBI offers a diverse range of visualization options, including various types of charts, geographic maps, interactive dashboards, and customizable visualization tools. These powerful tools help users understand and present their data more intuitively and comprehensively, enhancing the effectiveness and efficiency of data analysis.

Precondition

you can visit https://www.smartbi.com.cn to download and install Smartbi.

Data connection and application

1. Log into Smartbi and click Data Connections.



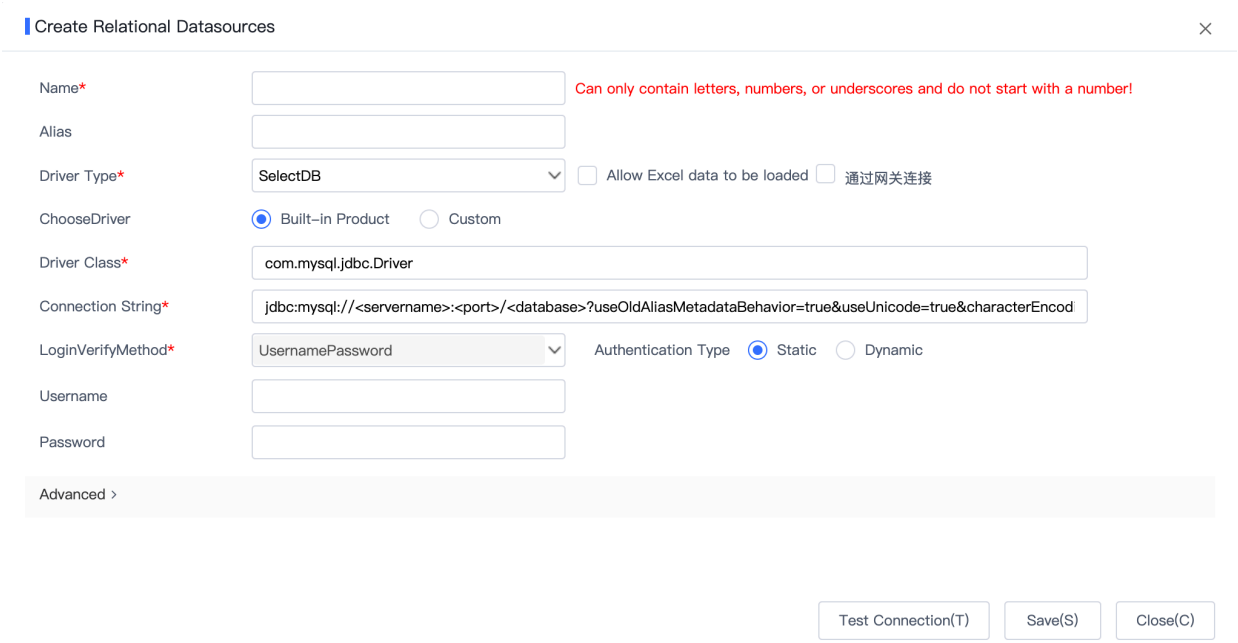
main page

2. After clicking the data connections, you can find a list of databases to be connected with and select the Doris database.



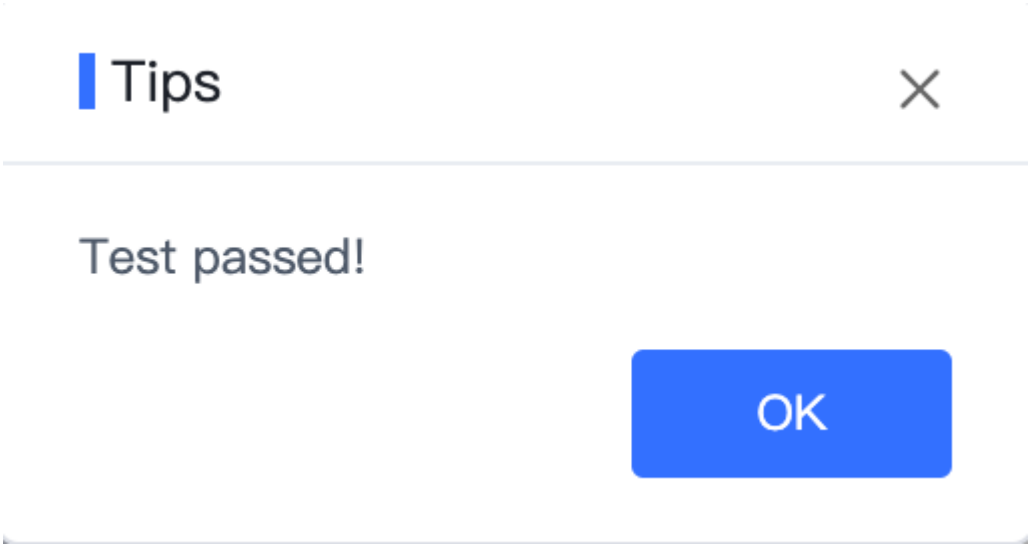
selectdb

3. After selecting the database, fill in the connection information of the Doris database



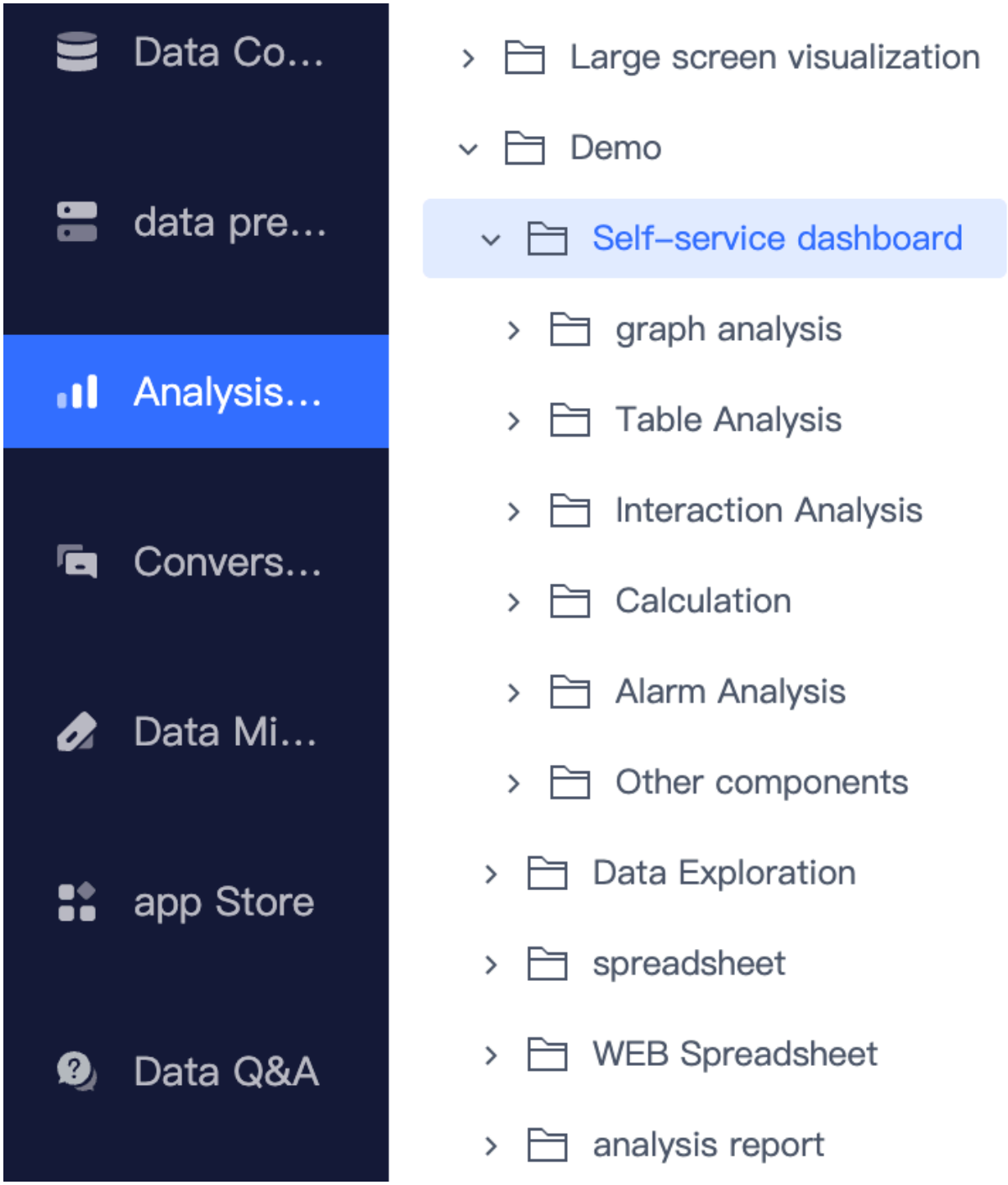
data source connection

4. If the information is correct and the network is connected, a message is displayed indicating that the Test passed.



test passed

5. After the connection is successful, you can customize and set the required report information in the data analysis module Self-service dashboard.



data analysis

CloudDM

Introduction

CloudDM, developed by Clougence, is a powerful cross-platform database management tool designed for both teams and individual users.

It provides dedicated support for key features of Apache Doris, offering capabilities such as data access, data masking, visual editing, and database CI/CD workflows.

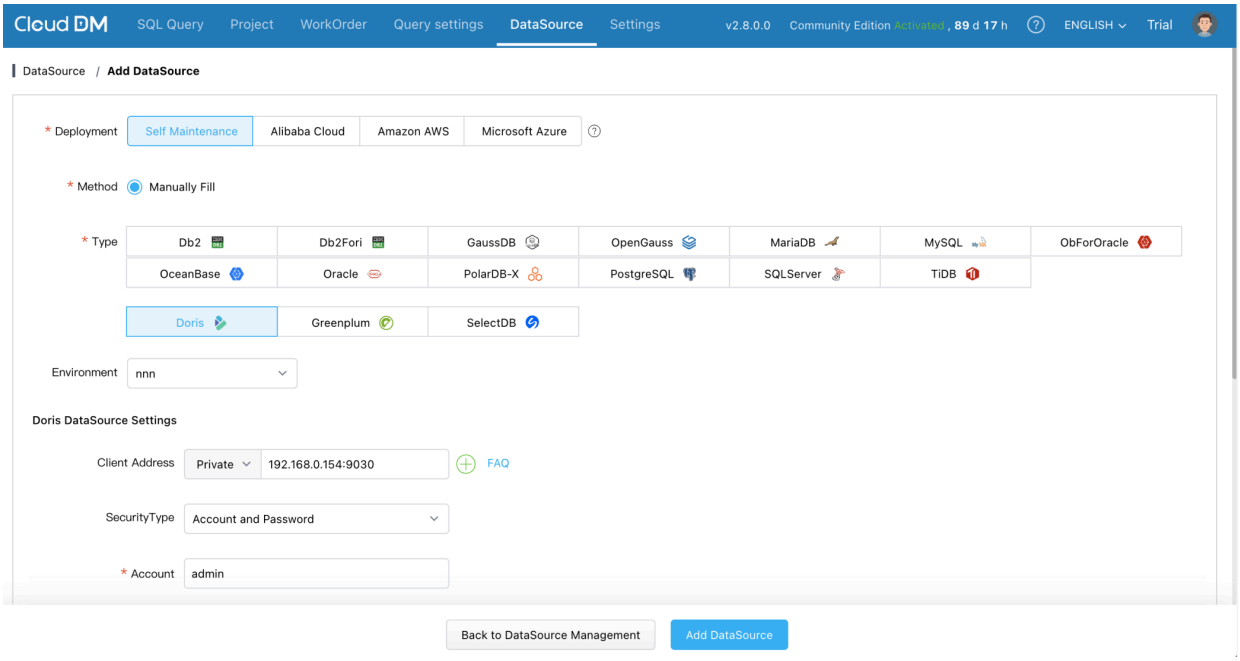
Preconditions

Install CloudDM. You can download and install CloudDM from https://www.cdmgr.com/.

Add data source

> **Note**
>
> CloudDM version should be 2.8.0.0 or above.

1. Log in to CloudDM.
2. Click **DataSources** > **Add DataSources**.
3. Select **Doris** for the Type.



add datasource

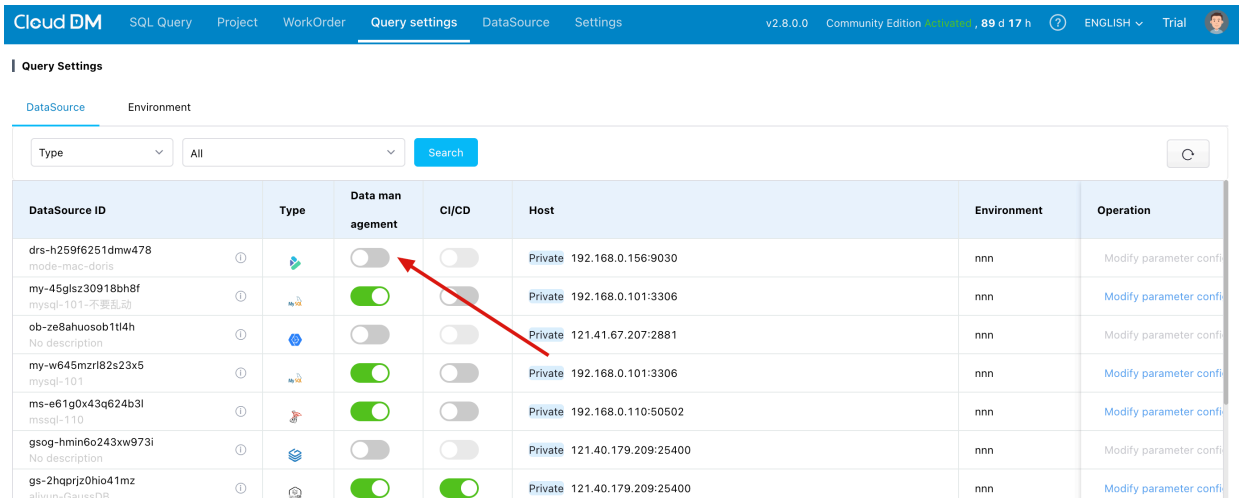
4. Enter the following required information to connect to the Doris instance.

- Client Address: The FE query port of the Doris cluster, for example: hostID:9030.
- Account: Username used to log in to the Doris cluster, for example: admin.
- Password: Password used to log in to the Doris cluster.

:::tip CloudDM can manage both internal catalogs and external catalogs in Doris. :::

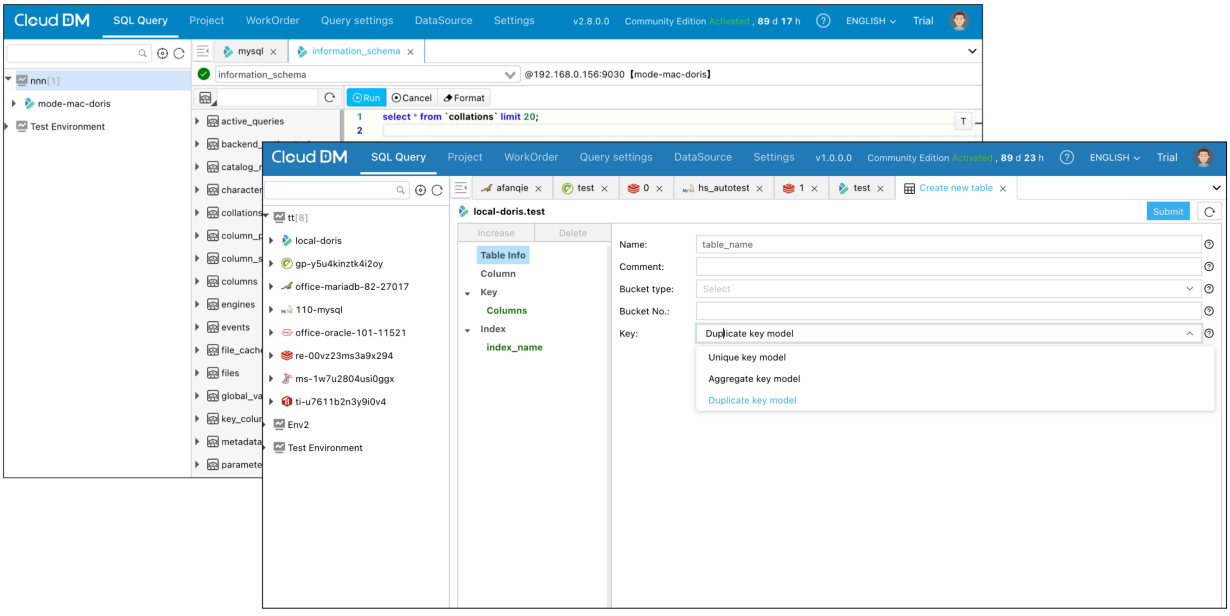
:::info Note Managing the external catalog connected to Doris through the Database form of catalog.db requires Doris version 2.1.0 and above. :::

5. Click **Query Settings** in the top navigation bar. Then enable **Data Management**.



enable data manage

6. Access data.
   In the database connection navigation panel on the left, you can view the added Doris connection. Then you can start to manage the data through **CloudDM**.



connect to data source

Function support

- Query Client

    - Visually manage database objects in Doris
    - Write and execute SQL in the console
    - Export query results
- Team Collaboration
    - Statement-level authorization with table-level granularity
    - Workflow and approval for SQL requests
    - Database CI/CD
    - Data masking for sensitive information
    - SQL review rules

DBeaver

introduce

DBeaver is a cross-platform database tool for developers, database administrators, analysts and anyone who works with data.

Apache Doris is highly compatible with the MySQL protocol. You can use DBeaver's MySQL driver to connect to Apache Doris and query data in the internal catalog and external catalog.

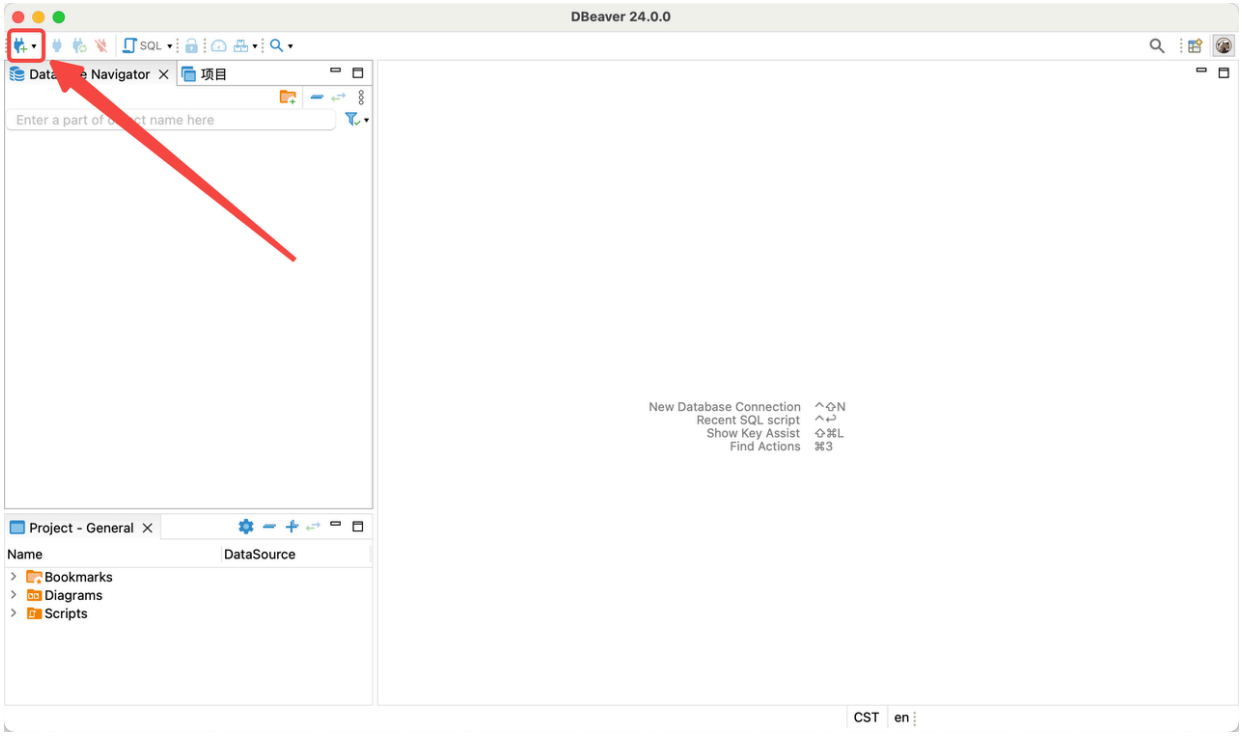
Preconditions

Dbeaver installed You can visit https://dbeaver.com to download and install DBeaver

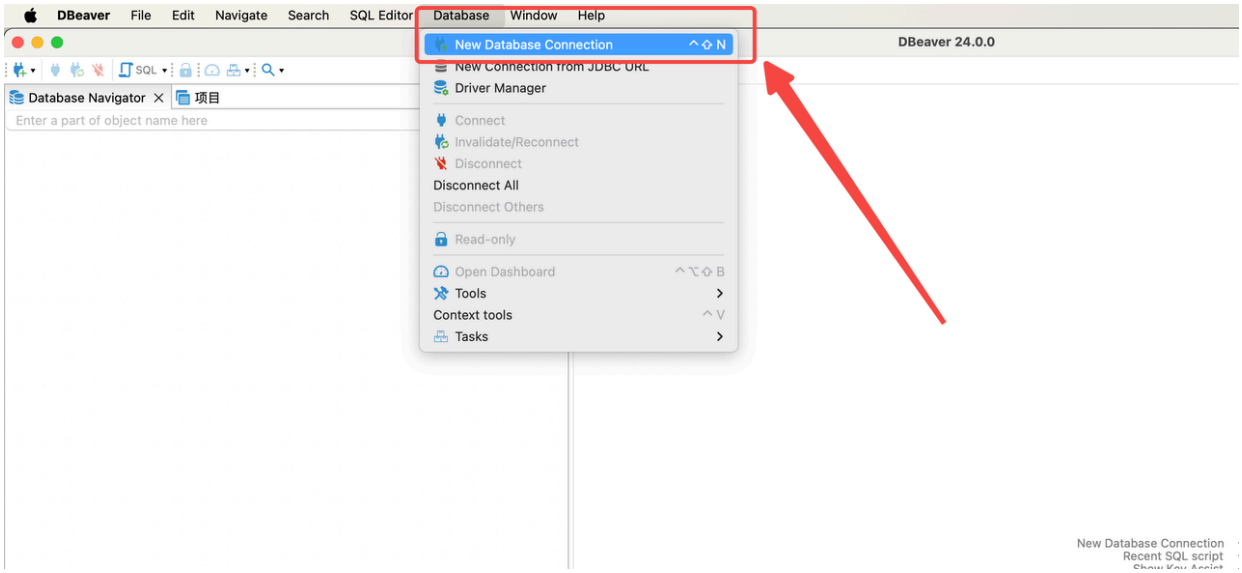
Add data source

> **Note**
>
> Currently verified using DBeaver version 24.0.0

1. Start DBeaver
2. Click the plus sign (**+**) icon in the upper left corner of the DBeaver window, or select **Database > New Database Connection** in the menu bar to open the **Connect to a database** interface.



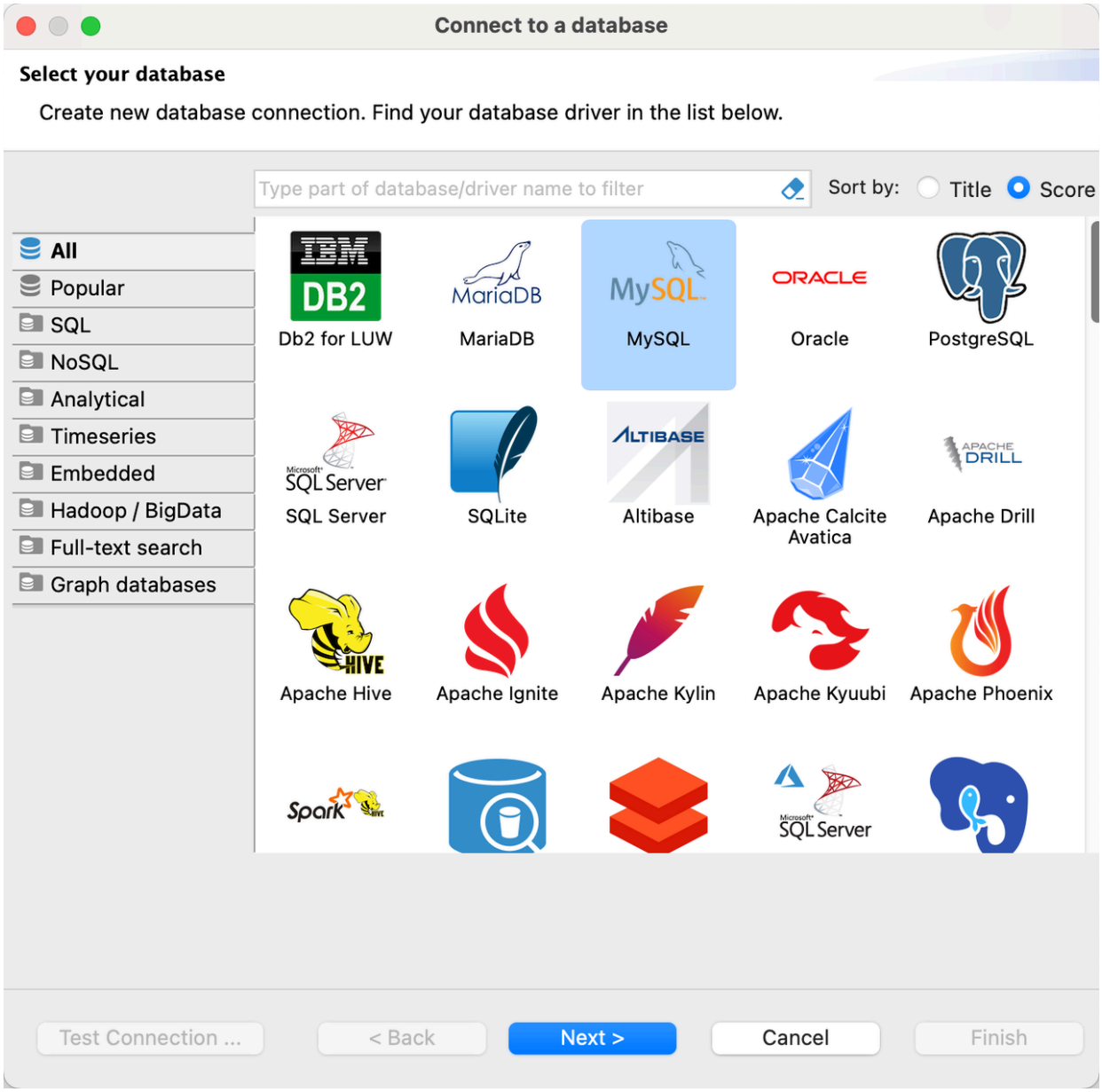
add connection 1



add connection 2

3. Select the MySQL driver

   In the **Select your database** window, select **MySQL**.



chose driver

4. Configure Doris connection

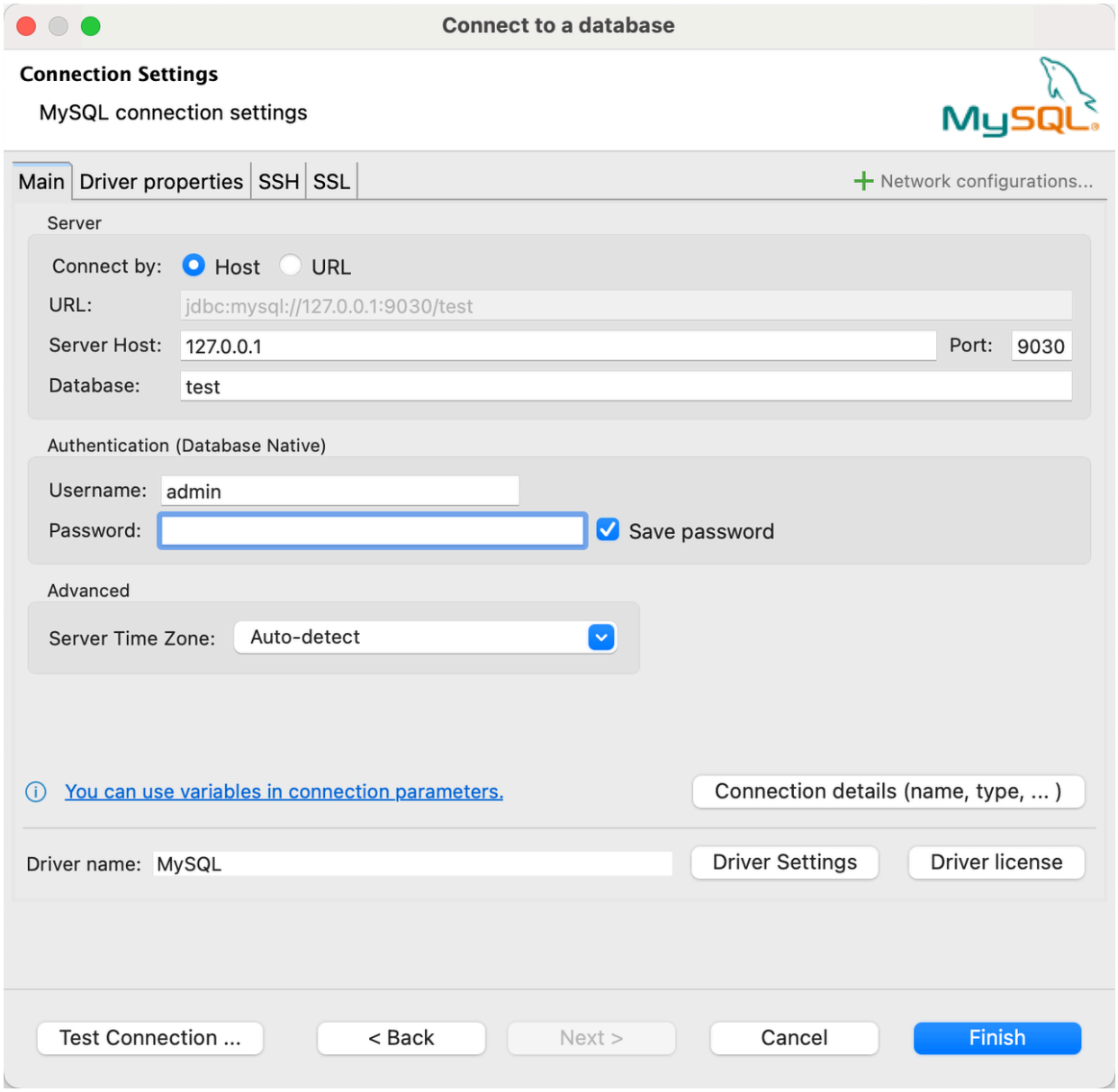
   In the **main** tab of the **Connection Settings** window, configure the following connection information:

- Server Host: FE host IP address of the Doris cluster.
- Port: FE query port of Doris cluster, such as 9030.
- Database: The target database in the Doris cluster.
- Username: The username used to log in to the Doris cluster, such as admin.
- Password: User password used to log in to the Doris cluster.

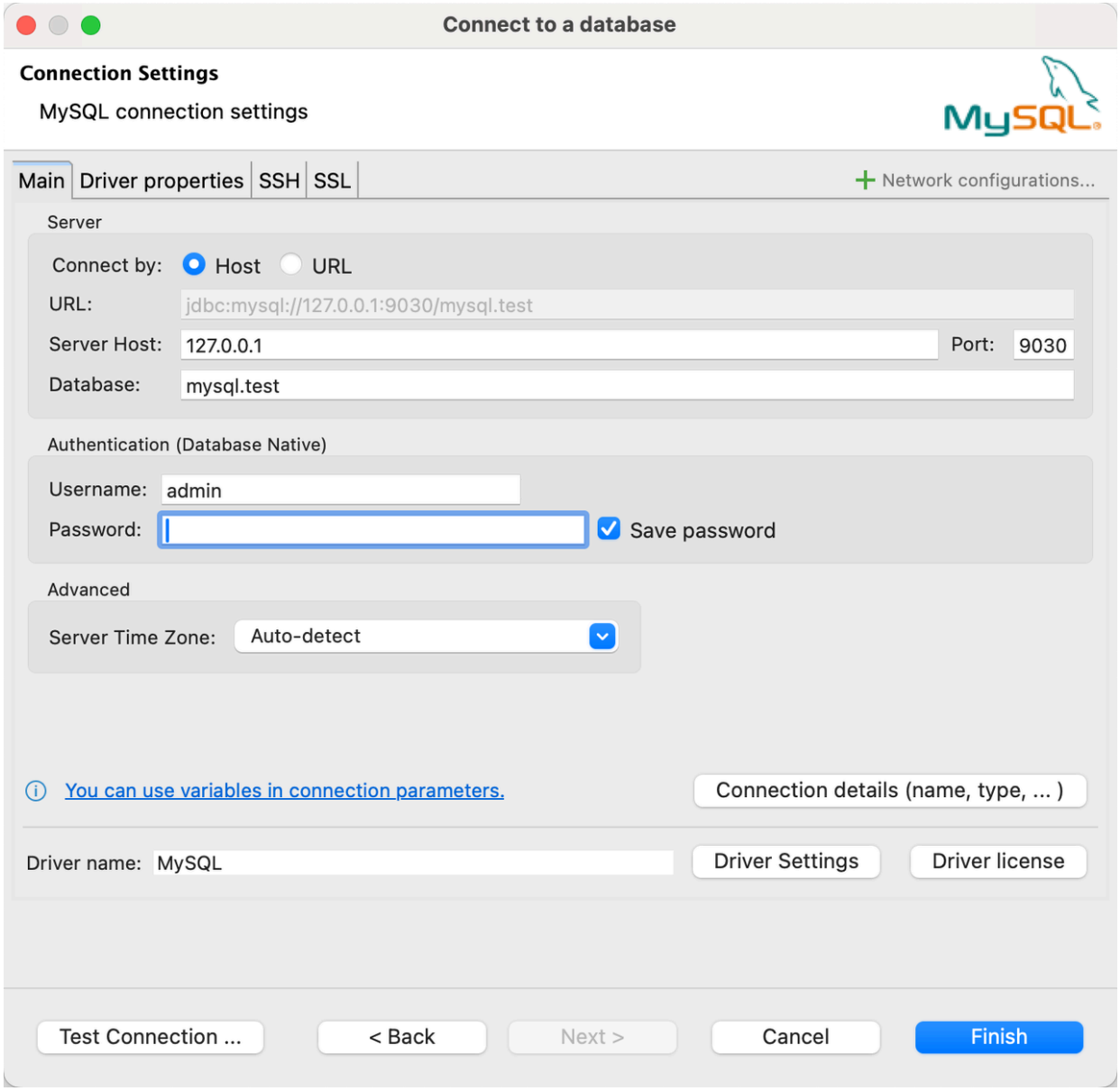
:::tip Database can be used to distinguish between internal catalog and external catalog. If only the Database name is filled in, the current data source will be connected to the internal catalog by default. If the format is catalog.db, the current data source will be connected to the catalog filled in Database by default, as shown in DBeaver The database tables are also database tables in the connected catalog, so you can use DBeaver's MySQL driver to create multiple Doris data sources to manage different Catalogs in Doris. :::

:::info Note Managing the external catalog connected to Doris through the Database form of catalog.db requires Doris version 2.1.0 and above. :::

- internal catalog



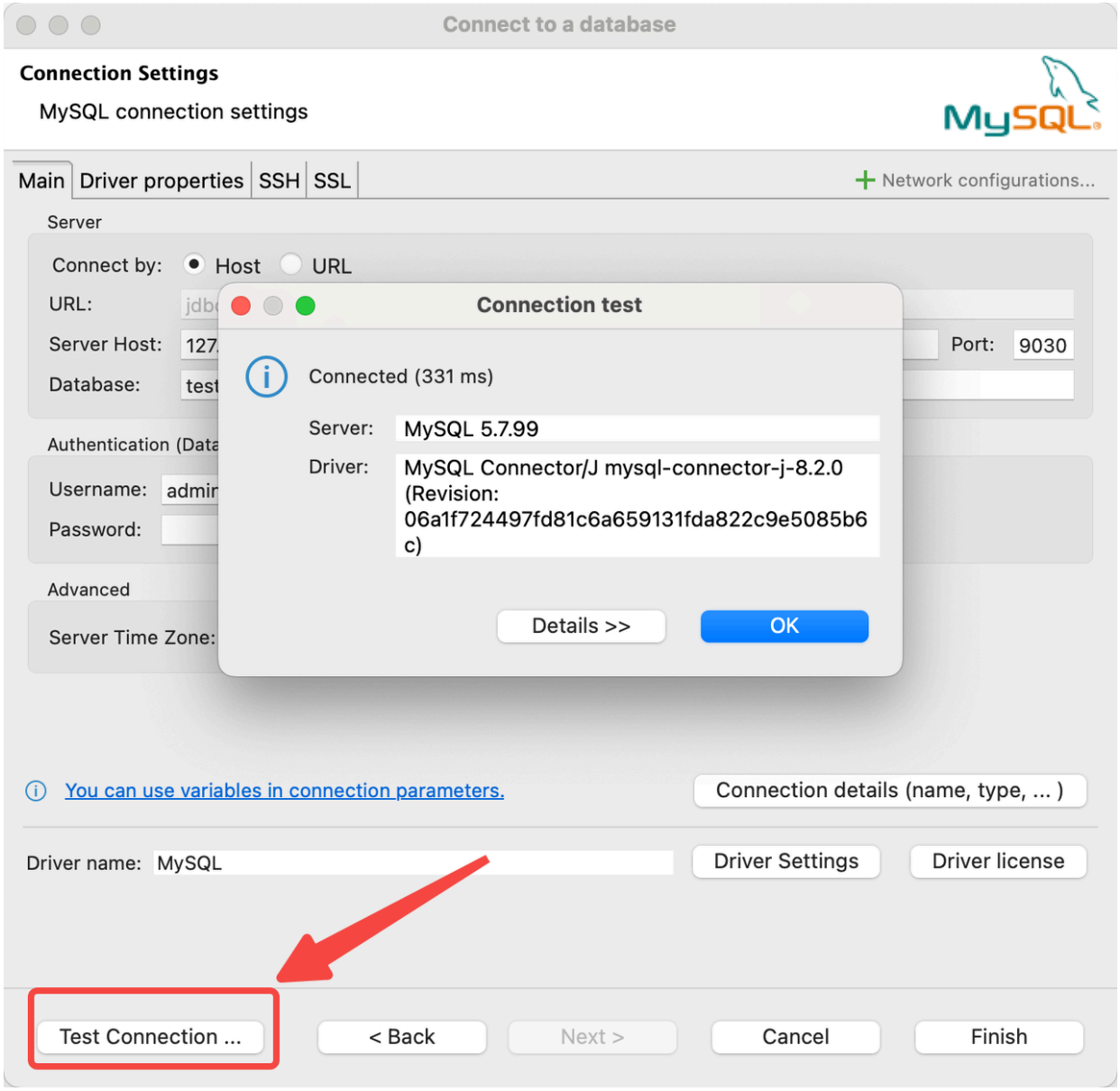
- external catalog



5. Test data source connection

   After filling in the connection information, click Test Connection in the lower left corner to verify the accuracy of the database connection information. DBeaver returns to the following dialog box to confirm the configuration of the connection information. Click OK to confirm that the configured connection information is correct. Then click Finish in the lower right corner to complete the connection

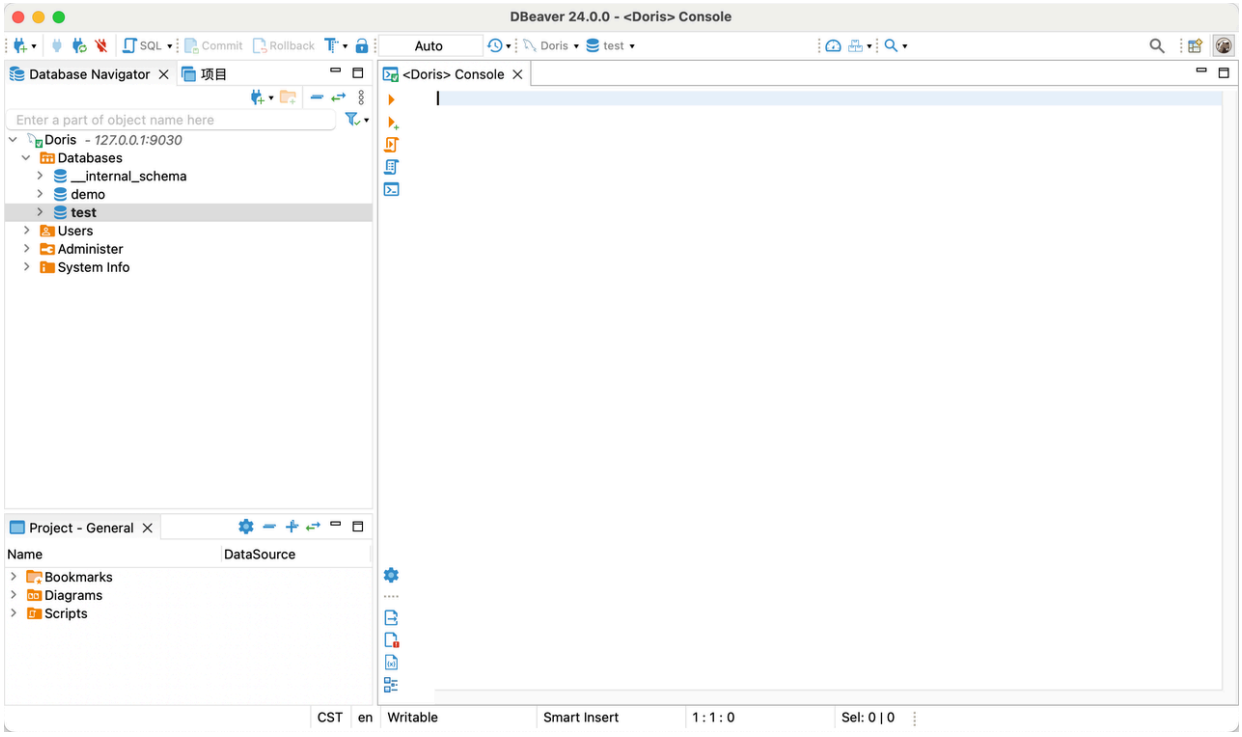
configuration.



6. Connect to database

   After the database connection is established, you can see the created data source connection in the database connection navigation on the left, and you can connect and manage the database through

DBeaver.



Function support

- fully support
    - Visual viewing class
        - Databases
            - Tables
            - Views
        - Users
            - Administer
        - Session Manager
        - System Info
            - Session Variables
            - Global Variables
            - Engines
            - Charsets
            - User Priviages
            - Plugin
        - Operation class
            - SQL editor
            - SQL console
- basic support

  The basic support part means that you can click to view without error, but due to protocol compatibility issues, there may be incomplete display.

    - Visual viewing class

        - dash board
        - Users/user/properties
        - Session Status
        - Global Status

- not support

  The unsupported part means that when using DBeaver to manage Doris, errors may be reported when performing certain visual operations, or some visual operations are not verified. Such as visual creation of database tables, schema change, addition, deletion and modification of data, etc.

DataGrip

introduce

DataGrip is a powerful cross-platform database tool for relational and NoSQL databases from JetBrains.

Apache Doris is highly compatible with the MySQL protocol. You can use DataGrip's MySQL data source to connect to Apache Doris and query data in the internal catalog and external catalog.

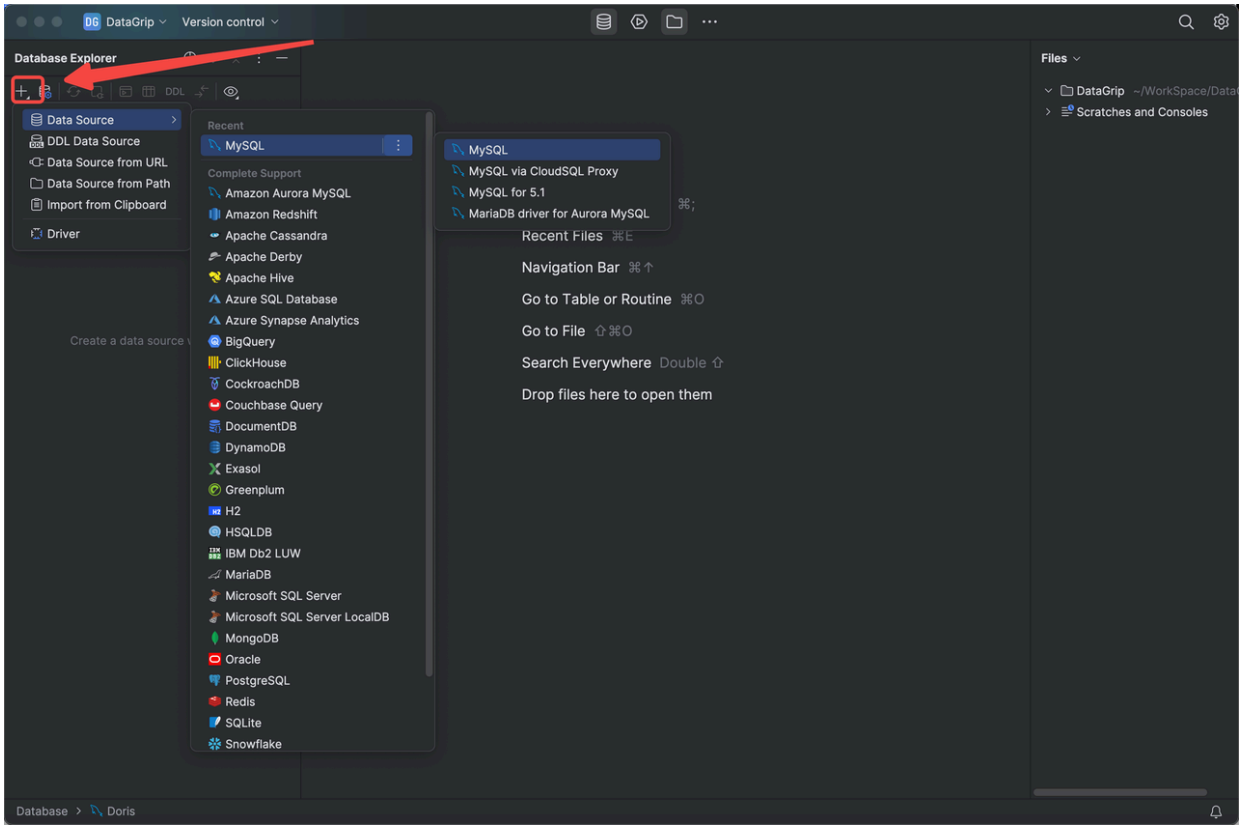
Preconditions

DataGrip installed You can visit www.jetbrains.com/datagrip/ to download and install DataGrip

Add data source

> **Note**
>
> Currently verified using DataGrip version 2023.3.4

1. Start DataGrip
2. Click the plus sign (**+**) icon in the upper left corner of the DataGrip window and select the MySQL data source



add data source

3. Configure Doris connection

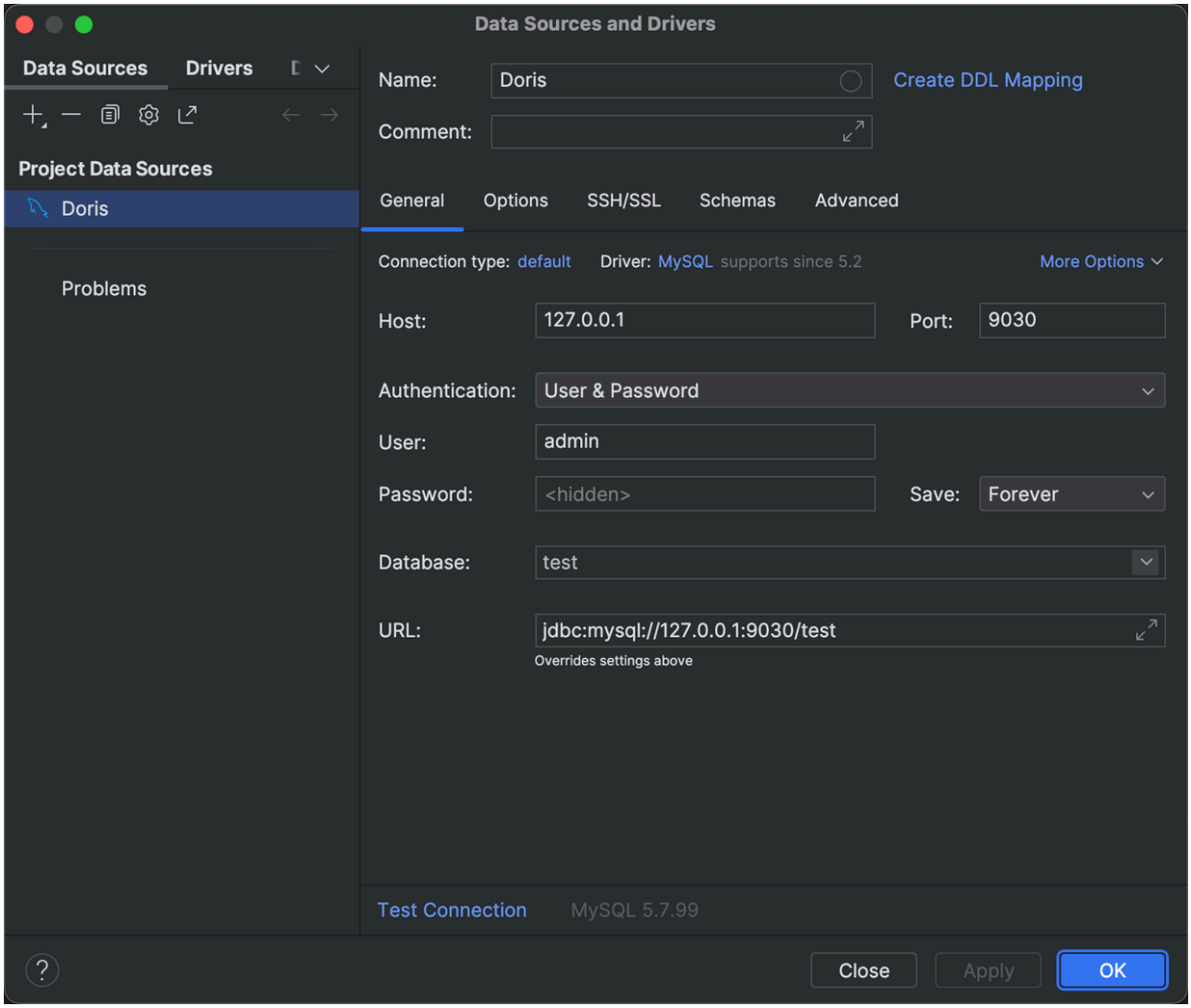
   On the General tab of the Data Sources and Drivers window, configure the following connection information:

- Host: FE host IP address of the Doris cluster.
- Port: FE query port of Doris cluster, such as 9030.
- Database: The target database in the Doris cluster.
- User: User name used to log in to the Doris cluster, such as admin.
- Password: User password used to log in to the Doris cluster.

  Database can be used to distinguish between internal catalog and external catalog. If only the Database name is filled in, the current data source will be connected to the internal catalog by default. If the format is catalog.db, the current data source will be connected to the catalog filled in Database by default, as shown in DataGrip The database table is also a database table in the connected catalog. In this way, you can use DataGrip's MySQL data source to create multiple Doris data sources to manage different Catalogs in Doris.

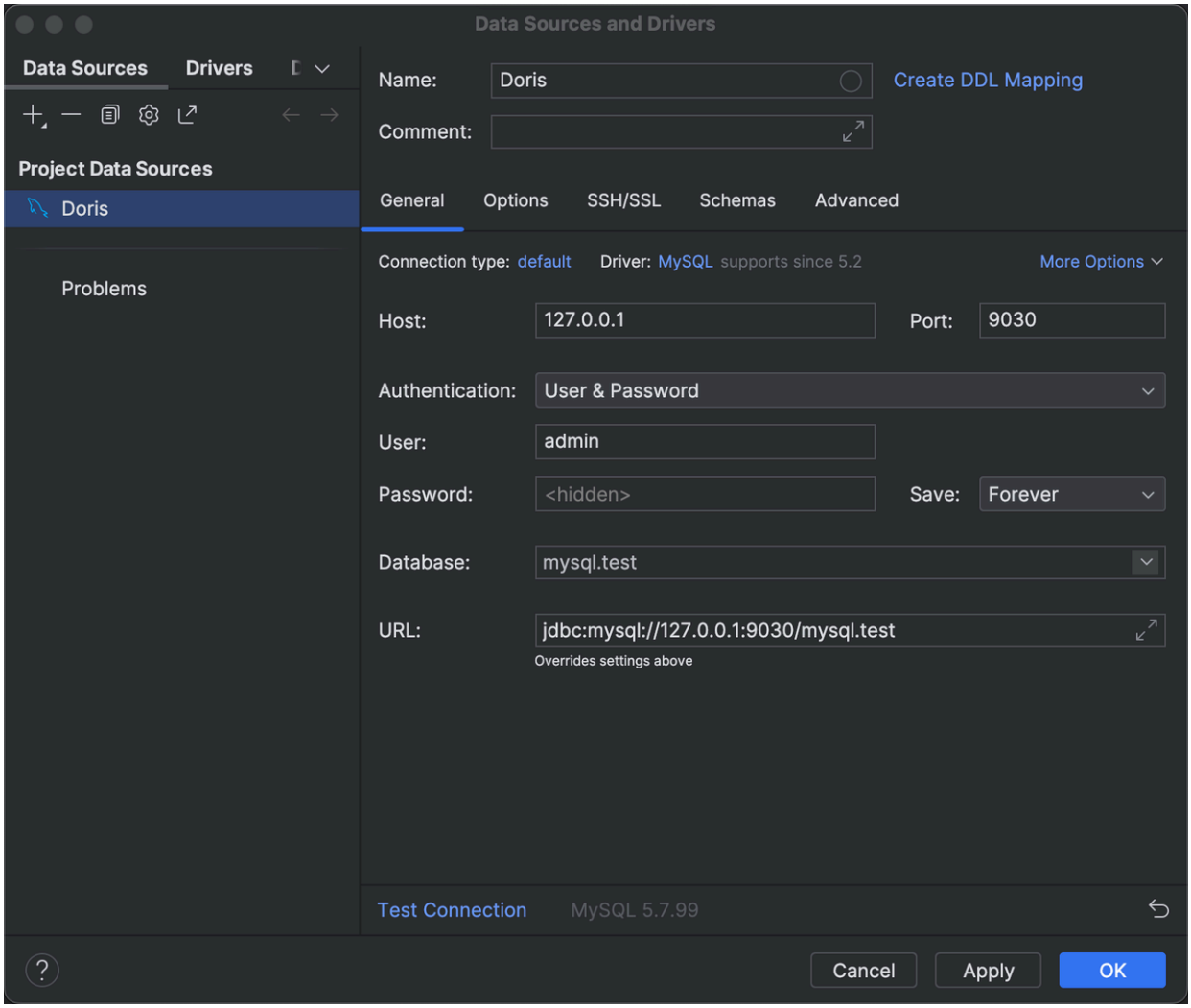
  :::info Note Managing the external catalog connected to Doris through the Database form of catalog.db requires Doris version 2.1.0 and above. :::

- internal catalog



connect internal catalog

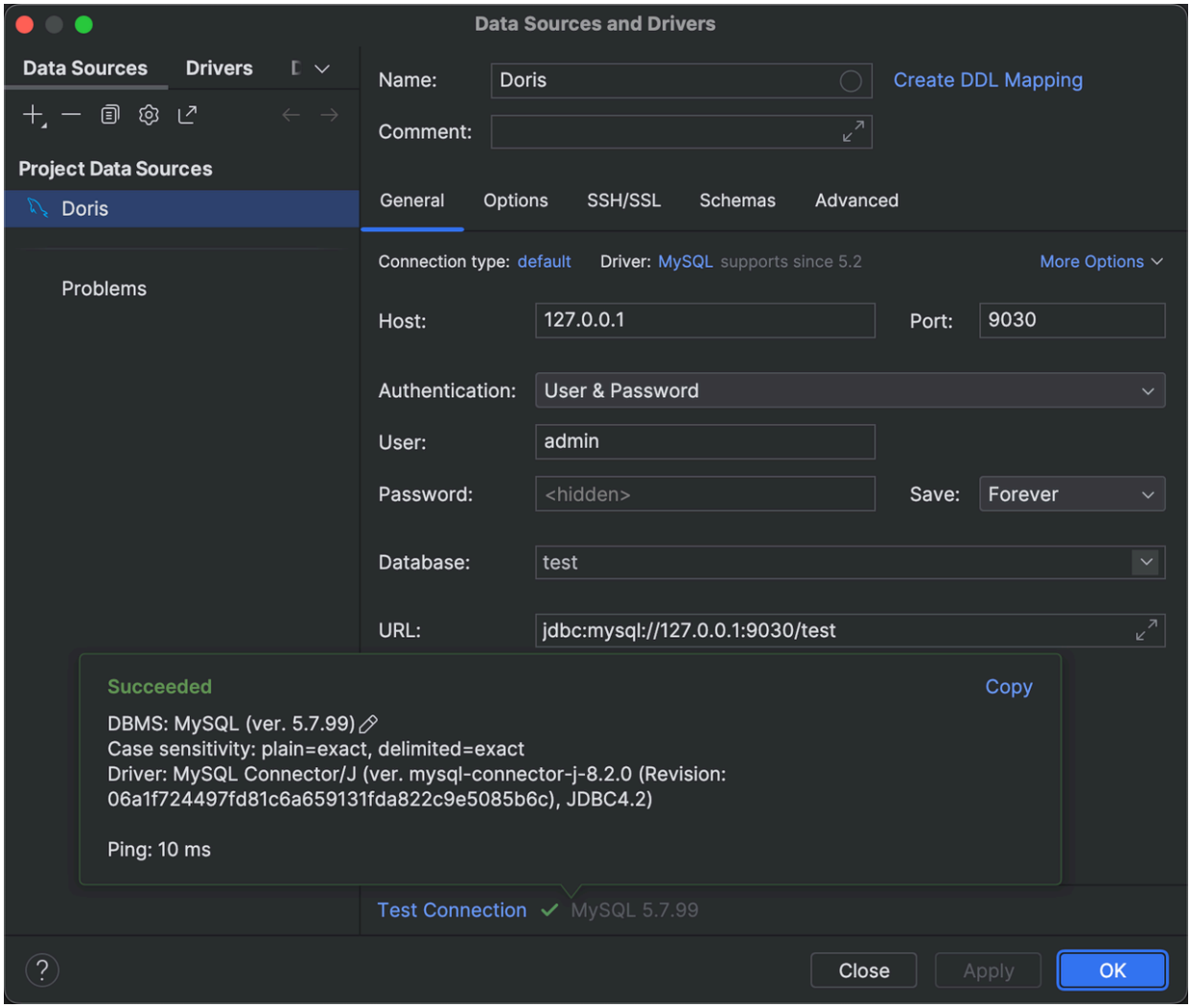
- external catalog



connect external catalog

5. Test data source connection

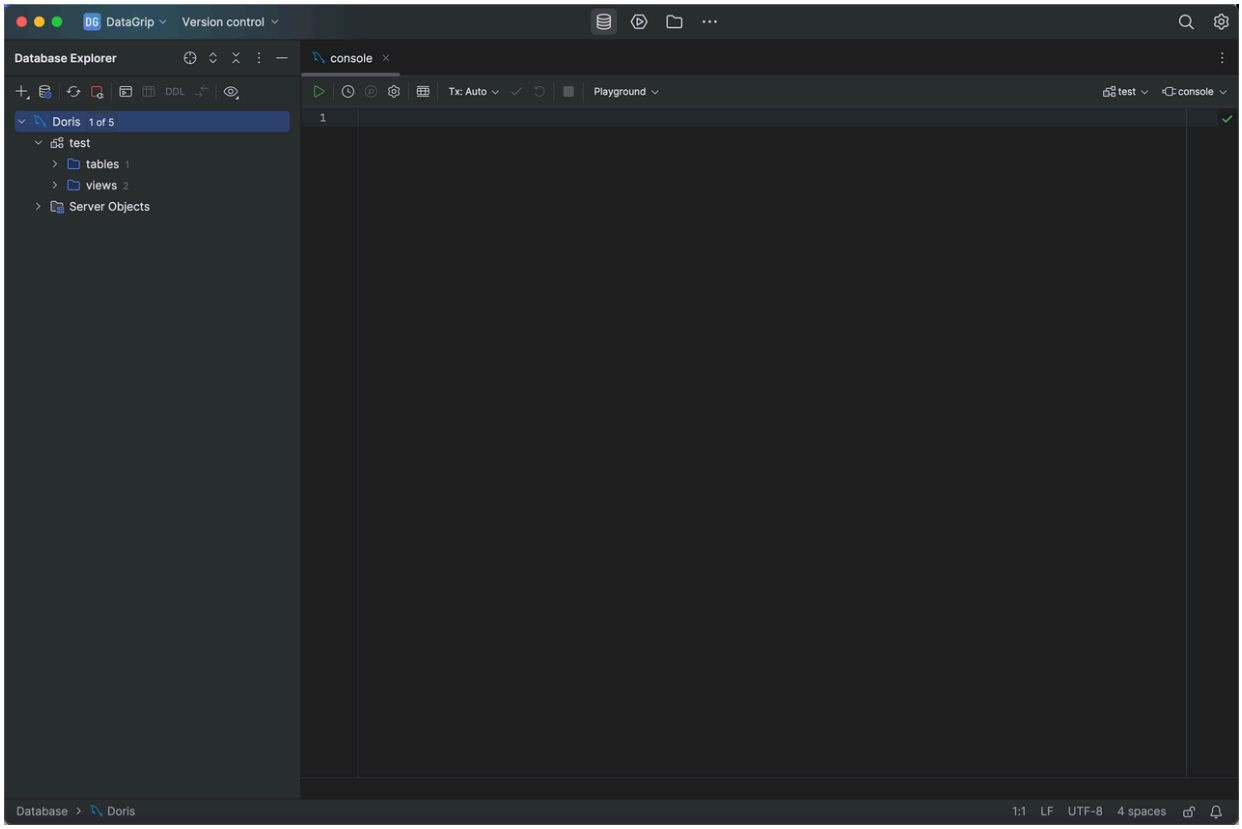
   After filling in the connection information, click Test Connection in the lower left corner to verify the accuracy of the database connection information. If DBeaver returns the following pop-up window, the test connection is successful. Then click OK in the lower right corner to complete the connection configuration.



test connection

6. Connect to database

   After the database connection is established, you can see the created data source connection in the database connection navigation on the left, and you can connect and manage the database through DataGrip.



create connection

Function support

Basically supports most visual viewing operations, as well as SQL console writing SQL operations Doris does not support or has not been verified various operations such as creating database tables, schema changes, and adding, deleting, and modifying data.

BladePipe

BladePipe is a **real-time end-to-end data replication** tool, moving data between **30+** databases, message queues, search engines, caching, real-time data warehouses, data lakes and more, with **ultra-low latency less than 3 seconds**. It features efficiency, stability and scalability, compatibility with diverse database engines, one-stop management, enhanced security, and complex data transformation.

Functions

BladePipe presents a visual management interface, allowing you to easily create DataJobs to realize **schema migration, data migration, synchronization, verification and correction**, etc. In addition, more refined and customized configurations are supported by setting parameters. Now BladePipe supports data movement from the following source DataSources to Doris:

Source DataSource	Schema Migration	Data Migration	Data Sync	Verification & Correction
MySQL/MariaDB/AuroraMySQL	Yes	Yes	Yes	Yes
Oracle	Yes	Yes	Yes	Yes
PostgreSQL/AuroraPostgreSQL	Yes	Yes	Yes	Yes
SQL Server	Yes	Yes	Yes	Yes
Kafka	No	No	Yes	No
AutoMQ	No	No	Yes	No
TiDB	Yes	Yes	Yes	Yes
Hana	Yes	Yes	Yes	Yes
PolarDB-X	Yes	Yes	Yes	Yes

> **For more functions and parameter settings, please refer to BladePipe Connections.**

Installation

Follow the instructions in Install Worker (Docker) or Install Worker (Binary) to download and install a BladePipe Worker.

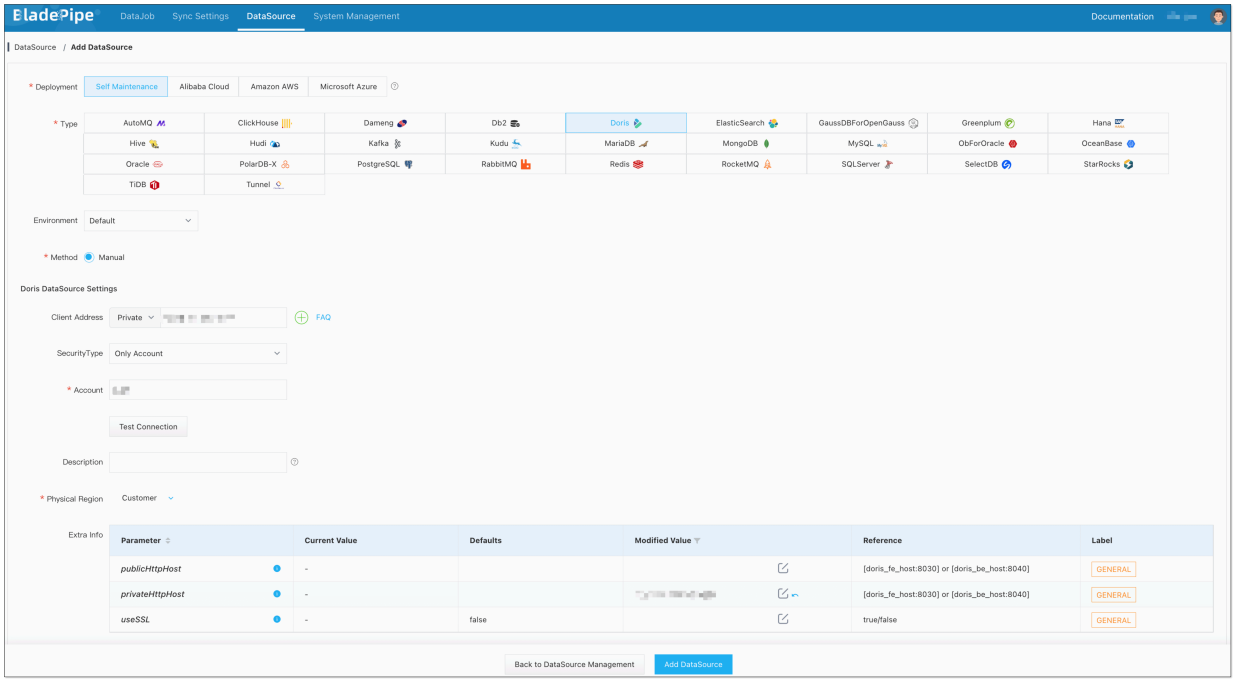
Example

Taking a MySQL instance as an example, the following part describes how to move data from MySQL to Doris.

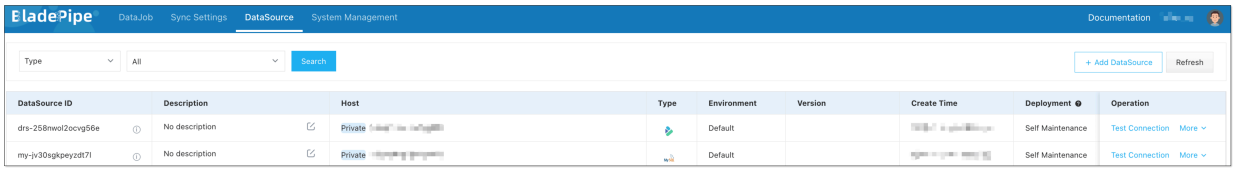
Add DataSources

1. Log in to the BladePipe Cloud. Click **DataSource** > **Add DataSource**.

2. Select MySQL and Doris as the Type respectively, and fill in the setup form accordingly.



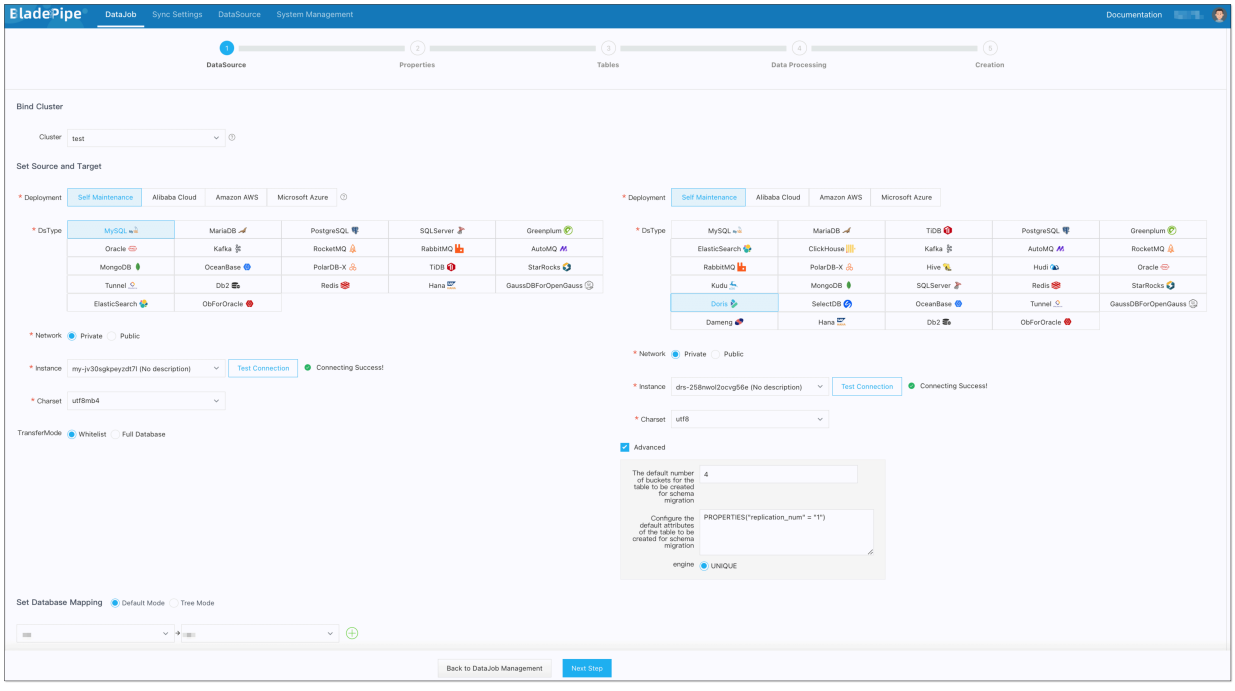
3. Click **Test Connection**. After successful connection, click **Add DataSource** to add the DataSource.



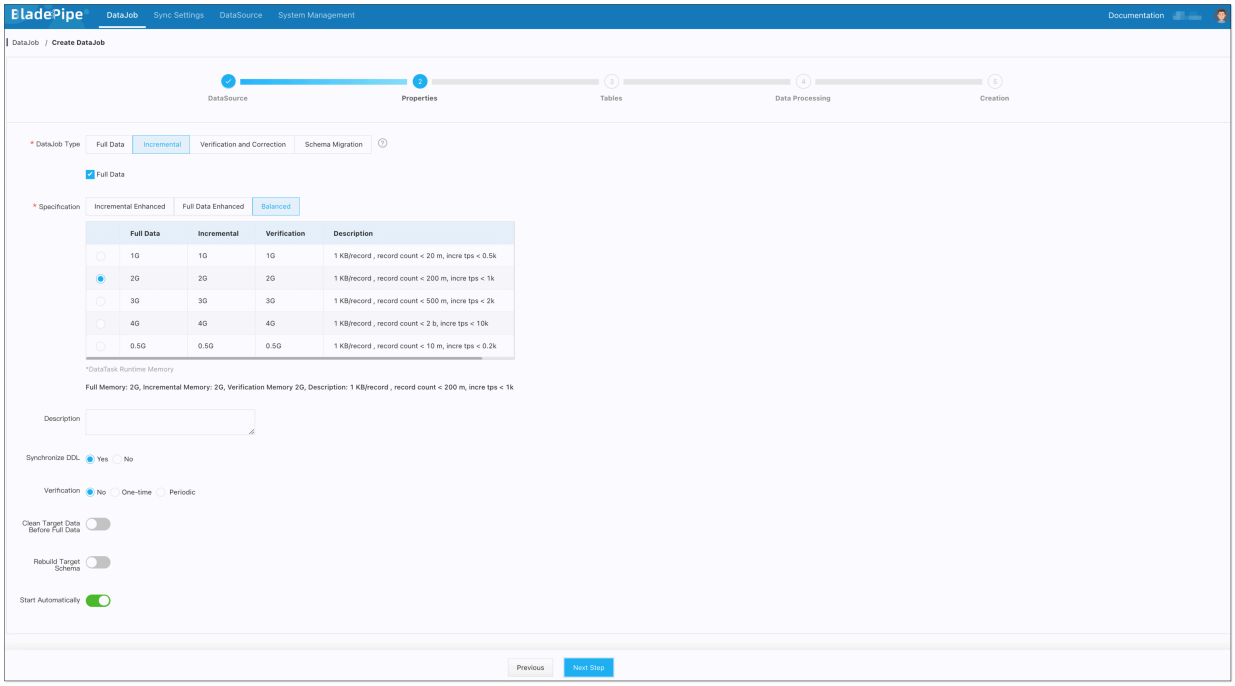
Create a DataJob

1. Click DataJob > Create DataJob.

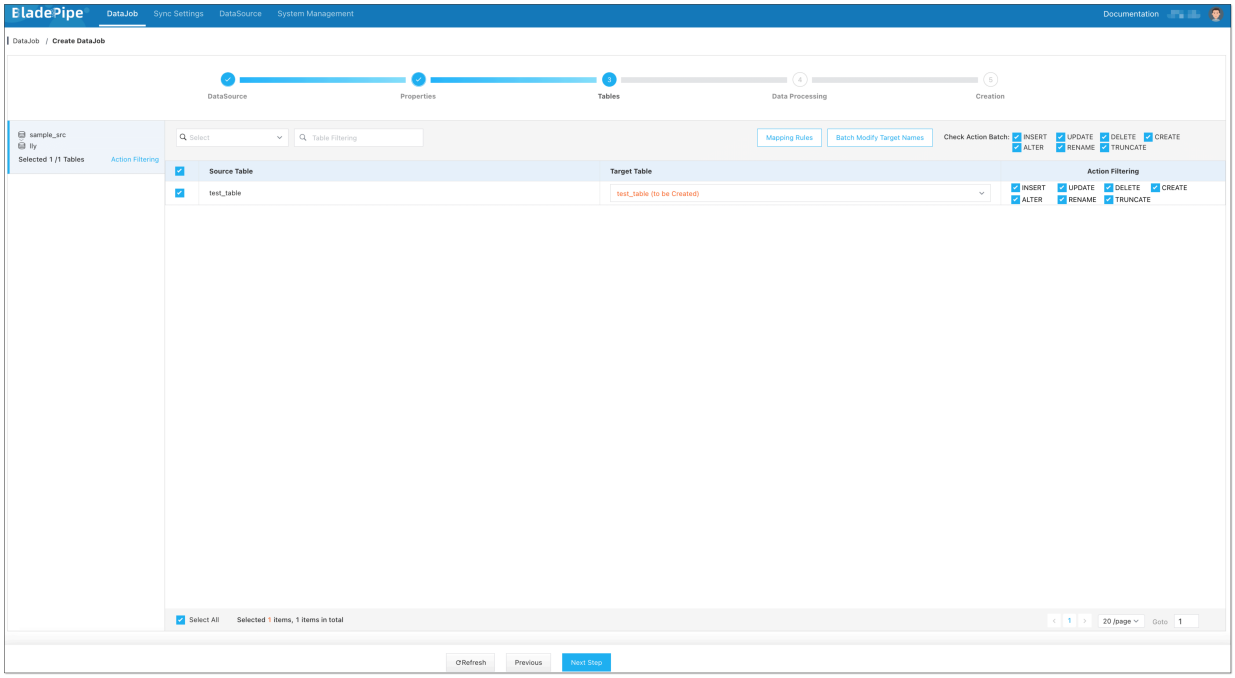
2. Select the source and target DataSources, and click **Test Connection** to ensure the connection to the source and target DataSources are both successful.



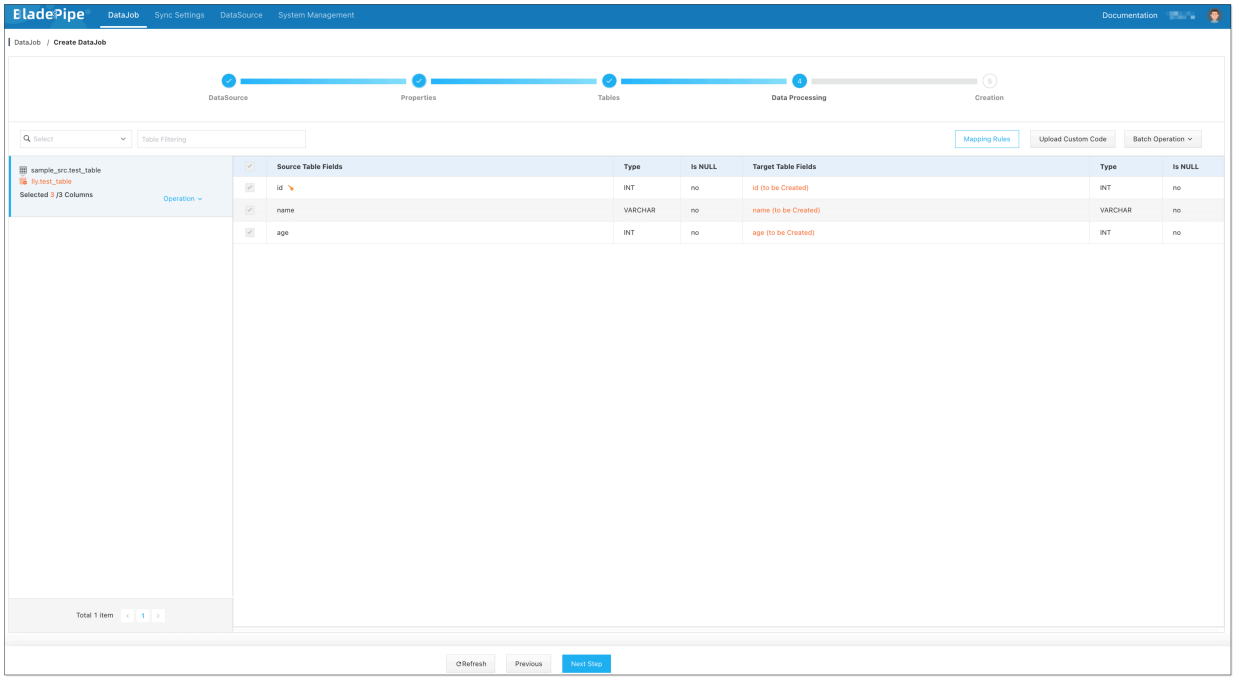
3. Select **Incremental** for DataJob Type, together with the **Full Data** option.



4. Select the tables to be replicated.



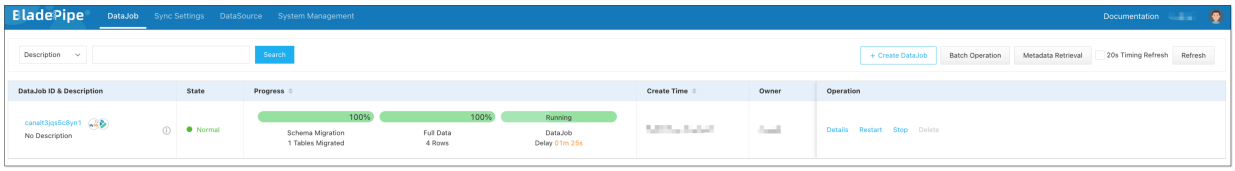
5. Select the columns to be replicated.



6. Confirm the DataJob creation.

7. The DataJob runs automatically. BladePipe will automatically run the following DataTasks:

- **Schema Migration**: The schemas of the source tables will be migrated to the target instance.
- **Full Data**: All existing data of the source tables will be fully migrated to the target instance.
- **Incremental**: Ongoing data changes will be continuously synchronized to the target instance.



DataX Doriswriter

DataX Doriswriter

The DataX Doriswriter plugin supports synchronizing data from various data sources, such as MySQL, Oracle, and SQL Server, into Doris using the Stream Load method.

> **Note**
>
> This plugin needs to be used together with the DataX service. DataX supports multiple data sources. For more details, see here.

Usage

Directly Download the DataX Installation Package

DataX provides an official installation package that already includes DataX, which can be downloaded and used directly. For more details, refer to here.

Compile the DorisWriter Plugin Manually

Download the source code for the DorisWriter plugin.

1. Run `init-env.sh`
2. Modify code of doriswriter in `DataX/doriswriter` if you need.
3. Build doriswriter

> Build doriswriter along:

```
`mvn clean install -pl plugin-rdbms-util,doriswriter -DskipTests`
```

> If you need to compile the entire DataX project, please refer to here

> Compilation error

```
If you encounter the following compilation errors:
```

```
Could not find artifact com.alibaba.datax:datax-all:pom:0.0.1-SNAPSHOT ...
```

You can try the following solutions:

1. Download [alibaba-datax-maven-m2-20210928.tar.gz](https://doris-thirdparty-repo.bj.bcebos.com/thirdparty/alibaba-datax-maven-m2-20210928.tar.gz)
2. After decompression, copy the resulting `alibaba/datax/` directory to `.m2/repository/com/alibaba/` corresponding to the maven used, and try to compile again.

Datax DorisWriter parameter introduction:

- **jdbcUrl**
  - Description: Doris's JDBC connection string, the user executes preSql or postSQL.
  - Mandatory: Yes
  - Default: None
- **loadUrl**
  - Description: As a connection target for Stream Load. The format is "ip:port". Where IP is the FE node IP, port is the http_port of the FE node. You can fill in more than one, separated by commas in English: ,, doriswriter will visit in a polling manner.
  - Mandatory: Yes
  - Default: None
- **username**
  - Description: The username to access the Doris database
  - Mandatory: Yes
  - Default: None
- **password**
  - Description: Password to access Doris database
  - Mandatory: No
  - Default: empty
- **connection.selectedDatabase**
  - Description: The name of the Doris database that needs to be written.
  - Mandatory: Yes
  - Default: None
- **connection. table**
  - Description: The name of the Doris table that needs to be written.
    - Mandatory: Yes
    - Default: None
- **flushInterval**

    - Description: The time interval at which data is written in batches. If this time interval is set too small, it will cause Doris write blocking problem, error code -235, and if you set this time interval too small, `maxBatchRows` and `batchSize` parameters are set too large, then it may not be able to reach you The data size set by this will also be imported.
    - Mandatory: No
    - Default: 30000 (ms)

- **column**

    - Description: The fields that the destination table needs to write data into, these fields will be used as the field names of the generated Json data. Fields are separated by commas. For example: “column”: [“id”,“name”,“age”].
    - Mandatory: Yes
    - Default: No

- **preSql**

    - Description: Before writing data to the destination table, the standard statement here will be executed first.
    - Mandatory: No
    - Default: None

- **postSql**

    - Description: After writing data to the destination table, the standard statement here will be executed.
    - Mandatory: No
    - Default: None

- **maxBatchRows**

    - Description: The maximum number of rows for each batch of imported data. Together with **batchSize**, it controls the number of imported record rows per batch. When each batch of data reaches one of the two thresholds, the data of this batch will start to be imported.
    - Mandatory: No
    - Default: 500000

- **batchSize**

    - Description: The maximum amount of data imported in each batch. Works with **maxBatchRows** to control the number of imports per batch. When each batch of data reaches one of the two thresholds, the data of this batch will start to be imported.
    - Mandatory: No
    - Default: 94371840

- **maxRetries**

    - Description: The number of retries after each batch of failed data imports.
    - Mandatory: No
    - Default: 3

- **labelPrefix**

    - Description: The label prefix for each batch of imported tasks. The final label will have `labelPrefix + UUID` to form a globally unique label to ensure that data will not be imported repeatedly
    - Mandatory: No
    - Default: `datax_doris_writer_`

- **loadProps**

    - Description: The request parameter of StreamLoad. For details, refer to the StreamLoad introduction page. Stream load - Apache Doris

      This includes the imported data format: format, etc. The imported data format defaults to csv, which supports JSON. For details, please refer to the type conversion section below, or refer to the official information of Stream load above.

    - Mandatory: No

    - Default: None

Example

1. Stream reads the data and imports it to Doris

For instructions on using the doriswriter plug-in, please refer to here.

2.Mysql reads the data and imports it to Doris

1.Mysql table structure

```
CREATE TABLE `t_test`(
 `id`bigint(30) NOT NULL,
 `order_code` varchar(30) DEFAULT NULL COMMENT '',
 `line_code` varchar(30) DEFAULT NULL COMMENT '',
 `remark` varchar(30) DEFAULT NULL COMMENT '',
 `unit_no` varchar(30) DEFAULT NULL COMMENT '',
 `unit_name` varchar(30) DEFAULT NULL COMMENT '',
 `price` decimal(12,2) DEFAULT NULL COMMENT '',
 PRIMARY KEY(`id`) USING BTREE
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 ROW_FORMAT=DYNAMIC COMMENT='';
```

2.Doris table structure

```
CREATE TABLE `ods_t_test` (
 `id`bigint(30) NOT NULL,
 `order_code` varchar(30) DEFAULT NULL COMMENT '',
 `line_code` varchar(30) DEFAULT NULL COMMENT '',
 `remark` varchar(30) DEFAULT NULL COMMENT '',
 `unit_no` varchar(30) DEFAULT NULL COMMENT '',
 `unit_name` varchar(30) DEFAULT NULL COMMENT '',
 `price` decimal(12,2) DEFAULT NULL COMMENT ''
) ENGINE=OLAP
UNIQUE KEY(id`, `order_code`)
DISTRIBUTED BY HASH(`order_code`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 3",
```

```
"in_memory" = "false",
"storage_format" = "V2"
);
```

3.Create datax script

my_import.json

```
{
    "job": {
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "column": ["id","order_code","line_code","remark","unit_no","unit_name","price"],
                        "connection": [
                            {
                                "jdbcUrl": ["jdbc:mysql://localhost:3306/demo"],
                                "table": ["employees_1"]
                            }
                        ],
                        "username": "root",
                        "password": "xxxxx",
                        "where": ""
                    }
                },
                "writer": {
                    "name": "doriswriter",
                    "parameter": {
                        "loadUrl": ["127.0.0.1:8030"],
                        "column": ["id","order_code","line_code","remark","unit_no","unit_name","price"],
                        "username": "root",
                        "password": "xxxxxx",
                        "postSql": ["select count(1) from all_employees_info"],
                        "preSql": [],
                        "flushInterval":30000,
                        "connection": [
                          {
                            "jdbcUrl": "jdbc:mysql://127.0.0.1:9030/demo",
                            "selectedDatabase": "demo",
                            "table": ["all_employees_info"]
                          }
                        ],
                        "loadProps": {
                            "format": "json",
                            "strip_outer_array":"true",
                            "line_delimiter": "\\x02"
                        }
                    }
                }
            }
        ],
        "setting": {
            "speed": {
                "channel": "1"
            }
        }
    }
}
```

Remark:

```
"loadProps": {
  "format": "json",
  "strip_outer_array": "true",
  "line_delimiter": "\\x02"
}
```

1. Here we use JSON format to import data
2. `line_delimiter` defaults to a newline character, which may conflict with the value in the data, we can use some special characters or invisible characters to avoid import errors
3. strip_outer_array : Represents multiple rows of data in a batch of imported data. Doris will expand the array when parsing, and then parse each Object in it as a row of data in turn.
4. For more parameters of Stream load, please refer to Stream load - Apache Doris
5. If it is in CSV format, we can use it like this

```
"loadProps": {
   "format": "csv",
   "column_separator": "\\x01",
   "line_delimiter": "\\x02"
}
```

**CSV format should pay special attention to row and column separators to avoid conflicts with special characters in the data. Hidden characters are recommended here. The default column separator is: row separator: *

4.Execute the datax task, refer to the specific datax official website

```
python bin/datax.py my_import.json
```

After execution, we can see the following information

```
2022-11-16 14:28:54.012 [job-0] INFO  JobContainer - jobContainer starts to do prepare ...
2022-11-16 14:28:54.012 [job-0] INFO  JobContainer - DataX Reader.Job [mysqlreader] do prepare work .
2022-11-16 14:28:54.013 [job-0] INFO  JobContainer - DataX Writer.Job [doriswriter] do prepare work .
2022-11-16 14:28:54.020 [job-0] INFO  JobContainer - jobContainer starts to do split ...
2022-11-16 14:28:54.020 [job-0] INFO  JobContainer - Job set Channel-Number to 1 channels.
2022-11-16 14:28:54.023 [job-0] INFO  JobContainer - DataX Reader.Job [mysqlreader] splits to [1] tasks.
2022-11-16 14:28:54.023 [job-0] INFO  JobContainer - DataX Writer.Job [doriswriter] splits to [1] tasks.
2022-11-16 14:28:54.033 [job-0] INFO  JobContainer - jobContainer starts to do schedule ...
2022-11-16 14:28:54.036 [job-0] INFO  JobContainer - Scheduler starts [1] taskGroups.
2022-11-16 14:28:54.037 [job-0] INFO  JobContainer - Running by standalone Mode.
2022-11-16 14:28:54.041 [taskGroup-0] INFO  TaskGroupContainer - taskGroupId=[0] start [1] channels for [1] tasks.
2022-11-16 14:28:54.043 [taskGroup-0] INFO  Channel - Channel set byte_speed_limit to -1, No bps activated.
2022-11-16 14:28:54.043 [taskGroup-0] INFO  Channel - Channel set record_speed_limit to -1, No tps activated.
2022-11-16 14:28:54.049 [taskGroup-0] INFO  TaskGroupContainer - taskGroup[0] taskId[0] attemptCount[1] is started
2022-11-16 14:28:54.052 [0-0-0-reader] INFO  CommonRdbmsReader$Task - Begin to read record by Sql: [select
taskid,projectid,taskflowid,templateid,template_name,status_task from dwd_universal_tb_task
] jdbcUrl:[jdbc:mysql://localhost:3306/demo?
yearIsDateType=false&zeroDateTimeBehavior=convertToNull&tinyInt1isBit=false&rewriteBatchedStatements=true].
Wed Nov 16 14:28:54 GMT+08:00 2022 WARN: Establishing SSL connection without server's identity verification is not recommended.
According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't
set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either
```

```
to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
2022-11-16 14:28:54.071 [0-0-0-reader] INFO  CommonRdbmsReader$Task - Finished read record by Sql: [select
taskid,projectid,taskflowid,templateid,template_name,status_task from dwd_universal_tb_task
] jdbcUrl:[jdbc:mysql://localhost:3306/demo?
yearIsDateType=false&zeroDateTimeBehavior=convertToNull&tinyInt1isBit=false&rewriteBatchedStatements=true].
2022-11-16 14:28:54.104 [Thread-1] INFO  DorisStreamLoadObserver - Start to join batch data: rows[2] bytes[438]
label[datax_doris_writer_c4e08cb9-c157-4689-932f-db34acc45b6f].
2022-11-16 14:28:54.104 [Thread-1] INFO  DorisStreamLoadObserver - Executing stream load to:
'http://127.0.0.1:8030/api/demo/dwd_universal_tb_task/_stream_load', size: '441'
2022-11-16 14:28:54.224 [Thread-1] INFO  DorisStreamLoadObserver - StreamLoad response :
{"Status":"Success","BeginTxnTimeMs":0,"Message":"OK","NumberUnselectedRows":0,"CommitAndPublishTimeMs":17,"Label":"datax_doris_writ
er_c4e08cb9-c157-4689-932f-
db34acc45b6f","LoadBytes":441,"StreamLoadPutTimeMs":1,"NumberTotalRows":2,"WriteDataTimeMs":11,"TxnId":217056,"LoadTimeMs":31,"TwoPh
aseCommit":"false","ReadDataTimeMs":0,"NumberLoadedRows":2,"NumberFilteredRows":0}
2022-11-16 14:28:54.225 [Thread-1] INFO  DorisWriterManager - Async stream load finished: label[datax_doris_writer_c4e08cb9-c157-
4689-932f-db34acc45b6f].
2022-11-16 14:28:54.249 [taskGroup-0] INFO  TaskGroupContainer - taskGroup[0] taskId[0] is successed, used[201]ms
2022-11-16 14:28:54.250 [taskGroup-0] INFO  TaskGroupContainer - taskGroup[0] completed it's tasks.
2022-11-16 14:29:04.048 [job-0] INFO  StandAloneJobContainerCommunicator - Total 2 records, 214 bytes | Speed 21B/s, 0 records/s |
Error 0 records, 0 bytes |  All Task WaitWriterTime 0.000s |  All Task WaitReaderTime 0.000s | Percentage 100.00%
2022-11-16 14:29:04.049 [job-0] INFO  AbstractScheduler - Scheduler accomplished all tasks.
2022-11-16 14:29:04.049 [job-0] INFO  JobContainer - DataX Writer.Job [doriswriter] do post work.
Wed Nov 16 14:29:04 GMT+08:00 2022 WARN: Establishing SSL connection without server's identity verification is not recommended.
According to MySQL 5.5.45+, 5.6.26+ and 5.7.6+ requirements SSL connection must be established by default if explicit option isn't
set. For compliance with existing applications not using SSL the verifyServerCertificate property is set to 'false'. You need either
to explicitly disable SSL by setting useSSL=false, or set useSSL=true and provide truststore for server certificate verification.
2022-11-16 14:29:04.187 [job-0] INFO  DorisWriter$Job - Start to execute preSqls:[select count(1) from dwd_universal_tb_task].
context info:jdbc:mysql://172.16.0.13:9030/demo.
2022-11-16 14:29:04.204 [job-0] INFO  JobContainer - DataX Reader.Job [mysqlreader] do post work.
2022-11-16 14:29:04.204 [job-0] INFO  JobContainer - DataX jobId [0] completed successfully.
2022-11-16 14:29:04.204 [job-0] INFO  HookInvoker - No hook invoked, because base dir not exists or is a file: /data/datax/hook
2022-11-16 14:29:04.205 [job-0] INFO  JobContainer -
         [total cpu info] =>
                averageCpu                     | maxDeltaCpu                    | minDeltaCpu
                -1.00%                         | -1.00%                         | -1.00%


         [total gc info] =>
                 NAME                 | totalGCCount       | maxDeltaGCCount    | minDeltaGCCount    | totalGCTime        |
maxDeltaGCTime     | minDeltaGCTime
                 PS MarkSweep         | 1                  | 1                  | 1                  | 0.017s             | 0.017s
| 0.017s
                 PS Scavenge          | 1                  | 1                  | 1                  | 0.007s             | 0.007s
| 0.007s

2022-11-16 14:29:04.205 [job-0] INFO  JobContainer - PerfTrace not enable!
2022-11-16 14:29:04.206 [job-0] INFO  StandAloneJobContainerCommunicator - Total 2 records, 214 bytes | Speed 21B/s, 0 records/s |
Error 0 records, 0 bytes |  All Task WaitWriterTime 0.000s |  All Task WaitReaderTime 0.000s | Percentage 100.00%
2022-11-16 14:29:04.206 [job-0] INFO  JobContainer -
Task Start Time                        : 2022-11-16 14:28:53
Task End Time                          : 2022-11-16 14:29:04
Total Task Duration                    : 10s
Average Task Throughput                : 21B/s
Record Write Speed                     : 0rec/s
Total Records Read                     : 2
Total Read/Write Failures              : 0
```

DBT Doris Adapter

DBT Doris Adapter

DBT(Data Build Tool) is a component that focuses on doing T (Transform) in ELT (extraction, loading, transformation) - the “transformation data” link The `dbt-doris` adapter is developed based on `dbt-core` and relies on the `mysql-connector-python` driver to convert data to doris.

git: https://github.com/apache/doris/tree/master/extension/dbt-doris

version

doris	python	dbt-core	dbt-doris
>=1.2.5	>=3.8,<=3.10	>=1.5.0	<=0.3
>=1.2.5	>=3.9	>=1.8.0	>=0.4

dbt-doris adapter Instructions

dbt-doris adapter install

use pip install:

```
pip install dbt-doris
```

check version:

```
dbt --version
```

if command not found: dbt:

```
ln -s /usr/local/python3/bin/dbt /usr/bin/dbt
```

dbt-doris adapter project init

```
dbt init
```

Users need to prepare the following information to init dbt project

name	default	meaning
project		project name

name	default	meaning
database		Enter the corresponding number to select the adapter
host		doris host
port	9030	doris MySQL Protocol Port
schema		In dbt-doris, it is equivalent to database, Database name
username		doris username
password		doris password
threads	1	Parallelism in dbt-doris (setting a parallelism that does not match the cluster capability will increase the risk of dbt running failure)

dbt-doris adapter run

For dbt run documentation, please refer to here. Go to the project directory and execute the default dbt model:

```
dbt run
```

model：`my_first_dbt_model`和`my_second_dbt_model`

They are materialized `table` and `view` respectively. then login to doris to view the data results and table creation statements of `my_first_dbt_model` and `my_second_dbt_model`. #### dbt-doris adapter Materialization dbt-doris Materialization support three: 1. view 2. table 3. incremental

View

Using `view` as the materialization, Models will be rebuilt as views each time they are run through the create view as statement. (By default, the materialization method of dbt is view)

```
Advantages: No extra data is stored, and views on top of the source data will always contain the latest records.
Disadvantages: View queries that perform large transformations or are nested on top of other views are slow.
Recommendation: Usually start with the view of the model and only change to another materialization if there are performance issues.
Views are best suited for models that do not undergo major transformations, such as renaming, column changes.
```

config：

```
models:
  <resource-path>:
    +materialized: view
```

Or write in the model file

```
{{ config(materialized = "view") }}
```

Table

When using the `table` materialization mode, your model is rebuilt as a table at each run with a `create table as select` statement. For the tablet materialization of dbt, dbt-doris uses the following steps to ensure the atomicity of data changes: 1. first create a temporary table: `create table this_table_temp as {{ model sql}}`. 2. Determine whether `this_table` does not exist, that is, it is created for the first time, execute `rename`, and change the temporary table to the final table. 3. if already exists, then `alter table this_table REPLACE WITH TABLE this_table_temp PROPERTIES('swap' = 'False')`, This operation can exchange the table name and delete the `this_table_temp` temporary table,this guarantees the atomicity of this operation through the transaction mechanism of the Doris.

```
Advantages: table query speed will be faster than view.
Disadvantages: The table takes a long time to build or rebuild, additional data will be stored, and incremental data synchronization cannot be performed.
Recommendation: It is recommended to use the table materialization method for models queried by BI tools or models with slow operations such as downstream queries and conversions.
```

config:

```
models:
  <resource-path>:
    +materialized: table
    +duplicate_key: [ <column-name>, ... ],
    +replication_num: int,
    +partition_by: [ <column-name>, ... ],
    +partition_type: <engine-type>,
    +partition_by_init: [<pertition-init>, ... ]
    +distributed_by: [ <column-name>, ... ],
    +buckets: int | 'auto',
    +properties: {<key>:<value>,...}
```

Or write in the model file:

```
{{ config(
    materialized = "table",
    duplicate_key = [ "<column-name>", ... ],
    replication_num = "<int>"
    partition_by = [ "<column-name>", ... ],
    partition_type = "<engine-type>",
    partition_by_init = ["<pertition-init>", ... ]
    distributed_by = [ "<column-name>", ... ],
    buckets = "<int>" | "auto",
    properties = {"<key>":"<value>",...}
      ...
    ]
) }}
```

The details of the above configuration items are as follows:

item	description	Required?
materialized	The materialized form of the table (Doris Duplicate table)	Required
duplicate_key	Doris Duplicate key	Optional
replication_num	Number of table replicas	Optional
partition_by	Table partition column	Optional
partition_type	Table partition type, range or list.(default: RANGE)	Optional
partition_by_init	Initialized table partitions	Optional
distributed_by	Table distributed column	Optional
buckets	Bucket size	Optional
properties	Doris table properties	Optional

Incremental

Based on the incremental model results of the last run of dbt, records are incrementally inserted or updated into the table. There are two ways to realize the increment of doris. incremental_strategy has two incremental strategies: * insert_overwrite: Depends on the doris unique model. If there is an incremental requirement, specify the materialization as incremental when initializing the data of the model, and aggregate by specifying the aggregation column to achieve incremental data coverage. * append: Depends on the doris duplicate model, it only appends incremental data and does not involve modifying any historical data. So no need to specify unique_key.

```
Advantages: Significantly reduces build time by only converting new records.
Disadvantages: incremental mode requires additional configuration, which is an advanced usage of dbt, and requires the support of complex scenarios and the adaptation of corresponding components.
Recommendation: The incremental model is best for event-based scenarios or when dbt runs become too slow
```

config:

```
models:
  <resource-path>:
    +materialized: incremental
    +incremental_strategy: <strategy>
    +unique_key: [ <column-name>, ... ],
    +replication_num: int,
    +partition_by: [ <column-name>, ... ],
    +partition_type: <engine-type>,
    +partition_by_init: [<pertition-init>, ... ]
    +distributed_by: [ <column-name>, ... ],
    +buckets: int | 'auto',
    +properties: {<key>:<value>,...}
```

Or write in the model file:

```
{{ config(
    materialized = "incremental",
    incremental_strategy = "<strategy>"
    unique_key = [ "<column-name>", ... ],
    replication_num = "<int>"
```

```
    partition_by = [ "<column-name>", ... ],
    partition_type = "<engine-type>",
    partition_by_init = ["<pertition-init>", ... ]
    distributed_by = [ "<column-name>", ... ],
    buckets = "<int>" | "auto",
    properties = {"<key>":"<value>",...}
      ...
    ]
) }}
```

The details of the above configuration items are as follows:

item	description	Required?
`materialized`	The materialized form of the table (Doris Duplicate/Unique table)	Required
`incremental_strategy`	Incremental_strategy	Optional
`unique_key`	Doris Unique key	Optional
`replication_num`	Number of table replicas	Optional
`partition_by`	Table partition column	Optional
`partition_type`	Table partition type, `range` or `list`.(default: `RANGE`)	Optional
`partition_by_init`	Initialized table partitions	Optional
`distributed_by`	Table distributed column	Optional
`buckets`	Bucket size	Optional
`properties`	Doris table properties	Optional

dbt-doris adapter seed

`seed` is a functional module used to load data files such as csv. It is a way to load files into the library and participate in model building, but there are the following precautions: 1. Seeds should not be used to load raw data (for example, large CSV exports from a production database). 2. Since seeds are version controlled, they are best suited to files that contain business-specific logic, for example a list of country codes or user IDs of employees. 3. Loading CSVs using dbt's seed functionality is not performant for large files. Consider using `streamload` to load these CSVs into doris.

Users can see the seeds directory under the dbt project directory, upload the csv file and seed configuration file in it and run

```
 dbt seed --select seed_name
```

Common seed configuration file writing method supports the definition of column types:

```
seeds:
  seed_name:
    config:
      schema: demo_seed
      full_refresh: true
      replication_num: 1
      column_types:
        id: bigint
```

```
        phone: varchar(32)
        ip: varchar(15)
        name: varchar(20)
        cost: DecimalV3(19,10)
```

Usage Examples

View Model Sample Reference

```
{{ config(materialized='view') }}

select
    u.user_id,
    max(o.create_time) as create_time,
    sum (o.cost) as balance
from {{ ref('sell_order') }} as o
left join {{ ref('sell_user') }} as u
on u.account_id=o.account_id
group by u.user_id
order by u.user_id
```

Table Model Sample Reference

```
{{ config(materialized='table') }}

select
    u.user_id,
    max(o.create_time) as create_time,
    sum (o.cost) as balance
from {{ ref('sell_order') }} as o
left join {{ ref('sell_user') }} as u
on u.account_id=o.account_id
group by u.user_id
order by u.user_id
```

Incremental model sample reference (duplicate mode)

Create a table in duplicate mode, without data aggregation, and without specifying unique_key

```
{{ config(
    materialized='incremental',
    replication_num=1
) }}

with source_data as (
    select
        *
    from {{ ref('sell_order2') }}
)

select * from source_data
```

Incremental model sample reference (unique mode)

Create a table in unique mode, data aggregation, unique_key must be specified

```
{{ config(
materialized='incremental',
unique_key=['account_id','create_time']
) }}

with source_data as (
    select
        *
    from {{ ref('sell_order2') }}
)

select * from source_data
```

Incremental model full refresh sample reference

```
{{ config(
    materialized='incremental',
    full_refresh = true
)}}

select * from
 {{ source('dbt_source', 'sell_user') }}
```

Example of setting bucketing rules

Here buckets can be filled with auto or a positive integer, representing automatic bucketing and setting a fixed number of buckets respectively.

```
{{ config(
    materialized='incremental',
    unique_key=['account_id',"create_time"],
    distributed_by=['account_id'],
    buckets='auto'
) }}

with source_data as (
    select
        *
    from {{ ref('sell_order') }}
)

select
    *
    from source_data

{% if is_incremental() %}
    where
    create_time > (select max(create_time) from {{this}})
{% endif %}
```

Setting the number of replicas example reference

```
{{ config(
    materialized='table',
    replication_num=1
)}}

with source_data as (
    select
        *
    from {{ ref('sell_order2') }}
)

select * from source_data
```

Dynamic partition sample reference

```
{{ config(
    materialized='incremental',
    partition_by = 'create_time',
    partition_type = 'range',
        -- The properties here are the properties in the create table statement, which contains the configuration related to dynamic
partitioning
    properties = {
        "dynamic_partition.time_unit":"DAY",
        "dynamic_partition.end":"8",
        "dynamic_partition.prefix":"p",
        "dynamic_partition.buckets":"4",
        "dynamic_partition.create_history_partition":"true",
        "dynamic_partition.history_partition_num":"3"
    }
) }}

with source_data as (
    select
        *
    from {{ ref('sell_order2') }}
)

select
    *
    from source_data

{% if is_incremental() %}
    where
    create_time = DATE_SUB(CURDATE(), INTERVAL 1 DAY)
{% endif %}
```

Conventional partition sample reference

```
{{ config(
    materialized='incremental',
    partition_by = 'create_time',
    partition_type = 'range',
        -- partition_by_init here refers to the historical partitions for creating partition tables. The historical partitions of
the current doris version need to be manually specified.
    partition_by_init = [
        "PARTITION `p20240601` VALUES [(\"2024-06-01\"),  (\"2024-06-02\"))",
        "PARTITION `p20240602` VALUES [(\"2024-06-02\"),  (\"2024-06-03\"))"
```

```
    ]
 )}}

with source_data as (
    select
        *
    from {{ ref('sell_order2') }}
)

select
    *
    from source_data

{% if is_incremental() %}
    where
    -- If the my_date variable is provided, use this path (via the dbt run --vars '{"my_date": "\"2024-06-03\""}' command). If the
my_date variable is not provided (directly using dbt run), use the day before the current date. For the incremental selection here,
it is recommended to directly use doris's CURDATE() function, which is also a common path in production environments.
    create_time = {{ var('my_date' , 'DATE_SUB(CURDATE(), INTERVAL 1 DAY)') }}

{% endif %}
```

Batch date setting parameter sample reference

```
{{ config(
    materialized='incremental',
    partition_by = 'create_time',
    partition_type = 'range',
    ...
)}}

with source_data as (
    select
        *
    from {{ ref('sell_order2') }}
)

select
    *
    from source_data

{% if is_incremental() %}
    where
    -- If the my_date variable is provided, use this path (via the dbt run --vars '{"my_date": "\"2024-06-03\""}' command). If the
my_date variable is not provided (directly using dbt run), use the day before the current date. For the incremental selection here,
it is recommended to directly use doris's CURDATE() function, which is also a common path in production environments.
    create_time = {{ var('my_date' , 'DATE_SUB(CURDATE(), INTERVAL 1 DAY)') }}

{% endif %}
```

Customizing table data column types and precision sample reference

The `schema.yaml` file configures `data_type` for `columns` in `models` as follows:

```
models:
  - name: sell_user
    description: "A dbt model named sell_user"
    columns:
      - name: user_id
        data_type: BIGINT
```

```
      - name: account_id
        data_type: VARCHAR(12)
      - name: status
      - name: cost_sum
        data_type: DECIMAL(38,9)
      - name: update_time
        data_type: DATETIME
      - name: create_time
        data_type: DATETIME
```

Access catalog sample reference

The Data Catalog is a reference to different data sources within the Doris data lake functionality, layered above the Database. It is recommended to access it through the dbt-doris built-in Macros: `catalog_source`

```
{{ config(materialized='table', replication_num=1) }}

select *
--  use macros 'catalog_source' not macros 'source'
--  catalog name is 'mysql_catalog'
--  database name is 'dbt_source'
--  table name is 'sell_user'
from {{ catalog_source('mysql_catalog', 'dbt_source', 'sell_user') }}
```

Seatunnel Doris Sink

About SeaTunnel

SeaTunnel is a very easy-to-use ultra-high-performance distributed data integration platform that supports real-time synchronization of massive data. It can synchronize tens of billions of data stably and efficiently every day.

Connector-V2

The connector-v2 for SeaTunnel supports Doris Sink since version 2.3.1 and supports exactly-once write and CDC data synchronization

Plugin Code

SeaTunnel Doris Sink Plugin Code

Options

name	type	required	default value
fenodes	string	yes	-
username	string	yes	-
password	string	yes	-
table.identifier	string	yes	-
sink.label-prefix	string	yes	-
sink.enable-2pc	bool	no	true
sink.enable-delete	bool	no	false
doris.config	map	yes	-

`fenodes [string]`

Doris cluster FE Nodes address, the format is `"fe_ip:fe_http_port, ..."`

`username [string]`

Doris user username

`password [string]`

Doris`user password

`table.identifier [string]`

The name of Doris table,The format is DBName.TableName

`sink.label-prefix [string]`

The label prefix used by stream load imports. In the 2pc scenario, global uniqueness is required to ensure the EOS semantics of SeaTunnel.

`sink.enable-2pc [bool]`

Whether to enable two-phase commit (2pc), the default is true, to ensure Exactly-Once semantics. For two-phase commit, please refer to here.

`sink.enable-delete [bool]`

Whether to enable deletion. This option requires Doris table to enable batch delete function (0.15+ version is enabled by default), and only supports Unique model. you can get more detail at this link:

batch delete

`doris.config [map]`

The parameter of the stream load `data_desc`, you can get more detail at this link:

More Stream Load parameters

Example

Use JSON format to import data

```
sink {
    Doris {
        fenodes = "doris_fe:8030"
        username = root
        password = ""
        table.identifier = "test.table_sink"
        sink.enable-2pc = "true"
        sink.label-prefix = "test_json"
        doris.config = {
            format="json"
            read_json_by_line="true"
        }
    }
}
```

Use CSV format to import data

```
sink {
    Doris {
        fenodes = "doris_fe:8030"
        username = root
        password = ""
        table.identifier = "test.table_sink"
        sink.enable-2pc = "true"
        sink.label-prefix = "test_csv"
        doris.config = {
          format = "csv"
          column_separator = ","
        }
    }
}
```

Connector-V1

Flink Sink Doris

Plugin Code

Seatunnel Flink Sink Doris plugin code

Options

name	type	required	default value	engine
fenodes	string	yes	-	Flink
database	string	yes	-	Flink
table	string	yes	-	Flink
user	string	yes	-	Flink
password	string	yes	-	Flink
batch_size	int	no	100	Flink
interval	int	no	1000	Flink
max_retries	int	no	1	Flink
doris.*	-	no	-	Flink

`fenodes [string]`

Doris Fe http url, eg: 127.0.0.1:8030

`database [string]`

Doris database

`table [string]`

Doris table

`user [string]`

Doris user

`password [string]`

Doris password

`batch_size [int]`

The maximum number of lines to write to Doris at a time, the default value is 100

`interval [int]`

The flush interval (in milliseconds), after which the asynchronous thread writes the data in the cache to Doris. Set to 0 to turn off periodic writes.

`max_retries [int]`

Number of retries after writing to Doris fails

`doris.* [string]`

Import parameters for Stream load. For example: 'doris.column_separator' = ',' etc.

More Stream Load parameter configuration

Examples

Socket To Doris

```
env {
  execution.parallelism = 1
}
source {
    SocketStream {
      host = 127.0.0.1
      port = 9999
      result_table_name = "socket"
      field_name = "info"
    }
}
transform {
}
sink {
  DorisSink {
      fenodes = "127.0.0.1:8030"
      user = root
      password = 123456
      database = test
      table = test_tbl
      batch_size = 5
      max_retries = 1
      interval = 5000
    }
}
```

Start command

```
sh bin/start-seatunnel-flink.sh --config config/flink.streaming.conf
```

Spark Sink Doris

Plugin Code

Seatunnel Spark Sink Doris plugin code

Options

name	type	required	default value	engine
fenodes	string	yes	-	Spark
database	string	yes	-	Spark
table	string	yes	-	Spark
user	string	yes	-	Spark
password	string	yes	-	Spark
batch_size	int	yes	100	Spark
doris.*	string	no	-	Spark

`fenodes [string]`

Doris FE address:8030

`database [string]`

Doris target database name

`table [string]`

Doris target table name

`user [string]`

Doris user name

`password [string]`

Doris user's password

`batch_size [string]`

Doris number of submissions per batch

`doris. [string]` Doris stream_load properties,you can use 'doris.' prefix + stream_load properties

More Doris stream_load Configurations

Examples

Hive to Doris

Config properties

```
env{
  spark.app.name = "hive2doris-template"
}

spark {
```

```
  spark.sql.catalogImplementation = "hive"
}

source {
  hive {
    preSql = "select * from tmp.test"
    result_table_name = "test"
  }
}

transform {
}


sink {

Console {

  }

Doris {
   fenodes="xxxx:8030"
   database="gl_mint_dim"
   table="dim_date"
   user="root"
   password="root"
   batch_size=1000
   doris.column_separator="\t"
   doris.columns="date_key,date_value,day_in_year,day_in_month"
   }
}
```

Start command

```
sh bin/start-waterdrop-spark.sh --master local[4] --deploy-mode client --config ./config/spark.conf
```

Kettle Doris Plugin

Kettle Doris Plugin

Kettle Doris Plugin is used to write data from other data sources to Doris through Stream Load in Kettle.

This plug-in uses the Stream Load function of Doris to import data. It needs to be used in conjunction with the Kettle service.

About Kettle

Kettle is an open source ETL (Extract, Transform, Load) tool, first developed by Pentaho, Kettle is one of the core components of the Pentaho product suite, mainly used for data integration and data processing, and can easily complete the tasks of extracting data from various sources, cleaning and transforming data, and loading it into the target system.

For more information, please refer to: `https://pentaho.com/`

User Manual

Download Kettle and install

Kettle download address: https://pentaho.com/download/#download-pentaho After downloading, unzip it and run spoon.sh to start kettle You can also compile it yourself, refer to the Compilation Chapter

Compile Kettle Doris Plugin

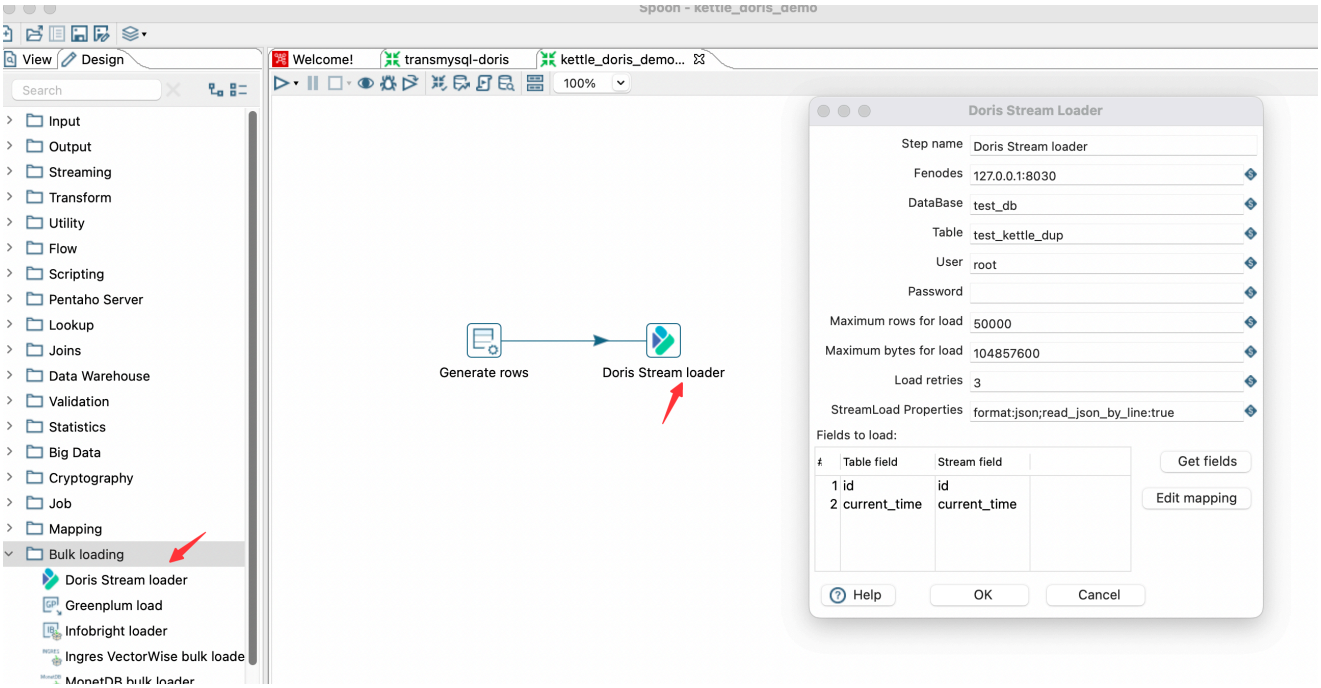
```
cd doris/extension/kettle
mvn clean package -DskipTests
```

After compiling, unzip the plug-in package and copy it to the plugins directory of kettle

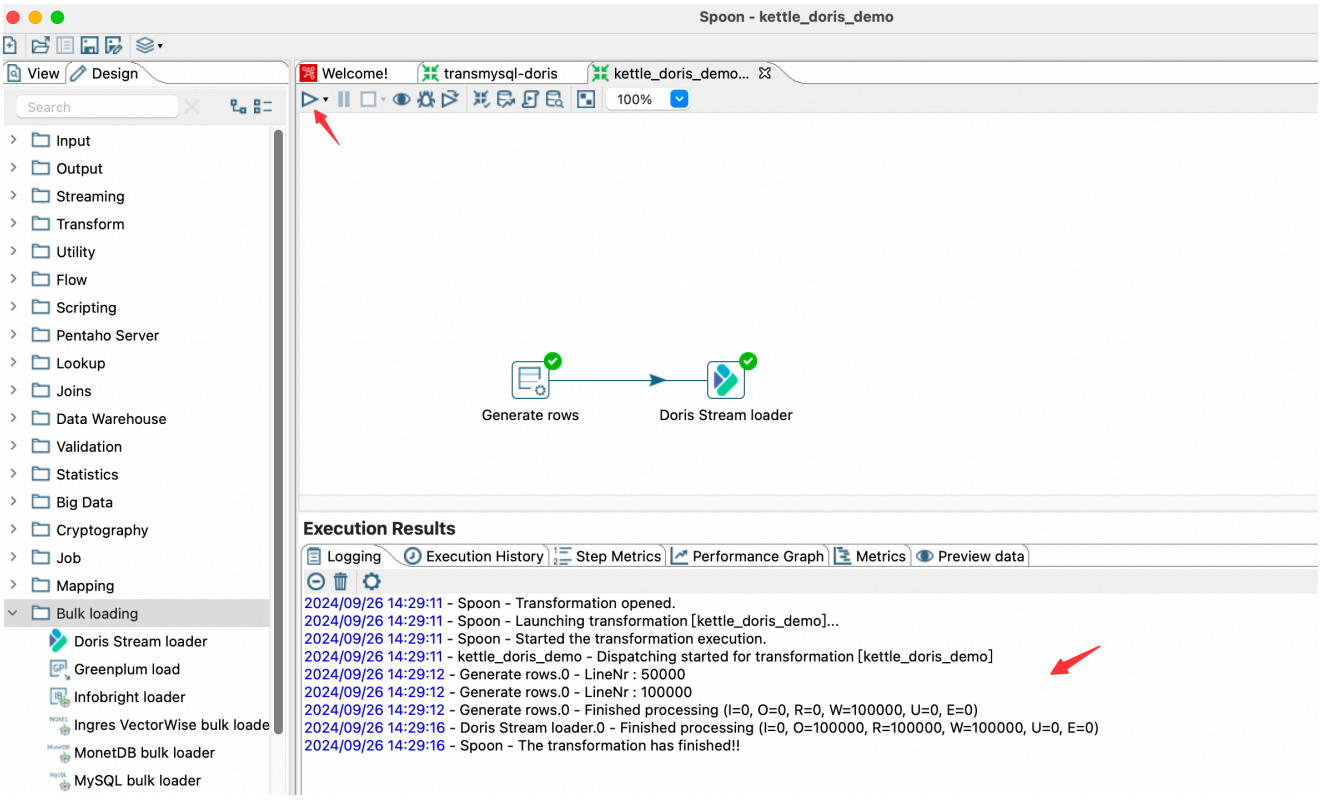
```
cd assemblies/plugin/target
unzip doris-stream-loader-plugins-9.4.0.0-343.zip
cp -r doris-stream-loader ${KETTLE_HOME}/plugins/
mvn clean package -DskipTests
```

Build a job

Find Doris Stream Loader in the batch loading in Kettle and build a job



Click Start Running the Job to complete data synchronization



Parameter Description

Key	Default Value	Required	Comment
Step name	–	Y	Step name

Key	Default Value	Required	Comment
fenodes	–	Y	Doris FE http address, supports multiple addresses, separated by commas
Database	–	Y	Doris write database
Target table	–	Y	Doris's write table
Username	–	Y	Username to access Doris
Password	–	N	Password to access Doris
Maximum number of rows for a single import	10000	N	Maximum number of rows for a single import
Maximum bytes for a single import	10485760 (10MB)	N	Maximum byte size for a single import
Number of import retries	3	N	Number of retries after import failure
StreamLoad properties	–	N	Streamload request header
Delete Mode	N	N	Whether to enable delete mode. By default, Stream Load performs insert operations. After the delete mode is enabled, all Stream Load writes are delete operations.

Kyuubi

Use Kyuubi with Doris

Introduction

Apache Kyuubi is a distributed and multi-tenant gateway to provide serverless SQL on Data Warehouses and Lakehouses. Apache Kyuubi is providing varied protocols like Thrift, Trino, MySQL etc., to the engines including Spark, Flink, Hive, JDBC, etc. Doris could be connected as JDBC data source with Doris dialect supported in Apache Kyuubi. Apache Kyuubi also provides a series of useful features including HA, service discovery, unified authentication, engine lifecycle management, etc.

Usage

Download Apache Kyuubi

Download Apache Kyuubi from https://kyuubi.apache.org/zh/releases.html

Get Apache Kyuubi 1.6.0 or above and extract it to folder.

Config Doris as Kyuubi data source

- Update Kyuubi configurations in `$KYUUBI_HOME/conf/kyuubi-defaults.conf`

```
kyuubi.engine.type=jdbc
kyuubi.engine.jdbc.type=doris
kyuubi.engine.jdbc.driver.class=com.mysql.cj.jdbc.Driver
kyuubi.engine.jdbc.connection.url=jdbc:mysql://xxx:xxx
kyuubi.engine.jdbc.connection.user=***
kyuubi.engine.jdbc.connection.password=***
```

Configuration	Description
kyuubi.engine.type	Engine Type, specify to `jdbc`
kyuubi.engine.jdbc.type	JDBC service type, specify to `doris`
kyuubi.engine.jdbc.driver.class	JDBC driver class name, specify to `com.mysql.cj.jdbc.Driver`
kyuubi.engine.jdbc.connection.url	JDBC url to Doris FE
kyuubi.engine.jdbc.connection.user	JDBC username
kyuubi.engine.jdbc.connection.password	JDBC password

- For other configuration in Apache Kyuubi, please refer to Apache Kyuubi Configuration Docs .

Add MySQL JDBC Driver

Copy the Mysql JDBC Driver `mysql-connector-j-8.X.X.jar` to `$KYUUBI_HOME/externals/engines/jdbc`.

Start Kyuubi Server

Run `$KYUUBI_HOME/bin/kyuubi start`. After started, port 10009 by default is listened by Kyuubi Server with Thrift protocol.

Example

The following example shows basic example of querying Doris with Kyuubi with beeline CLI in Thrift protocol.

Connect to Kyuubi with Beeline

```
$ $KYUUBI_HOME/bin/beeline -u "jdbc:hive2://xxxx:10009/"
```

Execute Query to Kyuubi

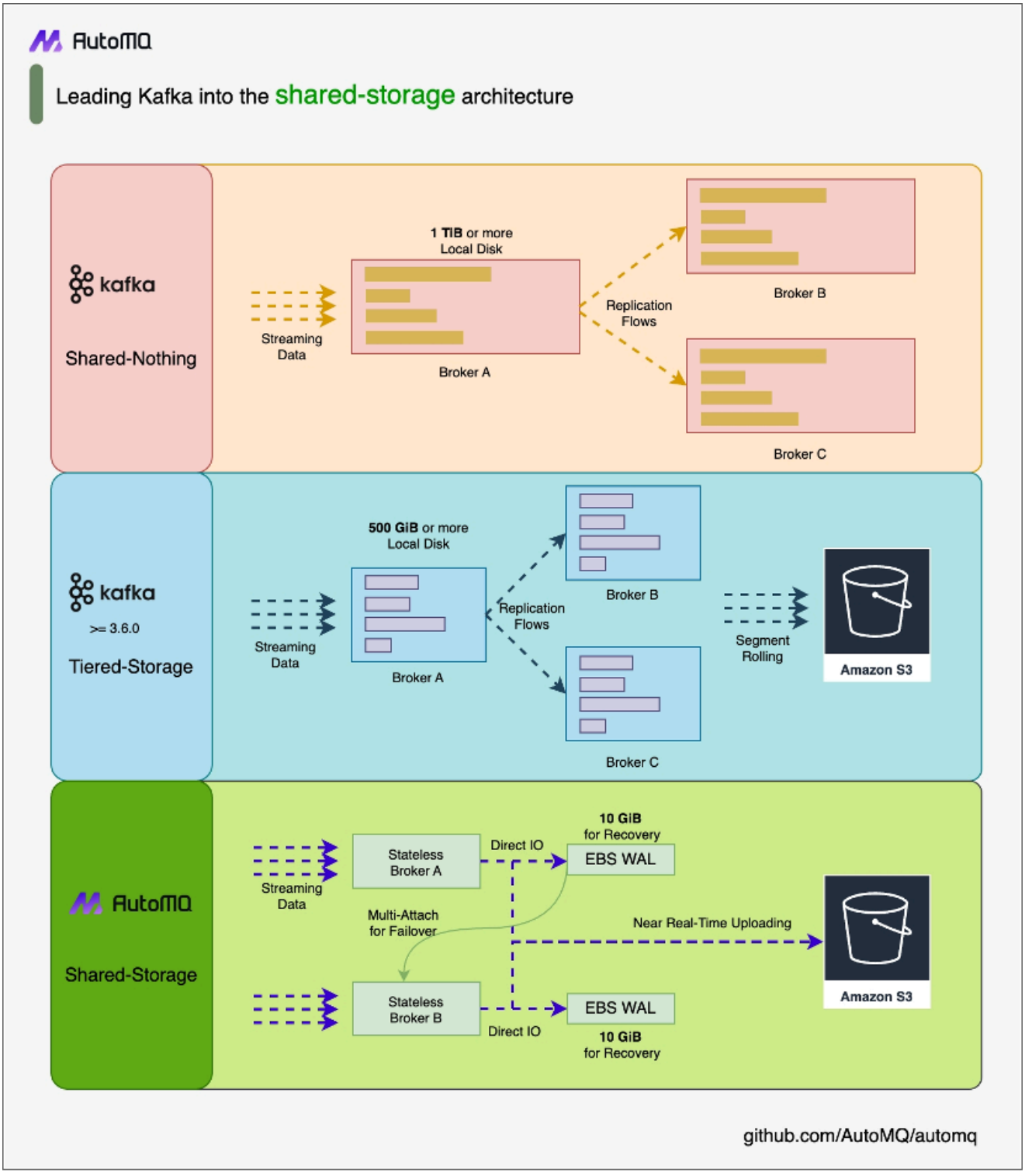
Execute query statement `select * from demo.expamle_tbl;` with query results returned.

```
0: jdbc:hive2://xxxx:10009/> select * from demo.example_tbl;

2023-03-07 09:29:14.771 INFO org.apache.kyuubi.operation.ExecuteStatement: Processing anonymous's query[bdc59dd0-ceea-4c02-8c3a-
23424323f5db]: PENDING_STATE -> RUNNING_STATE, statement:
select * from demo.example_tbl
2023-03-07 09:29:14.786 INFO org.apache.kyuubi.operation.ExecuteStatement: Query[bdc59dd0-ceea-4c02-8c3a-23424323f5db] in
FINISHED_STATE
2023-03-07 09:29:14.787 INFO org.apache.kyuubi.operation.ExecuteStatement: Processing anonymous's query[bdc59dd0-ceea-4c02-8c3a-
23424323f5db]: RUNNING_STATE -> FINISHED_STATE, time taken: 0.015 seconds
+----------+-------------+-------+------+------+----------------------+-------+-----------------+-----------------+
| user_id  |    date     | city  | age  | sex  |    last_visit_date     | cost  | max_dwell_time  | min_dwell_time  |
+----------+-------------+-------+------+------+----------------------+-------+-----------------+-----------------+
| 10000    | 2017-10-01  | Beijing | 20   | 0    | 2017-10-01 07:00:00.0  | 70    | 10              | 2               |
| 10001    | 2017-10-01  | Beijing | 30   | 1    | 2017-10-01 17:05:45.0  | 4     | 22              | 22              |
| 10002    | 2017-10-02  | Shanghai| 20   | 1    | 2017-10-02 12:59:12.0  | 400   | 5               | 5               |
| 10003    | 2017-10-02  | Guangzhou| 32   | 0    | 2017-10-02 11:20:00.0  | 60    | 11              | 11              |
| 10004    | 2017-10-01  | Shenzhen| 35   | 0    | 2017-10-01 10:00:15.0  | 200   | 3               | 3               |
| 10004    | 2017-10-03  | Shenzhen| 35   | 0    | 2017-10-03 10:20:22.0  | 22    | 6               | 6               |
+----------+-------------+-------+------+------+----------------------+-------+-----------------+-----------------+
6 rows selected (0.068 seconds)
```

AutoMQ Load

AutoMQ is a cloud-native fork of Kafka by separating storage to object storage like S3. It remains 100% compatible with Apache Kafka® while offering users up to a 10x cost-effective and 100x elasticity . Through its innovative shared storage architecture, it achieves capabilities such as reassign partitions in seconds, self-balancing and auto scaling in seconds while ensuring high throughput and low latency.



This article will explain how to use Apache Doris Routine Load to import data from AutoMQ into Doris. For more details on Routine Load, please refer to the Routine Load document.

Environment Preparation

Prepare Apache Doris and Test Data

Ensure that a working Apache Doris cluster is already set up. For demonstration purposes, we have deployed a test Apache Doris environment on Linux following the Quick Started document. Create databases and test tables:

```
create database automq_db;
CREATE TABLE automq_db.users (
                                id bigint NOT NULL,
                                name string NOT NULL,
                                timestamp string NULL,
                                status string NULL

) DISTRIBUTED BY hash (id) PROPERTIES ('replication_num' = '1');
```

Prepare Kafka Command Line Tools

Download the latest TGZ package from AutoMQ Releases and extract it. Assuming the extraction directory is $AUTOMQ_HOME, this article will use the scripts under $AUTOMQ_HOME/bin to create topics and generate test data.

Prepare AutoMQ and test data

Refer to the AutoMQ official deployment documentation to deploy a functional cluster, ensuring network connectivity between AutoMQ and Apache Doris. Quickly create a topic named example_topic in AutoMQ and write a test JSON data to it by following these steps.

Create Topic

Use the Apache Kafka® command line tool in AutoMQ to create the topic, ensuring that you have access to a Kafka environment and that the Kafka service is running. Here is an example command to create a topic:

```
$AUTOMQ_HOME/bin/kafka-topics.sh --create --topic exampleto_topic --bootstrap-server 127.0.0.1:9092  --partitions 1 --replication-factor 1
```

> Tips: When executing the command, replace `topic` and `bootstarp-server` with the actual AutoMQ Bootstrap Server address.

After creating the topic, you can use the following command to verify that the topic has been successfully created.

```
$AUTOMQ_HOME/bin/kafka-topics.sh --describe example_topic --bootstrap-server 127.0.0.1:9092
```

Generate test data

Create a JSON-formatted test data entry, corresponding to the table mentioned earlier.

```
{
  "id": 1,
  "name": "testuser",
  "timestamp": "2023-11-10T12:00:00",
  "status": "active"
}
```

Write test data

Use Kafka's command-line tools or a programming approach to write the test data to a topic named `example_topic`. Below is an example using the command-line tool:

```
echo '{"id": 1, "name": "testuser", "timestamp": "2023-11-10T12:00:00", "status": "active"}' | sh kafka-console-producer.sh --
broker-list 127.0.0.1:9092 --topic example_topic
```

To view the data just written to the topic, use the following command:

```
sh $AUTOMQ_HOME/bin/kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic example_topic --from-beginning
```

> Tips: When executing the command, replace `topic` and `bootstarp-server` with the actual AutoMQ Bootstrap Server address.

Create a Routine Load import job

In the Apache Doris command line, create a Routine Load job that accepts JSON data to continuously import data from an AutoMQ Kafka topic. For detailed parameter information of Routine Load, please refer to [Doris Routine Load].

```
CREATE ROUTINE LOAD automq_example_load ON users
COLUMNS(id, name, timestamp, status)
PROPERTIES
(
    "format" = "json",
    "jsonpaths" = "[\"$.id\",\"$.name\",\"$.timestamp\",\"$.status\"]"
 )
FROM KAFKA
(
    "kafka_broker_list" = "127.0.0.1:9092",
    "kafka_topic" = "example_topic",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

> Tips: When executing the command, you need to replace kafka_broker_list with the actual AutoMQ Bootstrap Server address.

Verify data import

First, check the status of the Routine Load import job to ensure that the task is running.

```
show routine load\G;
```

Then query the relevant tables in the Apache Doris database, and you will see that the data has been successfully imported.

```
select * from users;
+------+--------------+---------------------+--------+
| id   | name         | timestamp           | status |
+------+--------------+---------------------+--------+
|    1 | testuser     | 2023-11-10T12:00:00 | active |
|    2 | testuser     | 2023-11-10T12:00:00 | active |
+------+--------------+---------------------+--------+
2 rows in set (0.01 sec)
```

Hive Bitmap UDF

Hive UDF

Hive Bitmap UDF provides UDFs for generating bitmap and bitmap operations in hive tables. The bitmap in Hive is exactly the same as the Doris bitmap. The bitmap in Hive can be imported into doris through (spark bitmap load).

the main purpose: 1. Reduce the time of importing data into doris, and remove processes such as dictionary building and bitmap pre-aggregation; 2. Save hive storage, use bitmap to compress data, reduce storage cost; 3. Provide flexible bitmap operations in hive, such as: intersection, union, and difference operations, and the calculated bitmap can also be directly imported into doris; imported into doris;

How To Use

Create Bitmap type table in Hive

```
-- Example: Create Hive Bitmap Table
CREATE TABLE IF NOT EXISTS `hive_bitmap_table`(
  `k1`   int       COMMENT '',
  `k2`   String    COMMENT '',
  `k3`   String    COMMENT '',
  `uuid` binary    COMMENT 'bitmap'
) comment  'comment'

-- Example: Create Hive Table
CREATE TABLE IF NOT EXISTS `hive_table`(
  `k1`   int       COMMENT '',
  `k2`   String    COMMENT '',
  `k3`   String    COMMENT '',
  `uuid` int       COMMENT ''
) comment  'comment'
```

Hive Bitmap UDF Usage:

Hive Bitmap UDF used in Hive/Spark,First, you need to compile fe to get hive-udf-jar-with-dependencies.jar. Compilation preparation:If you have compiled the ldb source code, you can directly compile fe,If you have compiled the ldb source code, you can compile it directly. If you have not compiled the ldb source code, you need to manually install thrift, Reference:Setting Up dev env for FE.

```
--clone doris code
git clone https://github.com/apache/doris.git
cd doris
git submodule update --init --recursive
--install thrift
--Enter the fe directory
cd fe
--Execute the maven packaging command (All sub modules of fe will be packaged)
mvn package -Dmaven.test.skip=true
```

```
--You can also just package the hive-udf module
mvn package -pl hive-udf -am -Dmaven.test.skip=true
```

After packaging and compiling, enter the hive-udf directory and there will be a target directory,There will be hive-udf.jar package

```
-- Load the Hive Bitmap Udf jar package (Upload the compiled hive-udf jar package to HDFS)
add jar hdfs://node:9001/hive-udf-jar-with-dependencies.jar;
-- Create Hive Bitmap UDAF function
create temporary function to_bitmap as 'org.apache.doris.udf.ToBitmapUDAF' USING JAR 'hdfs://node:9001/hive-udf-jar-with-
dependencies.jar';
create temporary function bitmap_union as 'org.apache.doris.udf.BitmapUnionUDAF' USING JAR 'hdfs://node:9001/hive-udf-jar-with-
dependencies.jar';
-- Create Hive Bitmap UDF function
create temporary function bitmap_count as 'org.apache.doris.udf.BitmapCountUDF' USING JAR 'hdfs://node:9001/hive-udf-jar-with-
dependencies.jar';
create temporary function bitmap_and as 'org.apache.doris.udf.BitmapAndUDF' USING JAR 'hdfs://node:9001/hive-udf-jar-with-
dependencies.jar';
create temporary function bitmap_or as 'org.apache.doris.udf.BitmapOrUDF' USING JAR 'hdfs://node:9001/hive-udf-jar-with-
dependencies.jar';
create temporary function bitmap_xor as 'org.apache.doris.udf.BitmapXorUDF' USING JAR 'hdfs://node:9001/hive-udf-jar-with-
dependencies.jar';
-- Example: Generate bitmap by to_bitmap function and write to Hive Bitmap table
insert into hive_bitmap_table
select
    k1,
    k2,
    k3,
    to_bitmap(uuid) as uuid
from
    hive_table
group by
    k1,
    k2,
    k3
-- Example: The bitmap_count function calculate the number of elements in the bitmap
select k1,k2,k3,bitmap_count(uuid) from hive_bitmap_table
-- Example: The bitmap_union function calculate the grouped bitmap union
select k1,bitmap_union(uuid) from hive_bitmap_table group by k1
```

Hive Bitmap UDF Description

Hive Bitmap import into Doris

Method 1: Catalog (recommended)

When create a Hive table in the format specified as TEXT, for Binary type, Hive will be saved as a bash64 encoded string. Therefore, the binary data can be directly saved as Bitmap through bitmap_from_base64 function by using Doris's Hive Catalog.

Here is a full example:

1. Creating Hive Tables in Hive

```
CREATE TABLE IF NOT EXISTS `test`.`hive_bitmap_table`(
`k1`   int       COMMENT '',
`k2`   String    COMMENT '',
```

```
`k3`   String    COMMENT '',
`uuid` binary    COMMENT 'bitmap'
) stored as textfile
```

2. Creating a Catalog in Doris

```
CREATE CATALOG hive PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:9083'
);
```

3. Create Doris internal table

```
CREATE TABLE IF NOT EXISTS `test`.`doris_bitmap_table`(
    `k1`   int                   COMMENT '',
    `k2`   String                COMMENT '',
    `k3`   String                COMMENT '',
    `uuid` BITMAP  BITMAP_UNION  COMMENT 'bitmap'
)
AGGREGATE KEY(k1, k2, k3)
DISTRIBUTED BY HASH(`user_id`) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);
```

4. Inserting data from Hive into Doris

```
insert into doris_bitmap_table select k1, k2, k3, bitmap_from_base64(uuid) from hive.test.hive_bitmap_table;
```

Method 2: Spark Load

see details: Spark Load -> Basic operation -> Create load(Example 3: when the upstream data source is hive binary type table)

Install Error

Operation and Maintenance Error

This document is mainly used to record the common problems of operation and maintenance during the use of Doris. It will be updated from time to time.

The name of the BE binary that appears in this doc is `doris_be`, which was `palo_be` in previous versions.

Q1. Why is there always some tablet left when I log off the BE node through DECOMMISSION?

During the offline process, use show backends to view the tabletNum of the offline node, and you will observe that the number of tabletNum is decreasing, indicating that data shards are being migrated from this node. When the number is reduced to 0, the system will automatically delete the node. But in some cases, tabletNum will not change after it drops to a certain value. This is usually possible for two reasons:

1. The tablets belong to the table, partition, or materialized view that was just dropped. Objects that have just been deleted remain in the recycle bin. The offline logic will not process these shards. The time an object resides in the recycle bin can be modified by modifying the FE configuration parameter catalog_trash_expire_second. These tablets are disposed of when the object is removed from the recycle bin.
2. There is a problem with the migration task for these tablets. At this point, you need to view the errors of specific tasks through show proc `show proc "/cluster_balance"`.

For the above situation, you can first check whether there are unhealthy shards in the cluster through `show proc "/cluster_health/tablet_health";`. If it is 0, you can delete the BE directly through the drop backend statement. Otherwise, you also need to check the replicas of unhealthy shards in detail.

Q2. How should priorty_network be set?

priorty_network is a configuration parameter for both FE and BE. This parameter is mainly used to help the system select the correct network card IP as its own IP. It is recommended to explicitly set this parameter in any case to prevent the problem of incorrect IP selection caused by adding new network cards to subsequent machines.

The value of priorty_network is expressed in CIDR format. Divided into two parts, the first part is the IP address in dotted decimal, and the second part is a prefix length. For example 10.168.1.0/8 will match all 10.xx.xx.xx IP addresses, and 10.168.1.0/16 will match all 10.168.xx.xx IP addresses.

The reason why the CIDR format is used instead of specifying a specific IP directly is to ensure that all nodes can use a uniform configuration value. For example, there are two nodes: 10.168.10.1 and 10.168.10.2, then we can use 10.168.10.0/24 as the value of priorty_network.

Q3. What are the Master, Follower and Observer of FE?

First of all, make it clear that FE has only two roles: Follower and Observer. The Master is just an FE selected from a group of Follower nodes. Master can be regarded as a special kind of Follower. So when we were asked how many FEs a cluster had and what roles they were, the correct answer should be the number of all FE nodes, the number of Follower roles and the number of Observer roles.

All FE nodes of the Follower role will form an optional group, similar to the group concept in the Paxos consensus protocol. A Follower will be elected as the Master in the group. When the Master hangs up, a new Follower will be automatically selected as the Master. The Observer will not participate in the election, so the Observer will not be called Master.

A metadata log needs to be successfully written in most Follower nodes to be considered successful. For example, if there are 3 FEs, only 2 can be successfully written. This is why the number of Follower roles needs to be an odd number.

The role of Observer is the same as the meaning of this word. It only acts as an observer to synchronize the metadata logs that have been successfully written, and provides metadata reading services. He will not be involved in the logic of the majority writing.

Typically, 1 Follower + 2 Observer or 3 Follower + N Observer can be deployed. The former is simple to operate and maintain, and there is almost no consistency agreement between followers to cause such complex error situations (Most companies use this method). The latter can ensure the high availability of metadata writing. If it is a high concurrent query scenario, Observer can be added appropriately.

Q4. A new disk is added to the node, why is the data not balanced to the new disk?

The current Doris balancing strategy is based on nodes. That is to say, the cluster load is judged according to the overall load index of the node (number of shards and total disk utilization). And migrate data shards from high-load nodes to low-load nodes. If each node adds a disk, from the overall point of view of the node, the load does not change, so the balancing logic cannot be triggered.

In addition, Doris currently does not support balancing operations between disks within a single node. Therefore, after adding a new disk, the data will not be balanced to the new disk.

However, when data is migrated between nodes, Doris takes the disk into account. For example, when a shard is migrated from node A to node B, the disk with low disk space utilization in node B will be preferentially selected.

Here we provide 3 ways to solve this problem:

1. Rebuild the new table

   Create a new table through the create table like statement, and then use the insert into select method to synchronize data from the old table to the new table. Because when a new table is created, the data shards of the new table will be distributed in the new disk, so the data will also be written to the new disk. This method is suitable for situations where the amount of data is small (within tens of GB).

2. Through the Decommission command

The decommission command is used to safely decommission a BE node. This command will first migrate the data shards on the node to other nodes, and then delete the node. As mentioned earlier, during data migration, the disk with low disk utilization will be prioritized, so this method can “force” the data to be migrated to the disks of other nodes. When the data migration is completed, we cancel the decommission operation, so that the data will be rebalanced back to this node. When we perform the above steps on all BE nodes, the data will be evenly distributed on all disks of all nodes.

Note that before executing the decommission command, execute the following command to avoid the node being deleted after being offline.

```
admin set frontend config("drop_backend_after_decommission" = "false");
```

3. Manually migrate data using the API

   Doris provides HTTP API, which can manually specify the migration of data shards on one disk to another disk.

Q5. How to read FE/BE logs correctly?

In many cases, we need to troubleshoot problems through logs. The format and viewing method of the FE/BE log are described here.

1. FE

   FE logs mainly include:

   - fe.log: main log. Includes everything except fe.out.
   - fe.warn.log: A subset of the main log, only WARN and ERROR level logs are logged.
   - fe.out: log for standard/error output (stdout and stderr).
   - fe.audit.log: Audit log, which records all SQL requests received by this FE.

   A typical FE log is as follows:

   ```
   2021-09-16 23:13:22,502 INFO (tablet scheduler|43) [BeLoadRebalancer.selectAlternativeTabletsForCluster():85] cluster is
   balance: default_cluster with medium: HDD.skip
   ```

   - `2021-09-16 23:13:22,502`: log time.
   - `INFO`: log level, default is INFO.
   - `(tablet scheduler|43)`: thread name and thread id. Through the thread id, you can view the context information of this thread and check what happened in this thread.
   - `BeLoadRebalancer.selectAlternativeTabletsForCluster():85`: class name, method name and code line number.
   - `cluster is balance xxx`: log content.

   Usually, we mainly view the fe.log log. In special cases, some logs may be output to fe.out.

2. BE

   BE logs mainly include:

   - be.INFO: main log. This is actually a soft link, connected to the latest be.INFO.xxxx.

- be.WARNING: A subset of the main log, only WARN and FATAL level logs are logged. This is actually a soft link, connected to the latest be.WARN.xxxx.
- be.out: log for standard/error output (stdout and stderr).

A typical BE log is as follows:

```
I0916 23:21:22.038795 28087 task_worker_pool.cpp:1594] finish report TASK. master host: 10.10.10.10, port: 9222
```

- `I0916 23:21:22.038795`: log level and datetime. The capital letter I means INFO, W means WARN, and F means FATAL.
- `28087`: thread id. Through the thread id, you can view the context information of this thread and check what happened in this thread.
- `task_worker_pool.cpp:1594`: code file and line number.
- `finish report TASK xxx`: log content.

Usually we mainly look at the be.INFO log. In special cases, such as BE downtime, you need to check be.out.

Q6. How to troubleshoot the FE/BE node is down?

1. BE

   The BE process is a C/C++ process, which may hang due to some program bugs (memory out of bounds, illegal address access, etc.) or Out Of Memory (OOM). At this point, we can check the cause of the error through the following steps:

   1. View be.out

      The BE process realizes that when the program exits due to an exception, it will print the current error stack to be.out (note that it is be.out, not be.INFO or be.WARNING). Through the error stack, you can usually get a rough idea of where the program went wrong.

      Note that if there is an error stack in be.out, it is usually due to a program bug, and ordinary users may not be able to solve it by themselves. Welcome to the WeChat group, github discussion or dev mail group for help, and post the corresponding error stack, so that you can quickly Troubleshoot problems.

   2. dmesg

      If there is no stack information in be.out, the probability is that OOM was forcibly killed by the system. At this time, you can use the dmesg -T command to view the Linux system log. If a log similar to Memory cgroup out of memory: Kill process 7187 (doris_be) score 1007 or sacrifice child appears at the end, it means that it is caused by OOM.

      Memory problems can have many reasons, such as large queries, imports, compactions, etc. Doris is also constantly optimizing memory usage. Welcome to the WeChat group, github discussion or dev mail group for help.

   3. Check whether there are logs beginning with F in be.INFO.

Logs starting with F are Fatal logs. For example, F0916 , indicating the Fatal log on September 16th. Fatal logs usually indicate a program assertion error, and an assertion error will directly cause the process to exit (indicating a bug in the program). Welcome to the WeChat group, github discussion or dev mail group for help.

2. FE

FE is a java process, and the robustness is better than the C/C++ program. Usually the reason for FE to hang up may be OOM (Out-of-Memory) or metadata write failure. These errors usually have an error stack in fe.log or fe.out. Further investigation is required based on the error stack information.

Q7. About the configuration of data directory SSD and HDD, create table encounter error `Failed to find enough host with storage medium and tag`

Doris supports one BE node to configure multiple storage paths. Usually, one storage path can be configured for each disk. At the same time, Doris supports storage media properties that specify paths, such as SSD or HDD. SSD stands for high-speed storage device and HDD stands for low-speed storage device.

If the cluster only has one type of medium, such as all HDD or all SSD, the best practice is not to explicitly specify the medium property in be.conf. If encountering the error `Failed to find enough host with storage medium and tag` mentioned above, it is generally because be.conf only configures the SSD medium, while the table creation stage explicitly specifies `properties {"storage_medium" = "hdd"}`; similarly, if be.conf only configures the HDD medium, and the table creation stage explicitly specifies `properties {"storage_medium" = "ssd"}`, the same error will occur. The solution is to modify the properties parameter in the table creation to match the configuration; or remove the explicit configuration of SSD/HDD in be.conf.

By specifying the storage medium properties of the path, we can take advantage of Doris's hot and cold data partition storage function to store hot data in SSD at the partition level, while cold data is automatically transferred to HDD.

It should be noted that Doris does not automatically perceive the actual storage medium type of the disk where the storage path is located. This type needs to be explicitly indicated by the user in the path configuration. For example, the path "/path/to/data1.SSD" means that this path is an SSD storage medium. And "data1.SSD" is the actual directory name. Doris determines the storage media type based on the ".SSD" suffix after the directory name, not the actual storage media type. That is to say, the user can specify any path as the SSD storage medium, and Doris only recognizes the directory suffix and does not judge whether the storage medium matches. If no suffix is written, it will default to HDD.

In other words, ".HDD" and ".SSD" are only used to identify the "relative" "low speed" and "high speed" of the storage directory, not the actual storage medium type. Therefore, if the storage path on the BE node has no medium difference, the suffix does not need to be filled in.

Q8. Multiple FEs cannot log in when using Nginx to implement web UI load balancing

Doris can deploy multiple FEs. When accessing the Web UI, if Nginx is used for load balancing, there will be a constant prompt to log in again because of the session problem. This problem is actually a problem of session sharing. Nginx provides centralized session sharing. The solution, here we use the ip_hash technology in nginx, ip_hash can direct the request of an ip to the same backend, so that a client and a backend under this ip can establish a stable session, ip_hash is defined in the upstream configuration:

```
upstream doris.com {
   server 172.22.197.238:8030 weight=3;
   server 172.22.197.239:8030 weight=4;
   server 172.22.197.240:8030 weight=4;
   ip_hash;
}
```

The complete Nginx example configuration is as follows:

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/ngx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;
    #include /etc/nginx/custom/*.conf;
    upstream doris.com {
      server 172.22.197.238:8030 weight=3;
      server 172.22.197.239:8030 weight=4;
      server 172.22.197.240:8030 weight=4;
      ip_hash;
    }

    server {
        listen 80;
        server_name gaia-pro-bigdata-fe02;
        if ($request_uri ~ _load) {
           return 307 http://$host$request_uri ;
        }

        location / {
            proxy_pass http://doris.com;
            proxy_redirect default;
        }
        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
            root html;
```

```
        }
    }
}
```

Q9. FE fails to start, “wait catalog to be ready. FE type UNKNOWN” keeps scrolling in fe.log

There are usually two reasons for this problem:

1. The local IP obtained when FE is started this time is inconsistent with the last startup, usually because `priority_network` is not set correctly, which causes FE to match the wrong IP address when it starts. Restart FE after modifying `priority_network`.
2. Most Follower FE nodes in the cluster are not started. For example, there are 3 Followers, and only one is started. At this time, at least one other FE needs to be started, so that the FE electable group can elect the Master to provide services.

If the above situation cannot be solved, you can restore it according to the [metadata operation and maintenance document] (../admin-manual/trouble-shooting/metadata-operation.md) in the Doris official website document.

Q10. Lost connection to MySQL server at ‘reading initial communication packet’, system error: 0

If the following problems occur when using MySQL client to connect to Doris, this is usually caused by the different jdk version used when compiling FE and the jdk version used when running FE. Note that when using docker to compile the image, the default JDK version is openjdk 11, and you can switch to openjdk 8 through the command (see the compilation documentation for details).

Q11. recoveryTracker should overlap or follow on disk last VLSN of 4,422,880 recoveryFirst= 4,422,882 UNEXPECTED_STATE_FATAL

Sometimes when FE is restarted, the above error will occur (usually only in the case of multiple Followers). And the two values in the error differ by 2. Causes FE to fail to start.

This is a bug in bdbje that has not yet been resolved. In this case, you can only restore the metadata by performing the operation of failure recovery in Metadata Operation and Maintenance Documentation.

Q12. Doris compile and install JDK version incompatibility problem

When compiling Doris using Docker, start FE after compiling and installing, and the exception message `java.lang.Suchmethoderror: java.nio.ByteBuffer.limit (I)Ljava/nio/ByteBuffer;` appears, this is because the default in Docker It is JDK 11. If your installation environment is using JDK8, you need to switch the JDK environment to JDK8 in Docker. For the specific switching method, please refer to Compile Documentation

Q13. Error starting FE or unit test locally Cannot find external parser table action_table.dat

Run the following command

```
cd fe && mvn clean install -DskipTests
```

If the same error is reported, Run the following command

```
cp fe-core/target/generated-sources/cup/org/apache/doris/analysis/action_table.dat fe-core/target/classes/org/apache/doris/analysis
```

Q14. Doris upgrades to version 1.0 or later and reports error `Failed to set ciphers to use (2026) in MySQL appearance via ODBC.

This problem occurs after doris upgrades to version 1.0 and uses Connector/ODBC 8.0.x or higher. Connector/ODBC 8.0.x has multiple access methods, such as `/usr/lib64/libmyodbc8w.so` which is installed via yum and relies on `libssl.so.10` and `libcrypto.so.10`. In doris 1.0 onwards, openssl has been upgraded to 1.1 and is built into the doris binary package, so this can lead to openssl conflicts and errors like the following

```
ERROR 1105 (HY000): errCode = 2, detailMessage = driver connect Error: HY000 [MySQL][ODBC 8.0(w) Driver]SSL connection error: Failed to set ciphers to use (2026)
```

The solution is to use the `Connector/ODBC 8.0.28` version of ODBC Connector and select `Linux - Generic` in the operating system, this version of ODBC Driver uses openssl version 1.1. Or use a lower version of ODBC connector, e.g. Connector/ODBC 5.3.14. For details, see the ODBC exterior documentation.

You can verify the version of openssl used by MySQL ODBC Driver by

```
ldd /path/to/libmyodbc8w.so |grep libssl.so
```

If the output contains `libssl.so.10`, there may be problems using it, if it contains `libssl.so.1.1`, it is compatible with doris 1.0

Q14. After upgrading to version 1.2, the BE NoClassDefFoundError issue failed to start

Java UDF dependency error If the upgrade support starts be, the following Java `NoClassDefFoundError` error occurs

```
Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/doris/udf/IniUtil
Caused by: java.lang.ClassNotFoundException: org.apache.doris.udf.JniUtil
```

You need to download the Java UDF function dependency package of `apache-doris-java-udf-jar-with-dependencies-1.2.0` from the official website, put it in the lib directory under the BE installation directory, and then restart BE

Q15. After upgrading to version 1.2, BE startup shows Failed to initialize JNI

If the following `Failed to initialize JNI` error occurs when starting BE after upgrading

```
Failed to initialize JNI: Failed to find the library libjvm.so.
```

You need to set the `JAVA_HOME` environment variable, or set `JAVA_HOME` variable in be.conf and restart the BE node.

Q16. Docker: backend fails to start

This may be due to the CPU not supporting AVX2, check the backend logs with `docker logs -f be`. If the CPU does not support AVX2, the `apache/doris:1.2.2-be-x86_64-noavx2` image must be used, instead of `apache/doris:1.2.2-be-x86_64`. Note that the image version number will change over time, check Dockerhub for the most recent version.

Data Operation Error

Data Operation Error

This document is mainly used to record common problems of data operation during the use of Doris. It will be updated from time to time.

Q1. Use Stream Load to access FE's public network address to import data, but is redirected to the intranet IP?

When the connection target of stream load is the http port of FE, FE will only randomly select a BE node to perform the http 307 redirect operation, so the user's request is actually sent to a BE assigned by FE. The redirect returns the IP of the BE, that is, the intranet IP. So if you send the request through the public IP of FE, it is very likely that you cannot connect because it is redirected to the internal network address.

The usual way is to ensure that you can access the intranet IP address, or to assume a load balancer for all BE upper layers, and then directly send the stream load request to the load balancer, and the load balancer will transparently transmit the request to the BE node .

Q2. Does Doris support changing column names?

After version 1.2.0, when the `"light_schema_change"="true"` option is enabled, column names can be modified.

Before version 1.2.0 or when the `"light_schema_change"="true"` option is not enabled, modifying column names is not supported. The reasons are as follows:

Doris supports modifying database name, table name, partition name, materialized view (Rollup) name, as well as column type, comment, default value, etc. But unfortunately, modifying column names is currently not supported.

For some historical reasons, the column names are currently written directly to the data file. When Doris queries, it also finds the corresponding column through the class name. Therefore, modifying the column name is not only a simple metadata modification, but also involves data rewriting, which is a very heavy operation.

We do not rule out some compatible means to support lightweight column name modification operations in the future.

Q3. Does the table of the Unique Key model support creating a materialized view?

not support.

The table of the Unique Key model is a business-friendly table. Because of its unique function of deduplication according to the primary key, it can easily synchronize business databases with frequently changed data. Therefore, many users will first consider using the Unique Key model when accessing data into Doris.

But unfortunately, the table of the Unique Key model cannot establish a materialized view. The reason is that the essence of the materialized view is to "pre-compute" the data through pre-computation, so that the

calculated data is directly returned during the query to speed up the query. In the materialized view, the “pre-computed” data is usually some aggregated indicators, such as sum and count. At this time, if the data changes, such as update or delete, because the pre-computed data has lost detailed information, it cannot be updated synchronously. For example, a sum value of 5 may be 1+4 or 2+3. Because of the loss of detailed information, we cannot distinguish how this summation value is calculated, so we cannot meet the needs of updating.

Q4. tablet writer write failed, tablet_id=27306172, txn_id=28573520, err=-235 or -238

This error usually occurs during data import operations. The error code is -235. The meaning of this error is that the data version of the corresponding tablet exceeds the maximum limit (default 500, controlled by the BE parameter `max_tablet_version_num`), and subsequent writes will be rejected. For example, the error in the question means that the data version of the tablet 27306172 exceeds the limit.

This error is usually caused by the import frequency being too high, which is greater than the compaction speed of the backend data, causing versions to pile up and eventually exceed the limit. At this point, we can first pass the show tablet 27306172 statement, and then execute the show proc statement in the result to check the status of each copy of the tablet. The versionCount in the result represents the number of versions. If you find that a copy has too many versions, you need to reduce the import frequency or stop importing and observe whether the number of versions drops. If the number of versions does not decrease after the import is stopped, you need to go to the corresponding BE node to view the be.INFO log, search for the tablet id and compaction keyword, and check whether the compaction is running normally. For compaction tuning, you can refer to the ApacheDoris official account article: Doris Best Practices - Compaction Tuning (3)

The -238 error usually occurs when the same batch of imported data is too large, resulting in too many Segment files for a tablet (default is 200, controlled by the BE parameter `max_segment_num_per_rowset`). At this time, it is recommended to reduce the amount of data imported in one batch, or appropriately increase the BE configuration parameter value to solve the problem. Since version 2.0, users can enable segment compaction feature to reduce segment file number by setting `enable_segcompaction=true` in BE config.

Q5. tablet 110309738 has few replicas: 1, alive backends: [10003]

This error can occur during a query or import operation. Usually means that the copy of the corresponding tablet has an exception.

At this point, you can first check whether the BE node is down by using the show backends command. For example, the isAlive field is false, or the LastStartTime is a recent time (indicating that it has been restarted recently). If the BE is down, you need to go to the node corresponding to the BE and check the be.out log. If BE is down for abnormal reasons, the exception stack is usually printed in be.out to help troubleshoot the problem. If there is no error stack in be.out. Then you can use the linux command dmesg -T to check whether the process is killed by the system because of OOM.

If no BE node is down, you need to pass the show tablet 110309738 statement, and then execute the show proc statement in the result to check the status of each tablet copy for further investigation.

Q6. disk xxxxx on backend xxx exceed limit usage

Usually occurs in operations such as Import, Alter, etc. This error means that the usage of the corresponding disk corresponding to the BE exceeds the threshold (default 95%). In this case, you can first use the show

backends command, where MaxDiskUsedPct shows the usage of the disk with the highest usage on the corresponding BE. If If it exceeds 95%, this error will be reported.

At this point, you need to go to the corresponding BE node to check the usage in the data directory. The trash directory and snapshot directory can be manually cleaned to free up space. If the data directory occupies a large space, you need to consider deleting some data to free up space. For details, please refer to Disk Space Management.

Q7. Calling stream load to import data through a Java program may result in a Broken Pipe error when a batch of data is large.

Apart from Broken Pipe, some other weird errors may occur.

This situation usually occurs after enabling httpv2. Because httpv2 is an http service implemented using spring boot, and uses tomcat as the default built-in container. However, there seems to be some problems with tomcat's handling of 307 forwarding, so the built-in container was modified to jetty later. In addition, the version of apache http client in the java program needs to use the version after 4.5.13. In the previous version, there were also some problems with the processing of forwarding.

So this problem can be solved in two ways:

1. Disable httpv2

   Restart FE after adding enable_http_server_v2=false in fe.conf. However, the new version of the UI interface can no longer be used, and some new interfaces based on httpv2 can not be used. (Normal import queries are not affected).

2. Upgrade

   Upgrading to Doris 0.15 and later has fixed this issue.

Q8. Error -214 is reported when importing and querying

When performing operations such as import, query, etc., you may encounter the following errors:

```
failed to initialize storage reader. tablet=63416.1050661139.aa4d304e7a7aff9c-f0fa7579928c85a0, res=-214, backend=192.168.100.10
```

A -214 error means that the data version for the corresponding tablet is missing. For example, the above error indicates that the data version of the copy of tablet 63416 on the BE of 192.168.100.10 is missing. (There may be other similar error codes, which can be checked and repaired in the following ways).

Typically, if your data has multiple copies, the system will automatically repair these problematic copies. You can troubleshoot with the following steps:

First, check the status of each copy of the corresponding tablet by executing the `show tablet 63416` statement and executing the `show proc xxx` statement in the result. Usually we need to care about the data in the `Version` column.

Normally, the Version of multiple copies of a tablet should be the same. And it is the same as the VisibleVersion version of the corresponding partition.

You can view the corresponding partition version with `show partitions from tblx` (the partition corresponding to the tablet can be obtained in the `show tablet` statement.)

At the same time, you can also visit the URL in the CompactionStatus column in the `show proc` statement (just open it in a browser) to view more specific version information to check which versions are missing.

If there is no automatic repair for a long time, you need to use the `show proc "/cluster_balance"` statement to view the tablet repair and scheduling tasks currently being executed by the system. It may be because there are a large number of tablets waiting to be scheduled, resulting in a longer repair time. You can follow records in `pending_tablets` and `running_tablets`.

Further, you can use the `admin repair` statement to specify a table or partition to be repaired first. For details, please refer to `help admin repair`;

If it still can't be repaired, then in the case of multiple replicas, we use the `admin set replica status` command to force the replica in question to go offline. For details, see the example of setting the replica status to bad in `help admin set replica status`. (After set to bad, the copy will no longer be accessed. And it will be automatically repaired later. But before operation, you should make sure that other copies are normal)

Q9. Not connected to 192.168.100.1:8060 yet, server_id=384

We may encounter this error when importing or querying. If you go to the corresponding BE log, you may also find similar errors.

This is an RPC error, and there are usually two possibilities: 1. The corresponding BE node is down. 2. rpc congestion or other errors.

If the BE node is down, you need to check the specific downtime reason. Only the problem of rpc congestion is discussed here.

One case is OVERCROWDED, which means that the rpc source has a large amount of unsent data that exceeds the threshold. BE has two parameters associated with it:

1. `brpc_socket_max_unwritten_bytes`: The default value is 1GB. If the unsent data exceeds this value, an error will be reported. This value can be modified appropriately to avoid OVERCROWDED errors. (But this cures the symptoms but not the root cause, and there is still congestion in essence).
2. `tablet_writer_ignore_eovercrowded`: Default is false. If set to true, Doris will ignore OVERCROWDED errors during import. This parameter is mainly to avoid import failure and improve the stability of import.

The second is that the packet size of rpc exceeds max_body_size. This problem may occur if the query has a very large String type, or a bitmap type. It can be circumvented by modifying the following BE parameters:

```
brpc_max_body_size: default 3GB.
```

Q10. [Broker load] org.apache.thrift.transport.TTransportException: java.net.SocketException: Broken pipe

`org.apache.thrift.transport.TTransportException: java.net.SocketException: Broken pipe` during import.

The reason for this problem may be that when importing data from external storage (such as HDFS), because there are too many files in the directory, it takes too long to list the file directory. Here, the Broker RPC Timeout defaults to 10 seconds, and the timeout needs to be adjusted appropriately here. time.

Modify the `fe.conf` configuration file to add the following parameters:

```
broker_timeout_ms = 10000
##The default here is 10 seconds, you need to increase this parameter appropriately
```

Adding parameters here requires restarting the FE service.

Q11. [Routine load] ReasonOfStateChanged: ErrorReason{code=errCode = 104, msg='be 10004 abort task with reason: fetch failed due to requested offset not available on the broker: Broker: Offset out of range'}

The reason for this problem is that Kafka's cleanup policy defaults to 7 days. When a routine load task is suspended for some reason and the task is not restored for a long time, when the task is resumed, the routine load records the consumption offset, and This problem occurs when kafka has cleaned up the corresponding offset

So this problem can be solved with alter routine load:

View the smallest offset of kafka, use the ALTER ROUTINE LOAD command to modify the offset, and resume the task

```
ALTER ROUTINE LOAD FOR db.tb
FROM kafka
(
 "kafka_partitions" = "0",
 "kafka_offsets" = "xxx",
 "property.group.id" = "xxx"
);
```

Q12. ERROR 1105 (HY000): errCode = 2, detailMessage = (192.168.90.91)[CANCELLED][INTERNAL_ERROR]error setting certificate verify locations: CAfile: /etc/ssl/certs/ca-certificates.crt CApath: none

```
yum install -y ca-certificates
ln -s /etc/pki/ca-trust/extracted/openssl/ca-bundle.trust.crt /etc/ssl/certs/ca-certificates.crt
```

Q13. create partition failed. partition numbers will exceed limit variable max_auto_partition_num

To prevent accidental creation of too many partitions when importing data for auto-partitioned tables, we use the FE configuration item `max_auto_partition_num` to control the maximum number of partitions to be created automatically for such tables. If you really need to create more partitions, please modify this config item of FE Master node.

SQL Error

SQL Error

Q1. Query error: Failed to get scan range, no queryable replica found in tablet: xxxx

This happens because the corresponding tablet does not find a copy that can be queried, usually because the BE is down, the copy is missing, etc. You can first pass the `show tablet tablet_id` statement and then execute the following `show proc` statement to view the replica information corresponding to this tablet and check whether the replica is complete. At the same time, you can also query the progress of replica scheduling and repair in the cluster through `show proc "/cluster_balance"` information.

For commands related to data copy management, please refer to Data Copy Management.

Q2. Show backends/frontends The information viewed is incomplete

After executing certain statements such as `show backends/frontends`, some columns may be found to be incomplete in the results. For example, the disk capacity information cannot be seen in the show backends result.

Usually this problem occurs when the cluster has multiple FEs. If users connect to non-Master FE nodes to execute these statements, they will see incomplete information. This is because some information exists only on the Master FE node. For example, BE's disk usage information, etc. Therefore, complete information can only be obtained after a direct connection to the Master FE.

Of course, users can also execute `set forward_to_master=true;` before executing these statements. After the session variable is set to true, some information viewing statements executed subsequently will be automatically forwarded to the Master FE to obtain the results. In this way, no matter which FE the user is connected to, the complete result can be obtained.

Q3. invalid cluster id: xxxx

This error may appear in the results of the show backends or show frontends commands. Usually appears in the error message column of an FE or BE node. The meaning of this error is that after the Master FE sends the heartbeat information to the node, the node finds that the cluster id carried in the heartbeat information is different from the cluster id stored locally, so it refuses to respond to the heartbeat.

The Master FE node of Doris will actively send heartbeats to each FE or BE node, and will carry a cluster_id in the heartbeat information. cluster_id is the unique cluster ID generated by the Master FE when a cluster is initialized. When the FE or BE receives the heartbeat information for the first time, the cluster_id will be saved locally in the form of a file. The file of FE is in the image/ directory of the metadata directory, and the BE has a cluster_id file in all data directories. After that, each time the node receives the heartbeat, it will compare the content of the local cluster_id with the content in the heartbeat. If it is inconsistent, it will refuse to respond to the heartbeat.

This mechanism is a node authentication mechanism to prevent receiving false heartbeat messages sent by nodes outside the cluster.

If needed to recover from this error. The first thing to do is to make sure that all the nodes are in the correct cluster. After that, for the FE node, you can try to modify the cluster_id value in the image/VERSION file in the metadata directory and restart the FE. For the BE node, you can delete all the cluster_id files in the data directory and restart the BE.

Q4. Unique Key model query results are inconsistent

In some cases, when a user uses the same SQL to query a table with a Unique Key model, the results of multiple queries may be inconsistent. And the query results always change between 2-3 kinds.

This may be because, in the same batch of imported data, there are data with the same key but different values, which will lead to inconsistent results between different replicas due to the uncertainty of the sequence of data overwriting.

For example, the table is defined as k1, v1. A batch of imported data is as follows:

```
1, "abc"
1, "def"
```

Then maybe the result of copy 1 is `1, "abc"`, and the result of copy 2 is `1, "def"`. As a result, the query results are inconsistent.

To ensure that the data sequence between different replicas is unique, you can refer to the Sequence Column function.

Q5. The problem of querying bitmap/hll type data returns NULL

In version 1.1.x, when vectorization is enabled, and the bitmap type field in the query data table returns a NULL result,

1. First you have to `set return_object_data_as_binary=true;`
2. Turn off vectorization `set enable_vectorized_engine=false;`
3. Turn off SQL cache `set [global] enable_sql_cache = false;`

This is because the bitmap / hll type is in the vectorized execution engine: the input is all NULL, and the output result is also NULL instead of 0

Q6. Error when accessing object storage: curl 77: Problem with the SSL CA cert

If the `curl 77: Problem with the SSL CA cert` error appears in the be.INFO log. You can try to solve it in the following ways:

1. Download the certificate at https://curl.se/docs/caextract.html: cacert.pem
2. Copy the certificate to the specified location: `sudo cp /tmp/cacert.pem /etc/ssl/certs/ca-certificates.crt`
3. Restart the BE node.

Q7. import error:“Message”: “[INTERNAL_ERROR]single replica load is disabled on BE.”

1. Make sure this parameters `enable_single_replica_load` in be.conf is set true
2. Restart the BE node.

Data Lakehouse FAQ

Certificate Issues

1. When querying, an error `curl 77: Problem with the SSL CA cert.` occurs. This indicates that the current system certificate is too old and needs to be updated locally.
    - You can download the latest CA certificate from `https://curl.haxx.se/docs/caextract.html`.
    - Place the downloaded `cacert-xxx.pem` into the `/etc/ssl/certs/` directory, for example: `sudo cp cacert-xxx.pem /etc/ssl/certs/ca-certificates.crt`.
2. When querying, an error occurs: `ERROR 1105 (HY000): errCode = 2, detailMessage = (x.x.x.x)[CANCELLED][INTERNAL_ERROR]error setting certificate verify locations: CAfile: /etc/ssl/certs/ca-certificates.crt CApath: none`.

```
yum install -y ca-certificates
ln -s /etc/pki/ca-trust/extracted/openssl/ca-bundle.trust.crt /etc/ssl/certs/ca-certificates.crt
```

Kerberos

1. When connecting to a Hive Metastore authenticated with Kerberos, an error `GSS initiate failed` is encountered.

    This is usually due to incorrect Kerberos authentication information. You can troubleshoot by following these steps:

    1. In versions prior to 1.2.1, the libhdfs3 library that Doris depends on did not enable gsasl. Please update to versions 1.2.2 and later.

    2. Ensure that correct keytab and principal are set for each component and verify that the keytab file exists on all FE and BE nodes.

        - `hadoop.kerberos.keytab`/`hadoop.kerberos.principal`: Used for Hadoop hdfs access, fill in the corresponding values for hdfs.
        - `hive.metastore.kerberos.principal`: Used for hive metastore.

    3. Try replacing the IP in the principal with a domain name (do not use the default `_HOST` placeholder).

    4. Ensure that the `/etc/krb5.conf` file exists on all FE and BE nodes.

2. When connecting to a Hive database through the Hive Catalog, an error occurs: `RemoteException: SIMPLE authentication is not enabled. Available:[TOKEN, KERBEROS]`.

    If the error occurs during the query when there are no issues with `show databases` and `show tables`, follow these two steps:

    - Place core-site.xml and hdfs-site.xml in the fe/conf and be/conf directories.
    - Execute Kerberos kinit on the BE node, restart BE, and then proceed with the query.

When encountering the error `GSSException: No valid credentials provided (Mechanism level: Failed to find any Kerberos Ticket)` while querying a table configured with Kerberos, restarting FE and BE nodes usually resolves the issue.

- Before restarting all nodes, configure `-Djavax.security.auth.useSubjectCredsOnly=false` in the JAVA_OPTS parameter in "`${DORIS_HOME}/be/conf/be.conf`" to obtain JAAS credentials information through the underlying mechanism rather than the application.
- Refer to JAAS Troubleshooting for solutions to common JAAS errors.

To resolve the error `Unable to obtain password from user` when configuring Kerberos in the Catalog:

- Ensure the principal used is listed in klist by checking with `klist -kt your.keytab`.
- Verify the catalog configuration for any missing settings such as `yarn.resourcemanager.principal`.
- If the above checks are fine, it may be due to the JDK version installed by the system's package manager not supporting certain encryption algorithms. Consider installing JDK manually and setting the `JAVA_HOME` environment variable.
- Kerberos typically uses AES-256 for encryption. For Oracle JDK, JCE must be installed. Some distributions of OpenJDK automatically provide unlimited strength JCE, eliminating the need for separate installation.
- JCE versions correspond to JDK versions; download the appropriate JCE zip package and extract it to the `$JAVA_HOME/jre/lib/security` directory based on the JDK version:
  - JDK6: JCE6
  - JDK7: JCE7
  - JDK8: JCE8

When encountering the error `java.security.InvalidKeyException: Illegal key size` while accessing HDFS with KMS, upgrade the JDK version to >= Java 8 u162 or install the corresponding JCE Unlimited Strength Jurisdiction Policy Files.

If configuring Kerberos in the Catalog results in the error `SIMPLE authentication is not enabled. Available:[TOKEN, KERBEROS]`, place the `core-site.xml` file in the "`${DORIS_HOME}/be/conf`" directory.

If accessing HDFS results in the error `No common protection layer between client and server`, ensure that the `hadoop.rpc.protection` properties on the client and server are consistent.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

    <property>
        <name>hadoop.security.authentication</name>
        <value>kerberos</value>
    </property>

</configuration>
```

When using Broker Load with Kerberos configured and encountering the error `Cannot locate default realm.`:

Add the configuration item `-Djava.security.krb5.conf=/your-path` to the `JAVA_OPTS` in the `start_broker.sh` script for Broker Load.

3. When using Kerberos configuration in the Catalog, the `hadoop.username` property cannot be used simultaneously.

4. Accessing Kerberos with JDK 17

   When running Doris with JDK 17 and accessing Kerberos services, you may encounter issues accessing due to the use of deprecated encryption algorithms. You need to add the `allow_weak_crypto=true` property in krb5.conf or upgrade the encryption algorithm in Kerberos.

   For more details, refer to: https://seanjmullan.org/blog/2021/09/14/jdk17#kerberos

JDBC Catalog

1. Error connecting to SQLServer via JDBC Catalog: `unable to find valid certification path to requested target`

   Add the `trustServerCertificate=true` option in the `jdbc_url`.

2. Connecting to MySQL database via JDBC Catalog results in Chinese character garbling or incorrect Chinese character query conditions

   Add `useUnicode=true&characterEncoding=utf-8` in the `jdbc_url`.

   > Note: Starting from version 1.2.3, when connecting to MySQL database via JDBC Catalog, these parameters will be automatically added.

3. Error connecting to MySQL database via JDBC Catalog: `Establishing SSL connection without server's identity verification is not recommended`

   Add `useSSL=true` in the `jdbc_url`.

4. When synchronizing MySQL data to Doris using JDBC Catalog, date data synchronization error occurs. Verify if the MySQL version matches the MySQL driver package, for example, MySQL 8 and above require the driver com.mysql.cj.jdbc.Driver.

5. When a single field is too large, a Java memory OOM occurs on the BE side during a query.

   When Jdbc Scanner reads data through JDBC, the session variable `batch_size` determines the number of rows processed in the JVM per batch. If a single field is too large, it may cause `field_size * batch_size` (approximate value, considering JVM static memory and data copy overhead) to exceed the JVM memory limit, resulting in OOM.

   Solutions:

   - Reduce the `batch_size` value by executing `set batch_size = 512;`. The default value is 4064.
   - Increase the BE JVM memory by modifying the `-Xmx` parameter in `JAVA_OPTS`. For example: `-Xmx8g`.

Hive Catalog

1. Accessing Iceberg or Hive table through Hive Catalog reports an error: `failed to get schema` or `Storage schema reading not supported`

   You can try the following methods:

   - Put the `iceberg` runtime-related jar package in the lib/ directory of Hive.
   - Configure in `hive-site.xml`:

     ```
     metastore.storage.schema.reader.impl=org.apache.hadoop.hive.metastore.SerDeStorageSchemaReader
     ```

     After the configuration is completed, you need to restart the Hive Metastore.
   - Add `"get_schema_from_table" = "true"` in the Catalog properties

     This parameter is supported since versions 2.1.10 and 3.0.6.

2. Error connecting to Hive Catalog: `Caused by: java.lang.NullPointerException`

   If the fe.log contains the following stack trace:

   ```
   Caused by: java.lang.NullPointerException
       at
   org.apache.hadoop.hive.ql.security.authorization.plugin.AuthorizationMetaStoreFilterHook.getFilteredObjects(AuthorizationMe
   taStoreFilterHook.java:78) ~[hive-exec-3.1.3-core.jar:3.1.3]
       at
   org.apache.hadoop.hive.ql.security.authorization.plugin.AuthorizationMetaStoreFilterHook.filterDatabases(AuthorizationMetaS
   toreFilterHook.java:55) ~[hive-exec-3.1.3-core.jar:3.1.3]
       at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.getAllDatabases(HiveMetaStoreClient.java:1548) ~[doris-
   fe.jar:3.1.3]
       at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.getAllDatabases(HiveMetaStoreClient.java:1542) ~[doris-
   fe.jar:3.1.3]
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) ~[?:1.8.0_181]
   ```

   Try adding `"metastore.filter.hook" = "org.apache.hadoop.hive.metastore.DefaultMetaStoreFilterHookImpl"` in the `create catalog` statement to resolve.

3. If after creating Hive Catalog, `show tables` works fine but querying results in `java.net.UnknownHostException: xxxxx`

   Add the following in the CATALOG's PROPERTIES:

   ```
   'fs.defaultFS' = 'hdfs://<your_nameservice_or_actually_HDFS_IP_and_port>'
   ```

4. Tables in orc format in Hive 1.x may encounter system column names in the underlying orc file schema as `_col0`, `_col1`, `_col2`, etc. In this case, add `hive.version` as 1.x.x in the catalog configuration to map with the column names in the hive table.

   ```
   CREATE CATALOG hive PROPERTIES (
       'hive.version' = '1.x.x'
   );
   ```

5. When querying table data using Catalog, errors related to Hive Metastore such as `Invalid method name` are encountered, set the `hive.version` parameter.

6. When querying a table in ORC format, if the FE reports `Could not obtain block` or `Caused by: java.lang.NoSuchFieldError: types`, it may be due to the FE accessing HDFS to retrieve file information and perform file splitting by default. In some cases, the FE may not be able to access HDFS. This can be resolved by adding the following parameter: `"hive.exec.orc.split.strategy" = "BI"`. Other options include HYBRID (default) and ETL.

7. In Hive, you can find the partition field values of a Hudi table, but in Doris, you cannot. Doris and Hive currently have different ways of querying Hudi. In Doris, you need to add the partition fields in the avsc file structure of the Hudi table. If not added, Doris will query with partition_val being empty (even if `hoodie.datasource.hive_sync.partition_fields=partition_val` is set).

```
{
    "type": "record",
    "name": "record",
    "fields": [{
        "name": "partition_val",
        "type": [
            "null",
            "string"
            ],
        "doc": "Preset partition field, empty string when not partitioned",
        "default": null
        },
        {
        "name": "name",
        "type": "string",
        "doc": "Name"
        },
        {
        "name": "create_time",
        "type": "string",
        "doc": "Creation time"
        }
    ]
}
```

8. When querying a Hive external table, if you encounter the error `java.lang.ClassNotFoundException: Class com.hadoop.compression.lzo.LzoCodec not found`, search for `hadoop-lzo-*.jar` in the Hadoop environment, place it in the `"${DORIS_HOME}/fe/lib/"` directory, and restart the FE. Starting from version 2.0.2, you can place this file in the `custom_lib/` directory of the FE (if it does not exist, create it manually) to prevent file loss when upgrading the cluster due to the lib directory being replaced.

9. When creating a Hive table specifying the serde as `org.apache.hadoop.hive.contrib.serde2.MultiDelimitserDe`, and encountering the error `storage schema reading not supported` when accessing the table, add the following configuration to the hive-site.xml file and restart the HMS service:

```
<property>
  <name>metastore.storage.schema.reader.impl</name>
  <value>org.apache.hadoop.hive.metastore.SerDeStorageSchemaReader</value>
</property>
```

10. Error: `java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty`. The complete error message in the FE log is as follows:

```
org.apache.doris.common.UserException: errCode = 2, detailMessage = S3 list path failed. path=s3://bucket/part-*,msg=errors
while get file status listStatus on s3://bucket: com.amazonaws.SdkClientException: Unable to execute HTTP request:
Unexpected error: java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty: Unable to
execute HTTP request: Unexpected error: java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must
be non-empty
org.apache.doris.common.UserException: errCode = 2, detailMessage = S3 list path exception. path=s3://bucket/part-*, err:
errCode = 2, detailMessage = S3 list path failed. path=s3://bucket/part-*,msg=errors while get file status listStatus on
s3://bucket: com.amazonaws.SdkClientException: Unable to execute HTTP request: Unexpected error:
java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty: Unable to execute HTTP
request: Unexpected error: java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty
org.apache.hadoop.fs.s3a.AWSClientIOException: listStatus on s3://bucket: com.amazonaws.SdkClientException: Unable to
execute HTTP request: Unexpected error: java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must
be non-empty: Unable to execute HTTP request: Unexpected error: java.security.InvalidAlgorithmParameterException: the
trustAnchors parameter must be non-empty
Caused by: com.amazonaws.SdkClientException: Unable to execute HTTP request: Unexpected error:
java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty
Caused by: javax.net.ssl.SSLException: Unexpected error: java.security.InvalidAlgorithmParameterException: the trustAnchors
parameter must be non-empty
Caused by: java.lang.RuntimeException: Unexpected error: java.security.InvalidAlgorithmParameterException: the trustAnchors
parameter must be non-empty
Caused by: java.security.InvalidAlgorithmParameterException: the trustAnchors parameter must be non-empty
```

    Try updating the CA certificate on the FE node using `update-ca-trust (CentOS/RockyLinux)`, and then restart the FE process.

11. BE error: `java.lang.InternalError`. If you see an error similar to the following in `be.INFO`:

```
W20240506 15:19:57.553396 266457 jni-util.cpp:259] java.lang.InternalError
        at org.apache.hadoop.io.compress.zlib.ZlibDecompressor.init(Native Method)
        at org.apache.hadoop.io.compress.zlib.ZlibDecompressor.<init>(ZlibDecompressor.java:114)
        at org.apache.hadoop.io.compress.GzipCodec$GzipZlibDecompressor.<init>(GzipCodec.java:229)
        at org.apache.hadoop.io.compress.GzipCodec.createDecompressor(GzipCodec.java:188)
        at org.apache.hadoop.io.compress.CodecPool.getDecompressor(CodecPool.java:183)
        at org.apache.parquet.hadoop.CodecFactory$HeapBytesDecompressor.<init>(CodecFactory.java:99)
        at org.apache.parquet.hadoop.CodecFactory.createDecompressor(CodecFactory.java:223)
        at org.apache.parquet.hadoop.CodecFactory.getDecompressor(CodecFactory.java:212)
        at org.apache.parquet.hadoop.CodecFactory.getDecompressor(CodecFactory.java:43)
```

    It is because the Doris built-in `libz.a` conflicts with the system environment's `libz.so`. To resolve this issue, first execute `export LD_LIBRARY_PATH=/path/to/be/lib:$LD_LIBRARY_PATH`, and then restart the BE process.

12. When inserting data into Hive, an error occurred as `HiveAccessControlException Permission denied: user [user_a] does not have [UPDATE] privilege on [database/table]`.

    Since after inserting the data, the corresponding statistical information needs to be updated, and this update operation requires the alter privilege. Therefore, the alter privilege needs to be added for this user on Ranger.

HDFS

1. When accessing HDFS 3.x, if you encounter the error `java.lang.VerifyError: xxx`, in versions prior to 1.2.1, Doris depends on Hadoop version 2.8. You need to update to 2.10.2 or upgrade Doris to versions

after 1.2.2.

2. Using Hedged Read to optimize slow HDFS reads. In some cases, high load on HDFS may lead to longer read times for data replicas on a specific HDFS, thereby slowing down overall query efficiency. The HDFS Client provides the Hedged Read feature. This feature initiates another read thread to read the same data if a read request exceeds a certain threshold without returning, and the result returned first is used.

   Note: This feature may increase the load on the HDFS cluster, so use it judiciously.

   You can enable this feature by:

   ```
   create catalog regression properties (
       'type'='hms',
       'hive.metastore.uris' = 'thrift://172.21.16.47:7004',
       'dfs.client.hedged.read.threadpool.size' = '128',
       'dfs.client.hedged.read.threshold.millis' = "500"
   );
   ```

   `dfs.client.hedged.read.threadpool.size` represents the number of threads used for Hedged Read, which are shared by an HDFS Client. Typically, for an HDFS cluster, BE nodes will share an HDFS Client.

   `dfs.client.hedged.read.threshold.millis` is the read threshold in milliseconds. When a read request exceeds this threshold without returning, a Hedged Read is triggered.

   When enabled, you can see the related parameters in the Query Profile:

   `TotalHedgedRead`: Number of times Hedged Read was initiated.

   `HedgedReadWins`: Number of successful Hedged Reads (times when the request was initiated and returned faster than the original request)

   Note that these values are cumulative for a single HDFS Client, not for a single query. The same HDFS Client can be reused by multiple queries.

3. `Couldn't create proxy provider class org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider`

   In the start scripts of FE and BE, the environment variable `HADOOP_CONF_DIR` is added to the CLASSPATH. If `HADOOP_CONF_DIR` is set incorrectly, such as pointing to a non-existent or incorrect path, it may load the wrong xxx-site.xml file, resulting in reading incorrect information.

   Check if `HADOOP_CONF_DIR` is configured correctly or remove this environment variable.

4. `BlockMissingExcetpion: Could not obtain block: BP-XXXXXXXXX No live nodes contain current block`

   Possible solutions include:

   - Use `hdfs fsck file -files -blocks -locations` to check if the file is healthy.
   - Check connectivity with datanodes using `telnet`.
   - Check datanode logs.

   If you encounter the following error:

`org.apache.hadoop.hdfs.server.datanode.DataNode: Failed to read expected SASL data transfer protection handshake from client at /XXX.XXX.XXX.XXX:XXXXX. Perhaps the client is running an older version of Hadoop which does not support SASL data transfer protection` it means that the current hdfs has enabled encrypted transmission, but the client has not, causing the error.

Use any of the following solutions:

- Copy hdfs-site.xml and core-site.xml to be/conf and fe/conf directories. (Recommended)
- In hdfs-site.xml, find the corresponding configuration `dfs.data.transfer.protection` and set this parameter in the catalog.

DLF Catalog

1. When using the DLF Catalog, if `Invalid address` occurs during BE reading JindoFS data, add the domain name appearing in the logs to IP mapping in `/etc/hosts`.
2. If there is no permission to read data, use the `hadoop.username` property to specify a user with permission.
3. The metadata in the DLF Catalog should be consistent with DLF. When managing metadata using DLF, newly imported partitions in Hive may not be synchronized by DLF, leading to inconsistencies between DLF and Hive metadata. To address this, ensure that Hive metadata is fully synchronized by DLF.

Other Issues

1. Query results in garbled characters after mapping Binary type to Doris

   Doris natively does not support the Binary type, so when mapping Binary types from various data lakes or databases to Doris, it is usually done using the String type. The String type can only display printable characters. If you need to query the content of Binary data, you can use the `TO_BASE64()` function to convert it to Base64 encoding before further processing.

2. Analyzing Parquet files

   When querying Parquet files, due to potential differences in the format of Parquet files generated by different systems, such as the number of RowGroups, index values, etc., sometimes it is necessary to check the metadata of Parquet files for issue identification or performance analysis. Here is a tool provided to help users analyze Parquet files more conveniently:

   1. Download and unzip Apache Parquet Cli 1.14.0
   2. Download the Parquet file to be analyzed to your local machine, assuming the path is `/path/to/file.parquet`
   3. Use the following command to analyze the metadata of the Parquet file:

      ```
      ./parquet-tools meta /path/to/file.parquet
      ```

   4. For more functionalities, refer to Apache Parquet Cli documentation

BI FAQ

Power BI

Q1. An error occurs when you use JDBC to pull data into Desktop Power BI. “Timeout expired. The timeout period elapsed prior to completion of the operation or the server is not responding”.

Usually, this is Power BI pulling the time timeout of the data source. When filling in the data source server and database, click the advanced option, which has a timeout time, set the time higher.

Q2. When the 2.1.x version uses JDBC to connect to Power BI, an error occurs “An error happened while reading data from the provider: the given key was not present in the dictionary”.

Run “show collation” in the database first. Generally, only utf8mb4_900_bin is displayed, and the charset is utf8mb4. The main reason for this error is that ID 33 needs to be found when connecting to Power BI. That is, rows with 33ids in the table need to be upgraded to version 2.1.5 or later.

Q3. Connection Doris Times error “Reading data from the provider times error index and count must refer to the location within the string”.

The cause of the problem is that global parameters are loaded during the connection process, and the SQL column names and values are the same

```
SELECT
@@max_allowed_packet  as max_allowed_packet, @@character_set_client ,@@character_set_connection ,
@@license,@@sql_mode ,@@lower_case_table_names , @@autocommit ;
```

The new optimizer can be turned off in the current version or upgraded to version 2.0.7 or 2.1.6 or later.

Q4. JDBC connection version 2.1.x error message “Character set ‘utf8mb3’ is not supported by.net.Framework”.

This problem is easily encountered in version 2.1.x. If this problem occurs, you need to upgrade the JDBC Driver to 8.0.32.

Tableau

Q1. Version 2.0.x reports that Tableau cannot connect to the data source with error code 37CE01A3.

Turn off the new optimizer in the current version or upgrade to 2.0.7 or later

Q2. SSL connection error:protocol version mismatch Failed to connect to the MySQL server

The cause of this error is that SSL authentication is enabled on Doris, but SSL connections are not used during the connection. You need to disable the enable_ssl variable in fe.conf.

Q3. Connection error Unsupported command(COM_STMT_PREPARED)

The MySQL driver version is improperly installed. Install the MySQL 5.1.x connection driver instead.

Data Integrity FAQ

Data Integrity Issues

This document is mainly used to record common data integrity issues encountered during the use of Doris. It will be updated periodically.

The term “duplicate key data in the table” refers to the appearance of duplicate key data in the merge-on-write Unique table. Duplicate key issues in merge-on-write Unique tables can be fixed by triggering a full compaction, while other types of integrity issues may require specific solutions based on the situation. For assistance, please contact community support.

Issue Description	Occurrence Conditions	Affected Versions	Fix Versions	Impacted Scope	Fix PR
Partial column updates during import in merge-on-write Unique table reintroduce previously deleted data	A `__DORIS_DELETE_SIGN__` column is specified during partial column updates, and the historical data contains rows marked as deleted by the `__DORIS_DELETE_SIGN__` column	<2.1.8	>=2.1.8	Partial Column Update	#46194
System-generated values for auto-increment column are 0 or duplicate	Network anomalies between BE and FE	<2.1.8	>=2.1.8	Auto-increment column	#43774
Stream Load import does not delete data that meets the `delete` condition in merge-on-write Unique table	Stream Load import with `merge_type: MERGE`, `partial_columns: true`, and `delete` parameters	<2.0.15, <2.17	>=2.0.15, >=2.17	Partial Column Update	#40730
Partial column update import results in unintended update of auto-increment column values	The table has auto-increment column on the Value column, and the partial column update import does not specify values for those auto-increment column	<2.1.6	>=2.1.6	Auto-increment column	#39996

Issue Description	Occurrence Conditions	Affected Versions	Fix Versions	Impacted Scope	Fix PR
Duplicate key data in the table	User adds sequence column functionality to a merge-on-write Unique table that does not support sequence columns using `ALTER TABLE tbl ENABLE FEATURE "SEQUENCE_LOAD" WITH ...`, followed by new imports	<2.0.15, <2.1.6	>=2.0.15, >=2.1.6		#39958
Partial column update import results in data corruption in merge-on-write Unique table	Concurrent partial column updates in merge-on-write Unique table, with BE restart during the import process	<2.0.15, <2.1.6	>=2.0.15, >=2.1.6	Partial Column Update	#38331
Duplicate key data in the table	Large single import in merge-on-write Unique table with sequence column triggers segment compaction	<2.0.15, <2.1.6	>=2.0.15, >=2.1.6		#38369
Duplicate key data in the table	Failed full clone on merge-on-write Unique table	<2.0.13, <2.1.5	>=2.0.13, >=2.1.5		#37001
Inconsistent multi-replica data in merge-on-write Unique table	Partial column update import with `__DORIS_DELETE_SIGN__` column on merge-on-write Unique table, and inconsistent Base Compaction progress across replicas during import	<2.0.15, <2.1.5	>=2.0.15, >=2.1.5	Partial Column Update	#36210
Duplicate key data in the table	Concurrent partial column updates and BE restart during import on merge-on-write Unique table	<2.0.11, <2.1.4	>=2.0.11, >=2.1.4	Partial Column Update	#35739

Load FAQ

General Load FAQ

Error “[DATA_QUALITY_ERROR] Encountered unqualified data”

Problem Description: Data quality error during loading.

Solution: - Stream Load and Insert Into operations will return an error URL, while for Broker Load you can check the error URL through the `Show Load` command. - Use a browser or curl command to access the error URL to view the specific data quality error reasons. - Use the strict_mode and max_filter_ratio parameters to control the acceptable error rate.

Error “[E-235] Failed to init rowset builder”

Problem Description: Error -235 occurs when the load frequency is too high and data hasn’t been compacted in time, exceeding version limits.

Solution: - Increase the batch size of data loading and reduce loading frequency. - Increase the `max_tablet_version_num` parameter in `be.conf`, it is recommended not to exceed 5000.

Error “[E-238] Too many segments in rowset”

Problem Description: Error -238 occurs when the number of segments under a single rowset exceeds the limit.

Common Causes: - The bucket number configured during table creation is too small. - Data skew occurs; consider using more balanced bucket keys.

Error “Transaction commit successfully, BUT data will be visible later”

Problem Description: Data load is successful but temporarily not visible.

Cause: Usually due to transaction publish delay caused by system resource pressure.

Error “Failed to commit kv txn […] Transaction exceeds byte limit”

Problem Description: In shared-nothing mode, too many partitions and tablets are involved in a single load, exceeding the transaction size limit.

Solution: - Load data by partition in batches to reduce the number of partitions involved in a single load. - Optimize table structure to reduce the number of partitions and tablets.

Extra “ in the last column of CSV file

Problem Description: Usually caused by Windows line endings.

Solution: Specify the correct line delimiter: `-H "line_delimiter:\r\n"`

CSV data with quotes imported as null

Problem Description: CSV data with quotes becomes null after import.

Solution: Use the `trim_double_quotes` parameter to remove double quotes around fields.

Stream Load

Reasons for Slow Loading

- Bottlenecks in CPU, IO, memory, or network card resources.
- Slow network between client machine and BE machines, can be initially diagnosed through ping latency from client to BE machines.
- Webserver thread count bottleneck, too many concurrent Stream Loads on a single BE (exceeding be.conf webserver_num_workers configuration) may cause thread count bottleneck.
- Memtable Flush thread count bottleneck, check BE metrics doris_be_flush_thread_pool_queue_size to see if queuing is severe. Can be resolved by increasing the be.conf flush_thread_num_per_store parameter.

Handling Special Characters in Column Names

When column names contain special characters, use single quotes with backticks to specify the columns parameter:

```
curl --location-trusted -u root:"" \
    -H 'columns:`@coltime`,colint,colvar' \
    -T a.csv \
    -H "column_separator:," \
    http://127.0.0.1:8030/api/db/loadtest/_stream_load
```

Routine Load

Major Bug Fixes

Issue Description	Trigger Conditions	Impact Scope	Temporary Solution	Affected Versions	Fixed Versions	Fix PR
When at least one job times out while connecting to Kafka, it affects the import of other jobs, slowing down global Routine Load imports.	At least one job times out while connecting to Kafka.	Shared-nothing and shared-storage	Stop or manually pause the job to resolve the issue.	<2.1.9 <3.0.5	2.1.9 3.0.5	#47530

Issue Description	Trigger Conditions	Impact Scope	Temporary Solution	Affected Versions	Fixed Versions	Fix PR
User data may be lost after restarting the FE Master.	The job's offset is set to OFFSET_END, and the FE is restarted.	Shared-storage	Change the consumption mode to OFFSET_BEGINNING.	3.0.2-3.0.4	3.0.5	#46149
A large number of small transactions are generated during import, causing compaction to fail and resulting in continuous -235 errors.	Doris consumes data too quickly, or Kafka data flow is in small batches.	Shared-nothing and shared-storage	Pause the Routine Load job and execute the following command: `ALTER ROUTINE LOAD FOR jobname FROM kafka ("property.enable.partition.eof" = "false");`	<2.1.8 <3.0.4	2.1.8 3.0.4	#45528, #44949, #39975
Kafka third-party library destructor hangs, causing data consumption to fail.	Kafka topic deletion (possibly other conditions).	Shared-nothing and shared-storage	Restart all BE nodes.	<2.1.8 <3.0.4	2.1.8 3.0.4	#44913
Routine Load scheduling hangs.	Timeout occurs when FE aborts a transaction in Meta Service.	Shared-storage	Restart the FE node.	<3.0.2	3.0.2	#41267
Routine Load restart issue.	Restarting BE nodes.	Shared-nothing and shared-storage	Manually resume the job.	<2.1.7 <3.0.2	2.1.7 3.0.2	#40728

Default Configuration Optimizations

Optimization Content	Applied Versions	Corresponding PR
Increased the timeout duration for Routine Load.	2.1.7 3.0.3	#42042, #40818
Adjusted the default value of `max_batch_interval`.	2.1.8 3.0.3	#42491
Removed the restriction on `max_batch_interval`.	2.1.5 3.0.0	#29071
Adjusted the default values of `max_batch_rows` and `max_batch_size`.	2.1.5 3.0.0	#36632

Observability Optimizations

Optimization Content	Applied Versions	Corresponding PR
Added observability-related metrics.	3.0.5	#48209, #48171, #48963

Error “failed to get latest offset”

Problem Description: Routine Load cannot get the latest Kafka offset.

Common Causes: - Usually due to network connectivity issues with Kafka. Verify by pinging or using telnet to test the Kafka domain name. - Timeout caused by third-party library bug, error: java.util.concurrent.TimeoutException: Waited X seconds

Error “failed to get partition meta: Local:’Broker transport failure”

Problem Description: Routine Load cannot get Kafka Topic Partition Meta.

Common Causes: - Usually due to network connectivity issues with Kafka. Verify by pinging or using telnet to test the Kafka domain name. - If using domain names, try configuring domain name mapping in /etc/hosts

Error “Broker: Offset out of range”

Problem Description: The consumed offset doesn’t exist in Kafka, possibly because it has been cleaned up by Kafka.

Solution: - Need to specify a new offset for consumption, for example, set offset to OFFSET_BEGINNING. - Need to set appropriate Kafka log cleanup parameters based on import speed: log.retention.hours, log.retention.bytes, etc.

Overview | Sql Data Types

Overview

Numeric Types

Doris supports the following numeric data types:

BOOLEAN

There are two possible values: 0 represents false, and 1 represents true.

For more info, please refer BOOLEAN。

Integer

All are signed integers. The differences among the INT types are the number of bytes occupied and the range of values they can represent:

- **TINYINT**: 1 byte, [-128, 127]
- **SMALLINT**: 2 bytes, [-32768, 32767]
- **INT**: 4 bytes, [-2147483648, 2147483647]
- **BIGINT**: 8 bytes, [-9223372036854775808, 9223372036854775807]
- **LARGEINT**: 16 bytes, [-2^127, 2^127 - 1]

Floating-point

Including imprecise floating-point types FLOAT and DOUBLE, corresponding to the `float` and `double` in common programming languages

Fixed-point

The precise fixed-point type DECIMAL, used in financial and other cases that require strict accuracy.

Date Types

Date types include DATE, TIME and DATETIME, DATE type only stores the date accurate to the day, DATETIME type stores the date and time, which can be accurate to microseconds. TIME type only stores the time, and **does not support the construction of the table storage for the time being, can only be used in the query process**.

Do calculation for datetime types or converting them to numeric types, please use functions like TIME_TO_SEC, DATE_DIFF, UNIX_TIMESTAMP . The result of directly converting them as numeric types as

not guaranteed.

For more information refer to DATE, TIME and DATETIME documents.

String Types

Doris supports both fixed-length and variable-length strings, including:

- **CHAR(M)**: A fixed-length string, where M is the byte length. The range for M is [1, 255].
- **VARCHAR(M)**: A variable-length string, where M is the maximum length. The range for M is [1, 65533].
- **STRING**: A variable-length string with a default maximum length of 1,048,576 bytes (1 MB). This maximum length can be increased up to 2,147,483,643 bytes (2 GB) by configuring the `string_type_length_soft_limit_bytes`setting.

Semi-Structured Types

Doris supports different semi-structured data types for JSON data processing, each tailored to different use cases.

- **ARRAY** / **MAP** / **STRUCT**: They support nested data and fixed schema, making them well-suited for analytical workloads such as user behavior and profile analysis, as well as querying data lake formats like Parquet. Due to the fixed schema, there is no overhead for dynamic schema inference, resulting in high write and analysis performance.
- **VARIANT**: It supports nested data and flexible schema. It is well-suited for analytical workloads such as log, trace, and IoT data analysis. It can accommodate any legal JSON data, which will be automatically expanded into sub-columns in a columnar storage format. This approach enables high compression rate in storage and high performance in data aggregation, filtering, and sorting.
- **JSON**: It supports nested data and flexible schema. It is optimized for high-concurrency point query use cases. The flexible schema allows for ingesting any legal JSON data, which will be stored in a binary format. Extracting fields from this binary JSON format is more than 2X faster than using regular JSON strings.

Aggregation Types

The aggregation data types store aggregation results or intermediate results during aggregation. They are used for accelerating aggregation-heavy queries.

- **BITMAP**: It is used for exact deduplication, such as in (UV) statistics and audience segmentation. It works in conjunction with BITMAP functions like `bitmap_union`, `bitmap_union_count`, `bitmap_hash`, and `bitmap_hash64`.
- **HLL**: It is used for approximate deduplication and provides better performance than `COUNT DISTINCT`. It works in conjunction with HLL functions like `hll_union_agg`, `hll_raw_agg`, `hll_cardinality`, and `hll_hash`.
- **QUANTILE_STATE**: It is used for approximate percentile calculations and offers better performance than the `PERCENTILE` function. It works with functions like `QUANTILE_PERCENT`, `QUANTILE_UNION`, and `TO_QUANTILE_STATE`.

- **AGG_STATE**: It is used to accelerate aggregations, utilized in combination with aggregation function combinators like state/merge/union.

IP Types

IP data types store IP addresses in a binary format, which is faster and more space-efficient for querying compared to storing them as strings. There are two supported IP data types:

- **IPv4**: It stores IPv4 addresses as a 4-byte binary value. It is used in conjunction with the `ipv4_*` family of functions.
- **IPv6**: It stores IPv6 addresses as a 16-byte binary value. It is used in conjunction with the `ipv6_*` family of functions.

BOOLEAN

BOOLEAN

Description

BOOL, BOOLEAN Like TINYINT, 0 stands for false and 1 for true.

keywords

BOOLEAN

TINYINT

TINYINT

Description

TINYINT 1 byte signed integer, range [-128, 127]

keywords

TINYINT

SMALLINT

SMALLINT

Description

SMALLINT 2-byte signed integer, range [-32768, 32767]

keywords

SMALLINT

INT

INT

Description

INT 4-byte signed integer, range [-2147483648, 2147483647]

keywords

INT

BIGINT

BIGINT

Description

BIGINT 8-byte signed integer, range [-9223372036854775808, 9223372036854775807]

keywords

BIGINT

LARGEINT

LARGEINT

Description

LARGEINT 16-byte signed integer, range [-2^127 + 1 ~ 2^127 - 1]

keywords

LARGEINT

FLOAT

FLOAT

Description

FLOAT 4-byte floating point number

keywords

FLOAT

DOUBLE

DOUBLE

Description

DOUBLE 8-byte floating point number

keywords

DOUBLE

DECIMAL

DECIMAL

DECIMAL

Description

```
DECIMAL(P[,S])
High-precision fixed-point number, where P represents the total count of significant digits (precision), and S is the count of
decimal digits in the fractional part, to the right of the decimal point.
The range of significant digits P is [1, MAX_P], where MAX_P=38 when enable_decimal256=false, and MAX_P=76 when
enable_decimal256=true.
The range of decimal places S is [0, P].

By default, precision is 38, and scale is 9(that is DECIMAL(38, 9)).

The default value of enable_decimal256 is false. Setting it to true can get more accurate results, but it will bring some
performance loss.

When outputting decimal types, the decimal point is always followed by S digits, even if the trailing digits are 0. For example, the
number 123.456 of type decimal(18, 6) will be output as 123.456000.
```

Precision Deduction

DECIMAL has a very complex set of type inference rules. For different expressions, different rules will be applied for precision inference.

Arithmetic Operations

Assuming e1(p1, s1) and e2(p2, s2) are two DECIMAL numbers, the precision deduction rules for operation results are as follows:

Operation	Result precision	Result scale	Result precision if overflow	Result scale if overflow	Intermediate e1 type	Intermediate e2 type
e1 + e2	max(p1 - s1,p2 - s2) + max(s1, s2) + 1	max(s1, s2)	MAX_P	min(MAX_P, p) - max(p1 - s1,p2 - s2)	Cast according to result	Cast according to result
e1 - e2	max(p1 - s1,p2 - s2) + max(s1, s2) + 1	max(s1, s2)	MAX_P	min(MAX_P, p) - max(p1 - s1,p2 - s2)	Cast according to result	Cast according to result
e1 * e2	p1 + p2	s1 + s2	MAX_P		Unchanged	Unchanged

Operation	Result precision	Result scale	Result precision if overflow	Result scale if overflow	Intermediate e1 type	Intermediate e2 type
e1 / e2	p1 + s2 + `div_precision_increment`	s1 + `div_precision_increment`	MAX_P		p cast according to result, s cast according to result+e2.scale	
e1 % e2	max(p1 - s1,p2 - s2) + max(s1, s2)	max(s1, s2)	MAX_P	min(MAX_P, p) - max(p1 - s1,p2 - s2)	Cast according to result	Cast according to result

In the rules for calculating `Result scale if overflow` in the table, `precision` refers to `precision` in the `Result precision` column, and `scale` refers to `scale` in the `Result scale` column.

`div_precision_increment` is a configuration parameter of FE, see div_precision_increment.

`decimal_overflow_scale` is a session variable of FE, which indicates the maximum number of decimal places that can be retained in the calculation result when the precision of the decimal value calculation result overflows. The default value is 6.

It is worth noting that the process of division calculation is as follows: DECIMAL(p1, s1) / DECIMAL(p2, s2) is first converted to DECIMAL(p1 + s2 + div_precision_increment, s1 + s2) / DECIMAL(p2, s2) and then the calculation is performed. Therefore, it is possible that DECIMAL(p1 + s2 + div_precision_increment, s1 + div_precision_increment) satisfies the range of DECIMAL, but due to the conversion to DECIMAL(p1 + s2 + div_precision_increment, s1 + s2), it exceeds the range, Doris will report an `Arithmetic overflow` error by default.

Examples

Multiplication No Overflow

```
create table test_decimal_mul_no_overflow(f1 decimal(19, 9), f2 decimal(19, 9)) properties('replication_num'='1');
insert into test_decimal_mul_no_overflow values('9999999999.999999999', '9999999999.999999999');
```

According to the calculation rules of the multiplication result precision, the result type is decimal(38, 18) and will not overflow:

```
explain verbose select f1, f2, f1 * f2 from test_decimal_mul_no_overflow;
+----------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                    |
+----------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                    |
|   OUTPUT EXPRS:                                                                                                                                    |
|     f1[#2]                                                                                                                                         |
|     f2[#3]                                                                                                                                         |
```

```
|     f1 * f2[#4]                                                
```

According to the calculation rules of the precision of the multiplication result, under the default configuration (`enable_decimal256`=false, `decimal_overflow_scale`=6, `div_precision_increment`=4), the normal calculated result type is `decimal(41, 11)`. The precision overflows and needs to be recalculated according to the overflow rules: `MAX_P - decimal_overflow_scale` = 38 - 6 = 32, `precision - scale` = 41 - 11 = 30 < 32, Rule 1 applies, the final result scale = min(11, 38 - 30) = 8, and the final result type is decimal(38, 8):

```
 explain verbose select f1, f2, f1 * f2 from test_decimal_mul_overflow1;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                           |
|   OUTPUT EXPRS:                                                                                                                                           |
|     f1[#2]                                                                                                                                                |
|     f2[#3]                                                                                                                                                |
|     f1 * f2[#4]                                                                                                                                           |
|   PARTITION: UNPARTITIONED                                                                                                                                |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   VRESULT SINK                                                                                                                                            |
|      MYSQL_PROTOCAL                                                                                                                                       |
|                                                                                                                                                           |
|   1:VEXCHANGE                                                                                                                                             |
|      offset: 0                                                                                                                                            |
|      distribute expr lists:                                                                                                                               |
|      tuple ids: 1N                                                                                                                                        |
|                                                                                                                                                           |
| PLAN FRAGMENT 1                                                                                                                                           |
|                                                                                                                                                           |
|   PARTITION: RANDOM                                                                                                                                       |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   STREAM DATA SINK                                                                                                                                        |
|     EXCHANGE ID: 01                                                                                                                                       |
|     UNPARTITIONED                                                                                                                                         |
|                                                                                                                                                           |
|   0:VOlapScanNode(59)                                                                                                                                     |
|      TABLE: test.test_decimal_mul_overflow1(test_decimal_mul_overflow1), PREAGGREGATION: ON                                                               |
|      partitions=1/1 (test_decimal_mul_overflow1)                                                                                                          |
|      tablets=10/10, tabletList=1750210355791,1750210355793,1750210355795 ...                                                                             |
|      cardinality=1, avgRowSize=3115.0, numNodes=1                                                                                                         |
|      pushAggOp=NONE                                                                                                                                       |
|      desc: 0                                                                                                                                              |
|      final projections: f1[#0], f2[#1], (f1[#0] * f2[#1])                                                                                                 |
|      final project output tuple id: 1                                                                                                                     |
|      tuple ids: 0                                                                                                                                         |
|                                                                                                                                                           |
| Tuples:                                                                                                                                                   |
| TupleDescriptor{id=0, tbl=test_decimal_mul_overflow1}                                                                                                     |
|   SlotDescriptor{id=0, col=f1, colUniqueId=0, type=decimalv3(20,5), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=1, col=f2, colUniqueId=1, type=decimalv3(21,6), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|                                                                                                                                                           |
| TupleDescriptor{id=1, tbl=test_decimal_mul_overflow1}                                                                                                     |
|   SlotDescriptor{id=2, col=f1, colUniqueId=0, type=decimalv3(20,5), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=3, col=f2, colUniqueId=1, type=decimalv3(21,6), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=4, col=null, colUniqueId=null, type=decimalv3(38,8), nullable=true, isAutoIncrement=false, subColPath=null}                           |
```

Calculation Results:

```
select f1, f2, f1 * f2 from test_decimal_mul_overflow1;
+----------------------+----------------------+--------------------------------------+
| f1                   | f2                   | f1 * f2                              |
+----------------------+----------------------+--------------------------------------+
| 12345678901234.12345 | 12345678901234.123456 | 152415787532377393748917544.09724464 |
+----------------------+----------------------+--------------------------------------+
```

If the value of `decimal_overflow_scale` is increased, for example, `set decimal_overflow_scale=9;`, the calculation is performed according to the overflow rule: `MAX_P - decimal_overflow_scale = 38 - 9 = 29`, `precision - scale = 41 - 11 = 30 > 29`, and `scale > decimal_overflow_scale`, overflow rule 3 applies, and the final calculated result type is: `decimal(38,9)`:

```
explain verbose select f1, f2, f1 * f2 from test_decimal_mul_overflow1;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                                 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                                 |
|   OUTPUT EXPRS:                                                                                                                                                 |
|     f1[#2]                                                                                                                                                      |
|     f2[#3]                                                                                                                                                      |
|     f1 * f2[#4]                                                                                                                                                 |
|   PARTITION: UNPARTITIONED                                                                                                                                      |
|                                                                                                                                                                 |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                     |
|                                                                                                                                                                 |
|   VRESULT SINK                                                                                                                                                  |
|      MYSQL_PROTOCAL                                                                                                                                             |
|                                                                                                                                                                 |
|   1:VEXCHANGE                                                                                                                                                   |
|      offset: 0                                                                                                                                                  |
|      distribute expr lists:                                                                                                                                     |
|      tuple ids: 1N                                                                                                                                              |
|                                                                                                                                                                 |
| PLAN FRAGMENT 1                                                                                                                                                 |
|                                                                                                                                                                 |
|   PARTITION: RANDOM                                                                                                                                             |
|                                                                                                                                                                 |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                     |
|                                                                                                                                                                 |
|   STREAM DATA SINK                                                                                                                                              |
|     EXCHANGE ID: 01                                                                                                                                             |
|     UNPARTITIONED                                                                                                                                               |
|                                                                                                                                                                 |
|   0:VOlapScanNode(59)                                                                                                                                           |
|      TABLE: test.test_decimal_mul_overflow1(test_decimal_mul_overflow1), PREAGGREGATION: ON                                                                     |
|      partitions=1/1 (test_decimal_mul_overflow1)                                                                                                                |
|      tablets=10/10, tabletList=1750210355963,1750210355965,1750210355967 ...                                                                                   |
|      cardinality=1, avgRowSize=3145.0, numNodes=1                                                                                                               |
|      pushAggOp=NONE                                                                                                                                             |
|      desc: 0                                                                                                                                                    |
|      final projections: f1[#0], f2[#1], (f1[#0] * f2[#1])                                                                                                       |
|      final project output tuple id: 1                                                                                                                           |
|      tuple ids: 0                                                                                                                                               |
|                                                                                                                                                                 |
| Tuples:                                                                                                                                                         |
| TupleDescriptor{id=0, tbl=test_decimal_mul_overflow1}                                                                                                           |
|   SlotDescriptor{id=0, col=f1, colUniqueId=0, type=decimalv3(20,5), nullable=true, isAutoIncrement=false, subColPath=null}                                      |
|   SlotDescriptor{id=1, col=f2, colUniqueId=1, type=decimalv3(21,6), nullable=true, isAutoIncrement=false, subColPath=null}                                      |
|                                                                                                                                                                 |
| TupleDescriptor{id=1, tbl=test_decimal_mul_overflow1}                                                                                                           |
|   SlotDescriptor{id=2, col=f1, colUniqueId=0, type=decimalv3(20,5), nullable=true, isAutoIncrement=false, subColPath=null}                                      |
```

```
|   SlotDescriptor{id=3, col=f2, colUniqueId=1, type=decimalv3(21,6), nullable=true, isAutoIncrement=false, subColPath=null}      |
|   SlotDescriptor{id=4, col=null, colUniqueId=null, type=decimalv3(38,9), nullable=true, isAutoIncrement=false, subColPath=null} |
```

Calculation Results:

```
select f1, f2, f1 * f2 from test_decimal_mul_overflow1;
+----------------------+-----------------------+---------------------------------------+
| f1                   | f2                    | f1 * f2                               |
+----------------------+-----------------------+---------------------------------------+
| 12345678901234.12345 | 12345678901234.123456 | 152415787532377393748917544.097244643 |
+----------------------+-----------------------+---------------------------------------+
```

If we continue to increase the value of `decimal_overflow_scale`, for example, `set decimal_overflow_scale=12;`, we calculate according to the overflow rule: `MAX_P - decimal_overflow_scale = 38 - 12 = 26`, `precision - scale = 41 - 11 = 30 > 26`, and `scale < decimal_overflow_scale`. In this case, overflow rule 2 applies, and the final calculated result type is: `decimal(38,11)`:

```
explain verbose select f1, f2, f1 * f2 from test_decimal_mul_overflow1;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                                 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                                 |
|   OUTPUT EXPRS:                                                                                                                                                 |
|     f1[#2]                                                                                                                                                      |
|     f2[#3]                                                                                                                                                      |
|     f1 * f2[#4]                                                                                                                                                 |
|   PARTITION: UNPARTITIONED                                                                                                                                      |
|                                                                                                                                                                 |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                     |
|                                                                                                                                                                 |
|   VRESULT SINK                                                                                                                                                  |
|      MYSQL_PROTOCAL                                                                                                                                             |
|                                                                                                                                                                 |
|   1:VEXCHANGE                                                                                                                                                   |
|      offset: 0                                                                                                                                                  |
|      distribute expr lists:                                                                                                                                     |
|      tuple ids: 1N                                                                                                                                              |
|                                                                                                                                                                 |
| PLAN FRAGMENT 1                                                                                                                                                 |
|                                                                                                                                                                 |
|   PARTITION: RANDOM                                                                                                                                             |
|                                                                                                                                                                 |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                     |
|                                                                                                                                                                 |
|   STREAM DATA SINK                                                                                                                                              |
|     EXCHANGE ID: 01                                                                                                                                             |
|     UNPARTITIONED                                                                                                                                               |
|                                                                                                                                                                 |
|   0:VOlapScanNode(59)                                                                                                                                           |
|      TABLE: test.test_decimal_mul_overflow1(test_decimal_mul_overflow1), PREAGGREGATION: ON                                                                     |
|      partitions=1/1 (test_decimal_mul_overflow1)                                                                                                                |
|      tablets=10/10, tabletList=1750210355963,1750210355965,1750210355967 ...                                                                                    |
|      cardinality=1, avgRowSize=3145.0, numNodes=1                                                                                                               |
|      pushAggOp=NONE                                                                                                                                             |
|      desc: 0                                                                                                                                                    |
|      final projections: f1[#0], f2[#1], (f1[#0] * f2[#1])                                                                                                       |
|      final project output tuple id: 1                                                                                                                           |
|      tuple ids: 0                                                                                                                                               |
|                                                                                                                                                                 |
| Tuples:                                                                                                                                                         |
```

```
| TupleDescriptor{id=0, tbl=test_decimal_mul_overflow1}                                                                 |
|   SlotDescriptor{id=0, col=f1, colUniqueId=0, type=decimalv3(20,5), nullable=true, isAutoIncrement=false, subColPath=null}       |
|   SlotDescriptor{id=1, col=f2, colUniqueId=1, type=decimalv3(21,6), nullable=true, isAutoIncrement=false, subColPath=null}       |
|                                                                                                                       |
| TupleDescriptor{id=1, tbl=test_decimal_mul_overflow1}                                                                 |
|   SlotDescriptor{id=2, col=f1, colUniqueId=0, type=decimalv3(20,5), nullable=true, isAutoIncrement=false, subColPath=null}       |
|   SlotDescriptor{id=3, col=f2, colUniqueId=1, type=decimalv3(21,6), nullable=true, isAutoIncrement=false, subColPath=null}       |
|   SlotDescriptor{id=4, col=null, colUniqueId=null, type=decimalv3(38,11), nullable=true, isAutoIncrement=false, subColPath=null} |
```

Calculation Results:

```
select f1, f2, f1 * f2 from test_decimal_mul_overflow1;
+---------------------+----------------------+----------------------------------------+
| f1                  | f2                   | f1 * f2                                |
+---------------------+----------------------+----------------------------------------+
| 12345678901234.12345 | 12345678901234.123456 | 152415787532377393748917544.09724464320 |
+---------------------+----------------------+----------------------------------------+
```

Enable decimal256 when multiplication overflow

```
create table test_decimal_mul_overflow_dec256(f1 decimal(38, 19), f2 decimal(38, 19)) properties('replication_num'='1');
insert into test_decimal_mul_overflow_dec256 values('9999999999999999999.9999999999999999999',
'9999999999999999999.9999999999999999999');
```

By default (`enable_decimal256`=false), the actual result of the multiplication will overflow. In this case, you can enable decimal256: `set enable_decimal256=true` to calculate the exact result, and the result type is decimal(76, 38):

```
set enable_decimal256=true;

select f1, f2, f1 * f2 from test_decimal_mul_overflow_dec256;
+----------------------------------------+----------------------------------------+-----------------------------------------------
--------------------------------+
| f1                                     | f2                                     | f1 * f2
|
+----------------------------------------+----------------------------------------+-----------------------------------------------
--------------------------------+
| 9999999999999999999.9999999999999999999 | 9999999999999999999.9999999999999999999 |
99999999999999999999999999999999999998.00000000000000000000000000000000000001 |
+----------------------------------------+----------------------------------------+-----------------------------------------------
--------------------------------+
```

Division No Overflow

```
create table test_decimal_div_no_overflow(f1 decimal(19, 9), f2 decimal(19, 9)) properties('replication_num'='1');

insert into test_decimal_div_no_overflow values('1234567890.123456789', '234567890.123456789');
```

According to the calculation rules of the division result precision, under the default configuration (`enable_decimal256`=false, `decimal_overflow_scale`=6, `div_precision_increment`=4), the normal calculated result type is `decimal(19 + 9 + 4, 9 + 4)`, that is, `decimal(32, 13)`, and the precision does not overflow. The final type of the result is `decimal(32, 13)`:

```
 explain verbose select f1, f2, f1 / f2 from test_decimal_div_no_overflow;
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                     |
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                     |
|   OUTPUT EXPRS:                                                                                                                                     |
|     f1[#2]                                                                                                                                          |
|     f2[#3]                                                                                                                                          |
|     f1 / f2[#4]                                                                                                                                     |
|   PARTITION: UNPARTITIONED                                                                                                                          |
|                                                                                                                                                     |
|   HAS_COLO_PLAN_NODE: false                                                                                                                         |
|                                                                                                                                                     |
|   VRESULT SINK                                                                                                                                      |
|      MYSQL_PROTOCAL                                                                                                                                 |
|                                                                                                                                                     |
|   1:VEXCHANGE                                                                                                                                       |
|      offset: 0                                                                                                                                      |
|      distribute expr lists:                                                                                                                         |
|      tuple ids: 1N                                                                                                                                  |
|                                                                                                                                                     |
| PLAN FRAGMENT 1                                                                                                                                     |
|                                                                                                                                                     |
|   PARTITION: RANDOM                                                                                                                                 |
|                                                                                                                                                     |
|   HAS_COLO_PLAN_NODE: false                                                                                                                         |
|                                                                                                                                                     |
|   STREAM DATA SINK                                                                                                                                  |
|     EXCHANGE ID: 01                                                                                                                                 |
|     UNPARTITIONED                                                                                                                                   |
|                                                                                                                                                     |
|   0:VOlapScanNode(59)                                                                                                                               |
|      TABLE: test_decimal.test_decimal_div_no_overflow(test_decimal_div_no_overflow), PREAGGREGATION: ON
|
|      partitions=1/1 (test_decimal_div_no_overflow)
|
|      tablets=10/10, tabletList=1750210335692,1750210335694,1750210335696 ...                                                                        |
|      cardinality=1, avgRowSize=0.0, numNodes=1                                                                                                      |
|      pushAggOp=NONE                                                                                                                                 |
|      desc: 0                                                                                                                                        |
|      final projections: f1[#0], f2[#1], (CAST(f1[#0] AS decimalv3(32,22)) / f2[#1])                                                                 |
|      final project output tuple id: 1                                                                                                               |
|      tuple ids: 0                                                                                                                                   |
|                                                                                                                                                     |
| Tuples:                                                                                                                                             |
| TupleDescriptor{id=0, tbl=test_decimal_div_no_overflow}
|
|   SlotDescriptor{id=0, col=f1, colUniqueId=0, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                          |
|   SlotDescriptor{id=1, col=f2, colUniqueId=1, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                          |
|                                                                                                                                                     |
| TupleDescriptor{id=1, tbl=test_decimal_div_no_overflow}
|
|   SlotDescriptor{id=2, col=f1, colUniqueId=0, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                          |
|   SlotDescriptor{id=3, col=f2, colUniqueId=1, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                          |
|   SlotDescriptor{id=4, col=null, colUniqueId=null, type=decimalv3(32,13), nullable=true, isAutoIncrement=false, subColPath=null}                    |

select f1, f2, f1 / f2 from test_decimal_div_no_overflow;
+---------------------+--------------------+-----------------+
| f1                  | f2                 | f1 / f2         |
+---------------------+--------------------+-----------------+
| 1234567890.123456789 | 234567890.123456789 | 5.2631580966759 |
+---------------------+--------------------+-----------------+
```

If you want the result to retain more decimal places, you can increase `div_precision_increment`, for example, `admin set frontend config('div_precision_increment'='8');`. Then according to the above calculation rules, the calculated result type is `decimal(36, 17)`:

```
admin set frontend config('div_precision_increment'='8');
explain verbose select f1, f2, f1 / f2 from test_decimal_div_no_overflow;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                           |
|   OUTPUT EXPRS:                                                                                                                                           |
|     f1[#2]                                                                                                                                                |
|     f2[#3]                                                                                                                                                |
|     f1 / f2[#4]                                                                                                                                           |
|   PARTITION: UNPARTITIONED                                                                                                                                |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   VRESULT SINK                                                                                                                                            |
|      MYSQL_PROTOCAL                                                                                                                                       |
|                                                                                                                                                           |
|   1:VEXCHANGE                                                                                                                                             |
|      offset: 0                                                                                                                                            |
|      distribute expr lists:                                                                                                                               |
|      tuple ids: 1N                                                                                                                                        |
|                                                                                                                                                           |
| PLAN FRAGMENT 1                                                                                                                                           |
|                                                                                                                                                           |
|   PARTITION: RANDOM                                                                                                                                       |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   STREAM DATA SINK                                                                                                                                        |
|     EXCHANGE ID: 01                                                                                                                                       |
|     UNPARTITIONED                                                                                                                                         |
|                                                                                                                                                           |
|   0:VOlapScanNode(59)                                                                                                                                     |
|      TABLE: test.test_decimal_div_no_overflow(test_decimal_div_no_overflow), PREAGGREGATION: ON
|
|      partitions=1/1 (test_decimal_div_no_overflow)
|
|      tablets=10/10, tabletList=1750210354910,1750210354912,1750210354914 ...                                                                             |
|      cardinality=1, avgRowSize=3120.0, numNodes=1                                                                                                         |
|      pushAggOp=NONE                                                                                                                                       |
|      desc: 0                                                                                                                                              |
|      final projections: f1[#0], f2[#1], (CAST(f1[#0] AS decimalv3(36,26)) / f2[#1])                                                                       |
|      final project output tuple id: 1                                                                                                                    |
|      tuple ids: 0                                                                                                                                         |
|                                                                                                                                                           |
| Tuples:                                                                                                                                                   |
| TupleDescriptor{id=0, tbl=test_decimal_div_no_overflow}
|
|   SlotDescriptor{id=0, col=f1, colUniqueId=0, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=1, col=f2, colUniqueId=1, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|                                                                                                                                                           |
| TupleDescriptor{id=1, tbl=test_decimal_div_no_overflow}
|
|   SlotDescriptor{id=2, col=f1, colUniqueId=0, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=3, col=f2, colUniqueId=1, type=decimalv3(19,9), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=4, col=null, colUniqueId=null, type=decimalv3(36,17), nullable=true, isAutoIncrement=false, subColPath=null}                          |
```

Calculation Result:

```
select f1, f2, f1 / f2 from test_decimal_div_no_overflow;
+----------------------+--------------------+--------------------+
| f1                   | f2                 | f1 / f2            |
+----------------------+--------------------+--------------------+
| 1234567890.123456789 | 234567890.123456789 | 5.26315809667590986 |
+----------------------+--------------------+--------------------+
```

Division Overflow Rule 1

```
create table test_decimal_div_overflow1(f1 decimal(27, 8), f2 decimal(27, 8)) properties('replication_num'='1');

insert into test_decimal_div_overflow1 values('123456789012345678.12345678', '23456789012345678.12345678');
```

According to the calculation rules of the division result precision, under the default configuration (`enable_decimal256`=false, `decimal_overflow_scale`=6, `div_precision_increment`=4), the normal calculated result type is `decimal(27 + 8 + 4, 8 + 4)`, that is, `decimal(39, 12)`. The precision overflowed and needs to be recalculated according to the overflow rules: `MAX_P` - `decimal_overflow_scale` = 38 - 6 = 32, `precision` - s1 = 39 - 8 = 31 < 32, so the `Result scale if overflow` rule 1 applies, and the result scale is (`MAX_P` - (`precision` - `s1`)) + `div_precision_increment` = (38 - (39 - 8)) + 4 = 11, and the result type is `decimal(38, 11)`:

```
explain verbose select f1, f2, f1 / f2 from test_decimal_div_overflow1;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                                 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                                 |
|   OUTPUT EXPRS:                                                                                                                                                 |
|     f1[#2]                                                                                                                                                      |
|     f2[#3]                                                                                                                                                      |
|     f1 / f2[#4]                                                                                                                                                 |
|   PARTITION: UNPARTITIONED                                                                                                                                      |
|                                                                                                                                                                 |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                     |
|                                                                                                                                                                 |
|   VRESULT SINK                                                                                                                                                  |
|      MYSQL_PROTOCAL                                                                                                                                             |
|                                                                                                                                                                 |
|   1:VEXCHANGE                                                                                                                                                   |
|      offset: 0                                                                                                                                                  |
|      distribute expr lists:                                                                                                                                     |
|      tuple ids: 1N                                                                                                                                              |
|                                                                                                                                                                 |
| PLAN FRAGMENT 1                                                                                                                                                 |
|                                                                                                                                                                 |
|   PARTITION: RANDOM                                                                                                                                             |
|                                                                                                                                                                 |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                     |
|                                                                                                                                                                 |
|   STREAM DATA SINK                                                                                                                                              |
|     EXCHANGE ID: 01                                                                                                                                             |
|     UNPARTITIONED                                                                                                                                               |
|                                                                                                                                                                 |
|   0:VOlapScanNode(59)                                                                                                                                           |
|      TABLE: test_decimal.test_decimal_div_overflow1(test_decimal_div_overflow1), PREAGGREGATION: ON
|
|      partitions=1/1 (test_decimal_div_overflow1)
|
|      tablets=10/10, tabletList=1750210336251,1750210336253,1750210336255 ...                                                                                    |
```

```
|      cardinality=1, avgRowSize=3455.0, numNodes=1                                                
```

```
|                                                
```

```
| PLAN FRAGMENT 0                                                
```

```
admin set frontend config('div_precision_increment'='8');

explain verbose select f1, f2, f1 / f2 from test_decimal;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                           |
|   OUTPUT EXPRS:                                                                                                                                           |
|     f1[#2]                                                                                                                                                |
|     f2[#3]                                                                                                                                                |
|     f1 / f2[#4]                                                                                                                                           |
|   PARTITION: UNPARTITIONED                                                                                                                                |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   VRESULT SINK                                                                                                                                            |
|      MYSQL_PROTOCAL                                                                                                                                       |
|                                                                                                                                                           |
|   1:VEXCHANGE                                                                                                                                             |
|      offset: 0                                                                                                                                            |
|      distribute expr lists:                                                                                                                               |
|      tuple ids: 1N                                                                                                                                        |
|                                                                                                                                                           |
| PLAN FRAGMENT 1                                                                                                                                           |
|                                                                                                                                                           |
|   PARTITION: RANDOM                                                                                                                                       |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   STREAM DATA SINK                                                                                                                                        |
|     EXCHANGE ID: 01                                                                                                                                       |
|     UNPARTITIONED                                                                                                                                         |
|                                                                                                                                                           |
|   0:VOlapScanNode(59)                                                                                                                                     |
|      TABLE: test_decimal.test_decimal(test_decimal), PREAGGREGATION: ON                                                                                   |
|      partitions=1/1 (test_decimal)                                                                                                                        |
|      tablets=10/10, tabletList=1750210334096,1750210334098,1750210334100 ...                                                                             |
|      cardinality=2, avgRowSize=3240.0, numNodes=1                                                                                                         |
|      pushAggOp=NONE                                                                                                                                       |
|      desc: 0                                                                                                                                              |
|      final projections: f1[#0], f2[#1], (CAST(f1[#0] AS decimalv3(38,16)) / f2[#1])                                                                       |
|      final project output tuple id: 1                                                                                                                    |
|      tuple ids: 0                                                                                                                                         |
|                                                                                                                                                           |
| Tuples:                                                                                                                                                   |
| TupleDescriptor{id=0, tbl=test_decimal}                                                                                                                   |
|   SlotDescriptor{id=0, col=f1, colUniqueId=0, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=1, col=f2, colUniqueId=1, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|                                                                                                                                                           |
| TupleDescriptor{id=1, tbl=test_decimal}                                                                                                                   |
|   SlotDescriptor{id=2, col=f1, colUniqueId=0, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=3, col=f2, colUniqueId=1, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=4, col=null, colUniqueId=null, type=decimalv3(38,12), nullable=true, isAutoIncrement=false, subColPath=null}                          |
```

Calculation result:

```
select f1, f2, f1 / f2 from test_decimal;
+-------------------------+------------------------+-----------------+
| f1                      | f2                     | f1 / f2         |
+-------------------------+------------------------+-----------------+
| 123456789012345678.1234 | 23456789012345678.1234 |  5.263158096675 |
+-------------------------+------------------------+-----------------+
```

If decimal256 is enabled (`set enable_decimal256 = true;`), the result precision is calculated normally without overflow, and the result type is `decimal(46, 8)`:

```
set enable_decimal256=true;

admin set frontend config('div_precision_increment'='4');

explain verbose select f1, f2, f1 / f2 from test_decimal;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                           |
|   OUTPUT EXPRS:                                                                                                                                           |
|     f1[#2]                                                                                                                                                |
|     f2[#3]                                                                                                                                                |
|     f1 / f2[#4]                                                                                                                                           |
|   PARTITION: UNPARTITIONED                                                                                                                                |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   VRESULT SINK                                                                                                                                            |
|      MYSQL_PROTOCAL                                                                                                                                       |
|                                                                                                                                                           |
|   1:VEXCHANGE                                                                                                                                             |
|      offset: 0                                                                                                                                            |
|      distribute expr lists:                                                                                                                               |
|      tuple ids: 1N                                                                                                                                        |
|                                                                                                                                                           |
| PLAN FRAGMENT 1                                                                                                                                           |
|                                                                                                                                                           |
|   PARTITION: RANDOM                                                                                                                                       |
|                                                                                                                                                           |
|   HAS_COLO_PLAN_NODE: false                                                                                                                               |
|                                                                                                                                                           |
|   STREAM DATA SINK                                                                                                                                        |
|     EXCHANGE ID: 01                                                                                                                                       |
|     UNPARTITIONED                                                                                                                                         |
|                                                                                                                                                           |
|   0:VOlapScanNode(59)                                                                                                                                     |
|      TABLE: test_decimal.test_decimal(test_decimal), PREAGGREGATION: ON                                                                                   |
|      partitions=1/1 (test_decimal)                                                                                                                        |
|      tablets=10/10, tabletList=1750210334096,1750210334098,1750210334100 ...                                                                             |
|      cardinality=2, avgRowSize=3240.0, numNodes=1                                                                                                         |
|      pushAggOp=NONE                                                                                                                                       |
|      desc: 0                                                                                                                                              |
|      final projections: f1[#0], f2[#1], (CAST(f1[#0] AS decimalv3(46,12)) / f2[#1])                                                                       |
|      final project output tuple id: 1                                                                                                                     |
|      tuple ids: 0                                                                                                                                         |
|                                                                                                                                                           |
| Tuples:                                                                                                                                                   |
| TupleDescriptor{id=0, tbl=test_decimal}                                                                                                                   |
|   SlotDescriptor{id=0, col=f1, colUniqueId=0, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=1, col=f2, colUniqueId=1, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|                                                                                                                                                           |
| TupleDescriptor{id=1, tbl=test_decimal}                                                                                                                   |
|   SlotDescriptor{id=2, col=f1, colUniqueId=0, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=3, col=f2, colUniqueId=1, type=decimalv3(38,4), nullable=true, isAutoIncrement=false, subColPath=null}                                |
|   SlotDescriptor{id=4, col=null, colUniqueId=null, type=decimalv3(46,8), nullable=true, isAutoIncrement=false, subColPath=null}                           |

select f1, f2, f1 / f2 from test_decimal;
+------------------------+------------------------+-------------+
| f1                     | f2                     | f1 / f2     |
+------------------------+------------------------+-------------+
```

```
| 123456789012345678.1234 | 23456789012345678.1234 |  5.26315809 |
+-------------------------+------------------------+-------------+
```

Division Overflow Rule 3

```
create table test_decimal_div_overflow3(f1 decimal(38, 7), f2 decimal(38, 7)) properties('replication_num'='1');

insert into test_decimal_div_overflow3 values('123456789012345678.1234567', '23456789012345678.1234567');
```

According to the calculation rules of the division result precision, under the default configuration (`enable_decimal256`=false, `decimal_overflow_scale`=6, `div_precision_increment`=4), the normal calculated result type is `decimal(38 + 7 + 4, 7 + 4)`, that is, `decimal(49, 11)`. The precision overflows and needs to be recalculated according to the overflow rules: `MAX_P` - `decimal_overflow_scale` = 38 - 6 = 32, `precision` - s1 = 49 - 7 = 42 > 32, s1 = 7 > `decimal_overflow_scale`, so the `Result scale if overflow` rule 3 is applied, and the result scale is `decimal_overflow_scale` + `div_precision_increment` = 6 + 4 = 10, and the result type is `decimal(38, 10)`:

```
explain verbose select f1, f2, f1 / f2 from test_decimal_div_overflow3;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String(Nereids Planner)                                                                                                                                       |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                                                                                                                       |
|   OUTPUT EXPRS:                                                                                                                                                       |
|     f1[#2]                                                                                                                                                            |
|     f2[#3]                                                                                                                                                            |
|     f1 / f2[#4]                                                                                                                                                       |
|   PARTITION: UNPARTITIONED                                                                                                                                            |
|                                                                                                                                                                       |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                           |
|                                                                                                                                                                       |
|   VRESULT SINK                                                                                                                                                        |
|      MYSQL_PROTOCAL                                                                                                                                                   |
|                                                                                                                                                                       |
|   1:VEXCHANGE                                                                                                                                                         |
|      offset: 0                                                                                                                                                        |
|      distribute expr lists:                                                                                                                                           |
|      tuple ids: 1N                                                                                                                                                    |
|                                                                                                                                                                       |
| PLAN FRAGMENT 1                                                                                                                                                       |
|                                                                                                                                                                       |
|   PARTITION: RANDOM                                                                                                                                                   |
|                                                                                                                                                                       |
|   HAS_COLO_PLAN_NODE: false                                                                                                                                           |
|                                                                                                                                                                       |
|   STREAM DATA SINK                                                                                                                                                    |
|     EXCHANGE ID: 01                                                                                                                                                   |
|     UNPARTITIONED                                                                                                                                                     |
|                                                                                                                                                                       |
|   0:VOlapScanNode(59)                                                                                                                                                 |
|      TABLE: test_decimal.test_decimal_div_overflow3(test_decimal_div_overflow3), PREAGGREGATION: ON
|
|      partitions=1/1 (test_decimal_div_overflow3)
|
|      tablets=10/10, tabletList=1750210336825,1750210336827,1750210336829 ...                                                                                          |
|      cardinality=1, avgRowSize=0.0, numNodes=1                                                                                                                        |
|      pushAggOp=NONE                                                                                                                                                   |
|      desc: 0                                                                                                                                                          |
|      final projections: f1[#0], f2[#1], (CAST(f1[#0] AS decimalv3(38,17)) / f2[#1])                                                                                   |
|      final project output tuple id: 1                                                                                                                                |
```

```
|      tuple ids: 0                                                
```

expanded. 2. Higher performance. The old version of DECIMAL requires 16 bytes in memory and 12 bytes in storage, while DECIMAL has made adaptive adjustments as shown below.

precision	Space occupied (memory/disk)
0 < precision <= 9	4 bytes
9 < precision <= 18	8 bytes
18 < precision <= 38	16 bytes
38 < precision <= 76	32 bytes

3. More complete precision deduction. For different expressions, different precision inference rules are applied to deduce the precision of the results.

keywords

DECIMAL

DATE | Date Time

DATE

DATE

DATE

Description

Syntax

date Date type, the current range of values is [‘0000-01-01’,‘9999-12-31’], and the default print form is ‘yyyy-MM-dd’.

example

```
SELECT DATE('2003-12-31 01:02:03');
+----------------------------+
| DATE('2003-12-31 01:02:03') |
+----------------------------+
| 2003-12-31                 |
+----------------------------+
```

keywords

DATE

TIME

TIME

name

TIME

description

TIME type Time type, can appear as a query result, does not support table storage for the time being. The storage range is `[-838:59:59, 838:59:59]`. Currently in Doris, the correctness of TIME as a result of calculations is guaranteed (e.g., functions such as `timediff`), but **manual CAST generation of the TIME type is not recommended**. The calculation of TIME type in constant folding is prohibited.

example

```
mysql> select timediff('2020-01-01 12:05:03', '2020-01-01 08:02:15');
+-----------------------------------------------------------------------------------------------------+
| timediff(cast('2020-01-01 12:05:03' as DATETIMEV2(0)), cast('2020-01-01 08:02:15' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------+
| 04:02:48                                                                                            |
+-----------------------------------------------------------------------------------------------------+
1 row in set (0.12 sec)

mysql> select timediff('2020-01-01', '2000-01-01');
+-------------------------------------------------------------------------------+
| timediff(cast('2020-01-01' as DATETIMEV2(0)), cast('2000-01-01' as DATETIMEV2(0))) |
+-------------------------------------------------------------------------------+
| 838:59:59                                                                     |
+-------------------------------------------------------------------------------+
1 row in set (0.11 sec)
```

keywords

```
TIME
```

DATETIME

DATETIME

DATETIME

Description

DATETIME([P]) Date and time type. The optional parameter P indicates the time precision and the value range is [0, 6], that is, it supports up to 6 decimal places (microseconds). 0 when not set. Value range is [‘0000-01-01 00:00:00[.000000]’,‘9999-12-31 23:59:59[.999999]’]. The form of printing is ‘yyyy-MM-dd HH:mm:ss.SSSSSS’

note

DATETIME supports temporal precision up to microseconds. When parsing imported DATETIME type data using the BE side (e.g. using Stream load, Spark load, etc.), or using the FE side with the Nereids on, decimals exceeding the current precision will be **rounded**. Inserting a DATETIME value with a fractional seconds part into a column of the same type but having fewer fractional digits results in **rounded**. DATETIME reads support resolving the time zone in the format of the original DATETIME literal followed by the time zone:

```
<date> <time>[<timezone>]
```

For the specific supported formats for `<timezone>`, see timezone. Note that the `DATE`, `DATEV2`, `DATETIME`, and `DATETIMEV2` types **don’t** contain time zone information. For example, if an input time string “2012-12-12 08:00:00+08:00” is parsed and converted to the current time zone “+02:00”, and the actual value “2012-12-12 02:00:00” is stored in the DATETIME column, the value itself will not change, no matter how much the cluster environment variables are changed.

example

```
mysql> select @@time_zone;
+----------------+
| @@time_zone    |
+----------------+
| Asia/Hong_Kong |
+----------------+
1 row in set (0.11 sec)

mysql> insert into dtv23 values ("2020-12-12 12:12:12Z"), ("2020-12-12 12:12:12GMT"), ("2020-12-12 12:12:12+02:00"), ("2020-12-12
12:12:12America/Los_Angeles");
Query OK, 4 rows affected (0.17 sec)

mysql> select * from dtv23;
+-------------------------+
| k0                      |
+-------------------------+
| 2020-12-12 20:12:12.000 |
| 2020-12-12 20:12:12.000 |
| 2020-12-13 04:12:12.000 |
| 2020-12-12 18:12:12.000 |
```

```
+-------------------------+
4 rows in set (0.15 sec)
```

CHAR | String Type

CHAR

CHAR

Description

CHAR(M) A fixed-length string, M represents the byte length of a fixed-length string. The range of M is 1-255.

keywords

CHAR

VARCHAR

VARCHAR

Description

VARCHAR(M) A variable length string, M represents the byte length of a variable length string. The range of M is 1-65533.

Note: Variable length strings are stored in UTF-8 encoding, so usually English characters occupies 1 byte, and Chinese characters occupies 3 bytes.

keywords

VARCHAR

STRING

STRING

Description

A variable length string. Default support is 1048576 bytes (1M), adjustable up to 2147483643 bytes (2G),and the length of the String type is also limited by the configuration string_type_length_soft_limit_bytes(a soft limit of string type length) of be. the String type can only be used in the value column, not in the key column and the partition and bucket columns

Note: Variable length strings are stored in UTF-8 encoding, so usually English characters occupies 1 byte, and Chinese characters occupies 3 bytes.

keywords

STRING

ARRAY | Semi Structured

ARRAY

ARRAY

ARRAY

description

ARRAY<T>

An array of T-type items, it cannot be used as a key column. Now ARRAY can only used in Duplicate Model Tables.

After version 2.0, it supports the use of non-key columns in Unique model tables.

T-type could be any of:

```
BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, DATE,
DATEV2, DATETIME, DATETIMEV2, CHAR, VARCHAR, STRING
```

example

Create table example:

```
mysql> CREATE TABLE `array_test` (
  `id` int(11) NULL COMMENT "",
  `c_array` ARRAY<int(11)> NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1",
"in_memory" = "false",
"storage_format" = "V2"
);
```

Insert data example:

```
mysql> INSERT INTO `array_test` VALUES (1, [1,2,3,4,5]);
mysql> INSERT INTO `array_test` VALUES (2, [6,7,8]), (3, []), (4, null);
```

Select data example:

```
mysql> SELECT * FROM `array_test`;
+------+-----------------+
| id   | c_array         |
```

```
+------+-----------------+
|    1 | [1, 2, 3, 4, 5] |
|    2 | [6, 7, 8]       |
|    3 | []              |
|    4 | NULL            |
+------+-----------------+
```

keywords

```
ARRAY
```

MAP | Semi Structured

MAP

name

MAP

description

`MAP<K, V>`

A Map of K, V items, it cannot be used as a key column. Now MAP can only used in Duplicate and Unique Model Tables.

Need to manually enable the support, it is disabled by default.

```
admin set frontend config("enable_map_type" = "true");
```

K,V could be any of:

```
BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, DECIMALV3, DATE,
DATEV2, DATETIME, DATETIMEV2, CHAR, VARCHAR, STRING
```

example

Create table example:

```
CREATE TABLE IF NOT EXISTS test.simple_map (
  `id` INT(11) NULL COMMENT "",
  `m` Map<STRING, INT> NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1",
"storage_format" = "V2"
);
```

Insert data example:

```
mysql> INSERT INTO simple_map VALUES(1, {'a': 100, 'b': 200});
```

stream_load examples: See STREAM LOAD for syntax details.

```
# load the map data from json file
curl --location-trusted -uroot: -T events.json -H "format: json" -H "read_json_by_line: true"
http://fe_host:8030/api/test/simple_map/_stream_load
```

```
# 返回结果
{
    "TxnId": 106134,
    "Label": "5666e573-9a97-4dfc-ae61-2d6b61fdffd2",
    "Comment": "",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 10293125,
    "NumberLoadedRows": 10293125,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 2297411459,
    "LoadTimeMs": 66870,
    "BeginTxnTimeMs": 1,
    "StreamLoadPutTimeMs": 80,
    "ReadDataTimeMs": 6415,
    "WriteDataTimeMs": 10550,
    "CommitAndPublishTimeMs": 38
}
```

Select all data example:

```
mysql> SELECT * FROM simple_map;
+------+----------------------------+
| id   | m                          |
+------+----------------------------+
|    1 | {'a':100, 'b':200}         |
|    2 | {'b':100, 'c':200, 'd':300} |
|    3 | {'a':10, 'd':200}          |
+------+----------------------------+
```

Select map column example:

```
mysql> SELECT m FROM simple_map;
+----------------------------+
| m                          |
+----------------------------+
| {'a':100, 'b':200}         |
| {'b':100, 'c':200, 'd':300} |
| {'a':10, 'd':200}          |
+----------------------------+
```

Select map value according given key example:

```
mysql> SELECT m['a'] FROM simple_map;
+----------------------------+
| %element_extract%(`m`, 'a') |
+----------------------------+
|                        100 |
|                       NULL |
|                         10 |
+----------------------------+
```

map functions examples:

```
# map construct
```

```
mysql> SELECT map('k11', 1000, 'k22', 2000)['k11'];
+-----------------------------------------------------+
| %element_extract%(map('k11', 1000, 'k22', 2000), 'k11') |
+-----------------------------------------------------+
|                                                1000 |
+-----------------------------------------------------+

mysql> SELECT map('k11', 1000, 'k22', 2000)['nokey'];
+-------------------------------------------------------+
| %element_extract%(map('k11', 1000, 'k22', 2000), 'nokey') |
+-------------------------------------------------------+
|                                                  NULL |
+-------------------------------------------------------+
1 row in set (0.06 sec)

# map size

mysql> SELECT map_size(map('k11', 1000, 'k22', 2000));
+----------------------------------------+
| map_size(map('k11', 1000, 'k22', 2000)) |
+----------------------------------------+
|                                      2 |
+----------------------------------------+

mysql> SELECT id, m, map_size(m) FROM simple_map ORDER BY id;
+------+-----------------------------+--------------+
| id   | m                           | map_size(`m`) |
+------+-----------------------------+--------------+
|    1 | {"a":100, "b":200}          |             2 |
|    2 | {"b":100, "c":200, "d":300} |             3 |
|    2 | {"a":10, "d":200}           |             2 |
+------+-----------------------------+--------------+
3 rows in set (0.04 sec)

# map_contains_key

mysql> SELECT map_contains_key(map('k11', 1000, 'k22', 2000), 'k11');
+-------------------------------------------------------+
| map_contains_key(map('k11', 1000, 'k22', 2000), 'k11') |
+-------------------------------------------------------+
|                                                     1 |
+-------------------------------------------------------+
1 row in set (0.08 sec)

mysql> SELECT id, m, map_contains_key(m, 'k1') FROM simple_map ORDER BY id;
+------+-----------------------------+----------------------------+
| id   | m                           | map_contains_key(`m`, 'k1') |
+------+-----------------------------+----------------------------+
|    1 | {"a":100, "b":200}          |                          0 |
|    2 | {"b":100, "c":200, "d":300} |                          0 |
|    2 | {"a":10, "d":200}           |                          0 |
+------+-----------------------------+----------------------------+
3 rows in set (0.10 sec)

mysql> SELECT id, m, map_contains_key(m, 'a') FROM simple_map ORDER BY id;
+------+-----------------------------+---------------------------+
| id   | m                           | map_contains_key(`m`, 'a') |
+------+-----------------------------+---------------------------+
|    1 | {"a":100, "b":200}          |                         1 |
|    2 | {"b":100, "c":200, "d":300} |                         0 |
|    2 | {"a":10, "d":200}           |                         1 |
+------+-----------------------------+---------------------------+
3 rows in set (0.17 sec)

# map_contains_value
```

```
mysql> SELECT map_contains_value(map('k11', 1000, 'k22', 2000), NULL);
+-----------------------------------------------------------+
| map_contains_value(map('k11', 1000, 'k22', 2000), NULL) |
+-----------------------------------------------------------+
|                                                         0 |
+-----------------------------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT id, m, map_contains_value(m, '100') FROM simple_map ORDER BY id;
+------+-----------------------------+-----------------------------+
| id   | m                           | map_contains_value(`m`, 100) |
+------+-----------------------------+-----------------------------+
|    1 | {"a":100, "b":200}          |                           1 |
|    2 | {"b":100, "c":200, "d":300} |                           1 |
|    2 | {"a":10, "d":200}           |                           0 |
+------+-----------------------------+-----------------------------+
3 rows in set (0.11 sec)

# map_keys

mysql> SELECT map_keys(map('k11', 1000, 'k22', 2000));
+----------------------------------------+
| map_keys(map('k11', 1000, 'k22', 2000)) |
+----------------------------------------+
| ["k11", "k22"]                         |
+----------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT id, map_keys(m) FROM simple_map ORDER BY id;
+------+-----------------+
| id   | map_keys(`m`)   |
+------+-----------------+
|    1 | ["a", "b"]      |
|    2 | ["b", "c", "d"] |
|    2 | ["a", "d"]      |
+------+-----------------+
3 rows in set (0.19 sec)

# map_values

mysql> SELECT map_values(map('k11', 1000, 'k22', 2000));
+-----------------------------------------+
| map_values(map('k11', 1000, 'k22', 2000)) |
+-----------------------------------------+
| [1000, 2000]                            |
+-----------------------------------------+
1 row in set (0.03 sec)

mysql> SELECT id, map_values(m) FROM simple_map ORDER BY id;
+------+-----------------+
| id   | map_values(`m`) |
+------+-----------------+
|    1 | [100, 200]      |
|    2 | [100, 200, 300] |
|    2 | [10, 200]       |
+------+-----------------+
3 rows in set (0.18 sec)
```

keywords

```
MAP
```

STRUCT | Semi Structured

STRUCT

STRUCT

name

STRUCT

description

```
STRUCT<field_name:field_type [COMMENT 'comment_string'], ... >
```

Represents value with structure described by multiple fields, which can be viewed as a collection of multiple columns.

Need to manually enable the support, it is disabled by default.

```
admin set frontend config("enable_struct_type" = "true");
```

It cannot be used as a Key column. Now STRUCT can only used in Duplicate Model Tables.

The names and number of Fields in a Struct is fixed and always Nullable, and a Field typically consists of the following parts.

- field_name: Identifier naming the field, non repeatable.
- field_type: A data type.
- COMMENT: An optional string describing the field. (currently not supported)

The currently supported types are:

```
BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, DECIMAL, DECIMALV3, DATE,
DATEV2, DATETIME, DATETIMEV2, CHAR, VARCHAR, STRING
```

We have a todo list for future version:

```
TODO: Supports nested Struct or other complex types
```

example

Create table example:

```
mysql> CREATE TABLE `struct_test` (
  `id` int(11) NULL,
  `s_info` STRUCT<s_id:int(11), s_name:string, s_address:string> NULL
```

```
) ENGINE=OLAP
DUPLICATE KEY(`id`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`id`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1",
"storage_format" = "V2",
"light_schema_change" = "true",
"disable_auto_compaction" = "false"
);
```

Insert data example:

```
INSERT INTO `struct_test` VALUES (1, {1, 'sn1', 'sa1'});
INSERT INTO `struct_test` VALUES (2, struct(2, 'sn2', 'sa2'));
INSERT INTO `struct_test` VALUES (3, named_struct('s_id', 3, 's_name', 'sn3', 's_address', 'sa3'));
```

Stream load:

test.csv:

```
1|{"s_id":1, "s_name":"sn1", "s_address":"sa1"}
2|{s_id:2, s_name:sn2, s_address:sa2}
3|{"s_address":"sa3", "s_name":"sn3", "s_id":3}
```

example:

```
curl --location-trusted -u root -T test.csv  -H "label:test_label" http://host:port/api/test/struct_test/_stream_load
```

Select data example:

```
mysql> select * from struct_test;
+------+-------------------+
| id   | s_info            |
+------+-------------------+
|    1 | {1, 'sn1', 'sa1'} |
|    2 | {2, 'sn2', 'sa2'} |
|    3 | {3, 'sn3', 'sa3'} |
+------+-------------------+
3 rows in set (0.02 sec)
```

keywords

```
STRUCT
```

JSON | Semi Structured

JSON

JSON

The JSON data type stores JSON data efficiently in a binary format and allows access to its internal fields through JSON functions.

By default, it supports up to 1048576 bytes (1MB), and can be increased up to 2147483643 bytes (2GB). This can be adjusted via the `string_type_length_soft_limit_bytes` configuration.

Compared to storing JSON strings in a regular STRING type, the JSON type has two main advantages: 1. JSON format validation during data insertion. 2. More efficient binary storage format, enabling faster access to JSON internal fields using functions like `json_extract`, compared to `get_json_xx` functions.

Note: In version 1.2.x, the JSON type was named JSONB. To maintain compatibility with MySQL, it was renamed to JSON starting from version 2.0.0. Older tables can still use the previous name.

Syntax

Definition:

```
json_column_name JSON
```

Insertion: - Using `INSERT INTO VALUES` with the format as a string surrounded by quotes. For example:

```
INSERT INTO table_name(id, json_column_name) VALUES (1, '{"k1": "100"}')
```

- For STREAM LOAD, the format for the corresponding column is a string without additional quotes. For example:

```
12  {"k1":"v31", "k2": 300}
13  []
14  [123, 456]
```

- When the all special character with `'\'` such as `'\r'`, `'\t'` appears in JSON, you need to use the replace function to replace "\" with "\\", for example, you need replace "\n" to "\\n"

Query: - Directly select the entire JSON column:

```
SELECT json_column_name FROM table_name;
```

- Extract specific fields or other information from JSON using JSON functions. For example:

```
SELECT json_extract(json_column_name, '$.k1') FROM table_name;
```

- The JSON type can be cast to and from integers, strings, BOOLEAN, ARRAY, and MAP. For example:

```
SELECT CAST('{"k1": "100"}' AS JSON);
SELECT CAST(json_column_name AS STRING) FROM table_name;
SELECT CAST(json_extract(json_column_name, '$.k1') AS INT) FROM table_name;
```

> **The JSON type currently cannot be used for `GROUP BY`, `ORDER BY`, or comparison operations.**

example

A tutorial for JSON datatype including create table, load data and query.

create database and table

```
CREATE DATABASE testdb;

USE testdb;

CREATE TABLE test_json (
  id INT,
  j JSON
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 10
PROPERTIES("replication_num" = "1");
```

Load data

stream load test_json.csv test data

- there are 2 columns, the 1st column is id and the 2nd column is json string
- there are 25 rows, the first 18 rows are valid json and the last 7 rows are invalid

```
1   \N
2   null
3   true
4   false
5   100
6   10000
7   1000000000
8   1152921504606846976
9   6.18
10  "abcd"
11  {}
12  {"k1":"v31", "k2": 300}
13  []
14  [123, 456]
15  ["abc", "def"]
```

```
16  [null, true, false, 100, 6.18, "abc"]
17  [{"k1":"v41", "k2": 400}, 1, "a", 3.14]
18  {"k1":"v31", "k2": 300, "a1": [{"k1":"v41", "k2": 400}, 1, "a", 3.14]}
19  ''
20  'abc'
21  abc
22  100x
23  6.a8
24  {x
25  [123, abc]
```

- due to the 28% of rows are invalid, stream load with default configuration will fail with error message “too many filtered rows”

```
curl --location-trusted -u root: -T test_json.csv http://127.0.0.1:8840/api/testdb/test_json/_stream_load
{
    "TxnId": 12019,
    "Label": "744d9821-9c9f-43dc-bf3b-7ab048f14e32",
    "TwoPhaseCommit": "false",
    "Status": "Fail",
    "Message": "too many filtered rows",
    "NumberTotalRows": 25,
    "NumberLoadedRows": 18,
    "NumberFilteredRows": 7,
    "NumberUnselectedRows": 0,
    "LoadBytes": 380,
    "LoadTimeMs": 48,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 45,
    "CommitAndPublishTimeMs": 0,
    "ErrorURL": "http://172.21.0.5:8840/api/_load_error_log?file=__shard_2/error_log_insert_stmt_95435c4bf5f156df-
426735082a9296af_95435c4bf5f156df_426735082a9296af"
}
```

- stream load will success after set header configuration ‘max_filter_ratio: 0.3’

```
curl --location-trusted -u root: -H 'max_filter_ratio: 0.3' -T test_json.csv http://127.0.0.1:8840/api/testdb/test_json/_stream_load
{
    "TxnId": 12017,
    "Label": "f37a50c1-43e9-4f4e-a159-a3db6abe2579",
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 25,
    "NumberLoadedRows": 18,
    "NumberFilteredRows": 7,
    "NumberUnselectedRows": 0,
    "LoadBytes": 380,
    "LoadTimeMs": 68,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 2,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 45,
    "CommitAndPublishTimeMs": 19,
    "ErrorURL": "http://172.21.0.5:8840/api/_load_error_log?file=__shard_0/error_log_insert_stmt_a1463f98a7b15caf-
c79399b920f5bfa3_a1463f98a7b15caf_c79399b920f5bfa3"
}
```

- use SELECT to view the data loaded by stream load. The column with JSON type will be displayed as plain JSON string.

```
mysql> SELECT * FROM test_json ORDER BY id;
+------+-----------------------------------------------------------+
| id   | j                                                         |
+------+-----------------------------------------------------------+
|    1 |                                                      NULL |
|    2 |                                                      null |
|    3 |                                                      true |
|    4 |                                                     false |
|    5 |                                                       100 |
|    6 |                                                     10000 |
|    7 |                                                1000000000 |
|    8 |                                     1152921504606846976 |
|    9 |                                                      6.18 |
|   10 |                                                    "abcd" |
|   11 |                                                        {} |
|   12 |                                    {"k1":"v31","k2":300} |
|   13 |                                                        [] |
|   14 |                                                 [123,456] |
|   15 |                                           ["abc","def"] |
|   16 |                          [null,true,false,100,6.18,"abc"] |
|   17 |                        [{"k1":"v41","k2":400},1,"a",3.14] |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |
+------+-----------------------------------------------------------+
18 rows in set (0.03 sec)
```

write data using insert into

- total rows increae from 18 to 19 after insert 1 row

```
mysql> INSERT INTO test_json VALUES(26, '{"k1":"v1", "k2": 200}');
Query OK, 1 row affected (0.09 sec)
{'label':'insert_4ece6769d1b42fd_ac9f25b3b8f3dc02', 'status':'VISIBLE', 'txnId':'12016'}

mysql> SELECT * FROM test_json ORDER BY id;
+------+-----------------------------------------------------------+
| id   | j                                                         |
+------+-----------------------------------------------------------+
|    1 |                                                      NULL |
|    2 |                                                      null |
|    3 |                                                      true |
|    4 |                                                     false |
|    5 |                                                       100 |
|    6 |                                                     10000 |
|    7 |                                                1000000000 |
|    8 |                                     1152921504606846976 |
|    9 |                                                      6.18 |
|   10 |                                                    "abcd" |
|   11 |                                                        {} |
|   12 |                                    {"k1":"v31","k2":300} |
|   13 |                                                        [] |
|   14 |                                                 [123,456] |
|   15 |                                           ["abc","def"] |
|   16 |                          [null,true,false,100,6.18,"abc"] |
|   17 |                        [{"k1":"v41","k2":400},1,"a",3.14] |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |
|   26 |                                     {"k1":"v1","k2":200} |
```

```
+------+------------------------------------------------------------+
19 rows in set (0.03 sec)
```

Query

extract some filed from json by json_extract functions

1. extract the whole json, ‘$’ stands for root in json path

```
+------+------------------------------------------------------------+-------------------------------------------------------------
---+
| id   | j                                                          | json_extract(`j`, '$')
|
+------+------------------------------------------------------------+-------------------------------------------------------------
---+
|    1 |                                                       NULL |
NULL |
|    2 |                                                       null |
null |
|    3 |                                                       true |
true |
|    4 |                                                      false |
false |
|    5 |                                                        100 |
100 |
|    6 |                                                      10000 |
10000 |
|    7 |                                                 1000000000 |
1000000000 |
|    8 |                                           1152921504606846976 |
1152921504606846976 |
|    9 |                                                       6.18 |
6.18 |
|   10 |                                                     "abcd" |
"abcd" |
|   11 |                                                        {} |
{} |
|   12 |                                          {"k1":"v31","k2":300} |
{"k1":"v31","k2":300} |
|   13 |                                                        [] |
[] |
|   14 |                                                 [123,456] |
[123,456] |
|   15 |                                               ["abc","def"] |
["abc","def"] |
|   16 |                                [null,true,false,100,6.18,"abc"] |
[null,true,false,100,6.18,"abc"] |
|   17 |                              [{"k1":"v41","k2":400},1,"a",3.14] |
[{"k1":"v41","k2":400},1,"a",3.14] |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} | {"k1":"v31","k2":300,"a1":
[{"k1":"v41","k2":400},1,"a",3.14]} |
|   26 |                                           {"k1":"v1","k2":200} |
{"k1":"v1","k2":200} |
+------+------------------------------------------------------------+-------------------------------------------------------------
---+
19 rows in set (0.03 sec)
```

1. extract k1 field, return NULL if it does not exist

```
mysql> SELECT id, j, json_extract(j, '$.k1') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+---------------------------+
| id   | j                                                               | json_extract(`j`, '$.k1') |
+------+-----------------------------------------------------------------+---------------------------+
|    1 |                                                            NULL |                      NULL |
|    2 |                                                            null |                      NULL |
|    3 |                                                            true |                      NULL |
|    4 |                                                           false |                      NULL |
|    5 |                                                             100 |                      NULL |
|    6 |                                                           10000 |                      NULL |
|    7 |                                                      1000000000 |                      NULL |
|    8 |                                                1152921504606846976 |                      NULL |
|    9 |                                                            6.18 |                      NULL |
|   10 |                                                          "abcd" |                      NULL |
|   11 |                                                             {} |                      NULL |
|   12 |                                           {"k1":"v31","k2":300} |                     "v31" |
|   13 |                                                             [] |                      NULL |
|   14 |                                                     [123,456] |                      NULL |
|   15 |                                                ["abc","def"] |                      NULL |
|   16 |                                [null,true,false,100,6.18,"abc"] |                      NULL |
|   17 |                              [{"k1":"v41","k2":400},1,"a",3.14] |                      NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                     "v31" |
|   26 |                                           {"k1":"v1","k2":200} |                      "v1" |
+------+-----------------------------------------------------------------+---------------------------+
19 rows in set (0.03 sec)
```

1. extract element 0 of the top level array

```
mysql> SELECT id, j, json_extract(j, '$[0]') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+---------------------------+
| id   | j                                                               | json_extract(`j`, '$[0]') |
+------+-----------------------------------------------------------------+---------------------------+
|    1 |                                                            NULL |                      NULL |
|    2 |                                                            null |                      NULL |
|    3 |                                                            true |                      NULL |
|    4 |                                                           false |                      NULL |
|    5 |                                                             100 |                      NULL |
|    6 |                                                           10000 |                      NULL |
|    7 |                                                      1000000000 |                      NULL |
|    8 |                                                1152921504606846976 |                      NULL |
|    9 |                                                            6.18 |                      NULL |
|   10 |                                                          "abcd" |                      NULL |
|   11 |                                                             {} |                      NULL |
|   12 |                                           {"k1":"v31","k2":300} |                      NULL |
|   13 |                                                             [] |                      NULL |
|   14 |                                                     [123,456] |                       123 |
|   15 |                                                ["abc","def"] |                     "abc" |
|   16 |                                [null,true,false,100,6.18,"abc"] |                      null |
|   17 |                              [{"k1":"v41","k2":400},1,"a",3.14] |      {"k1":"v41","k2":400} |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                      NULL |
|   26 |                                           {"k1":"v1","k2":200} |                      NULL |
+------+-----------------------------------------------------------------+---------------------------+
19 rows in set (0.03 sec)
```

1. extract a whole json array of name a1

```
mysql> SELECT id, j, json_extract(j, '$.a1') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+-----------------------------------+
| id   | j                                                               | json_extract(`j`, '$.a1')         |
+------+-----------------------------------------------------------------+-----------------------------------+
|    1 |                                                            NULL |                              NULL |
```

```
|    2 |                                                        null |                                NULL |
|    3 |                                                        true |                                NULL |
|    4 |                                                       false |                                NULL |
|    5 |                                                         100 |                                NULL |
|    6 |                                                       10000 |                                NULL |
|    7 |                                                1000000000 |                                NULL |
|    8 |                                         1152921504606846976 |                                NULL |
|    9 |                                                        6.18 |                                NULL |
|   10 |                                                      "abcd" |                                NULL |
|   11 |                                                         {} |                                NULL |
|   12 |                                       {"k1":"v31","k2":300} |                                NULL |
|   13 |                                                         [] |                                NULL |
|   14 |                                                  [123,456] |                                NULL |
|   15 |                                              ["abc","def"] |                                NULL |
|   16 |                                [null,true,false,100,6.18,"abc"] |                                NULL |
|   17 |                             [{"k1":"v41","k2":400},1,"a",3.14] |                                NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} | [{"k1":"v41","k2":400},1,"a",3.14] |
|   26 |                                        {"k1":"v1","k2":200} |                                NULL |
+------+-------------------------------------------------------------+-----------------------------------+
19 rows in set (0.02 sec)
```

1. extract nested field from an object in an array

```
mysql> SELECT id, j, json_extract(j, '$.a1[0]'), json_extract(j, '$.a1[0].k1') FROM test_json ORDER BY id;
+------+-------------------------------------------------------------+-------------------------------+-----------------------------
------+
| id   | j                                                           | json_extract(`j`, '$.a1[0]') | json_extract(`j`,
'$.a1[0].k1') |
+------+-------------------------------------------------------------+-------------------------------+-----------------------------
------+
|    1 |                                                        NULL |                          NULL |
NULL |
|    2 |                                                        null |                          NULL |
NULL |
|    3 |                                                        true |                          NULL |
NULL |
|    4 |                                                       false |                          NULL |
NULL |
|    5 |                                                         100 |                          NULL |
NULL |
|    6 |                                                       10000 |                          NULL |
NULL |
|    7 |                                                1000000000 |                          NULL |
NULL |
|    8 |                                         1152921504606846976 |                          NULL |
NULL |
|    9 |                                                        6.18 |                          NULL |
NULL |
|   10 |                                                      "abcd" |                          NULL |
NULL |
|   11 |                                                         {} |                          NULL |
NULL |
|   12 |                                       {"k1":"v31","k2":300} |                          NULL |
NULL |
|   13 |                                                         [] |                          NULL |
NULL |
|   14 |                                                  [123,456] |                          NULL |
NULL |
|   15 |                                              ["abc","def"] |                          NULL |
NULL |
|   16 |                                [null,true,false,100,6.18,"abc"] |                          NULL |
NULL |
|   17 |                             [{"k1":"v41","k2":400},1,"a",3.14] |                          NULL |
```

```
NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |         {"k1":"v41","k2":400} |
"v41" |
|   26 |                                          {"k1":"v1","k2":200} |                          NULL |
NULL |
+------+-----------------------------------------------------------------+-------------------------------+----------------------------
------+
19 rows in set (0.02 sec)
```

1. extract field with specific datatype

- json_extract_string will extract field with string type, convert to string if the field is not string

```
mysql> SELECT id, j, json_extract_string(j, '$') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+-----------------------------------------------------------------
---+
| id   | j                                                               | json_extract_string(`j`, '$')
|
+------+-----------------------------------------------------------------+-----------------------------------------------------------------
---+
|    1 | NULL                                                            | NULL
|
|    2 | null                                                            | null
|
|    3 | true                                                            | true
|
|    4 | false                                                           | false
|
|    5 | 100                                                             | 100
|
|    6 | 10000                                                           | 10000
|
|    7 | 1000000000                                                      | 1000000000
|
|    8 | 1152921504606846976                                             | 1152921504606846976
|
|    9 | 6.18                                                            | 6.18
|
|   10 | "abcd"                                                          | abcd
|
|   11 | {}                                                              | {}
|
|   12 | {"k1":"v31","k2":300}                                           | {"k1":"v31","k2":300}
|
|   13 | []                                                              | []
|
|   14 | [123,456]                                                       | [123,456]
|
|   15 | ["abc","def"]                                                   | ["abc","def"]
|
|   16 | [null,true,false,100,6.18,"abc"]                                | [null,true,false,100,6.18,"abc"]
|
|   17 | [{"k1":"v41","k2":400},1,"a",3.14]                              | [{"k1":"v41","k2":400},1,"a",3.14]
|
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} | {"k1":"v31","k2":300,"a1":
[{"k1":"v41","k2":400},1,"a",3.14]} |
|   26 | {"k1":"v1","k2":200}                                            | {"k1":"v1","k2":200}
|
+------+-----------------------------------------------------------------+-----------------------------------------------------------------
---+
19 rows in set (0.02 sec)

mysql> SELECT id, j, json_extract_string(j, '$.k1') FROM test_json ORDER BY id;
```

```
+------+-------------------------------------------------------------+----------------------------------+
| id   | j                                                           | json_extract_string(`j`, '$.k1') |
+------+-------------------------------------------------------------+----------------------------------+
|    1 |                                                        NULL | NULL                             |
|    2 |                                                        null | NULL                             |
|    3 |                                                        true | NULL                             |
|    4 |                                                       false | NULL                             |
|    5 |                                                         100 | NULL                             |
|    6 |                                                       10000 | NULL                             |
|    7 |                                                  1000000000 | NULL                             |
|    8 |                                     1152921504606846976 | NULL                             |
|    9 |                                                        6.18 | NULL                             |
|   10 |                                                      "abcd" | NULL                             |
|   11 |                                                         {} | NULL                             |
|   12 |                                    {"k1":"v31","k2":300} | v31                              |
|   13 |                                                         [] | NULL                             |
|   14 |                                                 [123,456] | NULL                             |
|   15 |                                           ["abc","def"] | NULL                             |
|   16 |                             [null,true,false,100,6.18,"abc"] | NULL                             |
|   17 |                          [{"k1":"v41","k2":400},1,"a",3.14] | NULL                             |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} | v31                              |
|   26 |                                     {"k1":"v1","k2":200} | v1                               |
+------+-------------------------------------------------------------+----------------------------------+
19 rows in set (0.03 sec)
```

- json_extract_int will extract field with int type, return NULL if the field is not int

```
mysql> SELECT id, j, json_extract_int(j, '$') FROM test_json ORDER BY id;
+------+-------------------------------------------------------------+----------------------------+
| id   | j                                                           | json_extract_int(`j`, '$') |
+------+-------------------------------------------------------------+----------------------------+
|    1 |                                                        NULL |                       NULL |
|    2 |                                                        null |                       NULL |
|    3 |                                                        true |                       NULL |
|    4 |                                                       false |                       NULL |
|    5 |                                                         100 |                        100 |
|    6 |                                                       10000 |                      10000 |
|    7 |                                                  1000000000 |                 1000000000 |
|    8 |                                     1152921504606846976 |                       NULL |
|    9 |                                                        6.18 |                       NULL |
|   10 |                                                      "abcd" |                       NULL |
|   11 |                                                         {} |                       NULL |
|   12 |                                    {"k1":"v31","k2":300} |                       NULL |
|   13 |                                                         [] |                       NULL |
|   14 |                                                 [123,456] |                       NULL |
|   15 |                                           ["abc","def"] |                       NULL |
|   16 |                             [null,true,false,100,6.18,"abc"] |                       NULL |
|   17 |                          [{"k1":"v41","k2":400},1,"a",3.14] |                       NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                       NULL |
|   26 |                                     {"k1":"v1","k2":200} |                       NULL |
+------+-------------------------------------------------------------+----------------------------+
19 rows in set (0.02 sec)

mysql> SELECT id, j, json_extract_int(j, '$.k2') FROM test_json ORDER BY id;
+------+-------------------------------------------------------------+-----------------------------+
| id   | j                                                           | json_extract_int(`j`, '$.k2') |
+------+-------------------------------------------------------------+-----------------------------+
|    1 |                                                        NULL |                        NULL |
|    2 |                                                        null |                        NULL |
|    3 |                                                        true |                        NULL |
|    4 |                                                       false |                        NULL |
|    5 |                                                         100 |                        NULL |
|    6 |                                                       10000 |                        NULL |
```

```
|    7 |                                                1000000000 |                            NULL |
|    8 |                                           1152921504606846976 |                            NULL |
|    9 |                                                      6.18 |                            NULL |
|   10 |                                                    "abcd" |                            NULL |
|   11 |                                                        {} |                            NULL |
|   12 |                                      {"k1":"v31","k2":300} |                             300 |
|   13 |                                                        [] |                            NULL |
|   14 |                                                 [123,456] |                            NULL |
|   15 |                                           ["abc","def"] |                            NULL |
|   16 |                               [null,true,false,100,6.18,"abc"] |                            NULL |
|   17 |                             [{"k1":"v41","k2":400},1,"a",3.14] |                            NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                             300 |
|   26 |                                       {"k1":"v1","k2":200} |                             200 |
+------+------------------------------------------------------------+--------------------------------+
19 rows in set (0.03 sec)
```

- json_extract_bigint will extract field with bigint type, return NULL if the field is not bigint

```
mysql> SELECT id, j, json_extract_bigint(j, '$') FROM test_json ORDER BY id;
+------+------------------------------------------------------------+--------------------------------+
| id   | j                                                          | json_extract_bigint(`j`, '$') |
+------+------------------------------------------------------------+--------------------------------+
|    1 |                                                       NULL |                           NULL |
|    2 |                                                       null |                           NULL |
|    3 |                                                       true |                           NULL |
|    4 |                                                      false |                           NULL |
|    5 |                                                        100 |                            100 |
|    6 |                                                      10000 |                          10000 |
|    7 |                                                 1000000000 |                     1000000000 |
|    8 |                                            1152921504606846976 |            1152921504606846976 |
|    9 |                                                       6.18 |                           NULL |
|   10 |                                                     "abcd" |                           NULL |
|   11 |                                                         {} |                           NULL |
|   12 |                                       {"k1":"v31","k2":300} |                           NULL |
|   13 |                                                         [] |                           NULL |
|   14 |                                                  [123,456] |                           NULL |
|   15 |                                            ["abc","def"] |                           NULL |
|   16 |                                [null,true,false,100,6.18,"abc"] |                           NULL |
|   17 |                              [{"k1":"v41","k2":400},1,"a",3.14] |                           NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                           NULL |
|   26 |                                        {"k1":"v1","k2":200} |                           NULL |
+------+------------------------------------------------------------+--------------------------------+
19 rows in set (0.03 sec)

mysql> SELECT id, j, json_extract_bigint(j, '$.k2') FROM test_json ORDER BY id;
+------+------------------------------------------------------------+---------------------------------+
| id   | j                                                          | json_extract_bigint(`j`, '$.k2') |
+------+------------------------------------------------------------+---------------------------------+
|    1 |                                                       NULL |                            NULL |
|    2 |                                                       null |                            NULL |
|    3 |                                                       true |                            NULL |
|    4 |                                                      false |                            NULL |
|    5 |                                                        100 |                            NULL |
|    6 |                                                      10000 |                            NULL |
|    7 |                                                 1000000000 |                            NULL |
|    8 |                                            1152921504606846976 |                            NULL |
|    9 |                                                       6.18 |                            NULL |
|   10 |                                                     "abcd" |                            NULL |
|   11 |                                                         {} |                            NULL |
|   12 |                                       {"k1":"v31","k2":300} |                             300 |
|   13 |                                                         [] |                            NULL |
|   14 |                                                  [123,456] |                            NULL |
|   15 |                                            ["abc","def"] |                            NULL |
```

```
|   16 |                              [null,true,false,100,6.18,"abc"] |                              NULL |
|   17 |                            [{"k1":"v41","k2":400},1,"a",3.14] |                              NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                               300 |
|   26 |                                          {"k1":"v1","k2":200} |                               200 |
+------+-----------------------------------------------------------------+-----------------------------------+
19 rows in set (0.02 sec)
```

- json_extract_double will extract field with double type, return NULL if the field is not double

```
mysql> SELECT id, j, json_extract_double(j, '$') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+-------------------------------+
| id   | j                                                               | json_extract_double(`j`, '$') |
+------+-----------------------------------------------------------------+-------------------------------+
|    1 |                                                            NULL |                          NULL |
|    2 |                                                            null |                          NULL |
|    3 |                                                            true |                          NULL |
|    4 |                                                           false |                          NULL |
|    5 |                                                             100 |                           100 |
|    6 |                                                           10000 |                         10000 |
|    7 |                                                      1000000000 |                    1000000000 |
|    8 |                                                1152921504606846976 |         1.152921504606847e+18 |
|    9 |                                                            6.18 |                          6.18 |
|   10 |                                                          "abcd" |                          NULL |
|   11 |                                                             {} |                          NULL |
|   12 |                                          {"k1":"v31","k2":300} |                          NULL |
|   13 |                                                             [] |                          NULL |
|   14 |                                                      [123,456] |                          NULL |
|   15 |                                                ["abc","def"] |                          NULL |
|   16 |                              [null,true,false,100,6.18,"abc"] |                          NULL |
|   17 |                            [{"k1":"v41","k2":400},1,"a",3.14] |                          NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                          NULL |
|   26 |                                          {"k1":"v1","k2":200} |                          NULL |
+------+-----------------------------------------------------------------+-------------------------------+
19 rows in set (0.02 sec)

mysql> SELECT id, j, json_extract_double(j, '$.k2') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+---------------------------------+
| id   | j                                                               | json_extract_double(`j`, '$.k2') |
+------+-----------------------------------------------------------------+---------------------------------+
|    1 |                                                            NULL |                            NULL |
|    2 |                                                            null |                            NULL |
|    3 |                                                            true |                            NULL |
|    4 |                                                           false |                            NULL |
|    5 |                                                             100 |                            NULL |
|    6 |                                                           10000 |                            NULL |
|    7 |                                                      1000000000 |                            NULL |
|    8 |                                                1152921504606846976 |                            NULL |
|    9 |                                                            6.18 |                            NULL |
|   10 |                                                          "abcd" |                            NULL |
|   11 |                                                             {} |                            NULL |
|   12 |                                          {"k1":"v31","k2":300} |                             300 |
|   13 |                                                             [] |                            NULL |
|   14 |                                                      [123,456] |                            NULL |
|   15 |                                                ["abc","def"] |                            NULL |
|   16 |                              [null,true,false,100,6.18,"abc"] |                            NULL |
|   17 |                            [{"k1":"v41","k2":400},1,"a",3.14] |                            NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                             300 |
|   26 |                                          {"k1":"v1","k2":200} |                             200 |
+------+-----------------------------------------------------------------+---------------------------------+
19 rows in set (0.03 sec)
```

- json_extract_bool will extract field with boolean type, return NULL if the field is not boolean

```
mysql> SELECT id, j, json_extract_bool(j, '$') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+-----------------------------+
| id   | j                                                               | json_extract_bool(`j`, '$') |
+------+-----------------------------------------------------------------+-----------------------------+
|    1 |                                                            NULL |                        NULL |
|    2 |                                                            null |                        NULL |
|    3 |                                                            true |                           1 |
|    4 |                                                           false |                           0 |
|    5 |                                                             100 |                        NULL |
|    6 |                                                           10000 |                        NULL |
|    7 |                                                      1000000000 |                        NULL |
|    8 |                                             1152921504606846976 |                        NULL |
|    9 |                                                            6.18 |                        NULL |
|   10 |                                                          "abcd" |                        NULL |
|   11 |                                                             {} |                        NULL |
|   12 |                                          {"k1":"v31","k2":300} |                        NULL |
|   13 |                                                             [] |                        NULL |
|   14 |                                                      [123,456] |                        NULL |
|   15 |                                                ["abc","def"] |                        NULL |
|   16 |                                [null,true,false,100,6.18,"abc"] |                        NULL |
|   17 |                              [{"k1":"v41","k2":400},1,"a",3.14] |                        NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                        NULL |
|   26 |                                           {"k1":"v1","k2":200} |                        NULL |
+------+-----------------------------------------------------------------+-----------------------------+
19 rows in set (0.01 sec)

mysql> SELECT id, j, json_extract_bool(j, '$[1]') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+--------------------------------+
| id   | j                                                               | json_extract_bool(`j`, '$[1]') |
+------+-----------------------------------------------------------------+--------------------------------+
|    1 |                                                            NULL |                           NULL |
|    2 |                                                            null |                           NULL |
|    3 |                                                            true |                           NULL |
|    4 |                                                           false |                           NULL |
|    5 |                                                             100 |                           NULL |
|    6 |                                                           10000 |                           NULL |
|    7 |                                                      1000000000 |                           NULL |
|    8 |                                             1152921504606846976 |                           NULL |
|    9 |                                                            6.18 |                           NULL |
|   10 |                                                          "abcd" |                           NULL |
|   11 |                                                             {} |                           NULL |
|   12 |                                          {"k1":"v31","k2":300} |                           NULL |
|   13 |                                                             [] |                           NULL |
|   14 |                                                      [123,456] |                           NULL |
|   15 |                                                ["abc","def"] |                           NULL |
|   16 |                                [null,true,false,100,6.18,"abc"] |                              1 |
|   17 |                              [{"k1":"v41","k2":400},1,"a",3.14] |                           NULL |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                           NULL |
|   26 |                                           {"k1":"v1","k2":200} |                           NULL |
+------+-----------------------------------------------------------------+--------------------------------+
19 rows in set (0.01 sec)
```

- json_extract_isnull will extract field with json null type, return 1 if the field is json null , else 0
- json null is different from SQL NULL. SQL NULL stands for no value for a field, but json null stands for an field with special value null.

```
mysql> SELECT id, j, json_extract_isnull(j, '$') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+-------------------------------+
| id   | j                                                               | json_extract_isnull(`j`, '$') |
+------+-----------------------------------------------------------------+-------------------------------+
|    1 |                                                            NULL |                          NULL |
|    2 |                                                            null |                             1 |
```

```
|    3 |                                                        true |                              0 |
|    4 |                                                       false |                              0 |
|    5 |                                                         100 |                              0 |
|    6 |                                                       10000 |                              0 |
|    7 |                                                  1000000000 |                              0 |
|    8 |                                             1152921504606846976 |                              0 |
|    9 |                                                        6.18 |                              0 |
|   10 |                                                      "abcd" |                              0 |
|   11 |                                                        {} |                              0 |
|   12 |                                           {"k1":"v31","k2":300} |                              0 |
|   13 |                                                        [] |                              0 |
|   14 |                                                  [123,456] |                              0 |
|   15 |                                               ["abc","def"] |                              0 |
|   16 |                                 [null,true,false,100,6.18,"abc"] |                              0 |
|   17 |                               [{"k1":"v41","k2":400},1,"a",3.14] |                              0 |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                              0 |
|   26 |                                            {"k1":"v1","k2":200} |                              0 |
+------+-----------------------------------------------------------------+-----------------------------------+
19 rows in set (0.03 sec)
```

check if a field is existed in json by json_exists_path

```
mysql> SELECT id, j, json_exists_path(j, '$') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+-----------------------------+
| id   | j                                                               | json_exists_path(`j`, '$') |
+------+-----------------------------------------------------------------+-----------------------------+
|    1 |                                                            NULL |                        NULL |
|    2 |                                                            null |                           1 |
|    3 |                                                            true |                           1 |
|    4 |                                                           false |                           1 |
|    5 |                                                             100 |                           1 |
|    6 |                                                           10000 |                           1 |
|    7 |                                                      1000000000 |                           1 |
|    8 |                                                 1152921504606846976 |                           1 |
|    9 |                                                            6.18 |                           1 |
|   10 |                                                          "abcd" |                           1 |
|   11 |                                                            {} |                           1 |
|   12 |                                               {"k1":"v31","k2":300} |                           1 |
|   13 |                                                            [] |                           1 |
|   14 |                                                      [123,456] |                           1 |
|   15 |                                                   ["abc","def"] |                           1 |
|   16 |                                     [null,true,false,100,6.18,"abc"] |                           1 |
|   17 |                                   [{"k1":"v41","k2":400},1,"a",3.14] |                           1 |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                           1 |
|   26 |                                                {"k1":"v1","k2":200} |                           1 |
+------+-----------------------------------------------------------------+-----------------------------+
19 rows in set (0.02 sec)

mysql> SELECT id, j, json_exists_path(j, '$.k1') FROM test_json ORDER BY id;
+------+-----------------------------------------------------------------+------------------------------+
| id   | j                                                               | json_exists_path(`j`, '$.k1') |
+------+-----------------------------------------------------------------+------------------------------+
|    1 |                                                            NULL |                         NULL |
|    2 |                                                            null |                            0 |
|    3 |                                                            true |                            0 |
|    4 |                                                           false |                            0 |
|    5 |                                                             100 |                            0 |
|    6 |                                                           10000 |                            0 |
|    7 |                                                      1000000000 |                            0 |
|    8 |                                                 1152921504606846976 |                            0 |
|    9 |                                                            6.18 |                            0 |
```

```
|   10 |                                                    "abcd" |                              0 |
|   11 |                                                      {} |                              0 |
|   12 |                                    {"k1":"v31","k2":300} |                              1 |
|   13 |                                                      [] |                              0 |
|   14 |                                                [123,456] |                              0 |
|   15 |                                           ["abc","def"] |                              0 |
|   16 |                             [null,true,false,100,6.18,"abc"] |                              0 |
|   17 |                           [{"k1":"v41","k2":400},1,"a",3.14] |                              0 |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                              1 |
|   26 |                                     {"k1":"v1","k2":200} |                              1 |
+------+------------------------------------------------------------+-----------------------------+
19 rows in set (0.03 sec)

mysql> SELECT id, j, json_exists_path(j, '$[2]') FROM test_json ORDER BY id;
+------+------------------------------------------------------------+-----------------------------+
| id   | j                                                          | json_exists_path(`j`, '$[2]') |
+------+------------------------------------------------------------+-----------------------------+
|    1 |                                                       NULL |                        NULL |
|    2 |                                                       null |                           0 |
|    3 |                                                       true |                           0 |
|    4 |                                                      false |                           0 |
|    5 |                                                        100 |                           0 |
|    6 |                                                      10000 |                           0 |
|    7 |                                                 1000000000 |                           0 |
|    8 |                                    1152921504606846976 |                           0 |
|    9 |                                                       6.18 |                           0 |
|   10 |                                                    "abcd" |                           0 |
|   11 |                                                      {} |                           0 |
|   12 |                                    {"k1":"v31","k2":300} |                           0 |
|   13 |                                                      [] |                           0 |
|   14 |                                                [123,456] |                           0 |
|   15 |                                           ["abc","def"] |                           0 |
|   16 |                             [null,true,false,100,6.18,"abc"] |                           1 |
|   17 |                           [{"k1":"v41","k2":400},1,"a",3.14] |                           1 |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} |                           0 |
|   26 |                                     {"k1":"v1","k2":200} |                           0 |
+------+------------------------------------------------------------+-----------------------------+
19 rows in set (0.02 sec)
```

get the datatype of a field in json by json_type

- return the data type of the field specified by json path, NULL if not existed.

```
mysql> SELECT id, j, json_type(j, '$') FROM test_json ORDER BY id;
+------+------------------------------------------------------------+----------------------+
| id   | j                                                          | json_type(`j`, '$') |
+------+------------------------------------------------------------+----------------------+
|    1 |                                                       NULL | NULL                 |
|    2 |                                                       null | null                 |
|    3 |                                                       true | bool                 |
|    4 |                                                      false | bool                 |
|    5 |                                                        100 | int                  |
|    6 |                                                      10000 | int                  |
|    7 |                                                 1000000000 | int                  |
|    8 |                                    1152921504606846976 | bigint               |
|    9 |                                                       6.18 | double               |
|   10 |                                                    "abcd" | string               |
|   11 |                                                      {} | object               |
|   12 |                                    {"k1":"v31","k2":300} | object               |
|   13 |                                                      [] | array                |
```

```
|   14 |                                                [123,456] | array                |
|   15 |                                             ["abc","def"] | array                |
|   16 |                               [null,true,false,100,6.18,"abc"] | array                |
|   17 |                             [{"k1":"v41","k2":400},1,"a",3.14] | array                |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} | object               |
|   26 |                                          {"k1":"v1","k2":200} | object               |
+------+-------------------------------------------------------------+----------------------+
19 rows in set (0.02 sec)

mysql> select id, j, json_type(j, '$.k1') from test_json order by id;
+------+-------------------------------------------------------------+-------------------------+
| id   | j                                                           | json_type(`j`, '$.k1') |
+------+-------------------------------------------------------------+-------------------------+
|    1 |                                                        NULL | NULL                    |
|    2 |                                                        null | NULL                    |
|    3 |                                                        true | NULL                    |
|    4 |                                                       false | NULL                    |
|    5 |                                                         100 | NULL                    |
|    6 |                                                       10000 | NULL                    |
|    7 |                                                  1000000000 | NULL                    |
|    8 |                                           1152921504606846976 | NULL                    |
|    9 |                                                        6.18 | NULL                    |
|   10 |                                                      "abcd" | NULL                    |
|   11 |                                                         {} | NULL                    |
|   12 |                                       {"k1":"v31","k2":300} | string                  |
|   13 |                                                         [] | NULL                    |
|   14 |                                                  [123,456] | NULL                    |
|   15 |                                               ["abc","def"] | NULL                    |
|   16 |                                 [null,true,false,100,6.18,"abc"] | NULL                    |
|   17 |                               [{"k1":"v41","k2":400},1,"a",3.14] | NULL                    |
|   18 | {"k1":"v31","k2":300,"a1":[{"k1":"v41","k2":400},1,"a",3.14]} | string                  |
|   26 |                                          {"k1":"v1","k2":200} | string                  |
+------+-------------------------------------------------------------+-------------------------+
19 rows in set (0.03 sec)
```

FAQ

1. Is there a difference between null in JSON and NULL in SQL (i.e., IS NULL)?

Yes, there is a difference. In JSON, null (e.g., {“key1”: null}) means that the key key1 exists and its value is explicitly null. This is a special type that gets encoded into JSON binary.

In contrast, SQL NULL (when using IS NULL) can indicate that the key doesn’t exist at all in the JSON object.

For example:

```
mysql> SELECT JSON_EXTRACT_STRING('{"key1" : null}', "$.key1") IS NULL;
+-----------------------------------------------------------+
| JSON_EXTRACT_STRING('{"key1" : null}', "$.key1") IS NULL |
+-----------------------------------------------------------+
|                                                         0 |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_EXTRACT_STRING('{"key1" : null}', "$.key_not_exist") IS NULL;
+-------------------------------------------------------------------+
| JSON_EXTRACT_STRING('{"key1" : null}', "$.key_not_exist") IS NULL |
+-------------------------------------------------------------------+
|                                                                 1 |
```

```
+------------------------------------------------------------------+
1 row in set (0.01 sec)
```

2. What's the difference between `GET_JSON_XXX` and `JSON_EXTRACT_XXX` functions, and how should I choose between them?

The `GET_JSON_XXX` functions are designed for use on string types — they extract values directly from raw JSON strings. On the other hand, `JSON_EXTRACT_XXX` functions are implemented specifically for the JSON data type and are optimized for it.

keywords

JSONB, JSON, json_parse, json_parse_error_to_null, json_parse_error_to_value, json_extract, json_extract_isnull, json_extract_bool, json_extract_int, json_extract_bigint, json_extract_double, json_extract_string, json_exists_path, json_type

VARIANT

VARIANT

Description

Introduced a new data type VARIANT in Doris 2.1, which can store semi-structured JSON data. It allows storing complex data structures containing different data types (such as integers, strings, boolean values, etc.) without the need to define specific columns in the table structure beforehand. The VARIANT type is particularly useful for handling complex nested structures that may change at any time. During the writing process, this type can automatically infer column information based on the structure and types of the columns, dynamicly merge written schemas. It stores JSON keys and their corresponding values as columns and dynamic sub-columns.

Note

Advantages over JSON Type:

1. Different storage methods: The JSON type is stored in binary JSONB format, and the entire JSON is stored row by row in segment files. In contrast, the VARIANT type infers types during writing and stores the written JSON columns. It has a higher compression ratio compared to the JSON type, providing better storage efficiency.
2. Query: Querying does not require parsing. VARIANT fully utilizes columnar storage, vectorized engines, optimizers, and other components in Doris, providing users with extremely high query performance. Below are test results based on clickbench data:

	Storage Space
Predefined Static Columns	12.618 GB
VARIANT Type	12.718 GB
JSON Type	35.711 GB

Saves approximately 65% storage capacity

Query Counts	Predefined Static Columns	VARIANT Type	JSON Type
First Query (cold)	233.79s	248.66s	**Most queries timeout**
Second Query (hot)	86.02s	94.82s	789.24s
Third Query (hot)	83.03s	92.29s	743.69s

test case contains 43 queries

8x faster query, query performance comparable to static columns

Example

Demonstrate the functionality and usage of VARIANT with an example covering table creation, data import, and query cycle.

Table Creation Syntax Create a table, using the `VARIANT` keyword in the syntax.

```
-- Without index
CREATE TABLE IF NOT EXISTS ${table_name} (
    k BIGINT,
    v VARIANT
)
table_properties;

-- Create an index on the v column, optionally specify the tokenize method, default is untokenized
CREATE TABLE IF NOT EXISTS ${table_name} (
    k BIGINT,
    v VARIANT,
    INDEX idx_var(v) USING INVERTED [PROPERTIES("parser" = "english|unicode|chinese")] [COMMENT 'your comment']
)
table_properties;

-- Create an bloom filter on v column, to enhance query seed on sub columns
CREATE TABLE IF NOT EXISTS ${table_name} (
    k BIGINT,
    v VARIANT
)
...
properties("replication_num" = "1", "bloom_filter_columns" = "v");
```

Query Syntax

```
-- use v['a']['b'] format for example, v['properties']['title'] type is VARIANT
SELECT v['properties']['title'] from ${table_name}
```

Example based on the GitHub events dataset

Here, github events data is used to demonstrate the table creation, data import, and query using VARIANT. The below is a formatted line of data:

```
{
  "id": "14186154924",
  "type": "PushEvent",
  "actor": {
    "id": 282080,
    "login": "brianchandotcom",
    "display_login": "brianchandotcom",
    "gravatar_id": "",
    "url": "https://api.github.com/users/brianchandotcom",
    "avatar_url": "https://avatars.githubusercontent.com/u/282080?"
  },
  "repo": {
    "id": 1920851,
    "name": "brianchandotcom/liferay-portal",
    "url": "https://api.github.com/repos/brianchandotcom/liferay-portal"
  },
  "payload": {
    "push_id": 6027092734,
    "size": 4,
```

```
    "distinct_size": 4,
    "ref": "refs/heads/master",
    "head": "91edd3c8c98c214155191feb852831ec535580ba",
    "before": "abb58cc0db673a0bd5190000d2ff9c53bb51d04d",
    "commits": [""]
  },
  "public": true,
  "created_at": "2020-11-13T18:00:00Z"
}
```

Table Creation

- Created three columns of VARIANT type: `actor`, `repo`, and `payload`.
- Simultaneously created an inverted index, `idx_payload`, for the `payload` column while creating the table.
- Specified the index type as inverted using `USING INVERTED`, aimed at accelerating conditional filtering of sub-columns.
- `PROPERTIES("parser" = "english")` specified the adoption of English tokenization.

```
CREATE DATABASE test_variant;
USE test_variant;
CREATE TABLE IF NOT EXISTS github_events (
    id BIGINT NOT NULL,
    type VARCHAR(30) NULL,
    actor VARIANT NULL,
    repo VARIANT NULL,
    payload VARIANT NULL,
    public BOOLEAN NULL,
    created_at DATETIME NULL,
    INDEX idx_payload (`payload`) USING INVERTED PROPERTIES("parser" = "english") COMMENT 'inverted index for payload'
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(id) BUCKETS 10
properties("replication_num" = "1");
```

> **1. Creating an index on VARIANT columns, such as when there are numerous sub-columns in payload, might lead to an excessive number of index columns, impacting write performance.**
>
> 2. The tokenization properties for the same VARIANT column are uniform. If you have varied tokenization requirements, consider creating multiple VARIANT columns and specifying index properties separately for each.

Using Streamload for Import

Importing gh_2022-11-07-3.json, which contains one hour's worth of GitHub events data.

```
wget https://qa-build.oss-cn-beijing.aliyuncs.com/regression/variant/gh_2022-11-07-3.json

curl --location-trusted -u root:  -T gh_2022-11-07-3.json -H "read_json_by_line:true" -H "format:json"
http://127.0.0.1:18148/api/test_variant/github_events/_strea
m_load

{
    "TxnId": 2,
    "Label": "086fd46a-20e6-4487-becc-9b6ca80281bf",
    "Comment": "",
```

```
    "TwoPhaseCommit": "false",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 139325,
    "NumberLoadedRows": 139325,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 633782875,
    "LoadTimeMs": 7870,
    "BeginTxnTimeMs": 19,
    "StreamLoadPutTimeMs": 162,
    "ReadDataTimeMs": 2416,
    "WriteDataTimeMs": 7634,
    "CommitAndPublishTimeMs": 55
}
```

Confirm the successful import.

```
-- View the number of rows.
mysql> select count() from github_events;
+----------+
| count(*) |
+----------+
|   139325 |
+----------+
1 row in set (0.25 sec)

-- Random select one row
mysql> select * from github_events limit 1;
+-------------+-----------+-------------------
```

```
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+--------+--------------------+
1 row in set (0.23 sec)
```

Running desc command to view schema information, sub-columns will automatically expand at the storage layer and undergo type inference.

```
mysql> desc test_variant;
+-----------------------------------------------------------+------------+------+-------+---------+-------+
| Field                                                     | Type       | Null | Key   | Default | Extra |
+-----------------------------------------------------------+------------+------+-------+---------+-------+
| id                                                        | BIGINT     | No   | true  | NULL    |       |
| type                                                      | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| actor                                                     | VARIANT    | Yes  | false | NULL    | NONE  |
| created_at                                                | DATETIME   | Yes  | false | NULL    | NONE  |
| payload                                                   | VARIANT    | Yes  | false | NULL    | NONE  |
| public                                                    | BOOLEAN    | Yes  | false | NULL    | NONE  |
+-----------------------------------------------------------+------------+------+-------+---------+-------+
6 rows in set (0.07 sec)

mysql> set describe_extend_variant_column = true;
Query OK, 0 rows affected (0.01 sec)

mysql> desc test_variant;
+-----------------------------------------------------------+------------+------+-------+---------+-------+
| Field                                                     | Type       | Null | Key   | Default | Extra |
+-----------------------------------------------------------+------------+------+-------+---------+-------+
| id                                                        | BIGINT     | No   | true  | NULL    |       |
| type                                                      | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| actor                                                     | VARIANT    | Yes  | false | NULL    | NONE  |
| actor.avatar_url                                          | TEXT       | Yes  | false | NULL    | NONE  |
| actor.display_login                                       | TEXT       | Yes  | false | NULL    | NONE  |
| actor.id                                                  | INT        | Yes  | false | NULL    | NONE  |
| actor.login                                               | TEXT       | Yes  | false | NULL    | NONE  |
| actor.url                                                 | TEXT       | Yes  | false | NULL    | NONE  |
| created_at                                                | DATETIME   | Yes  | false | NULL    | NONE  |
| payload                                                   | VARIANT    | Yes  | false | NULL    | NONE  |
| payload.action                                            | TEXT       | Yes  | false | NULL    | NONE  |
| payload.before                                            | TEXT       | Yes  | false | NULL    | NONE  |
| payload.comment.author_association                        | TEXT       | Yes  | false | NULL    | NONE  |
| payload.comment.body                                      | TEXT       | Yes  | false | NULL    | NONE  |
....
+-----------------------------------------------------------+------------+------+-------+---------+-------+
406 rows in set (0.07 sec)
```

DESC can be used to specify partition and view the schema of a particular partition. The syntax is as follows:

```
DESCRIBE ${table_name} PARTITION ($partition_name);
```

Querying

> **When utilizing filtering and aggregation functionalities to query sub-columns, additional casting operations need to be performed on sub-columns (because the storage types are not necessarily fixed and require a unified SQL type).**

For instance, `SELECT * FROM tbl where CAST(var['titile'] as text) MATCH "hello world"` The simplified examples below illustrate how to use VARIANT for querying: The following are three typical query scenarios

1. Retrieve the top 5 repositories based on star count from the `github_events` table.

```
mysql> SELECT
    ->     cast(repo['name'] as text) as repo_name, count() AS stars
    -> FROM github_events
    -> WHERE type = 'WatchEvent'
    -> GROUP BY repo_name
    -> ORDER BY stars DESC LIMIT 5;
+--------------------------+-------+
| repo_name                | stars |
+--------------------------+-------+
| aplus-framework/app      |    78 |
| lensterxyz/lenster       |    77 |
| aplus-framework/database |    46 |
| stashapp/stash           |    42 |
| aplus-framework/image    |    34 |
+--------------------------+-------+
5 rows in set (0.03 sec)
```

2. Retrieve the count of comments containing “doris”.

```
-- implicit cast `payload['comment']['body']` to string type
mysql> SELECT
    ->     count() FROM github_events
    ->     WHERE payload['comment']['body'] MATCH 'doris';
+---------+
| count() |
+---------+
|       3 |
+---------+
1 row in set (0.04 sec)
```

3. Query the issue number with the highest number of comments along with its corresponding repository.

```
mysql> SELECT
    ->   cast(repo['name'] as string) as repo_name,
    ->   cast(payload['issue']['number'] as int) as issue_number,
    ->   count() AS comments,
    ->   count(
    ->     distinct cast(actor['login'] as string)
    ->   ) AS authors
    -> FROM  github_events
    -> WHERE type = 'IssueCommentEvent' AND (cast(payload["action"] as string) = 'created') AND (cast(payload["issue"]["number"] as
int) > 10)
    -> GROUP BY repo_name, issue_number
    -> HAVING authors >= 4
    -> ORDER BY comments DESC, repo_name
    -> LIMIT 50;
+-----------------------------------+--------------+----------+---------+
| repo_name                         | issue_number | comments | authors |
+-----------------------------------+--------------+----------+---------+
| facebook/react-native             |        35228 |        5 |       4 |
| swsnu/swppfall2022-team4          |           27 |        5 |       4 |
| belgattitude/nextjs-monorepo-example |         2865 |        4 |       4 |
```

```
+-------------------------------------+-------------+----------+---------+
3 rows in set (0.03 sec)
```

Usage Restrictions and Best Practices

There are several limitations when using the VARIANT type: Dynamic columns of VARIANT are nearly as efficient as predefined static columns. When dealing with data like logs, where fields are often added dynamically (such as container labels in Kubernetes), parsing JSON and inferring types can generate additional costs during write operations. Therefore, it's recommended to keep the number of columns for a single import below 1000.

Ensure consistency in types whenever possible. Doris automatically performs compatible type conversions. When a field cannot undergo compatible type conversion, it is uniformly converted to JSONB type. The performance of JSONB columns may degrade compared to columns like int or text.

1. tinyint -> smallint -> int -> bigint, integer types can be promoted following the direction of the arrows.
2. float -> double, floating-point numbers can be promoted following the direction of the arrow.
3. text, string type.
4. JSON, binary JSON type.

When the above types cannot be compatible, they will be transformed into JSON type to prevent loss of type information. If you need to set a strict schema in VARIANT, the VARIANT MAPPING mechanism will be introduced soon.

Other limitations include:

- VARIANT columns can only create inverted indexes or bloom filter to speed up query.
- Using the **RANDOM** mode or group commit mode is recommended for higher write performance.
- Non-standard JSON types such as date and decimal should ideally use static types for better performance, since these types are infered to text type.
- Arrays with dimensions of 2 or higher will be stored as JSONB encoding, which might perform less efficiently than native arrays.
- Not supported as primary or sort keys.
- Queries with filters or aggregations require casting. The storage layer eliminates cast operations based on storage type and the target type of the cast, speeding up queries.
- Reading a VARIANT column inherently involves scanning all its subfields. If the column contains numerous subfields, this can lead to substantial scan overhead and negatively impact query performance. To optimize performance when you need to retrieve the entire column, consider adding an additional column of type STRING or JSONB to store the raw JSON string. Example:

```
-- Lead to scan all subfields of data_variant
CREATE TABLE example_table (
  id INT,
  data_variant VARIANT,
);
SELECT * FROM example_table WHERE data_variant LIKE '%doris%'

-- Better performance for `LIKE`
CREATE TABLE example_table (
  id INT,
  data_variant VARIANT,
  data_string STRING
```

```
);
SELECT * FROM example_table WHERE data_string LIKE '%doris%'
```

Tuning Techniques for Column-Count Limits:

Note: If the number of sub-columns exceeds 5,000, higher requirements for memory and configuration apply. On a single machine, aim for at least 128 GB of RAM and 32 CPU cores.

1. In BE configuration, adjust `variant_max_merged_tablet_schema_size=n`, where n should be greater than the actual number of columns (not recommended to exceed 10,000).
2. Be aware that extracting too many columns will put heavy pressure on compaction (import throughput must be throttled accordingly). Increasing the client-side import `batch_size`—based on memory usage—can reduce write amplification during compaction. Alternatively, enable `group_commit` (a table property) and appropriately increase `group_commit_interval_ms` and `group_commit_data_bytes`.
3. If your queries do not require bucket pruning, use random bucketing and enable the load_to_single_tablet import setting (an import configuration) to reduce compaction write amplification.
4. In BE configuration, adjust `max_cumu_compaction_threads` according to import pressure; ensure at least 8 threads.
5. In BE configuration, set `vertical_compaction_num_columns_per_group=500` to improve grouped-compaction efficiency, although this increases memory overhead.
6. In BE configuration, set `segment_cache_memory_percentage=20` to increase segment cache capacity and improve metadata caching efficiency.
7. Monitor the Compaction Score closely. A continuously rising score indicates that compaction cannot keep up (import pressure should be reduced accordingly).
8. Using `SELECT *` or `SELECT variant` can significantly increase cluster-wide pressure, potentially causing timeouts or out-of-memory errors. It is recommended to include path information in queries—for example, `SELECT variant['path_1']`.

FAQ

1. Streamload Error: [CANCELLED][INTERNAL_ERROR] tablet error: [DATA_QUALITY_ERROR] Reached max column size limit 2048.

Due to compaction and metadata storage limitations, the VARIANT type imposes a limit on the number of columns, with the default being 2048 columns. You can adjust the BE configuration `variant_max_merged_tablet_schema_size` accordingly, but it is not recommended to exceed 4096 columns(Requires higher-spec hardware).

2. Is there a difference between null in the VARIANT type (e.g., `{"key": null}`) and SQL NULL (i.e., IS NULL)?

No, there is no difference — in the VARIANT type, they are considered equivalent.

Keywords

```
VARIANT
```

HLL (HyperLogLog)

HLL (HyperLogLog)

Description

HLL

HLL cannot be used as a key column. The columns of the QUANTILE_STATE type can be used in Aggregate tables, Duplicate tables and Unique tables. When used in an Aggregate table, the aggregation type is QUANTILE_UNION when building table. The user does not need to specify the length and default value. The length is controlled within the system according to the degree of data aggregation. And HLL columns can only be queried or used through the matching hll_union_agg, hll_raw_agg, hll_cardinality, and hll_hash.

HLL is approximate count of distinct elements, and its performance is better than Count Distinct when the amount of data is large. The error of HLL is usually around 1%, sometimes up to 2%.

example

```
select hour, HLL_UNION_AGG(pv) over(order by hour) uv from(
   select hour, HLL_RAW_AGG(device_id) as pv
   from metric_table -- Query the accumulated UV per hour
   where datekey=20200922
group by hour order by 1
) final;
```

keywords

HLL,HYPERLOGLOG

BITMAP

BITMAP

Description

BITMAP

The columns of the BITMAP type can be used in Aggregate table, Unique table or Duplicate table. When used in a Unique table or Duplicate table, they must be used as non-key columns. When used in an Aggregate table, they must be used as non-key columns, and the aggregation type is BITMAP_UNION when building the table. The user does not need to specify the length and default value. The length is controlled within the system according to the degree of data aggregation. And the BITMAP column can only be queried or used by supporting functions such as bitmap_union_count, bitmap_union, bitmap_hash and bitmap_hash64.

The use of BITMAP in offline scenarios will affect the import speed. In the case of a large amount of data, the query speed will be slower than HLL and better than Count Distinct. Note: If BITMAP does not use a global dictionary in real-time scenarios, using bitmap_hash() may cause an error of about one-thousandth. If the error rate is not tolerable, bitmap_hash64 can be used instead.

example

Create table example:

```
create table metric_table (
  datekey int,
  hour int,
  device_id bitmap BITMAP_UNION
)
aggregate key (datekey, hour)
distributed by hash(datekey, hour) buckets 1
properties(
  "replication_num" = "1"
);
```

Insert data example:

```
insert into metric_table values
(20200622, 1, to_bitmap(243)),
(20200622, 2, bitmap_from_array([1,2,3,4,5,434543])),
(20200622, 3, to_bitmap(287667876573));
```

Query data example:

```
select hour, BITMAP_UNION_COUNT(pv) over(order by hour) uv from(
   select hour, BITMAP_UNION(device_id) as pv
   from metric_table -- Query the accumulated UV per hour
   where datekey=20200622
```

```
group by hour order by 1
) final;
```

You can set the session variable `return_object_data_as_binary` to `true`. Consequently, the bitmap will be returned in binary format.

keywords

BITMAP

QUANTILE_STATE

QUANTILE_STATE

description

QUANTILE_STATE

In 2.0, we support the agg_state function, and it is recommended to use agg_state quantile_union(quantile_state not null) instead of this type.

QUANTILE_STATE cannot be used as a key column. The columns of the QUANTILE_STATE type can be used in Aggregate tables, Duplicate tables and Unique tables. When used in an Aggregate table, the aggregation type is QUANTILE_UNION when building table.

The user does not need to specify the length and default value. The length is controlled within the system according to the degree of data aggregation. And the QUANTILE_STATE column can only be queried or used through the supporting QUANTILE_PERCENT, QUANTILE_UNION and TO_QUANTILE_STATE functions. QUANTILE_STATE is a type for calculating the approximate value of quantiles. Different values with the same key are pre-aggregated during loading process. When the number of aggregated values does not exceed 2048, all data are recorded in detail. When the number of aggregated values is greater than 2048, [TDigest] is used. (https://github.com/tdunning/t-digest/blob/main/docs/t-digest-paper/histo.pdf) algorithm to aggregate (cluster) the data and save the centroid points after clustering.

related functions:

```
QUANTILE_UNION(QUANTILE_STATE):

  This function is an aggregation function, which is used to aggregate the intermediate results of different quantile calculations.
The result returned by this function is still QUANTILE_STATE


TO_QUANTILE_STATE(DOUBLE raw_data [,FLOAT compression]):

   This function converts a numeric type to a QUANTILE_STATE type
   The compression parameter is optional and can be set in the range [2048, 10000].
   The larger the value, the higher the precision of quantile approximation calculations, the greater the memory consumption, and
the longer the calculation time.
   An unspecified or set value for the compression parameter is outside the range [2048, 10000], run with the default value of 2048

QUANTILE_PERCENT(QUANTILE_STATE, percent):
   This function converts the intermediate result variable (QUANTILE_STATE) of the quantile calculation into a specific quantile
value
```

notice

Now QUANTILE_STATE can only used in Aggregate Model Tables. We should turn on the switch for the QUANTILE_STATE types feature with the following command before use:

```
$ mysql-client > admin set frontend config("enable_quantile_state_type"="true");
```

In this way the config will be reset after the FE process restarts. For permanent setting, you can add config `enable_quantile_state_type=true` inside fe.conf.

example

```
select QUANTILE_PERCENT(QUANTILE_UNION(v1), 0.5) from test_table group by k1, k2, k3;
```

keywords

```
QUANTILE_STATE, QUANTILE_UNION, TO_QUANTILE_STATE, QUANTILE_PERCENT
```

AGG_STATE

AGG_STATE

description

```
  AGG_STATE cannot be used as a key column, and the signature of the aggregation function must be declared at the same time when creating the table.
  User does not need to specify length and default value. The actual stored data size is related to the function implementation.
```

AGG_STATE can only be used with state /merge/union function combiner usage.

It should be noted that the signature of the aggregation function is also part of the type, and agg_state with different signatures cannot be mixed. For example, if the signature of the table creation statement is `max_by(int,int)`, then `max_by(bigint,int)` or `group_concat(varchar)` cannot be inserted. The nullable attribute here is also part of the signature. If you can confirm that you will not enter a null value, you can declare the parameter as not null, which can obtain a smaller storage size and reduce serialization/deserialization overhead.

Note: Because agg_state stores the intermediate result of the aggregation function, the read and write process is strongly dependent on the specific implementation of the aggregation function, if the implementation of the aggregation function is modified during the Doris version upgrade, it may cause incompatible situations. If the incompatibility occurs, the materialized view using the corresponding agg_state needs to be dropped and recreated, and the underlying aggregation table involved will be directly unavailable, so you need to use agg_state with caution.

example

Create table example:

```
-- after doris-2.1.1
create table a_table(
    k1 int null,
    k2 agg_state<max_by(int not null,int)> generic,
    k3 agg_state<group_concat(string)> generic
)
aggregate key (k1)
distributed BY hash(k1) buckets 3
properties("replication_num" = "1");

-- until doris-2.1.0
create table a_table(
    k1 int null,
    k2 agg_state max_by(int not null,int),
    k3 agg_state group_concat(string)
)
aggregate key (k1)
distributed BY hash(k1) buckets 3
properties("replication_num" = "1");
```

Here k2 and k3 use max_by and group_concat as aggregation types respectively.

Insert data example:

```
insert into a_table values(1,max_by_state(3,1),group_concat_state('a'));
insert into a_table values(1,max_by_state(2,2),group_concat_state('bb'));
insert into a_table values(2,max_by_state(1,3),group_concat_state('ccc'));
```

For the agg_state column, the insert statement must use the state function to generate the corresponding agg_state data, where the functions and input parameter types must completely correspond to agg_state.

Select data example:

```
mysql [test]>select k1,max_by_merge(k2),group_concat_merge(k3) from a_table group by k1 order by k1;
+------+--------------------+-------------------------+
| k1   | max_by_merge(`k2`) | group_concat_merge(`k3`) |
+------+--------------------+-------------------------+
|    1 |                  2 | bb,a                    |
|    2 |                  1 | ccc                     |
+------+--------------------+-------------------------+
```

If you need to get the actual result, you need to use the corresponding merge function.

```
mysql [test]>select max_by_merge(u2),group_concat_merge(u3) from (
select k1,max_by_union(k2) as u2,group_concat_union(k3) u3 from a_table group by k1 order by k1
) t;
+--------------------+-------------------------+
| max_by_merge(`u2`) | group_concat_merge(`u3`) |
+--------------------+-------------------------+
|                  1 | ccc,bb,a                |
+--------------------+-------------------------+
```

If you want to aggregate only the agg_state without getting the actual result during the process, you can use the union function.

For more examples, see datatype_p0/agg_state #### keywords

```
AGG_STATE
```

IPV4

IPV4

description

IPv4 type, stored in the form of UInt32 in 4 bytes, used to represent IPv4 addresses. The range of values is ['0.0.0.0', '255.255.255.255'].

Inputs that exceed the value range or have invalid format will return NULL

example

Create table example:

```
CREATE TABLE ipv4_test (
  `id` int,
  `ip_v4` ipv4
) ENGINE=OLAP
DISTRIBUTED BY HASH(`id`) BUCKETS 4
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

Insert data example:

```
insert into ipv4_test values(1, '0.0.0.0');
insert into ipv4_test values(2, '127.0.0.1');
insert into ipv4_test values(3, '59.50.185.152');
insert into ipv4_test values(4, '255.255.255.255');
insert into ipv4_test values(5, '255.255.255.256'); // invalid data
```

Select data example:

```
mysql> select * from ipv4_test order by id;
+------+-----------------+
| id   | ip_v4           |
+------+-----------------+
|    1 | 0.0.0.0         |
|    2 | 127.0.0.1       |
|    3 | 59.50.185.152   |
|    4 | 255.255.255.255 |
|    5 | NULL            |
+------+-----------------+
```

keywords

IPV4

IPV6

IPV6

description

IPv6 type, stored in UInt128 format in 16 bytes, used to represent IPv6 addresses. The range of values is ['::', 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff'].

Inputs that exceed the value range or have invalid format will return NULL

example

Create table example:

```
CREATE TABLE ipv6_test (
  `id` int,
  `ip_v6` ipv6
) ENGINE=OLAP
DISTRIBUTED BY HASH(`id`) BUCKETS 4
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

Insert data example:

```
insert into ipv6_test values(1, '::');
insert into ipv6_test values(2, '2001:16a0:2:200a::2');
insert into ipv6_test values(3, 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff');
insert into ipv6_test values(4, 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffffg'); // invalid data
```

Select data example:

```
mysql> select * from ipv6_test order by id;
+------+-----------------------------------------+
| id   | ip_v6                                   |
+------+-----------------------------------------+
|    1 | ::                                      |
|    2 | 2001:16a0:2:200a::2                     |
|    3 | ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff |
|    4 | NULL                                    |
+------+-----------------------------------------+
```

keywords

IPV6

Numeric Type Literal

Use numeric literals to specify fixed-point and floating-point numbers.

Integer Literal

Integers are represented as a sequence of digits. They may have a sign. For example: +1, -2, 345.

Doris determines the type used to store integer literals based on the input value. The range mapping is shown in the table below:

Value Range	Type
-2^7 to 2^7 - 1	TINYINT
-2^15 to 2^15 - 1	SMALLINT
-2^31 to 2^31 - 1	INT
-2^63 to 2^63 - 1	BIGINT
-2^127 to 2^127 - 1	LARGEINT

Fixed and Floating Point Numeric Literals

Fixed and floating point numeric literals can have an integer part, a fractional part, or both. They may have a sign. For example: 1, .2, 3.4, -5, -6.78, +9.10.

They can also be represented in scientific notation, including the significand and exponent. Either part or both can have a sign. For example: 1.2E3, 1.2E-3, -1.2E3, -1.2E-3.

Numbers represented in this way are preferentially parsed as fixed-point numbers. The range of supported fixed-point numbers is controlled by the variable `enable_decimal256`. When `enable_decimal256` is `TRUE`, the maximum precision is 76. When `enable_decimal256` is `FALSE`, the maximum precision is 38.

When the precision required by a number exceeds the maximum value representable by fixed-point numbers, it is parsed as a floating-point number, with the type DOUBLE.

String Type Literal

Description

A string is a sequence of bytes or characters enclosed in single quotes (') or double quotes (“). For example:

```
'a string'
"another string"
```

Escape Characters

In strings, certain sequences have special meanings unless the NO_BACKSLASH_ESCAPES SQL mode is enabled. These sequences start with a backslash (), which is known as an escape character. The escape characters recognized by Doris are listed in the table below:

Escape Character	Meaning
\0	ASCII character NUL (0x00)
\'	Single quote (')
\"	Double quote (“)
\b	Backspace
\n	Newline
\r	Carriage return
\t	Tab
\Z	ASCII 26 (Control+Z)
\\	Backslash (\)
\%	Percent sign %. For details, see the notes below
_	Underscore _. For details, see the notes below

Notes

> 1. In pattern matching contexts, % and _ are usually interpreted as wildcards, but using the sequences \% and _ can search for literal instances of % and _. For more information, see the description of the LIKE operator in the “Pattern Matching Operators” section. If \% or _ are used outside of pattern contexts, they will be calculated as the string \% and _, not as % and _.
> 2. Backslashes in escape characters not listed in the table are ignored. For example, `'\y'` and `'y'` are equivalent.

Using Quotes in String Literals

There are several ways to include quote characters in strings:

- In a string enclosed in single quotes (‘), single quotes can be written as two single quotes (’’).
- In a string enclosed in double quotes (“), double quotes can be written as two double quotes (”).
- Add an escape character (\) before the quote character.
- When including a single quote in a string enclosed in double quotes, no special treatment is necessary, nor must the single quote be doubled or escaped. Similarly, when including a double quote in a string enclosed in single quotes, no special treatment is required.

Date Type Literal

Description

Consistent with standard SQL, Doris requires the use of type keywords and strings to specify date literals. The space between the keyword and string is optional. For example:

```
DATE '2008-08-08'
TIMESTAMP '2008-08-08 20:08:08'
```

Date Formats

DATE Literal

- Use a string separated by - in the format `'YYYY-MM-DD'` or `'YY-MM-DD'`. Doris also supports MySQL's non-standard separator formats, but their use is not recommended.
- As a string without separators, use the format `'YYYYMMDD'` or `'YYMMDD'` (provided the string is meaningful as a date).

DATETIME Literal

- Use a string separated by - in the format `'YYYY-MM-DD hh:mm:ss'` or `'YY-MM-DD hh:mm:ss'`. Doris also supports MySQL's non-standard separator formats, but their use is not recommended. The separator between date and time can be a space () or `T`. **Unlike MySQL 8.4 and earlier versions, Doris does not support any other separators between time and date.**
- As a string without separators, use the format `'YYYYMMDDhhmmss'` or `'YYMMDDhhmmss'` (provided the string is meaningful as a date).

DATETIME literals can include a fractional second part with a precision up to microseconds (six digits). The fractional part should always be separated from the rest of the time with a dot (.); other fractional second separators are not recognized.

Two-Digit Years

Dates containing two-digit year values are ambiguous because the century is unknown. Doris uses the following rules to interpret two-digit year values:

- Year values between 70-99 are interpreted as years 1970-1999.
- Year values between 00-69 are interpreted as years 2000-2069.

Time Zones

DATE and DATETIME literals can use time zone suffixes. When using time zones, the time zone must be immediately adjacent to the previous date or time part, with no spaces in between. For example:

```
TIMESTAMP '2008-08-08 20:08:08+08:00'
```

The time zone formats supported by Doris are:

- Time zone offset: Format is `{+ | -}hh:mm`. For example, for UTC+8, it is `+08:00`.
- Time zone name. For example, Shanghai time zone is `Asia/Shanghai`.

Handling of Invalid Values

When encountering values that cannot be parsed into valid date literals, Doris will report an error directly. For example:

```
SELECT date '071332'
```

Will produce the following error:

```
date/datetime literal [071332] is invalid
```

NULL

Basic Introduction to NULL

If a column in a row has no value, it is said to be NULL. NULL can appear in any column not restricted by a “NOT NULL” constraint. Use NULL when the actual value is unknown or when a value has no meaning.

Do not use NULL to represent numerical zero or an empty string. They are not equivalent.

Any arithmetic expression that includes NULL will always result in NULL. For example, adding NULL to 10 still results in NULL. In fact, when given NULL as an operand, all operators return NULL.

NULL as a Function Argument

When a NULL is provided as an argument, most aggregate functions return NULL. You can use the IFNULL function to return a value when a NULL value occurs. For example, the expression IFNULL(arg, 0) returns 0 when arg is NULL and returns its value when arg is not NULL. For the specific behavior of each function, please refer to the “Functions” section.

NULL and Comparison Operators

To test for NULL results, only the comparison conditions IS NULL and IS NOT NULL can be used. If a condition that depends on NULL is used, the result is UNKNOWN. Since NULL represents missing data, NULL cannot be equal to or not equal any value or another NULL.

NULL in Conditions

Conditions that evaluate to UNKNOWN behave almost the same as FALSE. For example, a SELECT statement with a condition that evaluates to UNKNOWN in the WHERE clause will not return any rows. However, the difference between conditions that evaluate to UNKNOWN and FALSE is that further operations on the evaluation result of an UNKNOWN condition will also be evaluated as UNKNOWN. Therefore, the calculation result of NOT FALSE is TRUE, but the calculation result of NOT UNKNOWN is UNKNOWN.

The table below shows examples of various evaluations involving NULL in conditions. If a condition that evaluates to UNKNOWN is used in the WHERE clause of a SELECT statement, then the query will not return any rows.

Condition	Value of A	Evaluation
`a IS NULL`	`10`	`FALSE`
`a IS NOT NULL`	`10`	`TRUE`
`a IS NULL`	`NULL`	`TRUE`
`a IS NOT NULL`	`NULL`	`FALSE`
`a = NULL`	`10`	`UNKNOWN`

Condition	Value of A	Evaluation
a != NULL	10	UNKNOWN
a = NULL	NULL	UNKNOWN
a != NULL	NULL	UNKNOWN
a = 10	NULL	UNKNOWN
a != 10	NULL	UNKNOWN

Object Identifier

Description

Each database object, such as tables, columns, and indexes, has a name. In SQL statements, these names are referred to as object identifiers. Identifiers can be quoted or unquoted. If an identifier contains special characters or reserved keywords, it must be quoted with backticks (`) each time it is referenced. For more details on reserved keywords, please refer to the Reserved Keywords section.

Object Identifier Restrictions

In Doris, object identifiers can be controlled by the variable `enable_unicode_name_support` to determine whether Unicode characters are supported. When Unicode character support is enabled, identifiers can use any language characters in Unicode. However, punctuation marks and other characters are not allowed.

In Doris, different objects have different restrictions on identifiers, and the specific restrictions for different objects are listed below.

Description

Each database object, such as tables, columns, and indexes, has a name. In SQL statements, these names are referred to as object identifiers. Identifiers can be quoted or unquoted. If an identifier contains special characters or reserved keywords, it must be quoted with backticks (`) each time it is referenced. For more details on reserved keywords, please refer to the Reserved Keywords section.

Object Identifier Restrictions

In Doris, object identifiers can be controlled by the variable `enable_unicode_name_support` to determine whether Unicode characters are supported. When Unicode character support is enabled, identifiers can use any language characters in Unicode. However, punctuation marks and other characters are not allowed.

In Doris, different objects have different restrictions on identifiers, and the specific restrictions for different objects are listed below.

Table Names

Mode	Identifier Restrictions
Closed Unicode Mode	`^[a-zA-Z][a-zA-Z0-9-_]*$`
Enabled Unicode Mode	`^[a-zA-Z\\p{L}][a-zA-Z0-9-_\\p{L}]*$`

Column Names

Mode	Identifier Restrictions
Closed Unicode Mode	`^[_a-zA-Z@0-9\\s/][.a-zA-Z0-9_+-/?@#$%^&*\"\\s,:]{0,255}$`

Mode	Identifier Restrictions
Enabled Unicode Mode	`^[_a-zA-Z@0-9\\p{L}][.a-zA-Z0-9_+-/?@#$%^&*\\p{L}]{0,255}$`

OUTFILE Names

Mode	Identifier Restrictions
Closed Unicode Mode	`^[_a-zA-Z][a-zA-Z0-9-_]{0,63}$`
Enabled Unicode Mode	`^[_a-zA-Z\\p{L}][a-zA-Z0-9-_\\p{L}]{0,63}$`

User Names

Mode	Identifier Restrictions
Closed Unicode Mode	`^[a-zA-Z][a-zA-Z0-9.-_]*$`
Enabled Unicode Mode	`^[a-zA-Z\\p{L}][a-zA-Z0-9.-_\\p{L}]*$`

LABEL Names

Mode	Identifier Restrictions
Closed Unicode Mode	`^[-_A-Za-z0-9:]{1,N}$`, where `N` is determined by the `label_regex_length` configuration in FE, with a default value of 128.
Enabled Unicode Mode	`^[-_A-Za-z0-9:\\p{L}]{1,N}$`, where `N` is determined by the `label_regex_length` configuration in FE, with a default value of 128.

Others

Mode	Identifier Restrictions
Closed Unicode Mode	`^[a-zA-Z][a-zA-Z0-9-_]{0,63}$`
Enabled Unicode Mode	`^[a-zA-Z\\p{L}][a-zA-Z0-9-_\\p{L}]{0,63}$`

Reserved Keywords

Description

Reserved keywords are words that have special meanings in SQL. Reserved keywords are a special category of keywords; if you want to use them as object identifiers, you must use backticks (`). For example, `FROM`. For more details on object identifiers, please refer to the Object Identifiers section.

List of Reserved Keywords

- ACCOUNT_LOCK
- ACCOUNT_UNLOCK
- ADD
- ADMIN
- ALL
- ALTER
- ANALYZE
- AND
- ANTI
- APPEND
- AS
- ASC
- AUTO
- BACKEND
- BETWEEN
- BIGINT
- BINARY
- BINLOG
- BY
- CANCEL
- CASE
- CAST
- CHAR
- CHARACTER
- CLEAN
- COLLATE
- COLUMN
- CONSTRAINT_LIGHT_CHANGE_PROCESS
- CREATE
- CROSS
- CUBE
- CURRENT
- DATABASE
- DATABASES

- DECOMMISSION
- DEFAULT
- DELETE
- DESC
- DESCRIBE
- DISK
- DISTINCT
- DISTRIBUTED
- DISTRIBUTION
- DIV
- DOUBLE
- DROP
- DROPP
- DUMP
- DUPLICATE
- ELSE
- ENTER
- EXCEPT
- EXECUTE
- EXISTS
- EXPLAIN
- EXPORT
- EXTENDED
- EXTRACT
- FALSE
- FLOAT
- FOLLOWER
- FOLLOWING
- FOR
- FORCE
- FOREIGN
- FROM
- FRONTEND
- FULL
- FUNCTIONS
- GRANT
- GRANTS
- GROUP
- HAVING
- HLL
- IF
- IN
- INDEX
- INFILE
- INNER
- INSERT
- INSTALL

- INT
- INTEGER
- INTERMEDIATE
- INTERSECT
- INTERVAL
- INTO
- IS
- JOIN
- KEY
- KEYS
- KILL
- LARGEINT
- LATERAL
- LEFT
- LIKE
- LIMIT
- LIST
- LOAD
- LOW_PRIORITY
- MATCH
- MAXVALUE
- MINUS
- NATURAL
- NOT
- NULL
- OBSERVER
- ON
- OR
- ORDER
- OUTER
- OUTFILE
- OVERWRITE
- PARTITION
- PLAY
- PRECEDING
- PREPARE
- PRIMARY
- PROCEDURE
- RANGE
- READ
- REAL
- REBALANCE
- REFERENCES
- REGEXP
- RELEASE
- RENAME
- REPAIR

- REPLICA
- REVOKE
- RIGHT
- ROLE
- ROLES
- ROW
- ROWS
- SCHEMAS
- SELECT
- SEMI
- SET
- SETS
- SHOW
- SIGNED
- SMALLINT
- SQL_BLOCK_RULE
- SUPERUSER
- SWITCH
- SYNC
- SYSTEM
- TABLE
- TABLESAMPLE
- TABLET
- TABLETS
- TERMINATED
- THEN
- TINYINT
- TO
- TRASH
- TRIM
- TRUE
- TYPE_CAST
- UNBOUNDED
- UNINSTALL
- UNION
- UNIQUE
- UNSIGNED
- UPDATE
- USE
- USING
- VALUES
- WHEN
- WHERE
- WHITELIST
- WITH
- WORKLOAD
- WRITE

- XOR

Variables

Description

In Doris, variables are divided into system variables and user variables. Both are case-insensitive.

System variables affect the behavior of Doris. Both system variables and user variables can be used in user queries.

System Variables

System variables are a set of variables predefined by Doris to control database behavior and performance. The main features are as follows:

- Variable Types:
    - Read-only variables: These variables are set by the system and cannot be modified by users, such as `version`, `current_timestamp`, etc.
    - Modifiable variables: Users can modify the values of these variables at runtime, such as `exec_mem_limit`, `time_zone`, etc.
- Scope:
    - Global variables (Global): Affect all sessions and set with `SET GLOBAL`.
    - Session variables (Session): Only affect the current session and set with `SET`.
    - Some variables have both global and session scopes.
- Access Methods:
    - Use `SHOW VARIABLES` to view all system variables.
    - Use `SHOW VARIABLES LIKE 'pattern'` to view specific variables by pattern matching.
- Persistence:
    - Modifications to global variables are reset to default values after a system restart; session restarts do not reset to default values.
    - Changes to session variables are lost after the session ends.

User-Defined Variables

User-defined variables are a mechanism for temporarily storing data within a session. The main features are as follows:

- Naming Rules:

    - Must be prefixed with `@`.
    - Variable names can include letters, numbers, and underscores.
    - Case-insensitive.
- Scope:
    - Only valid within the current session.
    - Automatically destroyed after the session ends.
    - Variables with the same name in different sessions are independent of each other.
- Assignment Methods:
    - Use `SET @var_name = value` syntax for assignment.
    - Supports assigning values using expression results.
- Data Types:
    - Can store numbers (integers, floating-point numbers).
    - Can store strings.
    - Can store date-time values.
    - Can store NULL values.
    - The type is automatically determined at assignment.

System Variable Definitions and Query Statements

- SHOW VARIABLES

  You can view variables with `SHOW VARIABLES LIKE 'variable_name'`.

```
SHOW VARIABLES LIKE '%time_zone%';
+------------------+----------------+----------------+---------+
| Variable_name    | Value          | Default_Value  | Changed |
+------------------+----------------+----------------+---------+
| system_time_zone | Asia/Hong_Kong | Asia/Hong_Kong | 0       |
| time_zone        | Asia/Hong_Kong | Asia/Hong_Kong | 0       |
+------------------+----------------+----------------+---------+
```

  Or view all variables with `SHOW VARIABLES`.

```
SHOW VARIABLES
+-----------------------------------------------------------------+--------------------------------------+---------------
-----------------------+---------+
| Variable_name                                                   | Value                                | Default_Value
| Changed |
+-----------------------------------------------------------------+--------------------------------------+---------------
-----------------------+---------+
| DML_PLAN_RETRY_TIMES                                            | 3                                    | 3
```

```
| 0       |
| adaptive_pipeline_task_serial_read_on_limit                     | 10000                                | 10000
| 0       |
| allow_modify_materialized_view                                  | false                                | false
| 0       |
| allow_partition_column_nullable                                 | true                                 | true
| 0       |
| analyze_timeout                                                 | 43200                                | 43200
| 0       |
| version                                                         | 5.7.99                               | 5.7.99
| 0       |
| version_comment                                                 | Doris version doris-0.0.0--de61c58223 | Doris version
doris-0.0.0--de61c58223 | 0       |
| wait_full_block_schedule_times                                  | 2                                    | 2
| 0       |
| wait_timeout                                                    | 28800                                | 28800
| 0       |
| workload_group                                                  |                                      |
| 0       |
+-----------------------------------------------------------------+--------------------------------------+---------------
------------------------+---------+
360 rows in set (0.01 sec)
```

- SET

  Some variables can be set to take effect globally or only in the current session.

  Set to take effect only in the current session with `SET`. For example:

```
SET exec_mem_limit = 137438953472;
SHOW VARIABLES LIKE '%exec_mem_limit%';
+----------------+--------------+---------------+---------+
| Variable_name  | Value        | Default_Value | Changed |
+----------------+--------------+---------------+---------+
| exec_mem_limit | 137438953472 | 2147483648    | 1       |
+----------------+--------------+---------------+---------+
1 row in set (0.01 sec)
SET forward_to_master = true;
SHOW VARIABLES LIKE '%forward_to_master%';
+-------------------+-------+---------------+---------+
| Variable_name     | Value | Default_Value | Changed |
+-------------------+-------+---------------+---------+
| forward_to_master | true  | true          | 0       |
+-------------------+-------+---------------+---------+
1 row in set (0.00 sec)
SET time_zone = "Asia/Shanghai";
SHOW VARIABLES LIKE '%time_zone%';
+------------------+----------------+----------------+---------+
| Variable_name    | Value          | Default_Value  | Changed |
+------------------+----------------+----------------+---------+
| time_zone        | Asia/Shanghai  | Asia/Hong_Kong | 1       |
| system_time_zone | Asia/Hong_Kong | Asia/Hong_Kong | 0       |
+------------------+----------------+----------------+---------+
2 rows in set (0.00 sec)
```

  Set globally

```
SET GLOBAL exec_mem_limit = 137438953472;
SHOW VARIABLES LIKE '%exec_mem_limit%';
+----------------+--------------+---------------+---------+
| Variable_name  | Value        | Default_Value | Changed |
```

```
+----------------+-------------+---------------+---------+
| exec_mem_limit | 137438953472 | 2147483648    | 1       |
+----------------+-------------+---------------+---------+
1 row in set (0.01 sec)
```

- UNSET

  Syntax:

```
UNSET (GLOBAL | SESSION | LOCAL)? VARIABLE (ALL | identifier)
unset global variable exec_mem_limit;
SHOW VARIABLES LIKE '%exec_mem_limit%';
+----------------+------------+---------------+---------+
| Variable_name  | Value      | Default_Value | Changed |
+----------------+------------+---------------+---------+
| exec_mem_limit | 2147483648 | 2147483648    | 0       |
+----------------+------------+---------------+---------+
1 row in set (0.00 sec)
```

User Variable Definition and Query Statements

User-defined variables can be defined with the following statement:

```
SET @var_name = constant_value|constant_expr;
```

Setting examples:

```
SET @v1 = "A";
SET @v2 = 32+33;
SET @v3 = str_to_date("2024-12-29 10:11:12", '%Y-%m-%d %H:%i:%s');
```

Can be used in queries

```
SELECT @v1, @v2, @v3;
+------+------+---------------------+
| @v1  | @v2  | @v3                 |
+------+------+---------------------+
| A    |   65 | 2024-12-29 10:11:12 |
+------+------+---------------------+
1 row in set (0.01 sec)
```

Comments

Description

Comments can make your application easier to read and maintain. For example, you can include a comment within a statement to describe the purpose of that statement in your application. Comments in SQL statements (except for HINT) do not affect the execution of the statement. For information on using this specific form of comments with HINT, please refer to the HINT section.

In SQL statements, comments can appear between any keywords, parameters, or punctuation marks. You can include comments in statements in two ways:

- Multi-line comments: Start the comment with a slash and asterisk (/*), followed by the comment text. This text can span multiple lines. End the comment with an asterisk and a slash (*/). There is no need for spaces or line breaks between the start and end characters and the text.
- Single-line comments: Start the comment with two dashes (--), followed by the comment text. This text cannot extend to a new line. The comment ends with a newline character.

Examples

Multi-line Comment

```
SELECT /* This is a multi-line comment
          that can span multiple lines */
       column_name
FROM   table_name;
```

Single-line Comment

```
SELECT column_name -- This is a single-line comment
FROM   table_name;
```

Arithmetic Operators

Description

You can use arithmetic operators with one or two parameters to perform negation, addition, subtraction, multiplication, and division on numbers. Some of these operators are also used for arithmetic operations on date and time intervals. The parameters of the operators must be parsed as numeric data types, or any data type that can be implicitly converted to numeric data types.

Unary arithmetic operators return the same data type as the parameter numeric data type. For binary arithmetic operators, Doris will convert the parameters to the appropriate type for computation according to implicit type conversion rules, and return the appropriate type as the result. For specific conversion rules, please refer to the “Type Conversion” section.

Operators

Operators	Purpose	Example
• -	Unary operator. It represents taking the positive or negative value of an expression. Equivalent to 0 + a and 0 - a	SELECT +(5 + 3), -(5 + 3); +---------+---------------+ | (5 + 3) | (0 - (5 + 3)) | +---------+---------------+ | 8 | -8 | +---------+---------------+
• -	Binary operator. It represents adding or subtracting two expressions.	SELECT 5 + 3, 5 - 3; +---------+---------+ | (5 + 3) | (5 - 3) | +---------+---------+ | 8 | 2 | +---------+---------+

Operators	Purpose	Example

*	Binary operator. It represents multiplying or dividing two expressions. Special case: when the second argument of division is 0, it returns NULL	`SELECT 5 * 3, 5 / 3;`
		
		`+---------+----------------------`
		`----------------+`
		
		`| (5 * 3) | (cast(5 as DOUBLE) /`
		`cast(3 as DOUBLE)) |`
		
		`+---------+----------------------`
		`----------------+`
		
		`| 15 | 1.6666666666666667`
		`|`
		
		`+---------+----------------------`
		`----------------+`

DIV	Binary operator. It represents integer division of two expressions. Special case: when the second argument is 0, it returns NULL	`SELECT 5 DIV 3, 9 DIV 0;`
		
		`+-----------+-----------+`
		
		`| (5 DIV 3) | (9 DIV 0) |`
		
		`+-----------+-----------+`
		
		`| 1 | NULL |`
		
		`+-----------+-----------+`

%	Binary operator. It represents the remainder of two expressions. Special case: when the second argument is 0, it returns NULL	`SELECT 5 % 3, 9 % 0;`
		
		`+---------+---------+`
		
		`| (5 % 3) | (9 % 0) |`
		
		`+---------+---------+`
		
		`| 2 | NULL |`
		
		`+---------+---------+`

Usage Notes

In arithmetic expressions: 1. Do not use consecutive minus signs `--` to indicate double negation (this is the SQL comment symbol) 2. Use spaces or parentheses to separate as needed: `-(-5)` or `- -5` 3. For more comment standards, please refer to the “Comments” section

Logic Operators

Description

Logical conditions combine the results of two components' conditions to generate a single result based on them, or to invert the result of a condition.

Operator Introduction

Operator	Function	Example
NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if TRUE. If it is UNKNOWN, it remains UNKNOWN.	`SELECT NOT (TRUE)`
AND	Returns TRUE if both components' conditions are TRUE. Returns FALSE if either is FALSE. Otherwise, returns UNKNOWN.	`SELECT TRUE AND FALSE`
OR	Returns TRUE if either component's condition is TRUE. Returns FALSE if both are FALSE. Otherwise, returns UNKNOWN.	`SELECT TRUE OR NULL`

Truth Tables

NOT Truth Table

	TRUE	FALSE
NOT	FALSE	TRUE

AND Truth Table

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR Truth Table

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Comparison Operators

Description

Comparison conditions compare one expression with another. The result of the comparison can be TRUE, FALSE, or UNKNOWN.

Operator Introduction

Operator	Function	Example
=	Equality comparison. If either side of the comparison is UNKNOWN, the result is UNKNOWN.	`SELECT 1 = 1`
<=>	NULL-safe equality comparison. Unlike equality comparison, NULL-safe equality treats NULL as a comparable value. Returns TRUE when both sides are NULL. Returns FALSE when only one side is NULL. This operator never returns UNKNOWN.	`SELECT NULL <=> NULL`
!= <>	Inequality comparison	`SELECT 1 != 1`
< >	Greater than and less than comparison	`SELECT 1 > 1`
<= >=	Greater than or equal to and less than or equal to comparison	`SELECT 1 >= 1`
`<x> BETWEEN <y> AND <z>`	Equivalent to `<x> >= <y> and <x> <= <z>`. Greater than or equal to `<y>` and less than or equal to `<z>`	`SELECT 1 BETWEEN 0 AND 2`

Boolean Testing Operators

Description

This operator is used exclusively to check for TRUE, FALSE, or NULL. For an introduction to NULL, please refer to the “Nulls” section.

Operator Introduction

Operator	Function	Example
`x IS [NOT] TRUE`	Checks if x is TRUE. Returns TRUE if x is TRUE, otherwise returns FALSE.	`SELECT 1 IS NOT TRUE`
`x IS [NOT] FALSE`	Checks if x is FALSE. Returns TRUE if x is FALSE, otherwise returns FALSE.	`SELECT 1 IS NOT FALSE`
`x IS [NOT] NULL`	Checks if x is NULL. Returns TRUE if x is NULL, otherwise returns FALSE.	`SELECT 1 IS NOT NULL`

Pattern Matching Operators

Description

Pattern matching operators are used to compare character-type data.

Operator Introduction

Operator	Function	Example
<char1> [NOT] LIKE <char2>	If <char1> matches the pattern <char2>, it is TRUE (or does not match when NOT is specified). In <char2>, the character % matches any zero or multiple characters (except for an empty string). The character _ matches any single character. If there is an escape character before the wildcard character, it is treated as a literal character.	SELECT 'ABCD' LIKE '%C_'
<char1> [NOT] {REGEXP | RLIKE} <char2>	If <char1> matches the pattern <char2>, it is TRUE (or does not match when NOT is specified). For the specific rules of regular expressions, please refer to the subsequent REGEXP section.	SELECT 'ABCD' REGEXP 'A.*D'

LIKE

The LIKE condition specifies a test involving pattern matching. The equality comparison operator (=) precisely matches one character value to another character value, while the LIKE condition matches a part of one character value with another character value by searching for the pattern specified in the second value within the first value.

The syntax is as follows:

```
<char1> [ NOT ] LIKE <char2>
```

Where:

- char1 is a character expression (such as a character column), known as the search value.
- char2 is a character expression, usually a string literal, known as the pattern.

Both character expressions (char1, char2) can be any of CHAR, VARCHAR, or STRING data types. If they are different, Doris will convert them all to VARCHAR or STRING.

Patterns can include special pattern matching characters:

- The underscore (_) in the pattern matches exactly one character in the value.
- The percent sign (%) in the pattern can match zero or multiple characters in the value. The pattern % cannot match NULL.

REGEXP (RLIKE)

REGEXP is similar to the LIKE condition, differing in that REGEXP performs regular expression matching, rather than the simple pattern matching performed by LIKE. This condition uses a set of input characters defined by a character to evaluate strings.

The syntax is as follows:

```
<char1> [ NOT ] { REGEXP | RLIKE } <char2>
```

Where:

- `char1` is a character expression (such as a character column), known as the search value.
- `char2` is a character expression, usually a string literal, known as the pattern.

Both character expressions (`char1`, `char2`) can be any of CHAR, VARCHAR, or STRING data types. If they are different, Doris will convert them all to VARCHAR or STRING.

Full Text Search Operators

Description

Full-text search operators determine whether a column meets specified full-text search conditions (keywords, phrases, etc.), with results being TRUE, FALSE, or UNKNOWN.

Limitations:

1. The left operand must be a column name, and the right operand must be a string literal.
2. It can only be used in the WHERE clause and cannot be used in other clauses such as SELECT, GROUP BY, ORDER BY, etc. It can be combined with other conditions in the WHERE clause using AND, OR, NOT logical operations.

Operator Introduction

Operator	Function	Example
MATCH	Equivalent to MATCH_ANY	`SELECT * FROM t WHERE column1 MATCH 'word1 word2'`
`<column_name>` MATCH_ANY `<string_literal>`	Tokenizes column_name and string_literal according to column_name's index tokenizer. If any token from column_name contains any word after tokenization of string_literal, the result is TRUE, otherwise FALSE.	`SELECT * FROM t WHERE column1 MATCH_ANY 'word1 word2'`
`<column_name>` MATCH_ALL `<string_literal>`	Tokenizes column_name and string_literal according to column_name's index tokenizer. If all tokens from column_name contain every word after tokenization of string_literal, the result is TRUE, otherwise FALSE.	`SELECT * FROM t WHERE column1 MATCH_ALL 'word1 word2'`
`<column_name>` MATCH_PHRASE `<string_literal>`	Tokenizes column_name and string_literal according to column_name's index tokenizer. If all tokens from column_name contain every word after tokenization of string_literal and the order of words is consistent (i.e., a phrase), the result is TRUE, otherwise FALSE.	`SELECT * FROM t WHERE column1 MATCH_PHRASE 'word1 word2'`
`<column_name>` MATCH_PHRASE_PREFIX `<string_literal>`	An extension of MATCH_PHRASE, allowing the last word after tokenization of string_literal to match only a prefix rather than the whole word. Similar to the suggest feature of web search engines.	`SELECT * FROM t WHERE column1 MATCH_PHRASE_PREFIX 'word1 wor'`
`<column_name>` MATCH_PHRASE_EDGE `<string_literal>`	An extension of MATCH_PHRASE_PREFIX, allowing the first word after tokenization of string_literal to match a suffix and the last word to match a prefix.	`SELECT * FROM t WHERE column1 MATCH_PHRASE_EDGE 'rd wor'`
`<column_name>` MATCH_REGEXP `<string_literal>`	An extension of MATCH_ANY, where string_literal specifies a regular expression, and tokens from column_name match the regular expression.	`SELECT * FROM t WHERE column1 MATCH_REGEXP 'word.'`

In Operators

Description

The IN operator tests whether a value is a member of a list of values or a subquery.

Operator Introduction

Operator	Function	Example
IN	Tests if it is equal to any member. Returns TRUE if so.	`SELECT 1 IN (SELECT 2)`
NOT IN	Tests if it is not equal to all members. Returns TRUE if so.	`SELECT 1 NOT IN (3, 4, 5)`

Exists Operators

Description

The EXISTS condition is used to test for the existence of rows in a subquery.

Operator Introduction

Operator	Function	Example
EXISTS	Returns TRUE if the subquery returns at least one row of data	`SELECT department_id FROM departments d WHERE EXISTS (SELECT * FROM employees e WHERE d.department_id = e.department_id) ORDER BY department_id;`

Bitwise Operators

Description

Bitwise operators perform specified operations on one or two expressions at the bit level. These operators only accept arguments of the `BIGINT` type. Therefore, any expressions processed by bitwise operators will be converted to the `BIGINT` type.

Operator Overview

Operator	Function	Example
`&`	Performs a bitwise AND operation. If both corresponding bits are `1`, the result bit is `1`; otherwise, it is `0`.	`SELECT 1 & 2`
`|`	Performs a bitwise OR operation. If either corresponding bit is `1`, the result bit is `1`; otherwise, it is `0`.	`SELECT 1	2`
`^`	Performs a bitwise XOR operation. If the corresponding bits differ, the result bit is `1`; otherwise, it is `0`.	`SELECT 1 ^ 2`
`~`	Performs a bitwise NOT operation. Inverts each bit: `1` becomes `0`, and `0` becomes `1`.	`SELECT ~1`

Assignment Operators

Description

The function of the assignment operator is to assign the expression on the right-hand side of the operator to the expression on the left-hand side. In Doris, the assignment operator can only be used in the SET part of the UPDATE statement and in the SET statement. For details, please refer to the UPDATE statement and the SET statement.

Operators

Operator	Purpose	Example
=	Assign the result of to .	`SET enable_profile = true`

Operator Precedence

Description

Operator precedence determines the order in which operators are evaluated in an expression. When an expression contains multiple operators, Doris will perform calculations in descending order of operator precedence.

Operator Precedence

The precedence decreases from top to bottom, with the highest precedence at the top.

Precedence	Operator
1	!
2	+ (unary plus), - (unary minus), ~ (unary bitwise NOT), ^
3	*, /, %, DIV
4	-, +
5	&
6	|
7	=(comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, MATCH, IN
8	NOT
9	AND, &&
10	XOR
11	OR
12	||

ABS

Description

Returns the absolute value of x

Syntax

```
ABS(<x>)
```

Parameters

Parameter	Description
<x>	The value for which the absolute value is to be calculated

Return Value

The absolute value of parameter x.

Example

```
select abs(-2);
```

```
+---------+
| abs(-2) |
+---------+
|       2 |
+---------+
```

```
select abs(3.254655654);
```

```
+------------------+
| abs(3.254655654) |
+------------------+
|      3.254655654 |
+------------------+
```

```
select abs(-3254654236547654354654767);
```

```
+---------------------------------+
| abs(-3254654236547654354654767) |
+---------------------------------+
| 3254654236547654354654767       |
+---------------------------------+
```

ASIN

Description

Returns the arc sine of `x`, or `nan` if `x` is not in the range `-1` to `1`.

Syntax

```
ASIN(<x>)
```

Parameters

Parameter	Description
<x>	The value for which the asin value is to be calculated

Return Value

The asin value of parameter `x`.

Example

```
select asin(0.5);
```

```
+---------------------+
| asin(0.5)           |
+---------------------+
| 0.52359877559829893 |
+---------------------+
```

```
select asin(2);
```

```
+-----------+
| asin(2.0) |
+-----------+
|       nan |
+-----------+
```

ACOS

Description

Returns the arc cosine of `x`, or `NULL` if `x` is not in the range `-1` to `1`.

Syntax

```
ACOS(<x>)
```

Parameters

Parameter	Description
<x>	The value for which the acos value is to be calculated

Return Value

The acos value of parameter `x`.

Example

```
select acos(1);
```

```
+-----------+
| acos(1.0) |
+-----------+
|         0 |
+-----------+
```

```
select acos(0);
```

```
+--------------------+
| acos(0.0)          |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

```
select acos(-2);
```

```
+------------+
| acos(-2.0) |
+------------+
|        nan |
+------------+
```

ATAN

Description

Returns the arctangent of x, where x is in radians.

Syntax

```
ATAN(<x>)
```

Parameters

Parameter	Description
<x>	The value for which the atan value is to be calculated

Return Value

The atan value of parameter x.

Example

```
select atan(0);
```

```
+-----------+
| atan(0.0) |
+-----------+
|         0 |
+-----------+
```

```
select atan(2);
```

```
+--------------------+
| atan(2.0)          |
+--------------------+
| 1.1071487177940904 |
+--------------------+
```

ATAN2

Description

Returns the arc tangent of ‘y’ / ‘x’.

Syntax

```
ATAN2(<y>, <x>)
```

Parameters

Parameter	Description
<x>	The value representing the horizontal distance (coordinate) from the origin (0,0) along the x-axis.
<y>	The value representing the vertical distance (coordinate) from the origin (0,0) along the y-axis.

Return Value

The atan2 value of parameter `y` / `x`.

Example

```
select atan2(0.1, 0.2);
```

```
+--------------------+
| atan2(0.1, 0.2)    |
+--------------------+
| 0.46364760900080609 |
+--------------------+
```

```
select atan2(1.0, 1.0);
```

```
+--------------------+
| atan2(1.0, 1.0)    |
+--------------------+
| 0.78539816339744828 |
+--------------------+
```

BIN

Description

Converts decimal numbers to binary text.

Syntax

```
BIN(<a>)
```

Parameters

Parameter	Description
<a>	Decimal value to be converted

Return Value

The binary representation of the parameter <a>. When <a> is negative, the result is its 64-bit complement.

Examples

```
select bin(0);
```

```
+--------+
| bin(0) |
+--------+
| 0      |
+--------+
```

```
select bin(-1);
```

```
+------------------------------------------------------------------+
| bin(-1)                                                          |
+------------------------------------------------------------------+
| 1111111111111111111111111111111111111111111111111111111111111111 |
+------------------------------------------------------------------+
```

```
select bin(123);
```

```
+----------+
| bin(123) |
+----------+
| 1111011  |
+----------+
```

SIN

Description

Calculate the sine of the parameter

Syntax

```
SIN(<a>)
```

Parameters

Parameter	Description
<a>	floating point number, the radian value of the parameter to calculate

Return Value

The sine of the parameter <a>, expressed in radians.

Examples

```
select sin(1);
```

```
+------------------------+
| sin(cast(1 as DOUBLE)) |
+------------------------+
|     0.8414709848078965 |
+------------------------+
```

```
select sin(0);
```

```
+------------------------+
| sin(cast(0 as DOUBLE)) |
+------------------------+
|                    0.0 |
+------------------------+
```

```
select sin(Pi());
```

```
+-----------------------------------+
| sin(pi())                         |
+-----------------------------------+
```

```
| 0.00000000000000012246467991473532 |
+------------------------------------+
```

CBRT

Description

Calculate the cube root of the parameter

Syntax

```
CBRT(<a>)
```

Parameters

Parameter	Description
<a>	Floating point parameter

Return Value

Cubic root of parameter <a>, a floating point number.

Examples

```
select cbrt(0);
```

```
+-------------------------+
| cbrt(cast(0 as DOUBLE)) |
+-------------------------+
|                     0.0 |
+-------------------------+
```

```
select cbrt(-111);
```

```
+----------------------------+
| cbrt(cast(-111 as DOUBLE)) |
+----------------------------+
|         -4.805895533705333 |
+----------------------------+
```

```
select cbrt(1234);
```

```
+----------------------------+
| cbrt(cast(1234 as DOUBLE)) |
+----------------------------+
|         10.726014668827325 |
+----------------------------+
```

COS

Description

Calculate the cosine of the parameter

Syntax

```
COS(<a>)
```

Parameters

Parameter	Description
<a>	floating point number, the radian value of the parameter to calculate

Return Value

The cosine of the parameter <a>, expressed in radians.

Examples

```
select cos(1);
```

```
+--------------------+
| cos(1.0)           |
+--------------------+
| 0.54030230586813977 |
+--------------------+
```

```
select cos(0);
```

```
+------------------------+
| cos(cast(0 as DOUBLE)) |
+------------------------+
|                    1.0 |
+------------------------+
```

```
select cos(Pi());
```

```
+-----------+
| cos(pi()) |
+-----------+
```

```
|        -1 |
+-----------+
```

COSH

Description

Returns the hyperbolic cosine of x.

Syntax

```
COSH(<x>)
```

Parameters

Parameter	Description
<x>	The value for which the hyperbolic cosine is to be calculated

Return Value

The hyperbolic cosine of parameter x

Examples

```
select cosh(0);
```

```
+-------------------------+
| cosh(cast(0 as DOUBLE)) |
+-------------------------+
|                     1.0 |
+-------------------------+
```

```
select cosh(1);
```

```
+-------------------------+
| cosh(cast(1 as DOUBLE)) |
+-------------------------+
|       1.543080634815244 |
+-------------------------+
```

```
select cosh(-1);
```

```
+-------------------------+
| cosh(cast(-1 as DOUBLE)) |
+-------------------------+
|        1.543080634815244 |
+-------------------------+
```

CONV

Description

Do radix conversion for input parameter.

Syntax

```
CONV(<input>, <from_base>, <to_base>)
```

Parameters

Parameter	Description
`<input>`	Parameters to be converted, either as strings or integers
`<from_base>`	Numeric, the source base, within `[2,36]`.
`<to_base>`	Numeric, the target base, within `[2,36]`.

Return Value

The number under the converted target binary `<to_base>` is returned as a string.

Examples

```
SELECT CONV(15,10,2);
```

```
+-----------------+
| conv(15, 10, 2) |
+-----------------+
| 1111            |
+-----------------+
```

```
SELECT CONV('ff',16,10);
```

```
+--------------------+
| conv('ff', 16, 10) |
+--------------------+
| 255                |
+--------------------+
```

```
SELECT CONV(230,10,16);
```

```
+-------------------+
| conv(230, 10, 16) |
```

```
+-------------------+
| E6                |
+-------------------+
```

DEGREES

Description

Input a double-precision floating-point number and convert it from radians to degrees.

- When the parameter is NULL, return NULL.

Syntax

```
DEGREES(<a>)
```

Parameters

parameter	explain
<a>	The value that needs to be converted from radians to degrees.

Return Value

The angle of parameter a.

- When the parameter is NULL, return NULL.

Examples

```
select degrees(3.14),degrees(1),degrees(-1),degrees(NULL)
```

```
+------------------------------+---------------------------+----------------------------+--------------+
| degrees(cast(3.14 as DOUBLE)) | degrees(cast(1 as DOUBLE)) | degrees(cast(-1 as DOUBLE)) | degrees(NULL) |
+------------------------------+---------------------------+----------------------------+--------------+
|             179.9087476710785 |          57.29577951308232 |          -57.29577951308232 |          NULL |
+------------------------------+---------------------------+----------------------------+--------------+
```

E

Description

Returns the constant `e` value.

Syntax

```
E()
```

Example

```
mysql> select e();
+--------------------+
| e()                |
+--------------------+
| 2.718281828459045  |
+--------------------+
```

PI

Description

Returns the constant `Pi` value.

Syntax

```
PI()
```

Return value

Returns a constant Pi of type floating-point number.

Example

```
select Pi();
```

```
+-------------------+
| pi()              |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

EXP

Description

Returns x raised to the base e.

Alias

- DEXP

Syntax

```
EXP(<x>)
```

Parameters

Parameter	Description
<x>	independent variable

Return Value

Return a value of type double - If the parameter has a null value, it returns NULL

Example

```
select exp(2);
```

```
+------------------+
| exp(2.0)         |
+------------------+
| 7.38905609893065 |
+------------------+
```

```
select exp(3.4);
```

```
+--------------------+
| exp(3.4)           |
+--------------------+
| 29.964100047397011 |
+--------------------+
```

LOG

Description

Returns the logarithm of `x` based on base `b`.

Syntax

```
LOG(<b>,<x>)
```

Parameters

Parameter	Description
<b>	Base should be greater than 0 and not be exactly 1.0
<x>	Antilogarithm should be greater than 0

Return value

Return b float point number. Special cases:

- If a IS NULL or x IS NULL, return NULL

Example

```
select log(5,1);
```

```
+---------------+
| log(5.0, 1.0) |
+---------------+
|             0 |
+---------------+
```

```
select log(3,20);
```

```
+--------------------+
| log(3.0, 20.0)     |
+--------------------+
| 2.7268330278608417 |
+--------------------+
```

```
select log(2,65536);
```

```
+-------------------+
| log(2.0, 65536.0) |
+-------------------+
|                16 |
+-------------------+
```

```
select log(5,NULL);
```

```
+-----------------------------+
| log(cast(5 as DOUBLE), NULL) |
+-----------------------------+
|                        NULL |
+-----------------------------+
```

```
select log(NULL,3);
```

```
+-----------------------------+
| log(NULL, cast(3 as DOUBLE)) |
+-----------------------------+
|                        NULL |
+-----------------------------+
```

LOG2

Description

Returns the logarithm of `x` to base `2`.

Syntax

```
LOG2(<x>)
```

Parameters

Parameter	Description
<x>	Antilogarithm should be greater than 0

Return value

Returns a floating-point number.

- If x IS NULL: return `NULL`

Example

```
select log2(1);
```

```
+-------------------------+
| log2(cast(1 as DOUBLE)) |
+-------------------------+
|                     0.0 |
+-------------------------+
```

```
select log2(2);
```

```
+-------------------------+
| log2(cast(2 as DOUBLE)) |
+-------------------------+
|                     1.0 |
+-------------------------+
```

```
select log2(10);
```

```
+-------------------------+
| log2(cast(10 as DOUBLE)) |
+-------------------------+
```

```
|       3.3219280948873626 |
+--------------------------+
```

LOG10

Description

Returns the logarithm of `x` to base `10`.

Alias

- DLOG10

Syntax

```
LOG10(<x>)
```

Parameters

Parameter	Description
<x>	Antilogarithm should be greater than 0

Return value

Returns a floating-point number.

- If x IS NULL: return `NULL`

Example

```
select log10(1);
```

```
+--------------------------+
| log10(cast(1 as DOUBLE)) |
+--------------------------+
|                      0.0 |
+--------------------------+
```

```
select log10(10);
```

```
+---------------------------+
| log10(cast(10 as DOUBLE)) |
+---------------------------+
|                       1.0 |
+---------------------------+
```

```
select log10(16);
```

```
+--------------------------+
| log10(cast(16 as DOUBLE)) |
+--------------------------+
|        1.2041199826559248 |
+--------------------------+
```

LN

Description

Returns the natural logarithm of x to base e.

Alias

- DLOG1

Syntax

```
LN(<x>)
```

Parameters

Parameter	Description
<x>	Antilogarithm should be greater than 0

Return value

Return a float-point number. Special cases:

- If x IS NULL, return NULL

Example

```
select ln(1);
```

```
+----------------------+
| ln(cast(1 as DOUBLE)) |
+----------------------+
|                  0.0 |
+----------------------+
```

```
select ln(e());
```

```
+---------+
| ln(e()) |
+---------+
|     1.0 |
+---------+
```

```
select ln(10);
```

```
+------------------------+
| ln(cast(10 as DOUBLE)) |
+------------------------+
|      2.302585092994046 |
+------------------------+
```

CEIL

Description

Round up floating-point and fixed-point decimals to a specific number of places and return the rounded floating-point or fixed-point number.

Syntax

```
CEIL(<a>[, <d>])
```

Parameters

Parameter	Description
<a>	Floating-point (Double) or fixed-point (Decimal) parameter indicating the parameter to be rounded
<d>	Optional, integer, indicates rounding to the target number of digits, a positive number means rounding to the next decimal point, a negative number means rounding to the next decimal point, and 0 indicates rounding to an integer. When not filled, it is equivalent to <d> = 0.

Return Value

Returns the smallest rounded number greater than or equal to <a> according to the following rules.

Round to 1/(10^d) digit, i.e., make the result divisible by 1/(10^d). If 1/(10^d) is not exact, the rounding digit is the nearest number of the corresponding data type.

For an entry <a> of type Decimal, assuming it is of type Decimal(p, s), the return value is:

- Decimal(p, 0), if <d> <= 0
- Decimal(p, <d>), if 0 < <d> <= s
- Decimal(p, s), if <d> > s

Alias

- DCEIL
- CEILING

Examples

```
select ceil(123.456);
```

```
+---------------+
| ceil(123.456) |
+---------------+
|           124 |
+---------------+
```

```
select ceil(123.456, 2);
```

```
+------------------+
| ceil(123.456, 2) |
+------------------+
|           123.46 |
+------------------+
```

```
select ceil(123.456, -2);
```

```
+-------------------+
| ceil(123.456, -2) |
+-------------------+
|               200 |
+-------------------+
```

```
select ceil(123.45, 1), ceil(123.45), ceil(123.45, 0), ceil(123.45, -1);
```

```
+-----------------+--------------+-----------------+------------------+
| ceil(123.45, 1) | ceil(123.45) | ceil(123.45, 0) | ceil(123.45, -1) |
+-----------------+--------------+-----------------+------------------+
|           123.5 |          124 |             124 |              130 |
+-----------------+--------------+-----------------+------------------+
```

```
select ceil(x, 2) from ( select cast(123.456 as decimal(6,3)) as x from numbers("number"="5") )t;
```

```
+------------+
| ceil(x, 2) |
+------------+
|     123.46 |
|     123.46 |
|     123.46 |
|     123.46 |
|     123.46 |
+------------+
```

FLOOR

Description

Round down floating-point and fixed-point decimals to a specific number of digits and return the rounded floating-point or fixed-point number.

Syntax

```
FLOOR(<a>[, <d>])
```

Parameters

Parameter	Description
<a>	Floating-point (Double) or fixed-point (Decimal) parameter indicating the parameter to be rounded
<d>	Optional, integer, indicates rounding to the target number of digits, a positive number means rounding to the next decimal point, a negative number means rounding to the next decimal point, and 0 indicates rounding to an integer. When not filled, it is equivalent to <d> = 0.

Return Value

Returns the largest rounded number less than or equal to <a> according to the following rules.

Round to 1/(10^d) digit, i.e., make the result divisible by 1/(10^d). If 1/(10^d) is not exact, the rounding digit is the nearest number of the corresponding data type.

For an entry <a> of type Decimal, assuming it is of type Decimal(p, s), the return value is:

- Decimal(p, 0), if <d> <= 0
- Decimal(p, <d>), if 0 < <d> <= s
- Decimal(p, s), if <d> > s

Alias

- DFLOOR

Examples

```
select floor(123.456);
```

```
+----------------+
| floor(123.456) |
+----------------+
|            123 |
+----------------+
```

```
select floor(123.456, 2);
```

```
+-------------------+
| floor(123.456, 2) |
+-------------------+
|            123.45 |
+-------------------+
```

```
select floor(123.456, -2);
```

```
+--------------------+
| floor(123.456, -2) |
+--------------------+
|                100 |
+--------------------+
```

```
select floor(123.45, 1), floor(123.45), floor(123.45, 0), floor(123.45, -1);
```

```
+------------------+---------------+------------------+-------------------+
| floor(123.45, 1) | floor(123.45) | floor(123.45, 0) | floor(123.45, -1) |
+------------------+---------------+------------------+-------------------+
|            123.4 |           123 |              123 |               120 |
+------------------+---------------+------------------+-------------------+
```

```
select floor(x, 2) from ( select cast(123.456 as decimal(6,3)) as x from numbers("number"="5") )t;
```

```
+-------------+
| floor(x, 2) |
+-------------+
|      123.45 |
|      123.45 |
|      123.45 |
|      123.45 |
|      123.45 |
+-------------+
```

FMOD

Description

Find the remainder of a / b for the floating-point type. For the integer type, please use the mod function.

Syntax

```
MOD(<col_a> , <col_b>)
```

Parameters

Parameter	Description
<col_a>	Dividend
<col_b>	Divisor should not be 0

Return value

Return a float-point type. Special cases:

If col_a IS NULL or col_b IS NULL, return NULL.

Example

```
select fmod(10.1, 3.2);
```

```
+--------------------+
| fmod(10.1, 3.2)    |
+--------------------+
| 0.4999999999999991 |
+--------------------+
```

```
select fmod(10.1, 0);
```

```
+---------------+
| fmod(10.1, 0) |
+---------------+
|          NULL |
+---------------+
```

MONEY_FORMAT

Description

The number is output in currency format, the integer part is separated by commas every three bits, and the decimal part is reserved for two bits.

Syntax

```
MONEY_FORMAT(<number>)
```

Parameters

Parameter	Description
<number>	The numbers to be formatted

Return value

Returns a string in currency format. Special cases:

- If the parameter is NULL, return NULL

Example

```
select money_format(17014116);
```

```
+------------------------+
| money_format(17014116) |
+------------------------+
| 17,014,116.00          |
+------------------------+
```

```
select money_format(1123.456);
```

```
+------------------------+
| money_format(1123.456) |
+------------------------+
| 1,123.46               |
+------------------------+
```

```
select money_format(1123.4);
```

```
+----------------------+
| money_format(1123.4) |
```

```
+----------------------+
| 1,123.40             |
+----------------------+
```

TAN

Description

Returns the tangent of x, where x is the value in radians.

Syntax

```
TAN(<x>)
```

Parameters

Parameter	Description
<x>	The value for which the tangent is to be calculated

Return Value

Returns the tangent of x.

Example

```
select tan(0),tan(1),tan(-1);
```

```
+-----------------------+-----------------------+------------------------+
| tan(cast(0 as DOUBLE)) | tan(cast(1 as DOUBLE)) | tan(cast(-1 as DOUBLE)) |
+-----------------------+-----------------------+------------------------+
|                      0 |     1.5574077246549023 |     -1.5574077246549023 |
+-----------------------+-----------------------+------------------------+
```

TANH

Description

Returns the hyperbolic tangent of x.

Syntax

```
TANH(<x>)
```

Parameters

Parameter	Description
<x>	The value for which the hyperbolic tangent is to be calculated

Return Value

The hyperbolic tangent of parameter x.

Example

```
select tanh(0),tanh(1);
```

```
+-------------------------+-------------------------+
| tanh(cast(0 as DOUBLE)) | tanh(cast(1 as DOUBLE)) |
+-------------------------+-------------------------+
|                       0 |      0.7615941559557649 |
+-------------------------+-------------------------+
```

PMOD

Description

Returns the smallest positive solution of the modulo operation x mod y within the modular system, which is obtained by calculating (x % y + y) % y.

Syntax

```
PMOD(<x> , <y>)
```

Parameters

Parameter	Description
<x>	Dividend
<y>	Divisor should not be 0

Return value

Returns an integer or a floating-point number. Special cases:

- If x = 0, returns 0.
- If x is NULL or y is NULL, returns NULL.

Example

```
SELECT PMOD(13,5);
```

```
+-------------+
| pmod(13, 5) |
+-------------+
|           3 |
+-------------+
```

```
SELECT PMOD(-13,5);
```

```
+--------------+
| pmod(-13, 5) |
+--------------+
|            2 |
+--------------+
```

```
SELECT PMOD(0,-12);
```

```
+--------------+
| pmod(0, -12) |
+--------------+
|            0 |
+--------------+
```

```
SELECT PMOD(0,null);
```

```
+-------------------------------+
| pmod(cast(0 as DOUBLE), NULL) |
+-------------------------------+
|                          NULL |
+-------------------------------+
```

ROUND

Description

Rounds the number `x` to the specified number of digits. - If `d` is not specified, `x` is rounded to the nearest integer. - If `d` is negative, the result is rounded to the left of the decimal point by `d` places. - If `x` or `d` is `null`, returns `null`. - If `d` is a column and the first argument is of type `Decimal`, the resulting `Decimal` will have the same number of decimal places as the input `Decimal`.

Alias

- `DROUND`

Syntax

```
ROUND(<x> [ , <d> ])
```

Parameters

Parameter	Description
<x>	The number to be rounded
<d>	Optional, the number of decimal places to round to

Examples

```
select round(2.4);
```

```
+------------+
| round(2.4) |
+------------+
|          2 |
+------------+
```

```
select round(2.5);
```

```
+------------+
| round(2.5) |
+------------+
|          3 |
+------------+
```

```
select round(-3.4);
```

```
+-------------+
| round(-3.4) |
+-------------+
|          -3 |
+-------------+
```

```
select round(-3.5);
```

```
+-------------+
| round(-3.5) |
+-------------+
|          -4 |
+-------------+
```

```
select round(1667.2725, 2);
```

```
+---------------------+
| round(1667.2725, 2) |
+---------------------+
|             1667.27 |
+---------------------+
```

```
select round(1667.2725, -2);
```

```
+----------------------+
| round(1667.2725, -2) |
+----------------------+
|                 1700 |
+----------------------+
```

```
CREATE TABLE test_enhanced_round (
    rid int, flo float, dou double,
    dec90 decimal(9, 0), dec91 decimal(9, 1), dec99 decimal(9, 9),
    dec100 decimal(10,0), dec109 decimal(10,9), dec1010 decimal(10,10),
    number int DEFAULT 1)
DISTRIBUTED BY HASH(rid)
PROPERTIES("replication_num" = "1" );

INSERT INTO test_enhanced_round
VALUES
(1, 12345.123, 123456789.123456789,
    123456789, 12345678.1, 0.123456789,
    123456789.1, 1.123456789, 0.123456789, 1);

SELECT number, dec90, round(dec90, number), dec91, round(dec91, number), dec99, round(dec99, number) FROM test_enhanced_round order
by rid;
```

```
+--------+-----------+---------------------+------------+---------------------+-------------+----------------------+
| number | dec90     | round(dec90, number) | dec91      | round(dec91, number) | dec99       | round(dec99, number) |
+--------+-----------+---------------------+------------+---------------------+-------------+----------------------+
|      1 | 123456789 |            123456789 | 12345678.1 |           12345678.1 | 0.123456789 |          0.100000000 |
+--------+-----------+---------------------+------------+---------------------+-------------+----------------------+
```

Usage Note

2.5 will round to 3. If you want to round to 2, use the `round_bankers` function.

ROUND_BANKERS

Description

Round x using the banker's rounding method and keep d decimal places. The default value of d is 0.

If d is negative, the |d| digits to the left of the decimal point will be set to 0.

If x or d is null, return null.

If d represents a column and the first parameter is of the Decimal type, then the resulting Decimal will have the same number of decimal places as the input Decimal.

According to the rules of the banker's rounding algorithm, when rounding to a specified decimal place:

- If the digit to be rounded is 5 and there are no other non - zero digits following it, the digit immediately preceding it will be checked:
  - If the preceding digit is even, the 5 will be simply dropped.
  - If the preceding digit is odd, the number will be rounded up.
- If the digit to be rounded is greater than 5 or there are non - zero digits following it, the traditional rounding rules will apply: round up if the digit is greater than or equal to 5, otherwise round down.

For example:

- For the value 2.5, since the digit 2 before 5 is even, the result will be rounded to 2.
- For the value 3.5, since the digit 3 before 5 is odd, the result will be rounded to 4.
- For the value 2.51, since the digit after 5 is not 0, it will be rounded up directly, and the result is 3.

Syntax

```
ROUND_BANKERS(<x> [ , <d>])
```

Parameters

Parameter	Description
<x>	The number to be rounded
<d>	Optional, the number of decimal places to round to, with a default value of 0.

Return value

Returns an integer or a float-point number:

- By default, when the parameter d = 0, it returns an integer obtained by calculating x using the banker's rounding algorithm.
- If d is negative, it returns an integer with the first digit to the left of the decimal point being 0.
- If both x and d are NULL, it returns NULL.
- If d represents a column and x is of the Decimal type, it returns a floating-point number with the same precision.

Example

```
select round_bankers(0.4);
```

```
+-------------------+
| round_bankers(0.4) |
+-------------------+
|                 0 |
+-------------------+
```

```
select round_bankers(-3.5);
```

```
+--------------------+
| round_bankers(-3.5) |
+--------------------+
|                 -4 |
+--------------------+
```

```
select round_bankers(-3.4);
```

```
+--------------------+
| round_bankers(-3.4) |
+--------------------+
|                 -3 |
+--------------------+
```

```
select round_bankers(10.755, 2);
```

```
+-------------------------+
| round_bankers(10.755, 2) |
+-------------------------+
|                   10.76 |
+-------------------------+
```

```
select round_bankers(10.745, 2);
```

```
+-------------------------+
| round_bankers(10.745, 2) |
+-------------------------+
```

```
|                   10.74 |
+-------------------------+
```

```
select round_bankers(1667.2725, -2);
```

```
+------------------------------+
| round_bankers(1667.2725, -2) |
+------------------------------+
|                         1700 |
+------------------------------+
```

```
SELECT number
, round_bankers(number * 2.5, number - 1) AS rb_decimal_column
, round_bankers(number * 2.5, 0) AS rb_decimal_literal
, round_bankers(cast(number * 2.5 AS DOUBLE), number - 1) AS rb_double_column
, round_bankers(cast(number * 2.5 AS DOUBLE), 0) AS rb_double_literal
FROM test_enhanced_round
WHERE rid = 1;
```

```
+--------+------------------+--------------------+------------------+-------------------+
| number | rb_decimal_column | rb_decimal_literal | rb_double_column | rb_double_literal |
+--------+------------------+--------------------+------------------+-------------------+
|      1 |              2.0 |                  2 |                2 |                 2 |
+--------+------------------+--------------------+------------------+-------------------+
```

TRUNCATE

Description

Perform numerical truncation on x to the number of decimal places d

Syntax

```
TRUNCATE(<x>, <d>)
```

Parameters

Parameter	Description
<x>	The value that needs to be numerically truncated
<d>	The number of decimal places to retain

Return Value

Perform numerical truncation on x to the number of decimal places d. Truncation rules:

If d is a literal:

- When d > 0: Keep d decimal places of x.
- When d = 0: Remove the decimal part of x and retain only the integer part.
- When d < 0: Remove the decimal part of x and replace the integer part with the number of digits specified by d, using the digit 0.

If d is a column, and the first argument is of type Decimal, then the resulting Decimal will have the same number of decimal places as the input Decimal

Example

d is a literal

```
select truncate(124.3867, 2),truncate(124.3867, 0),truncate(124.3867, -2);
```

```
+----------------------+----------------------+-----------------------+
| truncate(124.3867, 2) | truncate(124.3867, 0) | truncate(124.3867, -2) |
+----------------------+----------------------+-----------------------+
|               124.38 |                  124 |                   100 |
+----------------------+----------------------+-----------------------+
```

d is a column

```
select cast("123.123456" as Decimal(9,6)), number, truncate(cast ("123.123456" as Decimal(9,6)), number) from numbers("number"="5");
```

```
+--------------------------------------+--------+-----------------------------------------------------------------------+
| cast('123.123456' as DECIMALV3(9, 6)) | number | truncate(cast('123.123456' as DECIMALV3(9, 6)), cast(number as INT)) |
+--------------------------------------+--------+-----------------------------------------------------------------------+
|                            123.123456 |      0 |                                                            123.000000 |
|                            123.123456 |      1 |                                                            123.100000 |
|                            123.123456 |      2 |                                                            123.120000 |
|                            123.123456 |      3 |                                                            123.123000 |
|                            123.123456 |      4 |                                                            123.123400 |
+--------------------------------------+--------+-----------------------------------------------------------------------+
```

SQRT

Description

Returns the square root of a value, where the input value must be greater than or equal to 0.

Aliases

- DSQRT

Syntax

```
SQRT(<a>)
```

Parameters

Parameter	Description
<a>	The value whose square root is to be calculated

Return Value

The square root of parameter a.

Examples

```
select sqrt(9),sqrt(2)
```

```
+-------------------------+-------------------------+
| sqrt(cast(9 as DOUBLE)) | sqrt(cast(2 as DOUBLE)) |
+-------------------------+-------------------------+
|                     3.0 |      1.4142135623730951 |
+-------------------------+-------------------------+
```

POW

Description

Returns the value of the first argument raised to the power of the second argument.

Alias

- POWER
- FPOW
- DPOW

Syntax

```
POW(<a>, <b>)
```

Parameters

Parameter	Description
<a>	Base
<b>	Power

Return value

Return an integer type or a floating-point type.

Special cases:

- If a `IS NULL` or b `IS NULL`, return `NULL`.
- If `b = 0` and a `IS NOT NULL`, it will always return `1`.

Examples

```
select pow(2, 0);
```

```
+-----------------------------------------+
| pow(cast(2 as DOUBLE), cast(0 as DOUBLE)) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

```
select pow(2, 10);
```

```
+-----------------------------------------+
| pow(cast(2 as DOUBLE), cast(10 as DOUBLE)) |
+-----------------------------------------+
|                                  1024 |
+-----------------------------------------+
```

```
select pow(1.2, 2);
```

```
+-----------------------------------------+
| pow(cast(1.2 as DOUBLE), cast(2 as DOUBLE)) |
+-----------------------------------------+
|                                   1.44 |
+-----------------------------------------+
```

```
select pow(1.2, 2.1);
```

```
+-------------------------------------------+
| pow(cast(1.2 as DOUBLE), cast(2.1 as DOUBLE)) |
+-------------------------------------------+
|                         1.4664951016517147 |
+-------------------------------------------+
```

```
select pow(2, null);
```

```
+----------------------------+
| pow(cast(2 as DOUBLE), NULL) |
+----------------------------+
|                       NULL |
+----------------------------+
```

```
select pow(null, 2);
```

```
+----------------------------+
| pow(NULL, cast(2 as DOUBLE)) |
+----------------------------+
|                       NULL |
+----------------------------+
```

RADIANS

Description

Returns the value of x in radians, converted from degrees to radians.

Syntax

```
RADIANS(<x>)
```

Parameters

Parameter	Description
<x>	The angle in degrees to be converted.

Return value

Returns an integer or a floating-point number. Special case:

- If the parameter x is NULL, it returns NULL.

Example

```
select radians(0);
```

```
+----------------------------+
| radians(cast(0 as DOUBLE)) |
+----------------------------+
|                        0.0 |
+----------------------------+
```

```
select radians(30);
```

```
+-----------------------------+
| radians(cast(30 as DOUBLE)) |
+-----------------------------+
|          0.5235987755982988 |
+-----------------------------+
```

```
select radians(90);
```

```
+-----------------------------+
| radians(cast(90 as DOUBLE)) |
+-----------------------------+
```

```
|          1.5707963267948966 |
+-----------------------------+
```

SIGN

Description

Returns the sign of x. Negative, zero or positive numbers correspond to -1, 0 or 1 respectively.

Syntax

```
SIGN(x)
```

Parameters

Parameter	Description
<x>	independent variable

Return value

Returns an integer:

- If x > 0, it returns 1, representing a positive number.
- If x = 0, it returns 0, representing zero.
- If x < 0, it returns -1, representing a negative number.
- If x is NULL, it returns NULL.

Example

```
select sign(3);
```

```
+-------------------------+
| sign(cast(3 as DOUBLE)) |
+-------------------------+
|                       1 |
+-------------------------+
```

```
select sign(0);
```

```
+-------------------------+
| sign(cast(0 as DOUBLE)) |
+-------------------------+
|                       0 |
+-------------------------+
```

```
select sign(-10.0);
```

```
+-----------------------------+
| sign(cast(-10.0 as DOUBLE)) |
+-----------------------------+
|                          -1 |
+-----------------------------+
```

```
select sign(null);
```

```
+------------+
| sign(NULL) |
+------------+
|       NULL |
+------------+
```

POSITIVE

Description

Returns the value itself.

Syntax

```
POSITIVE(<x>)
```

Parameters

Parameter	Description
<x>	The value that needs to be returned.

Return value

Returns an integer or a floating-point number. Special cases:

- If the parameter is NULL, return NULL.

Example

```
SELECT positive(-10);
```

```
+---------------+
| positive(-10) |
+---------------+
|           -10 |
+---------------+
```

```
SELECT positive(10.111);
```

```
+------------------+
| positive(10.111) |
+------------------+
|           10.111 |
+------------------+
```

```
SELECT positive(12);
```

```
+--------------+
| positive(12) |
+--------------+
```

```
|           12 |
+--------------+
```

```
SELECT positive(null);
```

```
+----------------+
| positive(NULL) |
+----------------+
|           NULL |
+----------------+
```

NEGATIVE

Description

Returns the negative value of the parameter x.

Syntax

```
NEGATIVE(<x>)
```

Parameters

Parameter	Description
<x>	The independent variable supports the types BIGINT, DOUBLE, and DECIMAL

Return value

Returns an integer or a floating-point number. Special cases:

- If the parameter is NULL, return NULL.
- If the parameter is 0, return 0.

Example

```
SELECT negative(-10);
```

```
+---------------+
| negative(-10) |
+---------------+
|            10 |
+---------------+
```

```
SELECT negative(12);
```

```
+--------------+
| negative(12) |
+--------------+
|          -12 |
+--------------+
```

```
SELECT negative(0);
```

```
+-------------+
| negative(0) |
+-------------+
|           0 |
+-------------+
```

```
SELECT negative(null);
```

```
+----------------+
| negative(NULL) |
+----------------+
|           NULL |
+----------------+
```

RANDOM

Description

Returns a random number between 0 and 1, or returns the required random number according to the parameters.

- Note: All parameters must be constants.

Alias

- RAND

Syntax

```
RANDOM() -- Generates a random number between 0 and 1

RANDOM(<seed>) -- Generates a fixed sequence of random numbers between 0 and 1 based on the seed value

RANDOM(<a> , <b>) -- Generates a random number between a and b
```

Parameters

Parameter	Description
<seed>	random number generator seed. Returns a fixed sequence of random numbers between 0 and 1.
<a>	The lower bound of a random number.
<b>	The upper bound of a random number. It must be greater than the lower bound.

Return value

- If no parameters are passed: Returns a random number between 0 and 1.
- If a single parameter seed is passed: Returns a fixed sequence of random numbers between 0 and 1.
- If two parameters a and b are passed: Returns a random integer between a and b.

Example

```
select random();
```

```
+--------------------+
| random()           |
+--------------------+
```

```
| 0.8047437125910604 |
+--------------------+
```

```
select rand(1.2);
```

```
+---------------------+
| rand(1)             |
+---------------------+
| 0.13387664401253274 |
+---------------------+
```

```
select rand(-20, -10);
```

```
+------------------+
| random(-20, -10) |
+------------------+
|              -10 |
+------------------+
```

MOD

Description

Find the remainder of a divided by b for the integer type. For the floating-point type, please use the fmod function.

Syntax

```
MOD(<col_a> , <col_b>)
```

Parameters

Parameter	Description
<col_a>	Dividend
<col_b>	Divisor should not be 0

Return value

Return an integer type. Special cases:

If col_a IS NULL or col_b IS NULL, return NULL.

Example

```
select mod(10, 3);
```

```
+----------+
| (10 % 3) |
+----------+
|        1 |
+----------+
```

```
select mod(10, 0);
```

```
+----------+
| (10 % 0) |
+----------+
|     NULL |
+----------+
```

NORMAL_CDF

Description

Computes the Cumulative distribution function (CDF) of the normal distribution at value `x`.

- Returns `NULL` when the standard deviation of the normal distribution is less than or equal to `0`.

Syntax

```
NORMAL_CDF(<mean>, <sd>, <x>)
```

Parameters

Parameter	Description
`<mean>`	The mean of the normal distribution
`<sd>`	The standard deviation of the normal distribution
`<x>`	The value to be evaluated

Return Value

Return the Cumulative distribution Function (CDF) for a Normal random variable at a value `x`.

- Return `NULL` when standard deviation of the normal distribution is less than or equal to `0`.

Examples

```
select normal_cdf(10, 9, 10);
```

```
+------------------------------------------------------------------------+
| normal_cdf(cast(10 as DOUBLE), cast(9 as DOUBLE), cast(10 as DOUBLE)) |
+------------------------------------------------------------------------+
|                                                                    0.5 |
+------------------------------------------------------------------------+
```

```
select NORMAL_CDF(10, 0, 10);
```

```
+------------------------------------------------------------------------+
| normal_cdf(cast(10 as DOUBLE), cast(0 as DOUBLE), cast(10 as DOUBLE)) |
+------------------------------------------------------------------------+
|                                                                   NULL |
+------------------------------------------------------------------------+
```

UUID_NUMERIC

Description

Return a LARGEINT type uuid

Syntax

```
UUID_NUMERIC()
```

Return Value

Return a LARGEINT type uuid. Note that LARGEINT is an Int128, so uuid_numeric() may produce negative values

Example

```
select uuid_numeric()
```

```
+--------------------------------------+
| uuid_numeric()                       |
+--------------------------------------+
| 82218484683747862468445277894131281464 |
+--------------------------------------+
```

WIDTH_BUCKET

Description

Construct an equal-width histogram, where the histogram range is divided into equal-sized intervals, and after calculation, return the bucket number where the value of the expression falls. Special cases:

- The function returns an integer value or NULL (if any input is NULL, it will return NULL)

Syntax

```
WIDTH_BUCKET(<expr>, <min_value>, <max_value>, <num_buckets>)
```

Parameters

Parameter	Description
`<expr>`	The expression for creating the histogram. This expression must evaluate to a numeric value or a value that can be implicitly converted to a numeric value. The value's range must be from `-(2^53 - 1)` to `2^53 - 1` (inclusive)
`<min_value>`	The lowest value point of the acceptable range for the expression. The parameter must be numeric and not equal to `<max_value>`. The range must be from `-(2^53 - 1)` to `2^53 - 1` (inclusive). Additionally, the difference between the highest and lowest value points must be less than `2^53` (for example: `abs(max_value - min_value) < 2^53`)
`<max_value>`	The highest value point of the acceptable range for the expression. The parameter must be numeric and not equal to `<min_value>`. The range must be from `-(2^53 - 1)` to `2^53 - 1` (inclusive). Additionally, the difference between the highest and lowest value points must be less than `2^53` (for example: `abs(max_value - min_value) < 2^53`)
`<num_buckets>`	The number of buckets. It must be a positive integer. The function assigns a value from the expression to each bucket and returns the corresponding bucket number

Return Value

Return the bucket number where the expression value falls. The function returns the following rules when the expression is out of range:

- If the value of the expression is less than `min_value`, return 0.
- If the value of the expression is greater than or equal to max_value, return `num_buckets + 1`.
- If any parameter is `null`, return `null`

Example

```
DROP TABLE IF EXISTS width_bucket_test;

CREATE TABLE IF NOT EXISTS width_bucket_test (
`k1` int NULL COMMENT "",
`v1` date NULL COMMENT "",
`v2` double NULL COMMENT "",
`v3` bigint NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`k1`)
DISTRIBUTED BY HASH(`k1`) BUCKETS 1
PROPERTIES (
"replication_allocation" = "tag.location.default: 1",
"storage_format" = "V2"
);

INSERT INTO width_bucket_test VALUES
(1, "2022-11-18", 290000.00, 290000),
(2, "2023-11-18", 320000.00, 320000),
(3, "2024-11-18", 399999.99, 399999),
(4, "2025-11-18", 400000.00, 400000),
(5, "2026-11-18", 470000.00, 470000),
(6, "2027-11-18", 510000.00, 510000),
(7, "2028-11-18", 610000.00, 610000),
(8, null, null, null);
```

```
SELECT * FROM width_bucket_test ORDER BY k1;
```

```
+------+------------+-----------+--------+
| k1   | v1         | v2        | v3     |
+------+------------+-----------+--------+
|    1 | 2022-11-18 |    290000 | 290000 |
|    2 | 2023-11-18 |    320000 | 320000 |
|    3 | 2024-11-18 | 399999.99 | 399999 |
|    4 | 2025-11-18 |    400000 | 400000 |
|    5 | 2026-11-18 |    470000 | 470000 |
|    6 | 2027-11-18 |    510000 | 510000 |
|    7 | 2028-11-18 |    610000 | 610000 |
|    8 | NULL       |      NULL |   NULL |
+------+------------+-----------+--------+
```

```
SELECT k1, v1, v2, v3, width_bucket(v1, date('2023-11-18'), date('2027-11-18'), 4) AS w FROM width_bucket_test ORDER BY k1;
```

```
+------+------------+-----------+--------+------+
| k1   | v1         | v2        | v3     | w    |
+------+------------+-----------+--------+------+
|    1 | 2022-11-18 |    290000 | 290000 |    0 |
|    2 | 2023-11-18 |    320000 | 320000 |    1 |
|    3 | 2024-11-18 | 399999.99 | 399999 |    2 |
|    4 | 2025-11-18 |    400000 | 400000 |    3 |
|    5 | 2026-11-18 |    470000 | 470000 |    4 |
|    6 | 2027-11-18 |    510000 | 510000 |    5 |
|    7 | 2028-11-18 |    610000 | 610000 |    5 |
|    8 | NULL       |      NULL |   NULL | NULL |
+------+------------+-----------+--------+------+
```

```
SELECT k1, v1, v2, v3, width_bucket(v2, 200000, 600000, 4) AS w FROM width_bucket_test ORDER BY k1;
```

```
+------+------------+-----------+--------+------+
| k1   | v1         | v2        | v3     | w    |
+------+------------+-----------+--------+------+
|    1 | 2022-11-18 |    290000 | 290000 |    1 |
|    2 | 2023-11-18 |    320000 | 320000 |    2 |
|    3 | 2024-11-18 | 399999.99 | 399999 |    2 |
|    4 | 2025-11-18 |    400000 | 400000 |    3 |
|    5 | 2026-11-18 |    470000 | 470000 |    3 |
|    6 | 2027-11-18 |    510000 | 510000 |    4 |
|    7 | 2028-11-18 |    610000 | 610000 |    5 |
|    8 | NULL       |      NULL |   NULL | NULL |
+------+------------+-----------+--------+------+
```

```
SELECT k1, v1, v2, v3, width_bucket(v3, 200000, 600000, 4) AS w FROM width_bucket_test ORDER BY k1;
```

```
+------+------------+-----------+--------+------+
| k1   | v1         | v2        | v3     | w    |
+------+------------+-----------+--------+------+
|    1 | 2022-11-18 |    290000 | 290000 |    1 |
|    2 | 2023-11-18 |    320000 | 320000 |    2 |
|    3 | 2024-11-18 | 399999.99 | 399999 |    2 |
|    4 | 2025-11-18 |    400000 | 400000 |    3 |
|    5 | 2026-11-18 |    470000 | 470000 |    3 |
|    6 | 2027-11-18 |    510000 | 510000 |    4 |
|    7 | 2028-11-18 |    610000 | 610000 |    5 |
|    8 | NULL       |      NULL |   NULL | NULL |
+------+------------+-----------+--------+------+
```

APPEND_TRAILING_CHAR_IF_ABSENT

Description

Used to add a specific character (such as a space, a specific symbol, etc.) to the end of a string if the character does not exist at the end of the string. The function is to ensure that the string ends with a specific character.

Syntax

```
APPEND_TRAILING_CHAR_IF_ABSENT ( <str> , <trailing_char> )
```

Parameters

Parameters	Description
<str>	Target string to be judged
<trailing_char>	Character to be added to the end of the string (if the character does not exist)

Return value

Parameters The string after concatenation of <str> and <trailing_char> (if <trailing_char> does not exist in <str>)

Example

1. Basic usage: Add character when it doesn't exist

```
SELECT APPEND_TRAILING_CHAR_IF_ABSENT('a', 'c');
```

```
+-----------------------------------------+
| APPEND_TRAILING_CHAR_IF_ABSENT('a', 'c') |
+-----------------------------------------+
| ac                                      |
+-----------------------------------------+
```

2. Character already exists, don't add

```
SELECT APPEND_TRAILING_CHAR_IF_ABSENT('ac', 'c');
```

```
+------------------------------------------+
| APPEND_TRAILING_CHAR_IF_ABSENT('ac', 'c') |
+------------------------------------------+
| ac                                       |
+------------------------------------------+
```

3. Empty string handling

```
SELECT APPEND_TRAILING_CHAR_IF_ABSENT('', '/');
```

```
+---------------------------------------+
| APPEND_TRAILING_CHAR_IF_ABSENT('', '/') |
+---------------------------------------+
| /                                     |
+---------------------------------------+
```

4. NULL value handling

```
SELECT APPEND_TRAILING_CHAR_IF_ABSENT(NULL, 'c');
```

```
+-----------------------------------------+
| APPEND_TRAILING_CHAR_IF_ABSENT(NULL, 'c') |
+-----------------------------------------+
| NULL                                    |
+-----------------------------------------+
```

5. UTF-8 character

```
SELECT APPEND_TRAILING_CHAR_IF_ABSENT('acf', 'ṛ');
```

```
+-------------------------------------------+
| APPEND_TRAILING_CHAR_IF_ABSENT('acf', 'ṛ')   |
+-------------------------------------------+
| acfṛ                                      |
+-------------------------------------------+
```

ASCII

Description

Returns the ASCII code of the first character of a string

Syntax

```
ASCII ( <str> )
```

Parameters

Parameter	Description
<str>	The string whose ASCII code of the first character needs to be calculated

Return value

Parameter ASCII code of the first character

Example

```
SELECT ASCII('1'),ASCII('234')
```

```
+------------+--------------+
| ascii('1') | ascii('234') |
+------------+--------------+
|         49 |           50 |
+------------+--------------+
```

AUTO_PARTITION_NAME

> **tip**
>
> Supported since Apache Doris 2.1.6

Description

Syntax

```
VARCHAR AUTO_PARTITION_NAME('RANGE', 'VARCHAR unit', DATETIME datetime)

VARCHAR AUTO_PARTITION_NAME('LIST', VARCHAR,...)
```

Generate datetime partition names by unit following RANGE's partition name rules

Convert strings to partition names following LIST's partition name rules

The datetime parameter is a legal date expression.

The unit parameter is the time interval you want, the available values are: [`second`, `minute`, `hour`, `day`, `month`, `year`]. If unit does not match one of these options, a syntax error will be returned.

Supported since Doris 2.1.6

Example

```
mysql> select auto_partition_name('range', 'years', '123');
ERROR 1105 (HY000): errCode = 2, detailMessage = range auto_partition_name must accept year|month|day|hour|minute|second for 2nd
argument

mysql> select auto_partition_name('range', 'year', '2022-12-12 19:20:30');
+-----------------------------------------------------------+
| auto_partition_name('range', 'year', '2022-12-12 19:20:30')   |
+-----------------------------------------------------------+
| p20220101000000                                           |
+-----------------------------------------------------------+

mysql> select auto_partition_name('range', 'month', '2022-12-12 19:20:30');
+-----------------------------------------------------------+
| auto_partition_name('range', 'month', '2022-12-12 19:20:30')  |
+-----------------------------------------------------------+
| p20221201000000                                           |
+-----------------------------------------------------------+

mysql> select auto_partition_name('range', 'day', '2022-12-12 19:20:30');
+-----------------------------------------------------------+
| auto_partition_name('range', 'day', '2022-12-12 19:20:30')    |
+-----------------------------------------------------------+
| p20221212000000                                           |
+-----------------------------------------------------------+
```

```
mysql> select auto_partition_name('range', 'hour', '2022-12-12 19:20:30');
+-----------------------------------------------------------------+
| auto_partition_name('range', 'hour', '2022-12-12 19:20:30')   |
+-----------------------------------------------------------------+
| p20221212190000                                                 |
+-----------------------------------------------------------------+

mysql> select auto_partition_name('range', 'minute', '2022-12-12 19:20:30');
+-------------------------------------------------------------------+
| auto_partition_name('range', 'minute', '2022-12-12 19:20:30') |
+-------------------------------------------------------------------+
| p20221212192000                                                   |
+-------------------------------------------------------------------+

mysql> select auto_partition_name('range', 'second', '2022-12-12 19:20:30');
+-------------------------------------------------------------------+
| auto_partition_name('range', 'second', '2022-12-12 19:20:30') |
+-------------------------------------------------------------------+
| p20221212192030                                                   |
+-------------------------------------------------------------------+

mysql> select auto_partition_name('list', 'helloworld');
+-----------------------------------------+
| auto_partition_name('list', 'helloworld') |
+-----------------------------------------+
| phelloworld10                           |
+-----------------------------------------+

mysql> select auto_partition_name('list', 'hello', 'world');
+-----------------------------------------------+
| auto_partition_name('list', 'hello', 'world') |
+-----------------------------------------------+
| phello5world5                                 |
+-----------------------------------------------+

mysql> select auto_partition_name('list', "你好");
+-----------------------------------+
| auto_partition_name('list', "你好") |
+-----------------------------------+
| p4f60597d2                        |
+-----------------------------------+
```

Keywords

```
AUTO_PARTITION_NAME,AUTO,PARTITION,NAME
```

CHAR | String Functions

CHAR

Description

Interpret each argument as an integer and return a string consisting of the characters given by the code values of those integers. Special cases:

- If the result string is illegal for the given character set, the corresponding conversion result is NULL.
- Arguments greater than `255` are converted to multiple result bytes. For example, `char(15049882)` is equivalent to `char(229, 164, 154)`.

Description

Interpret each argument as an integer and return a string consisting of the characters given by the code values of those integers. Special cases:

- If the result string is illegal for the given character set, the corresponding conversion results in the value NULL.
- Arguments greater than `255` are converted to multiple result bytes. For example, `char(15049882)` is equivalent to `char(229, 164, 154)`.

Syntax

```
CHAR ( <expr> [ , <expr> ... ] [ USING <charset_name> ] )
```

Parameters

Parameters	Description
`<expr>`	Integer to be calculated as a character
`<charset_name>`	Encoding of the return value, currently only supports `utf8`

Return value

Parameter list `<expr>` A string consisting of the corresponding characters. Special cases:

- If the result string is illegal for the given character set, the corresponding conversion result is NULL.
- Parameters greater than `255` will be converted to multiple result bytes. For example, `CHAR(15049882)` is equivalent to `CHAR(229, 164, 154)`.

Example

```
SELECT CHAR(68, 111, 114, 105, 115),CHAR(15049882, 15179199, 14989469),CHAR(255)
```

```
+-------------------------------------+-----------------------------------------+------------------+
| char('utf8', 68, 111, 114, 105, 115) | char('utf8', 15049882, 15179199, 14989469) | char('utf8', 255) |
+-------------------------------------+-----------------------------------------+------------------+
| Doris                               | 多睿丝                                  | NULL             |
+-------------------------------------+-----------------------------------------+------------------+
```

CHAR_LENGTH

Description

Calculates the length of a string. For multi-byte characters, returns the number of characters.

Currently only supports `utf8` encoding

Alias

- CHARACTER_LENGTH

Syntax

```
CHAR_LENGTH ( <str> )
```

Parameters

Parameter	Description
<str>	The string to calculate the length of

Return value

The length of the string .

Example

```
select CHAR_LENGTH("abc"),CHAR_LENGTH("中国")
```

```
+-------------------------+---------------------------+
| character_length('abc') | character_length('中国')   |
+-------------------------+---------------------------+
|                       3 |                         2 |
+-------------------------+---------------------------+
```

CONCAT

Description

Concatenates multiple strings. Special cases:

- If any of the parameter values is NULL, the result returned is NULL

Syntax

```
CONCAT ( <expr> [ , <expr> ... ] )
```

Parameters

Parameter	Description
<expr>	The strings to be concatenated

Return value

Parameter list <expr> The strings to be concatenated. Special cases:

- If any of the parameter values is NULL, the result returned is NULL

Example

```
SELECT  CONCAT("a", "b"),CONCAT("a", "b", "c"),CONCAT("a", null, "c")
```

```
+------------------+----------------------+-----------------------+
| concat('a', 'b') | concat('a', 'b', 'c') | concat('a', NULL, 'c') |
+------------------+----------------------+-----------------------+
| ab               | abc                  | NULL                  |
+------------------+----------------------+-----------------------+
```

CONCAT_WS

Description

Use the first parameter sep as the connector to concatenate the second parameter and all subsequent parameters (or all strings in ARRAY) into a string. Special cases:

- If the separator is NULL, NULL is returned.

The `CONCAT_WS` function does not skip empty strings, but skips NULL values.

Syntax

```
CONCAT_WS ( <sep> , <str> [ , <str> ] )
CONCAT_WS ( <sep> , <array> )
```

Parameters

Parameter	Description
<sep>	Connector for concatenating strings
<str>	String to be concatenated
<array>	Array to be concatenated

Return value

Parameter <sep> or <array> The string concatenated with <str>. Special cases:

- If delimiter is NULL, returns NULL.

Example

Concatenate strings together using or

```
SELECT CONCAT_WS("or", "d", "is"),CONCAT_WS(NULL, "d", "is"),CONCAT_WS('or', 'd', NULL, 'is')
```

```
+----------------------------+----------------------------+----------------------------------------+
| concat_ws('or', 'd', 'is') | concat_ws(NULL, 'd', 'is') | concat_ws('or', 'd', NULL, 'is') |
+----------------------------+----------------------------+----------------------------------------+
| doris                      | NULL                       | doris                                  |
+----------------------------+----------------------------+----------------------------------------+
```

Concatenate array arrays together using or

```
SELECT CONCAT_WS("or", ["d", "is"]),CONCAT_WS(NULL, ["d", "is"]),CONCAT_WS("or", ["d", NULL,"is"])
```

```
+-----------------------------+-----------------------------+-----------------------------------+
| concat_ws('or', ['d', 'is']) | concat_ws(NULL, ['d', 'is']) | concat_ws('or', ['d', NULL, 'is']) |
+-----------------------------+-----------------------------+-----------------------------------+
| doris                       | NULL                        | doris                             |
+-----------------------------+-----------------------------+-----------------------------------+
```

COUNT_SUBSTRINGS

Description

The COUNT_SUBSTRINGS function counts the number of occurrences of a specified substring within a string. Note: The current implementation continues searching after shifting by the length of the substring when a match is found. For example, when str='ccc' and pattern='cc', the result returned is 1.

Syntax

```
COUNT_SUBSTRINGS(<str>, <pattern>)
```

Parameters

Parameter	Description
<str>	The string to be searched. Type: STRING
<pattern>	The substring to match. Type: STRING

Return Value

Returns an INT type, representing the number of times the substring appears in the string.

Special cases: - If str is NULL, returns NULL - If pattern is an empty string, returns 0 - If str is an empty string, returns 0

Examples

1. Basic usage

```
SELECT count_substrings('a1b1c1d', '1');
```

```
+----------------------------------+
| count_substrings('a1b1c1d', '1') |
+----------------------------------+
|                                3 |
+----------------------------------+
```

2. Case with consecutive commas

```
SELECT count_substrings(',,a,b,c,', ',');
```

```
+---------------------------------+
| count_substrings(',,a,b,c,', ',') |
+---------------------------------+
```

```
|                                5 |
+----------------------------------+
```

3. Case with overlapping substrings

```
SELECT count_substrings('ccc', 'cc');
```

```
+--------------------------------+
| count_substrings('ccc', 'cc')  |
+--------------------------------+
|                              1 |
+--------------------------------+
```

4. NULL value handling

```
SELECT count_substrings(NULL, ',');
```

```
+----------------------------+
| count_substrings(NULL, ',') |
+----------------------------+
|                       NULL |
+----------------------------+
```

5. Empty string handling

```
SELECT count_substrings('a,b,c,abcde', '');
```

```
+-----------------------------------+
| count_substrings('a,b,c,abcde', '') |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

CUT_TO_FIRST_SIGNIFICANT_SUBDOMAIN

Description

The CUT_TO_FIRST_SIGNIFICANT_SUBDOMAIN function extracts the effective part of a domain from a URL, including the top-level domain up to the “first significant subdomain”. If the input URL is invalid, it returns an empty string.

Syntax

```
CUT_TO_FIRST_SIGNIFICANT_SUBDOMAIN(<url>)
```

Parameters

Parameter	Description
<url>	The URL string to be processed. Type: VARCHAR

Return Value

Returns VARCHAR type, representing the extracted domain part.

Special cases: - If url is NULL, returns NULL - If url is not a valid domain format, returns an empty string

Examples

1. Basic domain processing

```
SELECT cut_to_first_significant_subdomain('www.baidu.com');
```

```
+-----------------------------------------------------+
| cut_to_first_significant_subdomain('www.baidu.com') |
+-----------------------------------------------------+
| baidu.com                                           |
+-----------------------------------------------------+
```

2. Multi-level domain processing

```
SELECT cut_to_first_significant_subdomain('www.google.com.cn');
```

```
+--------------------------------------------------------+
| cut_to_first_significant_subdomain('www.google.com.cn') |
+--------------------------------------------------------+
| google.com.cn                                          |
+--------------------------------------------------------+
```

3. Invalid domain processing

```
SELECT cut_to_first_significant_subdomain('wwwwwwww');
```

```
+-----------------------------------------------+
| cut_to_first_significant_subdomain('wwwwwwww') |
+-----------------------------------------------+
|                                               |
+-----------------------------------------------+
```

DIGITAL_MASKING

Description

The `digital_masking` function is used for masking numbers. Based on the specified masking rule, certain characters in the number are replaced with *. This function is an alias for the original function `concat(left(id, 3), '****', right(id, 4))`.

Syntax

```
DIGITAL_MASKING( <digital_number> )
```

Parameters

Parameter	Description
`<digital_number>`	The digital string that needs to be masked

Return Value

Returns the masked digital string.

Examples

```
select digital_masking(13812345678);
```

```
+-----------------------------+
| digital_masking(13812345678) |
+-----------------------------+
| 138****5678                 |
+-----------------------------+
```

DOMAIN

Description

Extract the domain name from the string URL

Syntax

```
DOMAIN ( <url> )
```

Parameters

Parameter	Description
<url>	URL from which the domain name needs to be extracted

Return value

Domain name of parameter <url>

Example

```
SELECT DOMAIN("https://doris.apache.org/docs/gettingStarted/what-is-apache-doris")
```

```
+-----------------------------------------------------------------------------+
| domain('https://doris.apache.org/docs/gettingStarted/what-is-apache-doris') |
+-----------------------------------------------------------------------------+
| doris.apache.org                                                            |
+-----------------------------------------------------------------------------+
```

DOMAIN_WITHOUT_WWW

Description

Extract the domain name without the prefix www in the string URL

Syntax

```
DOMAIN_WITHOUT_WWW ( <url> )
```

Parameters

Parameter	Description
<url>	Need to extract the URL without the www domain name

Return value

Parameter <url> Domain name without the prefix www

```
SELECT DOMAIN_WITHOUT_WWW("https://www.apache.org/docs/gettingStarted/what-is-apache-doris")
```

```
+-----------------------------------------------------------------------------------+
| domain_without_www('https://www.apache.org/docs/gettingStarted/what-is-apache-doris') |
+-----------------------------------------------------------------------------------+
| apache.org                                                                        |
+-----------------------------------------------------------------------------------+
```

ELT

Description

Returns a string at the specified index. Special cases:

- If there is no string at the specified index, NULL is returned.

Syntax

```
ELT ( <pos> , <str> [ , <str> ] )
```

Parameters

Parameter	Description
<pos>	Specified index value
<str>	String to be indexed

Return value

Parameter <str> String to be indexed. Special cases:

- If there is no string at the specified index, NULL is returned.

Example

```
SELECT ELT(1, 'aaa', 'bbb'),ELT(2, 'aaa', 'bbb'), ELT(0, 'aaa', 'bbb'),ELT(2, 'aaa', 'bbb')
```

```
+----------------------+----------------------+----------------------+----------------------+
| elt(1, 'aaa', 'bbb') | elt(2, 'aaa', 'bbb') | elt(0, 'aaa', 'bbb') | elt(2, 'aaa', 'bbb') |
+----------------------+----------------------+----------------------+----------------------+
| aaa                  | bbb                  | NULL                 | bbb                  |
+----------------------+----------------------+----------------------+----------------------+
```

ENDS_WITH

Description

Returns true if the string ends with the specified suffix, otherwise returns false. Special cases:

- If either of the two parameters is NULL, returns NULL.

Syntax

```
ENDS_WITH ( <str> , <suffix> )
```

Parameters

Parameter	Description
str	Specifies the original string to be judged
suffix	Specifies the ending string to be judged

Return value

true or false, type is BOOLEAN. Special cases:

- If either of the two parameters is NULL, returns NULL.

Example

```
SELECT ENDS_WITH("Hello doris", "doris"),ENDS_WITH("Hello doris", "Hello")
```

```
+----------------------------------+----------------------------------+
| ends_with('Hello doris', 'doris') | ends_with('Hello doris', 'Hello') |
+----------------------------------+----------------------------------+
|                                1 |                                0 |
+----------------------------------+----------------------------------+
```

EXTRACT_URL_PARAMETER

Description

Returns the value of the `name` parameter in the URL, if it exists, or an empty string otherwise.

If there are multiple parameters with this name, the first one that occurs is returned.

This function works assuming that the parameter name is encoded in the URL exactly as it is in the passed parameter.

If you want to get other parts of the URL, you can use parse_url

Syntax

```
EXTRACT_URL_PARAMETER ( <url> , <name> )
```

Parameters

Parameters	Description
`<url>`	The url string of the parameter to be returned
`<name>`	The name of the parameter to be returned

Return Value

The value of the parameter `<name>` in `<url>`

Example

```
SELECT EXTRACT_URL_PARAMETER("http://doris.apache.org?k1=aa&k2=bb&test=cc#999", "k2")
```

```
+-----------------------------------------------------------------------------+
| extract_url_parameter('http://doris.apache.org?k1=aa&k2=bb&test=cc#999', 'k2') |
+-----------------------------------------------------------------------------+
| bb                                                                          |
+-----------------------------------------------------------------------------+
```

FIND_IN_SET

Description

Returns the position of the first occurrence of str in strlist (counting starts at 1).

strlist is a comma-delimited string. Special cases:

- If not found, returns 0.
- If any parameter is NULL, returns NULL.

Syntax

```
FIND_IN_SET ( <str> , <strlist> )
```

Parameters

Parameter	Description
<str>	String to be searched
<strlist>	String to be searched

Return value

The position of the first occurrence of parameter <str> in parameter <strlist>. Special cases:

- If not found, returns 0.
- If any parameter is NULL, returns NULL.

Example

```
SELECT FIND_IN_SET("b", "a,b,c")
```

```
| find_in_set('b', 'a,b,c') |
+---------------------------+
|                         2 |
+---------------------------+
```

FIRST_SIGNIFICANT_SUBDOMAIN

Description

Extract the “first valid subdomain” from the URL and return it. If it is illegal, an empty string will be returned.

Syntax

```
FIRST_SIGNIFICANT_SUBDOMAIN ( <url> )
```

Parameters

Parameter	Description
<url>	The URL from which the “first valid subdomain” needs to be extracted

Return value

The first valid subdomain in <url>.

Example

```
SELECT
FIRST_SIGNIFICANT_SUBDOMAIN("www.baidu.com"),FIRST_SIGNIFICANT_SUBDOMAIN("www.google.com.cn"),FIRST_SIGNIFICANT_SUBDOMAIN("wwwwwwww"
)
```

```
+-------------------------------------------+-------------------------------------------------+--------------------------------
--------+
| first_significant_subdomain('www.baidu.com') | first_significant_subdomain('www.google.com.cn') |
first_significant_subdomain('wwwwwwww') |
+-------------------------------------------+-------------------------------------------------+--------------------------------
--------+
| baidu                                     | google                                          |
|
+-------------------------------------------+-------------------------------------------------+--------------------------------
--------+
```

FROM_BASE64

Description

Returns the result of Base64 decoding the input string. Special cases:

- When the input string is incorrect (a string that is not possible after Base64 encoding appears), NULL will be returned

Syntax

```
FROM_BASE64 ( <str> )
```

Parameters

Parameters	Description
<str>	The string to be Base64 decoded

Return value

Parameter The result of Base64 decoding. Special cases:

- When the input string is incorrect (a string that is not possible after Base64 encoding appears), NULL will be returned.

Example

```
SELECT FROM_BASE64('MQ=='),FROM_BASE64('MjM0'),FROM_BASE64(NULL)
```

```
+--------------------+--------------------+-------------------+
| from_base64('MQ==') | from_base64('MjM0') | from_base64(NULL) |
+--------------------+--------------------+-------------------+
| 1                  | 234                | NULL              |
+--------------------+--------------------+-------------------+
```

HEX

Description

If the input parameter is a number, return the string representation of the hexadecimal value.

If the input parameter is a string, convert each character to two hexadecimal characters, concatenate all the converted characters into a string for output.

Syntax

```
HEX ( <str> )
```

Parameters

Parameter	Description
<str>	Input parameter is a number or a string

Return value

The hexadecimal result of parameter <str>.

Example

The input parameter is a number

```
SELECT HEX(12),HEX(-1)
```

```
+---------+------------------+
| hex(12) | hex(-1)          |
+---------+------------------+
| C       | FFFFFFFFFFFFFFFF |
+---------+------------------+
```

The input parameter is a string

```
SELECT HEX('1'),HEX('@'),HEX('12')
```

```
+----------+----------+-----------+
| hex('1') | hex('@') | hex('12') |
+----------+----------+-----------+
| 31       | 40       | 3132      |
+----------+----------+-----------+
```

INITCAP

Description

Capitalizes the first letter of the word contained in the parameter and converts the rest of the letters to lowercase. A word is a sequence of alphanumeric characters separated by non-alphanumeric characters.

Syntax

```
INITCAP ( <str> )
```

Parameters

Parameter	Description
<str>	The string to be converted

Return Value

The result of capitalizing the first letter of the word in the parameter <str> and lowering the rest of the letters.

Example

```
SELECT INITCAP('hello hello.,HELLO123HELlo')
```

```
+--------------------------------------+
| initcap('hello hello.,HELLO123HELlo') |
+--------------------------------------+
| Hello Hello.,hello123hello            |
+--------------------------------------+
```

INSTR

Description

Returns the first occurrence position of substr in str (counting starts from 1). Special cases:

- If substr does not appear in str, returns 0.

Syntax

```
INSTR ( <str> , <substr> )
```

Parameters

Parameter	Description
<str>	String to search for
<substr>	String to be searched for

Return value

Parameters The first occurrence position of <substr> in <str> (counting starts from 1). Special cases:

- If <substr> does not appear in <str>, returns 0.

Example

```
SELECT INSTR("abc", "b"),INSTR("abc", "d")
```

```
+------------------+------------------+
| instr('abc', 'b') | instr('abc', 'd') |
+------------------+------------------+
|                2 |                0 |
+------------------+------------------+
```

INT_TO_UUID

Description

For an encoded LARGEINT input, convert it to a raw uuid string.

Syntax

```
INT_TO_UUID ( <int128> )
```

Parameters

Parameter	Description
<int128>	Encoded LARGEINT value

Return value

Parameter <int128> Raw uuid string.

Example

```
SELECT INT_TO_UUID(957219555148694080917592900713939528 76)
```

```
+-----------------------------------------------------+
| int_to_uuid(957219555148694080917592900713939528 76) |
+-----------------------------------------------------+
| 6ce4766f-6783-4b30-b357-bba1c7600348                |
+-----------------------------------------------------+
```

LCASE/LOWER

Description

Convert all strings in the parameter to lowercase.

Alias

- LOWER

Syntax

```
LCASE ( <str> )
LOWER ( <str> )
```

Parameters

Parameter	Description
<str>	String to be converted to lowercase

Return Value

Parameter <str> String converted to lowercase

Example

```
SELECT LCASE("AbC123"),LOWER("AbC123")
```

```
+-----------------+-----------------+
| lower('AbC123') | lower('AbC123') |
+-----------------+-----------------+
| abc123          | abc123          |
+-----------------+-----------------+
```

```
SELECT LOWER("ҚAZAҚ123"),LCASE("ҒARЫШ");
```

```
+--------------------+--------------------+
| LOWER("ҚAZAҚ123")  | LCASE("ҒARЫШ")     |
+--------------------+--------------------+
| қazaқ123           | ғarыш              |
+--------------------+--------------------+
```

LENGTH

Description

Returns the number of bytes in a string.

Syntax

```
LENGTH ( <str> )
```

Parameters

Parameter	Description
<str>	The string whose bytes need to be calculated

Return Value

The number of bytes in the string <str>.

Example

```
SELECT LENGTH("abc"),length("中国")
```

```
+---------------+------------------+
| length('abc') | length('中国')   |
+---------------+------------------+
|             3 |                6 |
+---------------+------------------+
```

LOCATE

Description

Returns the position of substr in str (counting from 1). If the third parameter pos is specified, the position of substr is searched from the string starting with the pos subscript. If not found, 0 is returned

Syntax

```
LOCATE ( <substr> , <str> [, <pos> ] )
```

Parameters

Parameter	Description
substr	The substring to be searched
str	The string to be searched
pos	If this parameter is specified, the position of substr is searched from the string starting with the pos subscript

Return value

The position of substr in str (counting from 1)

Example

```
SELECT LOCATE('bar', 'foobarbar'),LOCATE('xbar', 'foobar'),LOCATE('bar', 'foobarbar', 5)
```

```
+---------------------------+-------------------------+-----------------------------+
| locate('bar', 'foobarbar') | locate('xbar', 'foobar') | locate('bar', 'foobarbar', 5) |
+---------------------------+-------------------------+-----------------------------+
|                          4 |                       0 |                           7 |
+---------------------------+-------------------------+-----------------------------+
```

LPAD

Description

Returns a string of length len (starting from the first letter) in str.

If len is greater than the length of str, pad characters are added to the front of str until the length of the string reaches len.

If len is less than the length of str, this function is equivalent to truncating the str string and returning only a string of length len. len refers to the character length rather than the byte length.

Special cases:

- In addition to containing NULL values, if pad is empty, the return value is an empty string.

Syntax

```
LPAD ( <str> , <len> , <pad>)
```

Parameters

Parameters	Description
<str>	The string to be padded
<len>	The length of the string to be padded, which refers to the length of characters rather than bytes
<pad>	The string to be padded on the left of the original string

Return value

The padded string. Special cases:

- If pad is empty, the return value is an empty string, except when it contains NULL values.

Example

```
SELECT LPAD("hi", 5, "xy"),LPAD("hi", 1, "xy"),LPAD("", 0, "")
```

```
+--------------------+--------------------+-----------------+
| lpad('hi', 5, 'xy') | lpad('hi', 1, 'xy') | lpad('', 0, '') |
+--------------------+--------------------+-----------------+
| xyxhi              | h                  |                 |
+--------------------+--------------------+-----------------+
```

LTRIM

Description

The LTRIM function is used to remove consecutive spaces or specified characters from the left side (leading) of a string.

Syntax

```
LTRIM( <str> [, <trim_chars> ] )
```

Parameters

Parameter	Description
<str>	String that need to be trimmed
<trim_chars>	Optional Parameter. If this parameter is provided, the LTRIM function will remove all characters from the <trim_chars> that appear on the left side of <str>. If this parameter is not provided, the LTRIM function will only remove the space character to the left of '<str>'.

Return Value

Returns a string with leading spaces or <trim_chars> removed. Special cases:

- If any Parameter is NULL, NULL will be returned.

Examples

```
SELECT ltrim('   ab d') str;
```

```
+------+
| str  |
+------+
| ab d |
+------+
```

```
SELECT ltrim('ababccaab','ab') str;
```

```
+-------+
| str   |
+-------+
```

```
| ccaab |
+-------+
```

LTRIM_IN

Description

The LTRIM_IN function removes specified characters from the left side of a string. When no character set is specified, it removes leading spaces by default. When a character set is specified, it removes all specified characters from the left side (regardless of their order in the set). The key feature of LTRIM_IN is that it removes any combination of characters from the specified set, while the LTRIM function removes characters based on exact string matching.

Syntax

```
LTRIM_IN(<str> [, <rhs>])
```

Parameters

Parameter	Description
<str>	The string to be processed. Type: VARCHAR
<rhs>	Optional parameter, the set of characters to be removed. Type: VARCHAR

Return Value

Returns VARCHAR type, representing the processed string.

Special cases: - If str is NULL, returns NULL - If rhs is not specified, removes all leading spaces - If rhs is specified, removes all characters from the left side that appear in rhs until encountering the first character not in rhs

Examples

1. Remove leading spaces

```
SELECT ltrim_in('   ab d') str;
```

```
+------+
| str  |
+------+
| ab d |
+------+
```

2. Remove specified character set

```
SELECT ltrim_in('ababccaab', 'ab') str;
```

```
+-------+
| str   |
+-------+
| ccaab |
+-------+
```

3. Comparison with LTRIM function

```
SELECT ltrim_in('abcd', 'ae'),ltrim('abcd', 'abe');
```

```
+------------------------+----------------------+
| ltrim_in('abcd', 'ae') | ltrim('abcd', 'abe') |
+------------------------+----------------------+
| bcd                    | abcd                 |
+------------------------+----------------------+
```

MASK

Description

The MASK function is to shield data to protect sensitive information, and it is commonly used in data anonymization scenarios. Its default behavior is to convert a uppercase letter in the input string to `X`, a lowercase letter to `x`, and a number to `n`.

Syntax

```
MASK(<str> [, <upper> [, <lower> [, <number> ]]])
```

Parameters

Parameter	Description
`<str>`	String that need to be masked
`<upper>`	Optional Parameter, replaces uppercase character to `X` by default. If a sequence of characters are input, the first character will be taken, and if non ASCII characters are input, the first byte will be taken
`<lower>`	Optional Parameter, replaces lowercase character to `x` by default. If a sequence of characters are input, the first character will be taken, and if non ASCII characters are input, the first byte will be taken
`<number>`	Optional Parameter, replaces numeric character to `n` by default. If a sequence of characters are input, the first character will be taken, and if non ASCII characters are input, the first byte will be taken

Return Value

Returns a string after masking uppercase character, lowercase character and lnumeric character. Special cases:

- If any Parameter is NULL, NULL will be returned.
- Non-alphabetic and non-numeric characters will do not masking

Examples

```
select mask('abc123EFG');
```

```
+------------------+
| mask('abc123EFG') |
+------------------+
| xxxnnnXXX         |
+------------------+
```

```
select mask(null);
```

```
+------------+
| mask(NULL) |
+------------+
| NULL       |
+------------+
```

```
select mask('abc123EFG', '*', '#', '$');
```

```
+---------------------------------+
| mask('abc123EFG', '*', '#', '$') |
+---------------------------------+
| ###$$$***                       |
+---------------------------------+
```

MASK_FIRST_N

Description

The MASK_FIRST_N function is mainly used to mask the first N bytes of data to protect sensitive information, and is commonly used in data anonymization scenarios. Its behavior is to replace a uppercase letter with `X`, a lowercase letter with `x`, and a number with `n` in the first N bytes.

Syntax

```
MASK_FIRST_N( <str> [, <n> ])
```

Parameters

Parameter	Description
<str>	String that need to be masked
<n>	Optional Parameter, limit data masking to only the first N bytes, default to masking the entire string

Return Value

Returns a string after masking uppercase character, lowercase character and lnumeric character in first N bytes. Special cases:

- If any Parameter is NULL, NULL will be returned.
- Non-alphabetic and non-numeric characters will do not masking

Examples

```
select mask_first_n("1234-5678-8765-4321", 4);
```

```
+----------------------------------------+
| mask_first_n('1234-5678-8765-4321', 4) |
+----------------------------------------+
| nnnn-5678-8765-4321                    |
+----------------------------------------+
```

```
select mask_first_n("1234-5678-8765-4321", null);
```

```
+----------------------------------------+
| mask_first_n('1234-5678-8765-4321', NULL) |
+----------------------------------------+
```

```
| NULL                                    |
+-----------------------------------------+
```

MASK_LAST_N

Description

The MASK_LAST_N function is mainly used to mask the last N bytes of data to protect sensitive information, and is commonly used in data anonymization scenarios. Its behavior is to replace a uppercase letter with `X`, a lowercase letter with `x`, and a number with `n` in the first N bytes.

Syntax

```
MASK_LAST_N( <str> [, <n> ])
```

Parameters

Parameter	Description
<str>	String that need to be masked
<n>	Optional Parameter, limit data masking to only the last N bytes, default to masking the entire string

Return Value

Returns a string after masking uppercase character, lowercase character and lnumeric character in last N bytes. Special cases:

- If any Parameter is NULL, NULL will be returned.
- Non-alphabetic and non-numeric characters will do not masking

Examples

```
select mask_last_n("1234-5678-8765-4321", 4);
```

```
+--------------------------------------+
| mask_last_n('1234-5678-8765-4321', 4) |
+--------------------------------------+
| 1234-5678-8765-nnnn                  |
+--------------------------------------+
```

```
select mask_last_n("1234-5678-8765-4321", null);
```

```
+----------------------------------------+
| mask_last_n('1234-5678-8765-4321', NULL) |
+----------------------------------------+
```

```
| NULL                                    |
+-----------------------------------------+
```

MULTI_MATCH_ANY

multi_match_any

Description

Syntax

```
TINYINT multi_match_any(VARCHAR haystack, ARRAY<VARCHAR> patterns)
```

Checks whether the string `haystack` matches the regular expressions `patterns` in re2 syntax. returns 0 if none of the regular expressions are matched and 1 if any of the patterns matches.

example

```
mysql> select multi_match_any('Hello, World!', ['hello', '!', 'world']);
+-----------------------------------------------------------+
| multi_match_any('Hello, World!', ['hello', '!', 'world']) |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+

mysql> select multi_match_any('abc', ['A', 'bcd']);
+--------------------------------------+
| multi_match_any('abc', ['A', 'bcd']) |
+--------------------------------------+
| 0                                    |
+--------------------------------------+
```

keywords

```
MULTI_MATCH,MATCH,ANY
```

MULTI_SEARCH_ALL_POSITIONS

multi_search_all_positions

Description

Syntax

```
ARRAY<INT> multi_search_all_positions(VARCHAR haystack, ARRAY<VARCHAR> needles)
```

Returns an `ARRAY` where the `i`-th element is the position of the `i`-th element in `needles`(i.e. `needle`)'s **first** occurrence in the string `haystack`. Positions are counted from 1, with 0 meaning the element was not found. **Case-sensitive**.

example

```
mysql> select multi_search_all_positions('Hello, World!', ['hello', '!', 'world']);
+-----------------------------------------------------------------------+
| multi_search_all_positions('Hello, World!', ['hello', '!', 'world']) |
+-----------------------------------------------------------------------+
| [0,13,0]                                                              |
+-----------------------------------------------------------------------+

select multi_search_all_positions("Hello, World!", ['hello', '!', 'world', 'Hello', 'World']);
+-----------------------------------------------------------------------------------------------+
| multi_search_all_positions('Hello, World!', ARRAY('hello', '!', 'world', 'Hello', 'World')) |
+-----------------------------------------------------------------------------------------------+
| [0, 13, 0, 1, 8]                                                                              |
+-----------------------------------------------------------------------------------------------+
```

keywords

```
MULTI_SEARCH,SEARCH,POSITIONS
```

NGRAM_SEARCH

Description

Calculate the N-gram similarity between `text` and `pattern`. The similarity ranges from 0 to 1, where a higher similarity indicates greater similarity between the two strings.

Both `pattern` and `gram_num` must be constants. If the length of either `text` or `pattern` is less than `gram_num`, return 0.

N-gram similarity is a method for calculating text similarity based on N-grams. An N-gram is a set of continuous N characters or words extracted from a text string. For example, for the string “text” with N=2 (bigram), the bigrams are: {“te”, “ex”, “xt”}.

The N-gram similarity is calculated as:

2 * |Intersection| / (|text set| + |pattern set|)

where |text set| and |pattern set| are the N-grams of `text` and `pattern`, and `Intersection` is the intersection of the two sets.

Note that, by definition, a similarity of 1 does not necessarily mean the two strings are identical.

Only supports ASCII encoding.

Syntax

```
DOUBLE ngram_search(VARCHAR text,VARCHAR pattern,INT gram_num)
```

Example

```
mysql> select ngram_search('123456789' , '12345' , 3);
+--------------------------------------+
| ngram_search('123456789', '12345', 3) |
+--------------------------------------+
|                                  0.6 |
+--------------------------------------+

mysql> select ngram_search("abababab","babababa",2);
+----------------------------------------+
| ngram_search('abababab', 'babababa', 2) |
+----------------------------------------+
|                                      1 |
+----------------------------------------+
```

keywords

```
NGRAM_SEARCH,NGRAM,SEARCH
```

PARSE_URL

Description

The PARSE_URL function is mainly used to parse URL strings and extract various components from them, such as protocols, hosts, paths, query parameters, etc.

Syntax

```
PARSE_URL( <url>, <name> )
```

Parameters

Parameter	Description
<url>	URL that need to be parsed
<name>	The parts to be extracted, and the optional values include PROTOCOL, HOST, PATH, REF, AUTHORITY, FILE, USERINFO, PORT, QUERY (case insensitive).

Return Value

Returns a specified part of <url>. Special cases:

- If any Parameter is NULL, NULL will be returned.
- If <name> is passed with other illegal values, an error will be occurred.

Examples

```
SELECT parse_url ('https://doris.apache.org/', 'HOST');
```

```
+-----------------------------------------------+
| parse_url('https://doris.apache.org/', 'HOST') |
+-----------------------------------------------+
| doris.apache.org                              |
+-----------------------------------------------+
```

```
SELECT parse_url ('https://doris.apache.org/', null);
```

```
+--------------------------------------------+
| parse_url('https://doris.apache.org/', NULL) |
+--------------------------------------------+
| NULL                                       |
+--------------------------------------------+
```

相关命令

如果想获取 QUERY 中的特定 Parameter，可使用extract_url_parameter。

PROTOCOL

Description

The PROTOCOL function is mainly used to extract the protocol part from a URL string.

Syntax

```
PROTOCOL( <url> )
```

Parameters

Parameter	Description
<url>	The URL to be parsed

Return Value

Returns the protocol part of the . Special cases:

- If any of the parameters is NULL, NULL is returned.

Examples

```
SELECT protocol('https://doris.apache.org/');
```

```
+---------------------------------------+
| protocol('https://doris.apache.org/') |
+---------------------------------------+
| https                                 |
+---------------------------------------+
```

```
SELECT protocol(null);
```

```
+----------------+
| protocol(NULL) |
+----------------+
| NULL           |
+----------------+
```

相关命令

If you want to extract other parts of the URL, you can use parse_url。

QUOTE

quote

description

Syntax

```
VARCHAR quote(VARCHAR str)
```

Output all the strings in the argument as is and wrap them with ’’

example

```
mysql> select quote('hello world!\\t');
+-------------------------+
| quote('hello world!\t') |
+-------------------------+
| 'hello world!\t'        |
+-------------------------+
```

keywords

```
QUOTE
```

RANDOM_BYTES

Description

The RANDOM_BYTES function is used to generate a random byte sequence of the specified length.

Syntax

```
RANDOM_BYTES( <len> )
```

Parameters

Parameter	Description
<len>	This parameter specifies the length of the random byte sequence to be generated. This value must be greater than 0; otherwise, an error will be occurred.

Return Value

Returns a random byte sequence of the specified length, encoded in hexadecimal. Special cases:

- If any of the parameters is NULL, NULL will be returned.

Examples

```
select random_bytes(7);
```

```
+------------------+
| random_bytes(7)  |
+------------------+
| 0x869684a082ab4b |
+------------------+
```

```
select random_bytes(-1);
```

```
(1105, 'errCode = 2, detailMessage = (127.0.0.1)[INVALID_ARGUMENT]argument -1 of function random_bytes at row 0 was invalid.')
```

REGEXP

Description

Performs a regular expression match on the string str, returning true if the match succeeds, otherwise false. pattern is the regular expression pattern. It should be noted that when handling character set matching, Utf-8 standard character classes should be used. This ensures that functions can correctly identify and process various characters from different languages.

If the `pattern` is not allowed regexp regular,throw error;

Support character match classes : https://github.com/google/re2/wiki/Syntax

Syntax

```
REGEXP(<str>, <pattern>)
```

Parameters

Parameter	Description
<str>	String type. Represents the string to be matched against the regular expression, which can be a column in a table or a literal string.
<pattern>	String type. The regular expression pattern used to match against the string . Regular expressions provide a powerful way to define complex search patterns, including character classes, quantifiers, and anchors.

Return Value

The REGEXP function returns a BOOLEAN value. If the string matches the regular expression pattern , the function returns true (represented as 1 in SQL); if not, it returns false (represented as 0 in SQL).

Examples

```
CREATE TABLE test ( k1 VARCHAR(255) ) properties("replication_num"="1")

INSERT INTO test (k1) VALUES ('billie eillish'), ('It\'s ok'), ('billie jean'), ('hello world');
```

```
--- Find all data in the k1 column starting with 'billie'
SELECT k1 FROM test WHERE k1 REGEXP '^billie'
--------------

+----------------+
| k1             |
+----------------+
```

```
| billie eillish |
| billie jean    |
+----------------+
2 rows in set (0.02 sec)

--- Find data in the k1 column ending with 'ok':
SELECT k1 FROM test WHERE k1 REGEXP 'ok$'
--------------

+---------+
| k1      |
+---------+
| It's ok |
+---------+
1 row in set (0.03 sec)
```

Chinese Character Example

```
mysql> select regexp('这是一段中文 This is a passage in English 1234567', '\\p{Han}');
+-------------------------------------------------------------------------------+
| ('这是一段中文 This is a passage in English 1234567' regexp '\p{Han}')        |
+-------------------------------------------------------------------------------+
|                                                                             1 |
+-------------------------------------------------------------------------------+
```

Insertion and Testing for Simple String Matching

```
CREATE TABLE test_regexp (
    id INT,
    name VARCHAR(255)
) PROPERTIES("replication_num"="1");

INSERT INTO test_regexp (id, name) VALUES
    (1, 'Alice'),
    (2, 'Bob'),
    (3, 'Charlie'),
    (4, 'David');

-- Find all names starting with 'A'
SELECT id, name FROM test_regexp WHERE name REGEXP '^A';
```

```
+------+-------+
| id   | name  |
+------+-------+
|    1 | Alice |
+------+-------+
```

Special Character Matching Test

```
-- Insert names with special characters
INSERT INTO test_regexp (id, name) VALUES
    (5, 'Anna-Maria'),
    (6, 'John_Doe');

-- Find names containing the '-' character
SELECT id, name FROM test_regexp WHERE name REGEXP '-';
```

```
+------+------------+
| id   | name       |
+------+------------+
|    5 | Anna-Maria |
+------+------------+
```

Test for Ending String Matching

```
-- Find names ending with 'e'
SELECT id, name FROM test_regexp WHERE name REGEXP 'e$';
```

```
+------+---------+
| id   | name    |
+------+---------+
|    1 | Alice   |
|    3 | Charlie |
+------+---------+
```

Emoji test

```
SELECT 'Hello' REGEXP '😀';
```

```
+-----------------------+
| 'Hello' REGEXP '😀'     |
+-----------------------+
|                     0 |
+-----------------------+
```

‘str’ is NULL,return NULL

```
mysql> SELECT REGEXP(NULL, '^billie');
+-------------------------+
| REGEXP(NULL, '^billie') |
+-------------------------+
|                    NULL |
+-------------------------+
```

‘pattern’ is NULL, return NULL

```
mysql> SELECT REGEXP('billie eillish', NULL);
+--------------------------------+
| REGEXP('billie eillish', NULL) |
+--------------------------------+
|                           NULL |
+--------------------------------+
```

All parameters are NULL,return NULL

```
mysql> SELECT REGEXP(NULL, NULL);
+--------------------+
| REGEXP(NULL, NULL) |
+--------------------+
```

```
|              NULL |
+-------------------+
```

If the `pattern` is not allowed regexp regular,throw error;

```
SELECT REGEXP('Hello, World!', '([a-z');
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.2)[INTERNAL_ERROR]Invalid regex expression: ([a-z
```

REGEXP_EXTRACT

Description

This is a function used to perform a regular match on a given string `STR` and extract the `POS`-th matching part that conforms to the specified pattern. For the function to return a matching result, the pattern must exactly match some part of the `STR`.

If no match is found, an empty string will be returned. It should be noted that when handling character set matching, Utf-8 standard character classes should be used. This ensures that functions can correctly identify and process various characters from different languages.

The `str` parameter is of 'string' type, representing the string to be subjected to regular matching. The `pattern` parameter is of 'string' type, representing the target regular expression pattern. The `pos` parameter is of 'integer' type, used to specify the position in the string from which to start searching for the regular expression match. The position starts from 1, and this parameter must be specified.

If the `pattern` is not allowed regexp regular,throw error;

Support character match classes : https://github.com/google/re2/wiki/Syntax

Syntax

```
REGEXP_EXTRACT(<str>, <pattern>, <pos>)
```

Parameters

Parameter	Description
`<str>`	The column that needs to undergo regular matching. It is of 'string' type.
`<pattern>`	The target regular expression pattern. It is of 'string' type.
`<pos>`	The parameter used to specify the position in the string from which to start searching for the regular expression match. It is an integer value representing the character position in the string (starting from 1). `pos` must be specified.

Return Value

The matching part of the pattern. It is of Varchar type. If no match is found, an empty string will be returned.

Example

Extract the first matching part.In this example, the regular expression ([[:lower:]]+)C([[:lower:]]+) matches the part of the string where one or more lowercase letters are followed by 'C' and then one or more lowercase

letters. The first capturing group ([[:lower:]]+) before ‘C’ matches ‘b’, so the result is ‘b’.

```
mysql> SELECT regexp_extract('AbCdE', '([[:lower:]]+)C([[:lower:]]+)', 1);
+-----------------------------------------------------------+
| regexp_extract('AbCdE', '([[:lower:]]+)C([[:lower:]]+)', 1) |
+-----------------------------------------------------------+
| b                                                         |
+-----------------------------------------------------------+
```

Extract the second matching part.Here, the second capturing group ([[:lower:]]+) after ‘C’ matches ‘d’, so the result is ‘d’.

```
mysql> SELECT regexp_extract('AbCdE', '([[:lower:]]+)C([[:lower:]]+)', 2);
+-----------------------------------------------------------+
| regexp_extract('AbCdE', '([[:lower:]]+)C([[:lower:]]+)', 2) |
+-----------------------------------------------------------+
| d                                                         |
+-----------------------------------------------------------+
```

Match Chinese characters.The pattern (+)(.+) first matches one or more Chinese characters (+) and then matches the remaining part of the string ((.+)). The second capturing group matches the non - Chinese part of the string, so the result is ‘This is a passage in English 1234567’.

```
mysql> select regexp_extract('这是一段中文 This is a passage in English 1234567', '(\\p{Han}+)(.+)', 2);
+-----------------------------------------------------------------------------------------------+
| regexp_extract('这是一段中文 This is a passage in English 1234567', '(\p{Han}+)(.+)', 2)       |
+-----------------------------------------------------------------------------------------------+
| This is a passage in English 1234567                                                          |
+-----------------------------------------------------------------------------------------------+
```

Insert variable values and perform matching.This example inserts data into a table and then uses the REGEXP_EXTRACT function to extract matching parts from the stored strings based on the stored patterns and positions.

```
CREATE TABLE test_table_for_regexp_extract (
        id INT,
        text_data VARCHAR(500),
        pattern VARCHAR(100),
        pos INT
    ) PROPERTIES ("replication_num"="1");

INSERT INTO test_table_for_regexp_extract VALUES
    (1, 'AbCdE', '([[:lower:]]+)C([[:lower:]]+)', 1),
    (2, 'AbCdE', '([[:lower:]]+)C([[:lower:]]+)', 2),
    (3, '这是一段中文 This is a passage in English 1234567', '(\\p{Han}+)(.+)', 2);

SELECT id, regexp_extract(text_data, pattern, pos) as extract_result FROM test_table_for_regexp_extract ORDER BY id;
```

```
+------+----------------+
| id   | extract_result |
+------+----------------+
|    1 | b              |
|    2 | d              |
```

```
|    3 | This is a passage in English 1234567 |
+------+---------------+
```

Test with a pattern that has no match.Since the pattern ([[:digit:]]+) (one or more digits) does not match any part of the string ‘AbCdE’, an empty string is returned.

```
SELECT regexp_extract('AbCdE', '([[:digit:]]+)', 1);
```

```
+-----------------------------------------------+
| regexp_extract('AbCdE', '([[:digit:]]+)', 1)  |
+-----------------------------------------------+
|                                               |
+-----------------------------------------------+
```

Emoji test case

```
SELECT regexp_extract('Text 😊 More 😀', '😊|😀',0);
```

```
+-----------------------------------------------------+
| regexp_extract('Text 😊 More 😀', '😊|😀',0)                |
+-----------------------------------------------------+
| 😊                                                   |
+-----------------------------------------------------+
```

‘str’ is NULL, return NULL

```
mysql> SELECT REGEXP_EXTRACT(NULL, '([a-z]+)', 1);
+-------------------------------------+
| REGEXP_EXTRACT(NULL, '([a-z]+)', 1) |
+-------------------------------------+
| NULL                                |
+-------------------------------------+
```

‘pattern’ is NULL,return NULL

```
mysql> SELECT REGEXP_EXTRACT('Hello World', NULL, 1);
+---------------------------------------+
| REGEXP_EXTRACT('Hello World', NULL, 1) |
+---------------------------------------+
| NULL                                  |
+---------------------------------------+
```

‘pos’ is NULL,return NULL

```
mysql> SELECT REGEXP_EXTRACT('Hello World', '([a-z]+)', NULL);
+-------------------------------------------------+
| REGEXP_EXTRACT('Hello World', '([a-z]+)', NULL) |
+-------------------------------------------------+
| NULL                                            |
+-------------------------------------------------+
```

All parameters are NULL,return NULL

```
mysql> SELECT REGEXP_EXTRACT(NULL, NULL, NULL);
+---------------------------------+
| REGEXP_EXTRACT(NULL, NULL, NULL) |
+---------------------------------+
| NULL                            |
+---------------------------------+
```

If the `pattern` is not allowed regexp regular,throw error;

```
SELECT regexp_extract('AbCdE', '([[:digit:]]+', 1);
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.2)[INVALID_ARGUMENT]Could not compile regexp pattern: ([[:digit:]]+
Error: missing ): ([[:digit:]]+
```

REGEXP_EXTRACT_ALL

Description

The `REGEXP_EXTRACT_ALL` function is used to perform a regular expression match on a given string `str` and extract all the parts that match the first sub - pattern of the specified `pattern`. For the function to return an array of strings representing the matched parts of the pattern, the pattern must exactly match a portion of the input string `str`. If there is no match, or if the pattern does not contain any sub - patterns, an empty string is returned.

It should be noted that when handling character set matching, Utf-8 standard character classes should be used. This ensures that functions can correctly identify and process various characters from different languages.

If the ‘pattern’ is not allowed regexp regular, throw error;

Support character match classes : https://github.com/google/re2/wiki/Syntax

Syntax

```
REGEXP_EXTRACT_ALL(<str>, <pattern>)
```

Parameters

Parameter	Description
`<str>`	This parameter is of type String. It represents the input string on which the regular expression matching will be performed. This can be a literal string value or a reference to a column in a table that contains string data.
`<pattern>`	This parameter is also of type String. It specifies the regular expression pattern that will be used to match against the input string . The pattern can include various regular expression constructs such as character classes, quantifiers, and sub - patterns.

Return value

The function returns an array of strings that represent the parts of the input string that match the first sub - pattern of the specified regular expression. The return type is an array of String values. If no matches are found, or if the pattern has no sub - patterns, an empty array is returned.

Example

Basic matching of lowercase letters around ‘C’.In this example, the pattern ([[:lower:]]+)C([[:lower:]]+) matches the part of the string where one or more lowercase letters are followed by ‘C’ and then one or more lowercase letters. The first sub - pattern ([[:lower:]]+) before ‘C’ matches ‘b’, so the result is [‘b’].

```
mysql> SELECT regexp_extract_all('AbCdE', '([[:lower:]]+)C([[:lower:]]+)');
+-----------------------------------------------------------+
| regexp_extract_all('AbCdE', '([[:lower:]]+)C([[:lower:]]+)') |
+-----------------------------------------------------------+
| ['b']                                                     |
+-----------------------------------------------------------+
```

Multiple matches in a string. Here, the pattern matches two parts in the string. The first match has the first sub - pattern matching ‘b’, and the second match has the first sub - pattern matching ‘f’. So the result is [‘b’, ‘f’].

```
mysql> SELECT regexp_extract_all('AbCdEfCg', '([[:lower:]]+)C([[:lower:]]+)');
+-----------------------------------------------------------+
| regexp_extract_all('AbCdEfCg', '([[:lower:]]+)C([[:lower:]]+)') |
+-----------------------------------------------------------+
| ['b','f']                                                 |
+-----------------------------------------------------------+
```

Extracting keys from key - value pairs.The pattern matches key - value pairs in the string. The first sub - pattern captures the keys, so the result is an array of the keys [‘abc’, ‘def’, ‘ghi’].

```
mysql> SELECT regexp_extract_all('abc=111, def=222, ghi=333','("[^"]+"|\\w+)=("[^"]+"|\\w+)');
+----------------------------------------------------------------------------+
| regexp_extract_all('abc=111, def=222, ghi=333', '("[^"]+"|\w+)=("[^"]+"|\w+)') |
+----------------------------------------------------------------------------+
| ['abc','def','ghi']                                                        |
+----------------------------------------------------------------------------+
```

Matching Chinese characters.The pattern (+)(.+) first matches one or more Chinese characters with the first sub - pattern (+), so the result is [‘这是一段中文’].

```
mysql> select regexp_extract_all('这是一段中文 This is a passage in English 1234567', '(\\p{Han}+)(.+)');
+-----------------------------------------------------------------------------------------------+
| regexp_extract_all('这是一段中文 This is a passage in English 1234567', '(\p{Han}+)(.+)')       |
+-----------------------------------------------------------------------------------------------+
| ['这是一段中文']                                                                                |
+-----------------------------------------------------------------------------------------------+
```

Inserting data and using REGEXP_EXTRACT_ALL

```
CREATE TABLE test_regexp_extract_all (
    id INT,
    text_content VARCHAR(255),
    pattern VARCHAR(255)
) PROPERTIES ("replication_num"="1");


INSERT INTO test_regexp_extract_all VALUES
(1, 'apple1, banana2, cherry3', '([a-zA-Z]+)\\d'),
(2, 'red#123, blue#456, green#789', '([a-zA-Z]+)#\\d+'),
(3, 'hello@example.com, world@test.net', '([a-zA-Z]+)@');


SELECT id, regexp_extract_all(text_content, pattern) AS extracted_data
FROM test_regexp_extract_all;
```

```
+------+----------------------+
| id   | extracted_data       |
+------+----------------------+
|    1 | ['apple', 'banana', 'cherry'] |
|    2 | ['red', 'blue', 'green']     |
|    3 | ['hello', 'world']           |
+------+----------------------+
```

No matched,return empty string

```
SELECT REGEXP_EXTRACT_ALL('ABC', '(\\d+)');
```

```
+-------------------------------------+
| REGEXP_EXTRACT_ALL('ABC', '(\\d+)') |
+-------------------------------------+
|                                     |
+-------------------------------------+
```

emoji match

```
mysql> SELECT REGEXP_EXTRACT_ALL('👩‍🚀,👩', '(💻|🚀)');
+-----------------------------------------------------------+
| REGEXP_EXTRACT_ALL('👩‍🚀,👩', '(💻|🚀)')                    |
+-----------------------------------------------------------+
| ['💻','🚀']                                                |
+-----------------------------------------------------------+
```

‘Str’ is NULL,return NULL

```
SELECT regexp_extract_all(NULL, '([a-z]+)');
```

```
+--------------------------------------+
| regexp_extract_all(NULL, '([a-z]+)') |
+--------------------------------------+
| NULL                                 |
+--------------------------------------+
```

‘pattern’ is NULL, return NULL

```
SELECT regexp_extract_all('Hello World', NULL);
```

```
+----------------------------------------+
| regexp_extract_all('Hello World', NULL) |
+----------------------------------------+
| NULL                                   |
+----------------------------------------+
```

All parameters are NULL,return NULL

```
SELECT regexp_extract_all(NULL,NULL);
```

```
+------------------------------+
| regexp_extract_all(NULL,NULL) |
+------------------------------+
| NULL                         |
+------------------------------+
```

If the `pattern` is not allowed regexp regular,throw error;

```
SELECT regexp_extract_all('hello (world) 123', '([[:alpha:]+');
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.2)[INVALID_ARGUMENT]Could not compile regexp pattern: ([[:alpha:]+
Error: missing ]: [[:alpha:]+
```

REGEXP_EXTRACT_OR_NULL

Description

Extract the first substring that matches the target regular expression pattern from the text string, and extract a specific group from it based on the expression group index.

It should be noted that when handling character set matching, Utf-8 standard character classes should be used. This ensures that functions can correctly identify and process various characters from different languages.

> **Support since Apache Doris 3.0.2**

If the 'pattern' is not allowed regexp regular,throw error

Support character match classes : https://github.com/google/re2/wiki/Syntax

Syntax

```
REGEXP_EXTRACT_OR_NULL(<str>, <pattern>, <pos>)
```

Parameters

Parameter	Description
<str>	A string parameter. It represents the text string in which the regular expression matching will be performed. This string can contain any combination of characters, and the function will search within it for substrings that match the .
<pattern>	A string parameter. It is the target regular expression pattern. This pattern can include various regular expression metacharacters and character classes, which precisely define the rules for the substring to be matched
<pos>	An integer parameter. It indicates the index of the expression group to be extracted. The indexing starts from 1. If is set to 0, the entire first matching substring will be returned. If is a negative number or exceeds the number of expression groups in the pattern, the function will return NULL.

Return Value

Return a string type, with the result being the part that matches <pattern>.

If the input <pos> is 0, return the entire first matching substring. If the input <pos> is invalid (negative or exceeds the number of expression groups), return NULL. If the regular expression fails to match, return NULL. If the

<pos> < 0,return NULL; If the pos > the length of <str>,return NULL;

Example

Extracting a specific group from a match. Explanation: The regular expression ([[:lower:]]+)C([[:lower:]]+) looks for sequences of one or more lowercase letters separated by ‘C’. The group with index 1 corresponds to the first sequence of lowercase letters, so ‘b’ is returned.

```
SELECT REGEXP_EXTRACT_OR_NULL('123AbCdExCx', '([[:lower:]]+)C([[:lower:]]+)', 1);
```

```
+-------------------------------------------------------------------------+
| REGEXP_EXTRACT_OR_NULL('123AbCdExCx', '([[:lower:]]+)C([[:lower:]]+)', 1) |
+-------------------------------------------------------------------------+
| b                                                                       |
+-------------------------------------------------------------------------+
```

Returning the entire matched substring.When is 0, the whole first substring that matches the pattern is returned.

```
SELECT REGEXP_EXTRACT_OR_NULL('123AbCdExCx', '([[:lower:]]+)C([[:lower:]]+)', 0);
```

```
+-------------------------------------------------------------------------+
| REGEXP_EXTRACT_OR_NULL('123AbCdExCx', '([[:lower:]]+)C([[:lower:]]+)', 0) |
+-------------------------------------------------------------------------+
| bCd                                                                     |
+-------------------------------------------------------------------------+
```

An invalid group index.Since the pattern has only 2 groups, an index of 5 is out of range, so NULL is returned.

```
SELECT REGEXP_EXTRACT_OR_NULL('123AbCdExCx', '([[:lower:]]+)C([[:lower:]]+)', 5);
```

```
+-------------------------------------------------------------------------+
| REGEXP_EXTRACT_OR_NULL('123AbCdExCx', '([[:lower:]]+)C([[:lower:]]+)', 5) |
+-------------------------------------------------------------------------+
| NULL                                                                    |
+-------------------------------------------------------------------------+
```

A non - matching regular expression.There is no part of the string ‘AbCdE’ that fully matches the pattern

```
SELECT REGEXP_EXTRACT_OR_NULL('AbCdE', '([[:lower:]]+)C([[:upper:]]+)', 1);
```

```
+-----------------------------------------------------------------+
| REGEXP_EXTRACT_OR_NULL('AbCdE', '([[:lower:]]+)C([[:upper:]]+)', 1) |
+-----------------------------------------------------------------+
| NULL                                                            |
+-----------------------------------------------------------------+
```

Chinese character matching.The pattern (+)(.+) first matches one or more Chinese characters and then any remaining characters. The group with index 2 represents the non - Chinese part of the string after the Chinese

characters.

```
select REGEXP_EXTRACT_OR_NULL('这是一段中文 This is a passage in English 1234567', '(\\p{Han}+)(.+)', 2);
```

```
+-----------------------------------------------------------------------------------------------------+
| REGEXP_EXTRACT_OR_NULL('这是一段中文 This is a passage in English 1234567', '(\\p{Han}+)(.+)', 2)       |
+-----------------------------------------------------------------------------------------------------+
|  This is a passage in English 1234567                                                               |
+-----------------------------------------------------------------------------------------------------+
```

Inserting data into a table and performing extraction

```
CREATE TABLE test_regexp_extract_or_null (
    id INT,
    text_column VARCHAR(255),
    pattern_column VARCHAR(255),
    position_column INT
) PROPERTIES ("replication_num"="1");

INSERT INTO test_regexp_extract_or_null VALUES
(1, 'abc123def', '([a-z]+)([0-9]+)([a-z]+)', 2),
(2, 'Hello World', '([A-Z][a-z]+) ([A-Z][a-z]+)', 0),
(3, '123-456-789', '([0-9]{3})-([0-9]{3})-([0-9]{3})', 3),
(4, 'example@example.com', '([a-z]+)@([a-z]+)\\.([a-z]+)', 1),
(5, '测试文本 test text', '(\\p{Han}+) (.+)', 1);

SELECT id, REGEXP_EXTRACT_OR_NULL(text_column, pattern_column, position_column) AS extracted_result
FROM test_regexp_extract_or_null
ORDER BY id;
```

```
+------+-----------------+
| id   | extracted_result|
+------+-----------------+
|    1 | 123             |
|    2 | Hello World     |
|    3 | 789             |
|    4 | example         |
|    5 | 测试文本          |
+------+-----------------+
```

Emoji case

```
SELECT regexp_extract_or_null('😀😊😎', '😀|😊|😎',0);
```

```
+-----------------------------------------------------------+
| regexp_extract_or_null('😀😊😎', '😀|😊|😎',0)           |
+-----------------------------------------------------------+
| 😀                                                         |
+-----------------------------------------------------------+
```

‘str’ is NULL,return NULL

```
SELECT REGEXP_EXTRACT_OR_NULL(NULL, '([a-z]+)', 1);
```

```
+-------------------------------------------+
| REGEXP_EXTRACT_OR_NULL(NULL, '([a-z]+)', 1) |
+-------------------------------------------+
| NULL                                      |
+-------------------------------------------+
```

‘pattern’ is NULL,return NULL

```
SELECT REGEXP_EXTRACT_OR_NULL('Hello World', NULL, 1);
```

```
+-----------------------------------------------+
| REGEXP_EXTRACT_OR_NULL('Hello World', NULL, 1) |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+
```

‘pos’ is NULL ,return NULL

```
SELECT REGEXP_EXTRACT_OR_NULL('Hello World', '([a-z]+)', NULL);
```

```
+-------------------------------------------------------+
| REGEXP_EXTRACT_OR_NULL('Hello World', '([a-z]+)', NULL) |
+-------------------------------------------------------+
| NULL                                                  |
+-------------------------------------------------------+
```

ALL parameters are NULL,return NULL;

```
SELECT REGEXP_EXTRACT_OR_NULL(NULL,NULL,NULL);
```

```
+---------------------------------------+
| REGEXP_EXTRACT_OR_NULL(NULL,NULL,NULL) |
+---------------------------------------+
| NULL                                  |
+---------------------------------------+
```

If the `pattern` is not allowed regexp regular,throw error;

```
mysql> SELECT REGEXP_EXTRACT_OR_NULL('123AbCdExCx', '([[:lower:]]+)C([[]ower:]]+)', 1);
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.2)[INVALID_ARGUMENT]Could not compile regexp pattern:
([[:lower:]]+)C([[:lower:]+)
Error: missing ]: [[:lower:]+)
```

REGEXP_REPLACE

Description

Regular matching of STR strings, replacing the part hitting pattern with a new string.

It should be noted that when handling character set matching, Utf-8 standard character classes should be used. This ensures that functions can correctly identify and process various characters from different languages.

If the `pattern` is not allowed regexp regular,throw error;

Support character match classes : https://github.com/google/re2/wiki/Syntax

Syntax

```
REGEXP_REPLACE(<str>, <pattern>, <repl>)
```

Parameters

Parameter	Description
<str>	This parameter is of Varchar type. It represents the string on which the regular expression matching will be performed. It can be a literal string or a column from a table containing string values.
<pattern>	This parameter is of Varchar type. It is the regular expression pattern used to match the string. The pattern can include various regular expression metacharacters and constructs to define complex matching rules.
<repl>	This parameter is of Varchar type. It is the string that will replace the parts of the that match the . If you want to reference captured groups in the pattern, you can use backreferences like \1, \2, etc., where \1 refers to the first captured group, \2 refers to the second captured group, and so on.

Return Value

The function returns the result string after the replacement operation. The return value is of Varchar type. If no part of the matches the , the original will be returned.

Example

Basic replacement example.In this example, all spaces in the string ‘a b c’ are replaced with hyphens.

```
mysql> SELECT regexp_replace('a b c', ' ', '-');
+-----------------------------------+
| regexp_replace('a b c', ' ', '-') |
```

```
+----------------------------------+
| a-b-c                            |
+----------------------------------+
```

Using captured groups.Here, the character ‘b’ is captured by the group (b) in the pattern, and then it is replaced with **using the backreference \1 in the replacement string.**

```
mysql> SELECT regexp_replace('a b c', '(b)', '<\\1>');
+----------------------------------------+
| regexp_replace('a b c', '(b)', '<\1>') |
+----------------------------------------+
| a <b> c                                |
+----------------------------------------+
```

Matching Chinese characters.This example replaces all consecutive Chinese characters in the string with ‘123’.

```
mysql> select regexp_replace('这是一段中文 This is a passage in English 1234567', '\\p{Han}+', '123');
+-----------------------------------------------------------------------------------------------+
| regexp_replace('这是一段中文 This is a passage in English 1234567', '\p{Han}+', '123')       |
+-----------------------------------------------------------------------------------------------+
| 123This is a passage in English 1234567                                                       |
+-----------------------------------------------------------------------------------------------+
```

Insert and test cases.In this set of test cases, we create a table to store original strings, patterns, and replacement strings. Then we insert various test data and perform REGEXP_REPLACE operations on the original strings using the corresponding patterns and replacement strings. Finally, we retrieve and display the replaced strings.

```
-- Create a table to store test data
CREATE TABLE test_table_for_regexp_replace (
        id INT,
        original_text VARCHAR(500),
        pattern VARCHAR(100),
        replacement VARCHAR(100)
    ) PROPERTIES ("replication_num"="1");

-- Insert test data
INSERT INTO test_table_for_regexp_replace VALUES
    (1, 'Hello, World!', ',', '-'),
    (2, 'apple123', '[0-9]', '*'),
    (3, 'aabbcc', '(aa|bb|cc)', 'XX'),
    (4, '123-456-7890', '-', ' '),
    (5, 'test,data', ',', ';'),
    (6, 'a1b2c3', '[0-9]', '#'),
    (7, 'book keeper', 'oo|ee', '**'),
    (8, 'ababab', '(ab)', 'XY'),
    (9, 'aabbcc', '(aa|bb|cc)', 'ZZ'),
    (10, 'apple,banana', ',', ' - ');

-- Perform replacement operations on the inserted data
SELECT id, regexp_replace(original_text, pattern, replacement) as replaced_text FROM test_table_for_regexp_replace ORDER BY id;
```

```
+------+------------------+
| id   | replaced_text    |
+------+------------------+
|    1 | Hello- World!    |
```

```
|    2 | apple***         |
|    3 | XXXXYY           |
|    4 | 123 456 7890     |
|    5 | test;data        |
|    6 | a#b#c#           |
|    7 | b**k k**per      |
|    8 | XYXYXY           |
|    9 | ZZZZYY           |
|   10 | apple - banana   |
+------+------------------+
```

Emoji replace

```
SELECT regexp_replace('🌍 Earth 🍔 Food', '🌍|🍔', '*');
```

```
+-----------------------------------------------------+
| regexp_replace('🌍 Earth 🍔 Food', '🌍|🍔', '*')               |
+-----------------------------------------------------+
| * Earth * Food                                      |
+-----------------------------------------------------+
```

‘str’ is NULL,return NULL

```
mysql> SELECT REGEXP_REPLACE(NULL, ' ', '-');
+--------------------------------+
| REGEXP_REPLACE(NULL, ' ', '-') |
+--------------------------------+
| NULL                           |
+--------------------------------+
```

‘pattern’ is NULL,return NULL

```
mysql> SELECT REGEXP_REPLACE('Hello World', NULL, '-');
+---------------------------------------+
| REGEXP_REPLACE('Hello World', NULL, '-') |
+---------------------------------------+
| NULL                                  |
+---------------------------------------+
```

‘repl’ is NULL,return NULL

```
mysql> SELECT REGEXP_REPLACE('Hello World', ' ', NULL);
+---------------------------------------+
| REGEXP_REPLACE('Hello World', ' ', NULL) |
+---------------------------------------+
| NULL                                  |
+---------------------------------------+
```

All parameters are NULL,return NULL

```
mysql> SELECT REGEXP_REPLACE(NULL, NULL, NULL);
+----------------------------------+
| REGEXP_REPLACE(NULL, NULL, NULL) |
+----------------------------------+
```

```
| NULL                             |
+----------------------------------+
```

If the `pattern` is not allowed regexp regular,throw error;

```
SELECT regexp_replace('a b c', '(b', '<\\1>');
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.2)[INVALID_ARGUMENT]Could not compile regexp pattern: (b
Error: missing ): (b
```

REGEXP_REPLACE_ONE

Description

The `REGEXP_REPLACE_ONE` function is a powerful tool designed to perform regular expression matching on a given string. It allows you to find and replace the first occurrence of a specific pattern within the string.

When working with text data, you often need to manipulate strings based on certain rules. Regular expressions provide a flexible and efficient way to define these rules. This function takes a string (`str`), a regular expression pattern (`pattern`), and a replacement string (`repl`). It then searches for the first part of the `str` that matches the `pattern` and substitutes it with the `repl`.

It should be noted that when handling character set matching, Utf-8 standard character classes should be used. This ensures that functions can correctly identify and process various characters from different languages.

If the `pattern` is not allowed regexp regular,throw error;

Support character match classes : https://github.com/google/re2/wiki/Syntax

Syntax

```
REGEXP_REPLACE_ONE(<str>, <pattern>, <repl>)
```

Parameters

Parameter	Description
`<str>`	This parameter is of type string. It represents the string on which the regular expression matching will be performed. This is the target string that you want to modify.
`<pattern>`	This parameter is also of type string. It is a regular expression pattern. The function will search for the first occurrence of this pattern within the string.
`<repl>`	This is a string parameter as well. It contains the string that will replace the first part of that matches the .

Return Value

The function returns the result string after the replacement operation. The return type is Varchar. If no part of the matches the , the original will be returned.

Example

Replace the first space with a hyphen Explanation: In this example, the input string is ‘a b c’, the regular expression pattern is a single space ’ ‘, and the replacement string is a hyphen’-‘. The function searches for the

first occurrence of a space in the string ’a b c’ and replaces it with a hyphen. So the output is ‘a-b c’.

```
mysql> SELECT regexp_replace_one('a b c', ' ', '-');

+-----------------------------------+
| regexp_replace_one('a b c', ' ', '-') |
+-----------------------------------+
| a-b c                             |
+-----------------------------------+
```

Replace the first matched group.Here, the input string is ‘a b b’, the regular expression pattern is ‘(b)’, which is a capturing group that matches the character ‘b’. The replacement string is ‘<\1>’, where \1 refers to the first capturing group (in this case, the matched ‘b’). The function finds the first occurrence of ‘b’ in the string ‘a b b’ and replaces it with ‘**’. Thus, the output is ‘a b’.**

```
mysql> SELECT regexp_replace_one('a b b', '(b)', '<\\1>');
+---------------------------------------+
| regexp_replace_one('a b b', '(b)', '<\1>') |
+---------------------------------------+
| a <b> b                               |
+---------------------------------------+
```

Replace the first Chinese character.The input string is a long string containing Chinese characters and English text. The regular expression pattern is ‘’, which is a Unicode character class that matches any Chinese character. The replacement string is ‘123’. The function searches for the first Chinese character in the string and replaces it with ‘123’. So the output is ‘123是一段中文This is a passage in English 1234567’.

```
mysql> select regexp_replace_one('这是一段中文 This is a passage in English 1234567', '\\p{Han}', '123');
+-----------------------------------------------------------------------------------------------+
| regexp_replace_one('这是一段中文 This is a passage in English 1234567', '\p{Han}', '123')       |
+-----------------------------------------------------------------------------------------------+
| 123是一段中文This is a passage in English 1234567                                              |
+-----------------------------------------------------------------------------------------------+
```

Insert data into a table and perform replacementFirst, a table named test_table_for_regexp_replace_one is created with four columns: id (an integer), text_data (a string where the replacement will be performed), pattern (the regular expression pattern for matching), and repl (the replacement string).Then, ten rows of data are inserted into the table, each containing different values for the four columns.Finally, a SELECT statement is used to query the table. For each row, the REGEXP_REPLACE_ONE function is applied to the text_data column using the corresponding pattern and repl values. The result of the replacement is aliased as replaced_result. The rows are ordered by the id column.

```
CREATE TABLE test_table_for_regexp_replace_one (
        id INT,
        text_data VARCHAR(500),
        pattern VARCHAR(100),
        repl VARCHAR(100)
    ) PROPERTIES ("replication_num"="1");

INSERT INTO test_table_for_regexp_replace_one VALUES
    (1, 'Hello World', ' ', '-'),
```

```
    (2, 'apple123', '[0-9]', 'X'),
    (3, 'aabbcc', '(aa)', 'AA'),
    (4, '123-456-7890', '[0-9][0-9][0-9]', 'XXX'),
    (5, 'test,data', ',', ';'),
    (6, 'a1b2c3', '[a-z][0-9]', 'X'),
    (7, 'book keeper', 'oo', 'OO'),
    (8, 'ababab', '(ab)', 'AB'),
    (9, 'aabbcc', '(bb)', 'BB'),
    (10, 'apple,banana', '[aeiou]', 'X');

SELECT id, regexp_replace_one(text_data, pattern, repl) as replaced_result FROM test_table_for_regexp_replace_one ORDER BY id;
```

```
+------+----------------+
| id   | replaced_result |
+------+----------------+
|    1 | Hello-World     |
|    2 | appleX23        |
|    3 | AAbbcc          |
|    4 | XXX-456-7890    |
|    5 | test;data       |
|    6 | Xb2c3           |
|    7 | BOOk keeper     |
|    8 | ABabab          |
|    9 | aaBBcc          |
|   10 | Xpple,banana    |
+------+----------------+
```

Emoji one replace case

```
SELECT regexp_replace_one('😀😊😀', '😀|😊', '[SMILE]');
```

```
+-----------------------------------------------------------+
| regexp_replace_one('😀😊😀', '😀|😊', '[SMILE]')                    |
+-----------------------------------------------------------+
| [SMILE]😊😀                                                  |
+-----------------------------------------------------------+
```

‘str’ is NULL,return NULL

```
mysql> SELECT REGEXP_REPLACE_ONE(NULL, ' ', '-');
+-----------------------------------+
| REGEXP_REPLACE_ONE(NULL, ' ', '-') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
```

‘pattern’ is NULL,return NULL

```
mysql> SELECT REGEXP_REPLACE_ONE('Hello World', NULL, '-');
+-------------------------------------------+
| REGEXP_REPLACE_ONE('Hello World', NULL, '-') |
+-------------------------------------------+
| NULL                                      |
+-------------------------------------------+
```

‘repl’ is NULL return NULL

```
mysql> SELECT REGEXP_REPLACE_ONE('Hello World', ' ', NULL);
+-----------------------------------------------+
| REGEXP_REPLACE_ONE('Hello World', ' ', NULL) |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+
```

All parameters are NULL,return NULL

```
mysql> SELECT REGEXP_REPLACE_ONE(NULL, NULL, NULL);
+--------------------------------------+
| REGEXP_REPLACE_ONE(NULL, NULL, NULL) |
+--------------------------------------+
| NULL                                 |
+--------------------------------------+
```

If the `pattern` is not allowed regexp regular,throw error

```
SELECT regexp_replace_one('a b b', '(b', '<\\1>');
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (10.16.10.2)[INVALID_ARGUMENT]Could not compile regexp pattern: (b
Error: missing ): (b
```

REPEAT

Description

The REPEAT function is used to repeat a string a specified number of times.

Syntax

```
REPEAT( <str>, <count> )
```

Parameters

Parameter	Description
<str>	The string to be repeated.
<count>	The number of times to repeat. It must be a non-negative integer. If it is less than 1, an empty string will be returned.

> **The repeat function defaults to a maximum of 10,000 repetitions. If this number is exceeded, an error will be reported. The limit can be adjusted through a session variable:**
>
> ```
> set repeat_max_num = 20000
> ```

Return Value

Returns the string repeated the specified number of times. Special cases:

- If any Parameter is NULL, NULL will be returned.

Examples

```
SELECT repeat("a", 3);
```

```
+----------------+
| repeat('a', 3) |
+----------------+
| aaa            |
+----------------+
```

```
SELECT repeat("a", -1);
```

```
+-----------------+
| repeat('a', -1) |
+-----------------+
|                 |
+-----------------+
```

REPLACE

Description

The REPLACE function is used to replace a part of characters in a string with other characters.

Syntax

```
REPLACE ( <str>, <old>, <new> )
```

Parameters

Parameter	Description
<str>	The string that needs to be replaced.
<old>	The substring that needs to be replaced. If <old> is not in <str>, no replacement will be made.
<new>	The new substring used to replace <old>.

Return Value

Returns the new string after replacing the substring. Special cases:

- If any Parameter is NULL, NULL will be returned.

Examples

```
SELECT replace('hello world', 'world', 'universe');
```

```
+-------------------------------------------+
| replace('hello world', 'world', 'universe') |
+-------------------------------------------+
| hello universe                            |
+-------------------------------------------+
```

REPLACE_EMPTY

Description

The REPLACE_EMPTY function is used to replace a part of the characters in a string with other characters. Unlike the REPLACE function, when old is an empty string, the new string will be inserted before each character of the str string and at the end of the str string.

Apart from this, all other behaviors are exactly the same as the REPLACE() function.

This function is mainly used to be compatible with Presto and Trino, and its behavior is exactly the same as the `REPLACE()` function in Presto and Trino.

Supported since version 2.1.5.

Syntax

```
REPLACE_EMPTY ( <str>, <old>, <new> )
```

Parameters

Parameter	Description
<str>	The string that needs to be replaced.
<old>	The substring that needs to be replaced. If <old> is not in <str>, no replacement will be performed. If <old> is an empty string, the <new> string will be inserted before each character of the str string.
<new>	The new substring used to replace <old>.

Return Value

Returns the new string after replacing the substring. Special cases:

- If any Parameter is NULL, NULL will be returned.
- If <old> is an empty string, the string with the <new> string inserted before each character of the <str> string will be returned.

Examples

```
SELECT replace('hello world', 'world', 'universe');
```

```
+-------------------------------------------+
| replace('hello world', 'world', 'universe') |
+-------------------------------------------+
| hello universe                            |
+-------------------------------------------+
```

```
SELECT replace_empty("abc", '', 'xyz');
```

```
+--------------------------------+
| replace_empty('abc', '', 'xyz') |
+--------------------------------+
| xyzaxyzbxyzcxyz                |
+--------------------------------+
```

```
SELECT replace_empty("", "", "xyz");
```

```
+-----------------------------+
| replace_empty('', '', 'xyz') |
+-----------------------------+
| xyz                         |
+-----------------------------+
```

REVERSE

Description

The REVERSE function is used to reverse the order of characters in a string or the order of elements in an array.

Syntax

```
REVERSE( <seq> )
```

Parameters

Parameter	Description
<seq>	The string or array whose order needs to be reversed.

Return Value

Returns the string or array with the reversed order. Special cases:

- If any Parameter is NULL, NULL will be returned.

Examples

```
SELECT reverse('hello');
```

```
+------------------+
| REVERSE('hello') |
+------------------+
| olleh            |
+------------------+
```

```
SELECT reverse(['hello', 'world']);
```

```
+-----------------------------+
| reverse(['hello', 'world']) |
+-----------------------------+
| ["world", "hello"]          |
+-----------------------------+
```

RPAD

Description

Used to pad the specified character on the right side of the original string until met the specified length.

Syntax

```
RPAD ( <str> , <len> , <pad>)
```

Parameters

Parameter	Description
<str>	The string to be padded.
<len>	The total length of the final result string, which represents character length rather than the byte length.
<pad>	The string used for padding.

> **The maximum value of the <len> parameter is 10000. If this limit is exceeded, an error will be occurred. You can adjust the limit by setting the session variable:**
>
> ```
> set repeat_max_num = 20000
> ```

Return Value

Returns the padded string. Special cases:

- If any Parameter is NULL, NULL will be returned.
- If <pad> is empty and <len> is greater than the length of <str>, the return value is an empty string.
- If <len> is less than the length of <str>, the string obtained by truncating <str> to <len> is returned.
- If <len> is less than 0, the return value is NULL.

Examples

```
SELECT rpad('hello', 1, '');
```

```
+----------------------+
| rpad('hello', 1, '') |
+----------------------+
| h                    |
+----------------------+
```

```
SELECT rpad('hello', 10, 'world');
```

```
+---------------------------+
| rpad('hello', 10, 'world') |
+---------------------------+
| helloworld                |
+---------------------------+
```

```
SELECT rpad('hello', 10, '');
```

```
+----------------------+
| rpad('hello', 10, '') |
+----------------------+
|                      |
+----------------------+
```

RTRIM

Description

The RTRIM function is used to remove consecutive spaces or specified characters from the right side (trailing) of a string.

Syntax

```
RTRIM( <str> [, <trim_chars> ] )
```

Parameters

Parameter	Description
<str>	The string that needs to be trimmed.
<trim_chars>	Optional parameter. If this parameter is provided, the RTRIM function will remove all characters in <trim_chars> that appear on the right side of <str>. If this parameter is not provided, the RTRIM function will only remove the space characters on the right side of <str>.

Return Value

Returns a string with trailing spaces or <trim_chars> removed. Special cases:

- If any Parameter is NULL, NULL will be returned.

Examples

```
SELECT rtrim('ab d   ') str;
```

```
+------+
| str  |
+------+
| ab d |
+------+
```

```
SELECT rtrim('ababccaab','ab') str;
```

```
+---------+
| str     |
+---------+
```

```
| ababcca |
+---------+
```

RTRIM_IN

Description

The RTRIM_IN function removes specified characters from the right side of a string. When no character set is specified, it removes trailing spaces by default. When a character set is specified, it removes all specified characters from the right side (regardless of their order in the set). The key feature of RTRIM_IN is that it removes any combination of characters from the specified set, while the RTRIM function removes characters based on exact string matching.

Syntax

```
RTRIM_IN(<str>[, <rhs>])
```

Parameters

Parameter	Description
<str>	The string to be processed. Type: VARCHAR
<rhs>	Optional parameter, the set of characters to be removed. Type: VARCHAR

Return Value

Returns VARCHAR type, representing the processed string.

Special cases: - If str is NULL, returns NULL - If rhs is not specified, removes all trailing spaces - If rhs is specified, removes all characters from the right side that appear in rhs until encountering the first character not in rhs

Examples

1. Remove trailing spaces

```
SELECT rtrim_in('ab d   ') str;
```

```
+------+
| str  |
+------+
| ab d |
+------+
```

2. Remove specified character set

```
-- RTRIM_IN removes any 'a' and 'b' characters from the right end
```

```
SELECT rtrim_in('ababccaab', 'ab') str;
```

```
+---------+
| str     |
+---------+
| ababcc  |
+---------+
```

3. Comparison with RTRIM function

```
SELECT rtrim_in('ababccaab', 'ab'),rtrim('ababccaab', 'ab');
```

```
+-----------------------------+--------------------------+
| rtrim_in('ababccaab', 'ab') | rtrim('ababccaab', 'ab') |
+-----------------------------+--------------------------+
| ababcc                      | ababcca                  |
+-----------------------------+--------------------------+
```

SPACE

Description

Generate a string consisting of a specified number of spaces.

Syntax

```
SPACE ( <len> )
```

Parameters

Parameter	Description
<len>	The number of spaces to generate

Return Value

Returns a string consisting of the specified number of spaces. Special cases:

- If any Parameter is NULL, NULL will be returned.
- When <len> is less than 0, an empty string is returned.

Examples

```
SELECT space(10);
```

```
+------------+
| space(10)  |
+------------+
|            |
+------------+
```

```
SELECT space(null);
```

```
+-------------+
| space(NULL) |
+-------------+
| NULL        |
+-------------+
```

SPLIT_BY_STRING

Description

Split the input string into a string array according to the specified string.

Syntax

```
SPLIT_BY_STRING ( <str>, <separator> )
```

Parameters

Parameter	Description
<str>	The string to be split.
<separator>	The string used for splitting.

Return Value

Returns a string array split according to the specified string. Special cases:

- If any of the parameters is NULL, NULL is returned.
- When <separator> is an empty string, <str> will be split to byte sequence.

Examples

```
SELECT split_by_string('hello','l');
```

```
+------------------------------+
| split_by_string('hello', 'l') |
+------------------------------+
| ["he", "", "o"]              |
+------------------------------+
```

```
SELECT split_by_string('hello','');
```

```
+----------------------------+
| split_by_string('hello', '') |
+----------------------------+
| ["h", "e", "l", "l", "o"]    |
+----------------------------+
```

SPLIT_PART

Description

The SPLIT_PART function splits a string into multiple parts according to the specified separator and return one of the parts.

Syntax

```
SPLIT_PART ( <str>, <separator>, <part_index> )
```

Parameters

Parameter	Description
<str>	The string to be split
<separator>	The string used for splitting
<part_index>	The index of the part to be returned. Starting from 1

Return Value

Returns the specified part of the string split according to the delimiter. Special cases:

- If any of the parameters is NULL, NULL is returned.
- When <part_index> is 0, NULL is returned.
- When <part_index> is out of range, NULL is returned.

Examples

```
select split_part("hello world", " ", 1);
```

```
+----------------------------------+
| split_part('hello world', ' ', 1) |
+----------------------------------+
| hello                            |
+----------------------------------+
```

```
SELECT split_part('apple,banana,cherry', ',', 0);
```

```
+-----------------------------------------+
| split_part('apple,banana,cherry', ',', 0) |
+-----------------------------------------+
| NULL                                    |
+-----------------------------------------+
```

5. Index out of range

```
SELECT split_part('apple,banana', ',', 5), split_part('apple,banana', ',', -5);
```

```
+-----------------------------------+------------------------------------+
| split_part('apple,banana', ',', 5) | split_part('apple,banana', ',', -5) |
+-----------------------------------+------------------------------------+
| NULL                              | NULL                               |
+-----------------------------------+------------------------------------+
```

STARTS_WITH

Description

The STARTS_WITH function checks if a string starts with a specified prefix. Returns true if the string starts with the specified prefix; otherwise returns false.

Syntax

```
STARTS_WITH(<str>, <prefix>)
```

Parameters

Parameter	Description
<str>	The string to check. Type: VARCHAR
<prefix>	The prefix string to match. Type: VARCHAR

Return Value

Returns BOOLEAN type.

Special cases: - Returns NULL if any argument is NULL

Examples

1. Successful match

```
SELECT starts_with('hello world', 'hello');
```

```
+-------------------------------------+
| starts_with('hello world', 'hello') |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

2. Failed match

```
SELECT starts_with('hello world', 'world');
```

```
+-------------------------------------+
| starts_with('hello world', 'world') |
+-------------------------------------+
|                                   0 |
+-------------------------------------+
```

STRCMP

Description

The STRCMP function compares two strings lexicographically. It returns an integer value indicating the result of the comparison.

Syntax

```
STRCMP(<str0>, <str1>)
```

Parameters

Parameter	Description
<str0>	The first string to compare. Type: VARCHAR
<str1>	The second string to compare. Type: VARCHAR

Return Value

Returns a TINYINT value indicating the comparison result: - Returns 0: if str0 equals str1 - Returns 1: if str0 is lexicographically greater than str1 - Returns -1: if str0 is lexicographically less than str1

Special cases: - Returns NULL if any argument is NULL

Examples

1. Comparing identical strings

```
SELECT strcmp('test', 'test');
```

```
+------------------------+
| strcmp('test', 'test') |
+------------------------+
|                      0 |
+------------------------+
```

2. First string is greater

```
SELECT strcmp('test1', 'test');
```

```
+-------------------------+
| strcmp('test1', 'test') |
+-------------------------+
```

```
|                      1 |
+------------------------+
```

3. First string is smaller

```
SELECT strcmp('test', 'test1');
```

```
+-------------------------+
| strcmp('test', 'test1') |
+-------------------------+
|                      -1 |
+-------------------------+
```

STRLEFT

Description

The STRLEFT function returns a specified number of characters from the left side of a string. The length is measured in UTF8 characters.

Alias

LEFT

Syntax

```
STRLEFT(<str>, <len>)
```

Parameters

Parameter	Description
<str>	The string to extract from. Type: VARCHAR
<len>	The number of characters to return. Type: INT

Return Value

Returns VARCHAR type, representing the extracted substring.

Special cases: - Returns NULL if any argument is NULL - Returns empty string “” if len is less than or equal to 0 - Returns the entire string if len is greater than the string length

Examples

1. Basic usage

```
SELECT strleft('Hello doris', 5);
```

```
+--------------------------+
| strleft('Hello doris', 5) |
+--------------------------+
| Hello                    |
+--------------------------+
```

2. Handling negative length

```
SELECT strleft('Hello doris', -5);
```

```
+---------------------------+
| strleft('Hello doris', -5) |
+---------------------------+
|                           |
+---------------------------+
```

3. Handling NULL parameter

```
SELECT strleft('Hello doris', NULL);
```

```
+-----------------------------+
| strleft('Hello doris', NULL) |
+-----------------------------+
| NULL                        |
+-----------------------------+
```

4. Handling NULL string

```
SELECT strleft(NULL, 3);
```

```
+------------------+
| strleft(NULL, 3) |
+------------------+
| NULL             |
+------------------+
```

STRRIGHT

Description

The STRRIGHT function returns a specified number of characters from the right side of a string. The length is measured in UTF8 characters.

Alias

RIGHT

Syntax

```
STRRIGHT(<str>, <len>)
```

Parameters

Parameter	Description
<str>	The string to extract from. Type: VARCHAR
<len>	The number of characters to return. Type: INT

Return Value

Returns VARCHAR type, representing the extracted substring.

Special cases: - Returns NULL if any argument is NULL - If len is negative, returns the substring starting from the abs(len)th character from the right - Returns the entire string if len is greater than the string length

Examples

1. Basic usage

```
SELECT strright('Hello doris', 5);
```

```
+----------------------------+
| strright('Hello doris', 5) |
+----------------------------+
| doris                      |
+----------------------------+
```

2. Handling negative length

```
SELECT strright('Hello doris', -7);
```

```
+-----------------------------+
| strright('Hello doris', -7) |
+-----------------------------+
| doris                       |
+-----------------------------+
```

3. Handling NULL parameter

```
SELECT strright('Hello doris', NULL);
```

```
+------------------------------+
| strright('Hello doris', NULL) |
+------------------------------+
| NULL                          |
+------------------------------+
```

4. Handling NULL string

```
SELECT strright(NULL, 5);
```

```
+-------------------+
| strright(NULL, 5) |
+-------------------+
| NULL              |
+-------------------+
```

SUB_REPLACE

Description

The `sub_replace` function is used to replace substrings within a string. You can specify the substring to be replaced and the target string to replace it with. It returns a new string where the substring starting from `start` with length `len` in `str` is replaced by `new_str`. If `start` or `len` is a negative integer, it returns NULL. The default value for `len` is the length of `new_str`.

Syntax

```
sub_replace(<str>, <new_str>, [ ,<start> [ , <len> ] ])
```

Parameters

Parameter	Description
`<str>`	The target string in which the replacement will occur
`<new_str>`	The string that will replace the specified substring
`<start>`	`start` is the position where the replacement operation begins, indicating from which position in the string the replacement will start
`<len>`	`len` is an optional parameter that specifies the length of the substring to be replaced

Return Value

Returns the string after replacement.

Examples

```
select sub_replace("this is origin str","NEW-STR",1);
```

```
+-----------------------------------------------+
| sub_replace('this is origin str', 'NEW-STR', 1) |
+-----------------------------------------------+
| tNEW-STRorigin str                            |
+-----------------------------------------------+
```

```
select sub_replace("doris","***",1,2);
```

```
+----------------------------------+
| sub_replace('doris', '***', 1, 2) |
+----------------------------------+
```

```
| d***is                          |
+---------------------------------+
```

SUBSTRING

Description

The SUBSTRING function is used to extract a substring from a string. You can specify the starting position and length, supporting both forward and backward extraction. The position of the first character in the string is 1.

Alias

SUBSTR

Syntax

```
SUBSTRING(<str>, <pos> [, <len>])
```

Parameters

Parameter	Description
<str>	Source string. Type: VARCHAR
<pos>	Starting position, can be negative. Type: INT
<len>	Optional parameter, length to extract. Type: INT

Return Value

Returns VARCHAR type, representing the extracted substring.

Special cases: - If any parameter is NULL, returns NULL - If pos is 0, returns an empty string - If pos is negative, counts from the end of the string backwards - If pos exceeds the string length, returns an empty string - If len is not specified, returns all characters from pos to the end of the string

Examples

1. Basic usage (specify starting position)

```
SELECT substring('abc1', 2);
```

```
+-----------------------------+
| substring('abc1', 2)        |
+-----------------------------+
| bc1                         |
+-----------------------------+
```

2. Using negative position

```
SELECT substring('abc1', -2);
```

```
+-----------------------------+
| substring('abc1', -2)       |
+-----------------------------+
| c1                          |
+-----------------------------+
```

3. Case when position is 0

```
SELECT substring('abc1', 0);
```

```
+----------------------+
| substring('abc1', 0) |
+----------------------+
|                      |
+----------------------+
```

4. Position exceeds string length

```
SELECT substring('abc1', 5);
```

```
+-----------------------------+
| substring('abc1', 5)        |
+-----------------------------+
|                             |
+-----------------------------+
```

5. Specifying length parameter

```
SELECT substring('abc1def', 2, 2);
```

```
+-----------------------------+
| substring('abc1def', 2, 2)  |
+-----------------------------+
| bc                          |
+-----------------------------+
```

SUBSTRING_INDEX

Description

The SUBSTRING_INDEX function is used to extract a substring from a string based on a specified delimiter and occurrence count. This function supports counting from either left or right.

Syntax

```
SUBSTRING_INDEX(<content>, <delimiter>, <field>)
```

Parameters

Parameter	Description
<content>	The string to be extracted from. Type: VARCHAR
<delimiter>	The delimiter string, case-sensitive and multi-byte safe. Type: VARCHAR
<field>	Number of delimiter occurrences. Positive numbers count from left, negative numbers count from right. Type: INT

Return Value

Returns VARCHAR type, representing the extracted substring.

Special cases: - If field > 0, returns the substring before the field-th delimiter from the left - If field < 0, returns the substring after the |field|-th delimiter from the right - If field = 0, returns empty string when content is not NULL, returns NULL when content is NULL - If any parameter is NULL, returns NULL

Examples

1. Extract content before the first space from the left

```
SELECT substring_index('hello world', ' ', 1);
```

```
+---------------------------------------+
| substring_index('hello world', ' ', 1) |
+---------------------------------------+
| hello                                 |
+---------------------------------------+
```

2. Extract all content from the left (delimiter count greater than actual occurrences)

```
SELECT substring_index('hello world', ' ', 2);
```

```
+---------------------------------------+
| substring_index('hello world', ' ', 2) |
+---------------------------------------+
| hello world                           |
+---------------------------------------+
```

3. Extract content after the last space from the right

```
SELECT substring_index('hello world', ' ', -1);
```

```
+----------------------------------------+
| substring_index('hello world', ' ', -1) |
+----------------------------------------+
| world                                  |
+----------------------------------------+
```

4. Case when field is 0

```
SELECT substring_index('hello world', ' ', 0);
```

```
+---------------------------------------+
| substring_index('hello world', ' ', 0) |
+---------------------------------------+
|                                       |
+---------------------------------------+
```

TO_BASE64

Description

The TO_BASE64 function is used to convert an input string to Base64 encoded format. Base64 encoding can convert any binary data into a string composed of 64 characters.

Syntax

```
TO_BASE64(<str>)
```

Parameters

Parameter	Description
<str>	The string to be Base64 encoded. Type: VARCHAR

Return Value

Returns VARCHAR type, representing the Base64 encoded string.

Special cases: - If input is NULL, returns NULL - If input is an empty string, returns an empty string

Examples

1. Single character encoding

```
SELECT to_base64('1');
```

```
+----------------+
| to_base64('1') |
+----------------+
| MQ==           |
+----------------+
```

2. Multiple character encoding

```
SELECT to_base64('234');
```

```
+------------------+
| to_base64('234') |
+------------------+
| MjM0             |
+------------------+
```

TOKENIZE

Description

The `TOKENIZE` function tokenizes a string using a specified parser and returns the tokenization results as a string array. This function is particularly useful for testing and understanding how text will be analyzed when using inverted indexes with full-text search capabilities.

Syntax

```
ARRAY<VARCHAR> TOKENIZE(VARCHAR str, VARCHAR properties)
```

Parameters

- `str`: The input string to be tokenized. Type: `VARCHAR`
- `properties`: A property string specifying the parser configuration. Type: `VARCHAR`

The `properties` parameter supports the following key-value pairs (format: `'key1'='value1', 'key2'='value2'` or `"key1"="value1", "key2"="value2"`):

Supported Properties

Property	Description	Example Values
`parser`	Built-in parser type	`"chinese"`, `"english"`, `"unicode"`
`parser_mode`	Parser mode for Chinese parser	`"fine_grained"`, `"coarse_grained"`
`char_filter_type`	Character filter type	`"char_replace"`
`char_filter_pattern`	Characters to be replaced (used with `char_filter_type`)	`"._=:,"`
`char_filter_replacement`	Replacement character (used with `char_filter_type`)	`" "` (space)
`stopwords`	Stop words configuration	`"none"`

Return Value

Returns an `ARRAY<VARCHAR>` containing the tokenized strings as individual array elements.

Examples

Example 1: Using the Chinese parser

```
SELECT TOKENIZE('我来到北京清华大学', "'parser'='chinese'");
```

```
["我", "来到", "北京", "清华大学"]
```

Example 2: Chinese parser with fine-grained mode

```
SELECT TOKENIZE('我来到北京清华大学', "'parser'='chinese', 'parser_mode'='fine_grained'");
```

```
["我", "来到", "北京", "清华", "清华大学", "华大", "大学"]
```

Example 3: Using the Unicode parser

```
SELECT TOKENIZE('Apache Doris数据库', "'parser'='unicode'");
```

```
["apache", "doris", "数", "据", "库"]
```

Example 4: Using character filters

```
SELECT TOKENIZE('GET /images/hm_bg.jpg HTTP/1.0 test:abc=bcd',
    '"parser"="unicode","char_filter_type" = "char_replace","char_filter_pattern" = "._=:,","char_filter_replacement" = " "');
```

```
["get", "images", "hm", "bg", "jpg", "http", "1", "0", "test", "abc", "bcd"]
```

Example 5: Stopwords configuration

```
SELECT TOKENIZE('华夏智胜新税股票A', '"parser"="unicode"');
```

```
["华", "夏", "智", "胜", "新", "税", "股", "票"]
```

```
SELECT TOKENIZE('华夏智胜新税股票A', '"parser"="unicode","stopwords" = "none"');
```

```
["华", "夏", "智", "胜", "新", "税", "股", "票", "a"]
```

Notes

1. **Parser Configuration**: The `properties` parameter must be a valid property string. Only built-in parsers are supported in this version.
2. **Supported Parsers**: Version 2.1 supports the following built-in parsers:
    - `chinese`: Chinese text parser with optional `parser_mode` (`fine_grained` or `coarse_grained`)
    - `english`: English language parser with stemming
    - `unicode`: Unicode-based parser for multilingual text

3. **Parser Mode**: The `parser_mode` property is primarily used with the `chinese` parser:
    - `fine_grained`: Produces more detailed tokens with overlapping segments
    - `coarse_grained`: Default mode with standard segmentation
4. **Character Filters**: Use `char_filter_type`, `char_filter_pattern`, and `char_filter_replacement` together to replace specific characters before tokenization.
5. **Performance**: The `TOKENIZE` function is primarily intended for testing and debugging parser configurations. For production full-text search, use inverted indexes with the `MATCH` predicate.
6. **Compatibility with Inverted Indexes**: The same parser configuration used in `TOKENIZE` can be applied to inverted indexes when creating tables:

    ```
    CREATE TABLE example (
        content TEXT,
        INDEX idx_content(content) USING INVERTED PROPERTIES("parser"="chinese")
    )
    ```

7. **Testing Parser Behavior**: Use `TOKENIZE` to preview how text will be tokenized before creating inverted indexes, helping to choose the most appropriate parser for your data.

Keywords

TOKENIZE, STRING, FULL-TEXT SEARCH, INVERTED INDEX, PARSER

TOP_LEVEL_DOMAIN

Description

The TOP_LEVEL_DOMAIN function is used to extract the top-level domain from a URL. If the input URL is invalid, it returns an empty string.

Syntax

```
TOP_LEVEL_DOMAIN(<url>)
```

Parameters

Parameter	Description
<url>	The URL string from which to extract the top-level domain. Type: VARCHAR

Return Value

Returns VARCHAR type, representing the extracted top-level domain.

Special cases: - Returns NULL if url is NULL - Returns an empty string if url is not a valid URL format - For multi-level domains (e.g., .com.cn), returns the last level domain

Examples

1. Basic domain processing

```
SELECT top_level_domain('www.baidu.com');
```

```
+----------------------------------+
| top_level_domain('www.baidu.com') |
+----------------------------------+
| com                              |
+----------------------------------+
```

2. Multi-level domain processing

```
SELECT top_level_domain('www.google.com.cn');
```

```
+--------------------------------------+
| top_level_domain('www.google.com.cn') |
+--------------------------------------+
| cn                                   |
+--------------------------------------+
```

3. Invalid URL processing

```
SELECT top_level_domain('wwwwwwww');
```

```
+-----------------------------+
| top_level_domain('wwwwwwww') |
+-----------------------------+
|                             |
+-----------------------------+
```

TRANSLATE

Description

The TRANSLATE function is used for string replacement, converting characters in the source string according to mapping rules. It replaces characters in the 'from' string with corresponding characters in the 'to' string.

Syntax

```
TRANSLATE(<source>, <from>, <to>)
```

Parameters

Parameter	Description
<source>	The source string to be converted. Type: VARCHAR
<from>	The set of characters to be replaced. Type: VARCHAR
<to>	The set of replacement characters. Type: VARCHAR

Return Value

Returns VARCHAR type, representing the transformed string.

Special cases: - Returns NULL if any parameter is NULL - If there are duplicate characters in the 'from' string, only the first occurrence is used - If a source character is not in the 'from' string, it remains unchanged - If a character's position in the 'from' string exceeds the length of the 'to' string, the corresponding source character will be deleted - If both 'from' and 'to' are empty strings, returns the original string

Examples

1. Basic character replacement

```
SELECT translate('abcd', 'a', 'z');
```

```
+---------------------------+
| translate('abcd', 'a', 'z') |
+---------------------------+
| zbcd                      |
+---------------------------+
```

2. Multiple replacements of the same character

```
SELECT translate('abcda', 'a', 'z');
```

```
+----------------------------+
| translate('abcda', 'a', 'z') |
+----------------------------+
| zbcdz                      |
+----------------------------+
```

3. Special character replacement

```
SELECT translate('Palhoça', 'ç', 'c');
```

```
+-------------------------------+
| translate('Palhoça', 'ç', 'c') |
+-------------------------------+
| Palhoca                       |
+-------------------------------+
```

4. Character deletion (empty ‘to’ string)

```
SELECT translate('abcd', 'a', '');
```

```
+----------------------------+
| translate('abcd', 'a', '') |
+----------------------------+
| bcd                        |
+----------------------------+
```

5. Duplicate characters in ‘from’ string

```
SELECT translate('abcd', 'aac', 'zq');
```

```
+-------------------------------+
| translate('abcd', 'aac', 'zq') |
+-------------------------------+
| zbd                           |
+-------------------------------+
```

TRIM

Description

This command is used to delete Spaces or specified characters at both ends of the string. If no rhs parameter is specified, delete the Spaces that appear continuously at the beginning of the right and left parts of str. Otherwise, delete rhs

Syntax

```
TRIM( <str> [ , <rhs>])
```

Required Parameters

Parameters	Description
<str>	Deletes the Spaces at both ends of the string

Optional Parameters

Parameters	Description
<rhs>	removes the specified character

Return Value

Deletes Spaces at both ends or the string after a specified character

Example

```
SELECT trim('   ab d   ') str;
```

```
+------+
| str  |
+------+
| ab d |
+------+
```

```
SELECT trim('ababccaab','ab') str;
```

```
+------+
| str  |
+------+
```

```
| cc   |
+------+
```

TRIM_IN

Description

When there is no `rhs` parameter, this function removes the leading and trailing spaces from the `str` string. When the `rhs` parameter is provided, it removes any characters from both ends of the string that are found in the `rhs` character set (order does not matter).

Syntax

```
TRIM_IN( <str> [ , <rhs>])
```

Required Parameters

Parameters	Description
`<str>`	Delete spaces at both ends of the string

Optional Parameters

Parameters	Description
`<rhs>`	Remove the specified character

Return Value

Delete spaces at both ends or the string after the specified character

Returns VARCHAR type, representing the processed string.

Special cases: - If str is NULL, returns NULL - If rhs is not specified, removes all leading and trailing spaces - If rhs is specified, removes all characters from both ends that appear in rhs until encountering the first character not in rhs

Examples

1. Remove spaces from both ends:

```
SELECT trim_in('   ab d   ') str;
```

```
+------+
| str  |
+------+
```

```
|  ab d|
+------+
```

```
SELECT trim_in('ababccaab','ab') str;
```

```
+------+
| str  |
+------+
| cc   |
+------+
```

UCASE/UPPER

Description

Used to convert a string to uppercase letters

Syntax

```
UCASE( <str> )
UPPER( <str> )
```

Required Parameters

Parameters	Description
<str>	The string to convert to uppercase

Return Value

The value after converting the uppercase

Example

```
SELECT UCASE("aBc123"),UPPER("aBc123");
```

```
+-----------------+-----------------+
| UCASE("aBc123") | UPPER("aBc123") |
+-----------------+-----------------+
| ABC123          | ABC123          |
+-----------------+-----------------+
```

```
SELECT UCASE("Кириллица"),UPPER("Бәйтерек");
```

```
+----------------------------+--------------------------+
| UCASE("Кириллица")         | UPPER("Бәйтерек")        |
+----------------------------+--------------------------+
| КИРИЛЛИЦА                  | БӘЙТЕРЕК                 |
+----------------------------+--------------------------+
```

UNHEX

Description

The `unhex` function is used to convert a hexadecimal string back into the original string. It converts every two hexadecimal characters into one byte. When a NULL value is passed as a parameter, it will return NULL; otherwise invalid values return an empty string.

Syntax

```
UNHEX(<str>)
```

Parameters

Parameter	Description
`<str>`	The hexadecimal character string

Return Value

If the input string has a length of 0 or is odd, `unhex` function returns an empty string. If the string contains characters other than [0-9], [a-f], or [A-F], `unhex` function returns an empty string. In other cases, every two characters are converted to their hexadecimal representation and concatenated into a string for output.

Examples

```
select unhex('@');
```

```
+------------+
| unhex('@') |
+------------+
|            |
+------------+
```

```
select unhex(NULL);
```

```
+-------------+
| unhex(NULL) |
+-------------+
| NULL        |
+-------------+
```

```
select unhex('41');
```

```
+-------------+
| unhex('41') |
+-------------+
| A           |
+-------------+
```

```
select unhex('4142');
```

```
+---------------+
| unhex('4142') |
+---------------+
| AB            |
+---------------+
```

URL_DECODE

Description

Converts an url to a decode string.

Syntax

```
URL_DECODE( <str> )
```

Required Parameters

Parameters	Description
<str>	the string to decode. If url is not a string type.

Return Value

The decoded value

Example

```
select url_decode('https%3A%2F%2Fdoris.apache.org%2Fzh-CN%2Fdocs%2Fsql-manual%2Fsql-functions%2Fstring-functions');
```

```
+-------------------------------------------------+
| url_decode('https%3A%2F%2Fdoris.apache.org%2Fzh-CN%2Fdocs%2Fsql-manual%2Fsql-functions%2Fstring-functions') |
+-------------------------------------------------+
| https://doris.apache.org/zh-CN/docs/sql-manual/sql-functions/string-functions                               |
+-------------------------------------------------+
```

URL_ENCODE

Description

Use UTF-8 encoding to complete the URL encoding of the provided text. Typically used to encode parameter information passed as part of a URL

Syntax

```
URL_ENCODE(  <str>  )
```

Required Parameters

Required Parameters

Parameters	Description
<str>	String to be encoded

Return Value

UTF-8 encoding completes the URL encoding of the provided text

Example

```
select  URL_ENCODE('Doris Q&A');
```

```
+-------------------------+
| url_encode('Doris Q&A') |
+-------------------------+
| Doris+Q%26A             |
+-------------------------+
```

UUID

Description

Used to return a randomly generated UUID string.

Syntax

```
UUID()
```

Return Value

Returns a randomly generated UUID string.

Examples

```
select UUID();
```

```
+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| 29077778-fc5e-4603-8368-6b5f8fd55c24 |
+--------------------------------------+
```

CONVERT_TZ

Description

Converts a datetime value from the time zone specified by from_tz to the time zone specified by to_tz, and returns the resulting value. Special cases: - If the parameters are invalid, the function returns NULL.

Syntax

```
CONVERT_TZ(<dt>, <from_tz>, <to_tz>)
```

Parameters

Parameter	Description
<dt>	The datetime value to be converted
<from_tz>	The original time zone of dt
<to_tz>	The target time zone to convert to

Return Value

Returns the calculated date.

Examples

```
select CONVERT_TZ('2019-08-01 13:21:03', 'Asia/Shanghai', 'America/Los_Angeles');
```

```
+------------------------------------------------------------------------+
| convert_tz('2019-08-01 13:21:03', 'Asia/Shanghai', 'America/Los_Angeles') |
+------------------------------------------------------------------------+
| 2019-07-31 22:21:03                                                    |
+------------------------------------------------------------------------+
```

```
select CONVERT_TZ('2019-08-01 13:21:03', '+08:00', 'America/Los_Angeles');
```

```
+---------------------------------------------------------------+
| convert_tz('2019-08-01 13:21:03', '+08:00', 'America/Los_Angeles') |
+---------------------------------------------------------------+
| 2019-07-31 22:21:03                                           |
+---------------------------------------------------------------+
```

CURDATE,CURRENT_DATE

Description

Retrieves the current date and returns it as a DATE type.

Alias

- curdate
- current_date

Syntax

```
CURDATE()
```

Return Value

The current date.

Examples

```
SELECT CURDATE();
```

```
+------------+
| CURDATE()  |
+------------+
| 2019-12-20 |
+------------+
```

```
SELECT CURDATE() + 0;
```

```
+---------------+
| CURDATE() + 0 |
+---------------+
|      20191220 |
+---------------+
```

CURTIME,CURRENT_TIME

Description

Retrieves the current time and returns it as a TIME type.

Alias

- CURTIME
- CURRENT_TIME

Syntax

```
CURTIME()
```

Return Value

Returns the current time.

Examples

```
mysql> select current_time();
```

```
+----------------+
| current_time() |
+----------------+
| 15:25:47       |
+----------------+
```

DATE | Date Time Functions

DATE

Description

Extracts the date from the datetime.

Syntax

```
DATE(<datetime>)
```

Parameters

Parameter	Description
<datetime>	A valid date expression

Return Value

Returns the date part of the datetime.

Examples

```
select date('2010-12-02 19:28:30');
```

```
+-----------------------------------------------------+
| date(cast('2010-12-02 19:28:30' as DATETIMEV2(0))) |
+-----------------------------------------------------+
| 2010-12-02                                          |
+-----------------------------------------------------+
```

DAYS_ADD

Description

Add a specified time interval to the date.

Alias

- date_add
- days_add
- adddate

Syntax

```
DATE_ADD(<date>, <expr> <time_unit>)
```

Parameters

Parameter	Description
<date>	A valid date value
<expr>	The time interval you want to add
<time_unit>	Enumerated values: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND

Return Value

Returns the calculated date.

Examples

```
select date_add('2010-11-30 23:59:59', INTERVAL 2 DAY);
```

```
+-----------------------------------------------+
| date_add('2010-11-30 23:59:59', INTERVAL 2 DAY) |
+-----------------------------------------------+
| 2010-12-02 23:59:59                           |
+-----------------------------------------------+
```

DATE_CEIL

Description

`date_ceil` rounds a given date to the next upper boundary of the specified time interval.

Syntax

```
DATE_CEIL(<datetime>, INTERVAL <period> <type>)
```

Parameters

Parameter	Description
`datetime`	The argument is a valid date expression
`period`	The argument specifies how many units make up each period, with the start time being 0001-01-01T00:00:00
`type`	The argument can be: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND

Return Value

The return value is a date or time, representing the result of rounding the input value up to the specified unit.

Examples

```
select date_ceil("2023-07-13 22:28:18",interval 5 second);
```

```
+-----------------------------------------------------------+
| second_ceil('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-----------------------------------------------------------+
| 2023-07-13 22:28:20                                       |
+-----------------------------------------------------------+
```

```
select date_ceil("2023-07-13 22:28:18",interval 5 minute);
+-----------------------------------------------------------+
| minute_ceil('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-----------------------------------------------------------+
| 2023-07-13 22:30:00                                       |
+-----------------------------------------------------------+
```

```
select date_ceil("2023-07-13 22:28:18",interval 5 hour);
```

```
+-----------------------------------------------------------+
| hour_ceil('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-----------------------------------------------------------+
```

```
| 2023-07-13 23:00:00                                      |
+----------------------------------------------------------+
```

```
select date_ceil("2023-07-13 22:28:18",interval 5 day);
```

```
+----------------------------------------------------------+
| day_ceil('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+----------------------------------------------------------+
| 2023-07-15 00:00:00                                      |
+----------------------------------------------------------+
```

```
select date_ceil("2023-07-13 22:28:18",interval 5 month);
```

```
+------------------------------------------------------------+
| month_ceil('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+------------------------------------------------------------+
| 2023-12-01 00:00:00                                        |
+------------------------------------------------------------+
```

```
select date_ceil("2023-07-13 22:28:18",interval 5 year);
```

```
+-----------------------------------------------------------+
| year_ceil('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-----------------------------------------------------------+
| 2026-01-01 00:00:00                                       |
+-----------------------------------------------------------+
```

DATE_FLOOR

Description

`date_floor` rounds a given date to the closest lower boundary of the specified time interval.

Syntax

```
DATE_FLOOR(<datetime>, INTERVAL <period> <type>)
```

Parameters

Parameter	Description
`datetime`	The argument is a valid date expression
`period`	The argument specifies how many units make up each period, with the start time being 0001-01-01T00:00:00
`type`	The argument can be: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND

Return Value

The return value is a date or time, representing the result of rounding the input value down to the specified unit.

Examples

```
select date_floor("0001-01-01 00:00:16",interval 5 second);
```

```
+------------------------------------------------------------+
| second_floor('0001-01-01 00:00:16', 5, '0001-01-01 00:00:00') |
+------------------------------------------------------------+
| 0001-01-01 00:00:15                                        |
+------------------------------------------------------------+
```

```
select date_floor("0001-01-01 00:00:18",interval 5 second);
```

```
+------------------------------------------------------------+
| second_floor('0001-01-01 00:00:18', 5, '0001-01-01 00:00:00') |
+------------------------------------------------------------+
| 0001-01-01 00:00:15                                        |
+------------------------------------------------------------+
```

```
select date_floor("2023-07-13 22:28:18",interval 5 minute);
```

```
+-------------------------------------------------------------+
| minute_floor('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-------------------------------------------------------------+
| 2023-07-13 22:25:00                                         |
+-------------------------------------------------------------+
```

```
select date_floor("2023-07-13 22:28:18",interval 5 hour);
```

```
+-------------------------------------------------------------+
| hour_floor('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-------------------------------------------------------------+
| 2023-07-13 18:00:00                                         |
+-------------------------------------------------------------+
```

```
select date_floor("2023-07-13 22:28:18",interval 5 day);
```

```
+-------------------------------------------------------------+
| day_floor('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-------------------------------------------------------------+
| 2023-07-10 00:00:00                                         |
+-------------------------------------------------------------+
```

```
select date_floor("2023-07-13 22:28:18",interval 5 month);
```

```
+-------------------------------------------------------------+
| month_floor('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-------------------------------------------------------------+
| 2023-07-01 00:00:00                                         |
+-------------------------------------------------------------+
```

```
select date_floor("2023-07-13 22:28:18",interval 5 year);
```

```
+-------------------------------------------------------------+
| year_floor('2023-07-13 22:28:18', 5, '0001-01-01 00:00:00') |
+-------------------------------------------------------------+
| 2021-01-01 00:00:00                                         |
+-------------------------------------------------------------+
```

DATE_FORMAT

Description

Convert the date type to a bit string according to the format type.

Syntax

```
DATE_FORMAT(<date>, <format>)
```

Parameters

Parameter	Description
<date>	A valid date value
<format>	Specifies the output format for the date/time

The formats available are:

Format	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, …)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM

Format	Description
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week; WEEK mode 0
%u	Week (00..53), where Monday is the first day of the week; WEEK mode 1
%V	Week (01..53), where Sunday is the first day of the week; WEEK mode 2; used with %X
%v	Week (01..53), where Monday is the first day of the week; WEEK mode 3; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal % character
%**x**	**x**, for any "**x**" not listed above

Also support 3 formats:

```
yyyyMMdd
yyyy-MM-dd
yyyy-MM-dd HH:mm:ss
```

Return Value

The formatted date string, with the following special case:

- Currently, a maximum of 128 bytes of string is supported. If the returned value exceeds 128 bytes, it will return NULL.

Examples

```
select date_format('2009-10-04 22:23:00', '%W %M %Y');
```

```
+------------------------------------------------+
| date_format('2009-10-04 22:23:00', '%W %M %Y') |
+------------------------------------------------+
| Sunday October 2009                            |
+------------------------------------------------+
```

```
select date_format('2007-10-04 22:23:00', '%H:%i:%s');
```

```
+-------------------------------------------+
| date_format('2007-10-04 22:23:00', '%H:%i:%s') |
+-------------------------------------------+
| 22:23:00                                  |
+-------------------------------------------+
```

```
select date_format('1900-10-04 22:23:00', '%D %y %a %d %m %b %j');
```

```
+------------------------------------------------------------+
| date_format('1900-10-04 22:23:00', '%D %y %a %d %m %b %j') |
+------------------------------------------------------------+
| 4th 00 Thu 04 10 Oct 277                                   |
+------------------------------------------------------------+
```

```
select date_format('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
```

```
+------------------------------------------------------------+
| date_format('1997-10-04 22:23:00', '%H %k %I %r %T %S %w') |
+------------------------------------------------------------+
| 22 22 10 10:23:00 PM 22:23:00 00 6                         |
+------------------------------------------------------------+
```

```
select date_format('1999-01-01 00:00:00', '%X %V');
```

```
+-----------------------------------------+
| date_format('1999-01-01 00:00:00', '%X %V') |
+-----------------------------------------+
| 1998 52                                 |
+-----------------------------------------+
```

```
select date_format('2006-06-01', '%d');
```

```
+----------------------------------------+
| date_format('2006-06-01 00:00:00', '%d') |
+----------------------------------------+
| 01                                     |
+----------------------------------------+
```

```
select date_format('2006-06-01', '%%%d');
```

```
+----------------------------------------+
| date_format('2006-06-01 00:00:00', '%%%d') |
+----------------------------------------+
| %01                                    |
+----------------------------------------+
```

DAYS_SUB

Description

Subtracts a specified time interval to the date.

Alias

别名

- days_sub
- date_sub
- subdate

Syntax

```
DATE_SUB(<date>, <expr> <time_unit>)
```

Parameters

Parameter	Description
<date>	A valid date value
<expr>	The time interval you want to subtract
<type>	Enumerated values: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND

Return Value

Returns the calculated date.

Examples

```
select date_sub('2010-11-30 23:59:59', INTERVAL 2 DAY);
```

```
+------------------------------------------------+
| date_sub('2010-11-30 23:59:59', INTERVAL 2 DAY) |
+------------------------------------------------+
| 2010-11-28 23:59:59                            |
+------------------------------------------------+
```

DATE_TRUNC

Description

Truncates the datetime according to the specified time unit.

Syntax

```
DATE_TRUNC(<datetime>, <time_unit>)
DATE_TRUNC(<time_unit>, <datetime>)
```

Parameters

Parameter	Description
<datetime>	A valid date expression
<time_unit>	The time interval to truncate to. The optional values are: [second, minute, hour, day, week, month, quarter, year]

Return Value

The time truncated according to the specified time unit.

Examples

```
select date_trunc('2010-12-02 19:28:30', 'second');
```

```
+-------------------------------------------------+
| date_trunc('2010-12-02 19:28:30', 'second')     |
+-------------------------------------------------+
| 2010-12-02 19:28:30                             |
+-------------------------------------------------+
```

```
select date_trunc('2010-12-02 19:28:30', 'minute');
```

```
+-------------------------------------------------+
| date_trunc('2010-12-02 19:28:30', 'minute')     |
+-------------------------------------------------+
| 2010-12-02 19:28:00                             |
+-------------------------------------------------+
```

```
select date_trunc('2010-12-02 19:28:30', 'hour');
```

```
+-----------------------------------------------+
| date_trunc('2010-12-02 19:28:30', 'hour')       |
+-----------------------------------------------+
| 2010-12-02 19:00:00                           |
+-----------------------------------------------+
```

```
select date_trunc('2010-12-02 19:28:30', 'day');
```

```
+-----------------------------------------------+
| date_trunc('2010-12-02 19:28:30', 'day')        |
+-----------------------------------------------+
| 2010-12-02 00:00:00                           |
+-----------------------------------------------+
```

```
select date_trunc('2023-4-05 19:28:30', 'week');
```

```
+-----------------------------------------+
| date_trunc('2023-04-05 19:28:30', 'week') |
+-----------------------------------------+
| 2023-04-03 00:00:00                       |
+-----------------------------------------+
```

```
select date_trunc('2010-12-02 19:28:30', 'month');
```

```
+-----------------------------------------------+
| date_trunc('2010-12-02 19:28:30', 'month')      |
+-----------------------------------------------+
| 2010-12-01 00:00:00                           |
+-----------------------------------------------+
```

```
select date_trunc('2010-12-02 19:28:30', 'quarter');
```

```
+-----------------------------------------------+
| date_trunc('2010-12-02 19:28:30', 'quarter')    |
+-----------------------------------------------+
| 2010-10-01 00:00:00                           |
+-----------------------------------------------+
```

```
select date_trunc('2010-12-02 19:28:30', 'year');
```

```
+-----------------------------------------------+
| date_trunc('2010-12-02 19:28:30', 'year')       |
+-----------------------------------------------+
| 2010-01-01 00:00:00                           |
+-----------------------------------------------+
```

DATEDIFF

Description

Calculates the difference between two given dates.

Syntax

```
DATEDIFF(<expr1>, <expr2>)
```

Parameters

Parameter	Description
<expr1>	The minuend (the date to be subtracted from)
<expr2>	The subtrahend (the date to subtract)

Return Value

Returns the value of `expr1 - expr2`, with the result rounded to the nearest day.

Examples

```
select datediff(CAST('2007-12-31 23:59:59' AS DATETIME), CAST('2007-12-30' AS DATETIME));
```

```
+-----------------------------------------------------------------------------+
| datediff(CAST('2007-12-31 23:59:59' AS DATETIME), CAST('2007-12-30' AS DATETIME)) |
+-----------------------------------------------------------------------------+
|                                                                           1 |
+-----------------------------------------------------------------------------+
```

```
select datediff(CAST('2010-11-30 23:59:59' AS DATETIME), CAST('2010-12-31' AS DATETIME));
```

```
+-----------------------------------------------------------------------------+
| datediff(CAST('2010-11-30 23:59:59' AS DATETIME), CAST('2010-12-31' AS DATETIME)) |
+-----------------------------------------------------------------------------+
|                                                                         -31 |
+-----------------------------------------------------------------------------+
```

DAY

Description

Obtain the day information from the date, with return values ranging from 1 to 31.

Alias

- dayofmonth

Syntax

```
DAY(<dt>)
```

Parameters

Parameter	Description
<dt>	A valid date expression

Return Value

Returns the day information from the given date.

Examples

```
select day('1987-01-31');
```

```
+----------------------------+
| day('1987-01-31 00:00:00') |
+----------------------------+
|                         31 |
+----------------------------+
```

DAY_CEIL

Description

Rounds the date up to the nearest specified time interval period.

Syntax

```
DAY_CEIL(<datetime>)
DAY_CEIL(<datetime>, <origin>)
DAY_CEIL(<datetime>, <period>)
DAY_CEIL(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	A valid date expression
<period>	Specifies how many days make up each period
<origin>	The starting point of time. If not provided, the default is 0001-01-01T00:00:00

Return Value

Returns the date of the nearest rounded-up timestamp.

Examples

```
select day_ceil("2023-07-13 22:28:18", 5);
```

```
+-----------------------------------------------------------+
| day_ceil(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2023-07-17 00:00:00                                       |
+-----------------------------------------------------------+
```

DAY_FLOOR

Description

Rounds the date down to the nearest timestamp of the specified time interval period.

Syntax

```
DAY_FLOOR(<datetime>)
DAY_FLOOR(<datetime>, <origin>)
DAY_FLOOR(<datetime>, <period>)
DAY_FLOOR(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	A valid date expression
<period>	Specifies how many days make up each period
<origin>	The starting point of time. If not provided, the default is 0001-01-01T00:00:00

Return Value

Returns the date of the nearest rounded-up timestamp.

Examples

```
select day_floor("2023-07-13 22:28:18", 5);
```

```
+-----------------------------------------------------------+
| day_floor(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2023-07-12 00:00:00                                       |
+-----------------------------------------------------------+
```

DAYNAME

Description

Calculates the name of the day corresponding to the given date expression.

Syntax

```
DAYNAME(<dt>)
```

Parameters

Parameter	Description
<dt>	The date expression to be calculated

Return Value

Returns the name of the day corresponding to the given date expression.

Examples

```
select dayname('2007-02-03 00:00:00');
```

```
+--------------------------------+
| dayname('2007-02-03 00:00:00') |
+--------------------------------+
| Saturday                       |
+--------------------------------+
```

DAYOFWEEK

Description

Returns the weekday index of the date, where Sunday is 1, Monday is 2, and Saturday is 7.

Syntax

```
DAYOFWEEK(<dt>)
```

Parameters

Parameter	Description
<dt>	The date expression to be calculated

Return Value

Returns the weekday index of the date.

Examples

```
select dayofweek('2019-06-25');
```

```
+----------------------------------+
| dayofweek('2019-06-25 00:00:00') |
+----------------------------------+
|                                3 |
+----------------------------------+
```text

```sql
select dayofweek(cast(20190625 as date));
```

```
+-----------------------------------+
| dayofweek(CAST(20190625 AS DATE)) |
+-----------------------------------+
|                                 3 |
+-----------------------------------+
```

DAYOFYEAR

Description

Obtains the corresponding day of the year for the given date.

Syntax

```
DAYOFYEAR(<dt>)
```

Parameters

Parameter	Description
<dt>	The date expression to be calculated

Return Value

Returns the day of the year corresponding to the given date.

Examples

```
select dayofyear('2007-02-03 00:00:00');
```

```
+----------------------------------+
| dayofyear('2007-02-03 00:00:00') |
+----------------------------------+
|                               34 |
+----------------------------------+
```

EXTRACT

Description

The `extract` function is used to extract a specified part of a date or time value, such as the year, month, day, hour, minute, second, etc. This function is commonly used to extract specific time components from a datetime field for calculation, comparison, or display.

Syntax

```
EXTRACT(<unit> FROM <datetime>)
```

Parameters

Parameter	Description
`unit`	The unit to extract from the DATETIME. Possible values are year, month, day, hour, minute, second, or microsecond
`datetime`	The argument is a valid date expression

Return Value

The return value is the extracted part of the date or time (such as an integer), depending on the unit being extracted.

Examples

```
select extract(year from '2022-09-22 17:01:30') as year,
extract(month from '2022-09-22 17:01:30') as month,
extract(day from '2022-09-22 17:01:30') as day,
extract(hour from '2022-09-22 17:01:30') as hour,
extract(minute from '2022-09-22 17:01:30') as minute,
extract(second from '2022-09-22 17:01:30') as second,
extract(microsecond from cast('2022-09-22 17:01:30.000123' as datetimev2(6))) as microsecond;
```

```
+------+-------+------+------+--------+--------+-------------+
| year | month | day  | hour | minute | second | microsecond |
+------+-------+------+------+--------+--------+-------------+
| 2022 |     9 |   22 |   17 |      1 |     30 |         123 |
+------+-------+------+------+--------+--------+-------------+
```

FROM_DAYS

Description

Given a number of days, returns a DATE.

- Note: To maintain consistent behavior with MySQL, the date 0000-02-29 does not exist.

Syntax

```
FROM_DAYS(<dt>)
```

Parameters

Parameter	Description
<dt>	Days

Return Value

Returns the date corresponding to the given number of days.

Examples

```
select from_days(730669),from_days(5),from_days(59), from_days(60);
```

```
+------------------+-------------+--------------+--------------+
| from_days(730669) | from_days(5) | from_days(59) | from_days(60) |
+------------------+-------------+--------------+--------------+
| 2000-07-03        | 0000-01-05   | 0000-02-28    | 0000-03-01    |
+------------------+-------------+--------------+--------------+
```

FROM_ISO8601_DATE | Date Time Functions

FROM_ISO8601_DATE

Description

Converts an ISO8601 formatted date expression to a DATE type date expression.

Syntax

```
from_iso8601_date(<dt>)
```

Parameters

Parameter	Description
<dt>	An ISO8601 formatted date

Return Value

A DATE type date expression.

Examples

```
SELECT from_iso8601_date('0000-01'),from_iso8601_date('0000-W01'),from_iso8601_date('0000-059');
```

```
+-----------------------------+-----------------------------+-----------------------------+
| from_iso8601_date('0000-01') | from_iso8601_date('0000-W01') | from_iso8601_date('0000-059') |
+-----------------------------+-----------------------------+-----------------------------+
| 0000-01-01                  | 0000-01-03                  | 0000-02-28                  |
+-----------------------------+-----------------------------+-----------------------------+
```

FROM_MICROSECOND

Description

The function converts a Unix timestamp (in microseconds) into a `DATETIME` value.

Syntax

```
FROM_MICROSECOND(<unix_timestamp>)
```

Parameters

Parameter	Description
`<unix_timestamp>`	Required. The Unix timestamp representing the number of microseconds elapsed since 1970-01-01 00:00:00 UTC.

Return Value

- Returns a DATETIME value representing the date and time corresponding to the given Unix timestamp.
- If `<unix_timestamp>` is NULL, the function returns NULL.
- If `<unix_timestamp>` is out of valid range, the function returns an error.

Example

```
SELECT FROM_MICROSECOND(1700000000000000);
```

```
+-----------------------------------+
| from_microsecond(1700000000000000) |
+-----------------------------------+
| 2023-11-15 06:13:20               |
+-----------------------------------+
```

FROM_MILLISECOND

Description

The function converts a Unix timestamp (in milliseconds) into a `DATETIME` value.

Syntax

```
FROM_MILLISECOND(<millisecond>)
```

Parameters

Parameter	Description
`<millisecond>`	Required. The Unix timestamp representing the number of milliseconds elapsed since 1970-01-01 00:00:00 UTC.

Return Value

- Returns a DATETIME value representing the date and time corresponding to the given Unix timestamp.
- If is NULL, the function returns NULL.
- If is out of valid range, the function returns an error.

Example

```
SELECT FROM_MILLISECOND(1700000000000);
```

```
+--------------------------------+
| from_millisecond(1700000000000) |
+--------------------------------+
| 2023-11-15 06:13:20            |
+--------------------------------+
```

FROM_SECOND

Description

The function converts a Unix timestamp (in seconds) into a `DATETIME` value.

Syntax

```
FROM_SECOND(<unix_timestamp>)
```

Parameters

Parameter	Description
`<unix_timestamp>`	Required. The Unix timestamp representing the number of seconds elapsed since 1970-01-01 00:00:00 UTC.

Return Value

- Returns a DATETIME value representing the date and time corresponding to the given Unix timestamp.
- If `<unix_timestamp>` is NULL, the function returns NULL.
- If `<unix_timestamp>` is out of valid range, the function returns an error.

Example

```
SELECT FROM_SECOND(1700000000);
```

```
+-------------------------+
| from_second(1700000000) |
+-------------------------+
| 2023-11-15 06:13:20     |
+-------------------------+
```

FROM_UNIXTIME

Description

Converts a Unix timestamp to the corresponding TIME format. Special cases:

- The currently supported range for unix_timestamp is [0, 32536771199]. Unix timestamps outside this range will return NULL.

Syntax

```
FROM_UNIXTIME(<unix_timestamp>[, <string_format>])
```

Parameters

Parameter	Description
<unix_timestamp>	unix timestamp
<string_format>	The format format, with the default being %Y-%m-%d %H:%i:%s.

Return Value

Returns the date in the specified format.

Examples

```
mysql> select from_unixtime(1196440219);
+---------------------------+
| from_unixtime(1196440219) |
+---------------------------+
| 2007-12-01 00:30:19       |
+---------------------------+

mysql> select from_unixtime(1196440219, 'yyyy-MM-dd HH:mm:ss');
+--------------------------------------------------+
| from_unixtime(1196440219, 'yyyy-MM-dd HH:mm:ss') |
+--------------------------------------------------+
| 2007-12-01 00:30:19                              |
+--------------------------------------------------+

mysql> select from_unixtime(1196440219, '%Y-%m-%d');
+---------------------------------------+
| from_unixtime(1196440219, '%Y-%m-%d') |
+---------------------------------------+
| 2007-12-01                            |
+---------------------------------------+

mysql> select from_unixtime(1196440219, '%Y-%m-%d %H:%i:%s');
+------------------------------------------------+
| from_unixtime(1196440219, '%Y-%m-%d %H:%i:%s') |
```

```
+-------------------------------------------------+
| 2007-12-01 00:30:19                             |
+-------------------------------------------------+
```

HOUR

Description

Obtains the hour information from the given datetime.

Syntax

```
HOUR(<dt>)
```

Parameters

Parameter	Description
<dt>	The date to be calculated.

Return Value

Returns the hour information from the given date. The return value ranges from 0 to 23. When the parameter is of type TIME, the return value can be greater than 24.

Examples

```
select hour('2018-12-31 23:59:59');
```

```
+-----------------------------+
| hour('2018-12-31 23:59:59') |
+-----------------------------+
|                          23 |
+-----------------------------+
```

```
select cast(4562632 as time),hour(cast(4562632 as time)), minute(cast(4562632 as time)),second(cast(4562632 as time));
```

```
+----------------------+----------------------------+------------------------------+------------------------------+
| cast(4562632 as TIME) | hour(cast(4562632 as TIME)) | minute(cast(4562632 as TIME)) | second(cast(4562632 as TIME)) |
+----------------------+----------------------------+------------------------------+------------------------------+
| 456:26:32             |                        456 |                           26 |                           32 |
+----------------------+----------------------------+------------------------------+------------------------------+
```

HOUR_CEIL

Description

Converts the date to the nearest rounded-up timestamp of the specified time interval period.

Syntax

```
HOUR_CEIL(<datetime>)
HOUR_CEIL(<datetime>, <origin>)
HOUR_CEIL(<datetime>, <period>)
HOUR_CEIL(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	A valid date expression
<period>	Specifies how many hours make up each period
<origin>	The starting point of time. If not provided, the default is 0001-01-01T00:00:00

Return Value

Returns the nearest rounded-up timestamp of the specified time interval period.

Examples

```
select hour_ceil("2023-07-13 22:28:18", 5);
```

```
+-----------------------------------------------------------+
| hour_ceil(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2023-07-14 02:00:00                                       |
+-----------------------------------------------------------+
```

HOUR_FLOOR

Description

Converts the date to the nearest rounded-down timestamp of the specified time interval period.

Syntax

```
HOUR_FLOOR(<datetime>)
HOUR_FLOOR(<datetime>, <origin>)
HOUR_FLOOR(<datetime>, <period>)
HOUR_FLOOR(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	A valid date expression
<period>	Specifies how many hours make up each period
<origin>	The starting point of time. If not provided, the default is 0001-01-01T00:00:00

Return Value

Returns the nearest rounded-down timestamp of the specified time interval period.

Examples

```
select hour_floor("2023-07-13 22:28:18", 5);
```

```
+-----------------------------------------------------------+
| hour_floor(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2023-07-13 21:00:00                                       |
+-----------------------------------------------------------+
```

HOURS_ADD

Description

Returns a new datetime value that is the result of adding a specified number of hours to the input datetime.

Syntax

```
HOURS_ADD(<date>, <hours>)
```

Parameters

Parameter	Description
<date>	The input datetime value, which can be of type DATETIME or DATE
<hours>	The number of hours to add, of type INT

Return Value

Returns a value of type DATETIME, representing the time value after adding the specified number of hours to the input datetime.

Example

```
SELECT HOURS_ADD('2020-02-02 02:02:02', 1);
```

```
+-----------------------------------------------------------+
| hours_add(cast('2020-02-02 02:02:02' as DATETIMEV2(0)), 1) |
+-----------------------------------------------------------+
| 2020-02-02 03:02:02                                       |
+-----------------------------------------------------------+
```

HOURS_DIFF

Description

Calculates the difference in hours between the start time and the end time.

Syntax

```
HOURS_DIFF(<end_date>, <start_date>)
```

Parameters

Parameter	Description
<end_date>	The end time, which can be of type DATETIME or DATE
<start_date>	The start time, which can be of type DATETIME or DATE

Return Value

Returns an INT type representing the number of hours between the start time and the end time.

Example

```
SELECT HOURS_DIFF('2020-12-25 22:00:00', '2020-12-25 21:00:00');
```

```
+-----------------------------------------------------------------------------------------------------+
| hours_diff(cast('2020-12-25 22:00:00' as DATETIMEV2(0)), cast('2020-12-25 21:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------+
|                                                                                                   1 |
+-----------------------------------------------------------------------------------------------------+
```

HOURS_SUB

Description

Returns a new datetime value that is the result of subtracting a specified number of hours from the input datetime.

Syntax

```
HOURS_SUB(<date>, <hours>)
```

Parameters

Parameter	Description
<date>	The input datetime value, which can be of type DATETIME or DATE
<hours>	The number of hours to subtract, of type INT

Return Value

Returns a value of type DATETIME, representing the time value after subtracting the specified number of hours from the input datetime.

Example

```
SELECT HOURS_SUB('2020-02-02 02:02:02', 1);
```

```
+-----------------------------------------------------------+
| hours_sub(cast('2020-02-02 02:02:02' as DATETIMEV2(0)), 1) |
+-----------------------------------------------------------+
| 2020-02-02 01:02:02                                       |
+-----------------------------------------------------------+
```

LAST_DAY

Description

Returns the date of the last day of the month for the given input date. The returned day varies depending on the month: - 28th - For February in non-leap years - 29th - For February in leap years - 30th - For April, June, September, and November - 31st - For January, March, May, July, August, October, and December

Syntax

```
LAST_DAY(<date>)
```

Parameters

Parameter	Description
<date>	Input datetime value, type can be DATETIME or DATE

Return Value

Returns a value of type DATE representing the last day of the month for the given input date.

Example

```
SELECT LAST_DAY('2000-02-03');
```

```
+-----------------------------------------------+
| last_day(cast('2000-02-03' as DATETIMEV2(0))) |
+-----------------------------------------------+
| 2000-02-29                                    |
+-----------------------------------------------+
```

MAKEDATE

Description

Returns a date based on the specified year and the day of the year (dayofyear).

Special cases: - Returns NULL when `dayofyear` is less than or equal to 0. - Automatically rolls over to the next year if `dayofyear` exceeds the number of days in the year.

Syntax

```
MAKEDATE(<year>, <day_of_year>)
```

Parameters

Parameter	Description
`<year>`	The specified year, of type INT
`<day_of_year>`	The day of the year (1-366), of type INT

Return Value

Returns a value of type DATE, constructed from the specified year and the given day of the year.

Example

```
SELECT MAKEDATE(2021, 1), MAKEDATE(2021, 100), MAKEDATE(2021, 400);
```

```
+------------------+--------------------+--------------------+
| makedate(2021, 1) | makedate(2021, 100) | makedate(2021, 400) |
+------------------+--------------------+--------------------+
| 2021-01-01        | 2021-04-10          | 2022-02-04          |
+------------------+--------------------+--------------------+
```

MICROSECOND

Description

Extracts the microsecond part from a datetime value. The returned range is from 0 to 999999.

Syntax

```
MICROSECOND(<date>)
```

Parameters

Parameter	Description
<date>	The input datetime value, of type DATETIMEV2, with a precision greater than 0

Return Value

Returns an INT type representing the microsecond part of the datetime value. The range is from 0 to 999999. For inputs with a precision less than 6, the missing digits are padded with zeros.

Example

```
SELECT MICROSECOND(CAST('1999-01-02 10:11:12.000123' AS DATETIMEV2(6))) AS microsecond;
```

```
+-------------+
| microsecond |
+-------------+
|         123 |
+-------------+
```

MICROSECOND_TIMESTAMP

Description

The function converts a `DATETIME` value into a Unix timestamp (in microseconds) starting from `1970-01-01 00:00:00 UTC`.

Syntax

```
MICROSECOND_TIMESTAMP(<datetime>)
```

Parameters

Parameter	Description
`<datetime>`	Required. The DATETIME value to be converted into a Unix timestamp.

Return Value

- Returns an integer representing the Unix timestamp (in microseconds) corresponding to the input datetime value.
- If `<datetime>` is NULL, the function returns NULL.
- If `<datetime>` is out of valid range, the function may return an error or unexpected value.

Example

```
SELECT MICROSECOND_TIMESTAMP('2025-01-23 12:34:56.123456');
```

```
+-----------------------------------------------------------------------+
| microsecond_timestamp(cast('2025-01-23 12:34:56.123456' as DATETIMEV2(6))) |
+-----------------------------------------------------------------------+
|                                                      1737606896123456 |
+-----------------------------------------------------------------------+
```

MICROSECONDS_ADD

Description

Adds a specified number of microseconds to a datetime value and returns a new datetime value.

Syntax

```
MICROSECONDS_ADD(<basetime>, <delta>)
```

Parameters

Parameter	Description
<basetime>	The input datetime value, of type DATETIMEV2
<delta>	The number of microseconds to add, of type INT; 1 second = 1,000,000 microseconds

Return Value

Returns a value of type DATETIMEV2, representing the time value after adding the specified number of microseconds to the input datetime. The precision of the return value is the same as that of the input parameter basetime.

Example

```
SELECT NOW(3) AS current_time, MICROSECONDS_ADD(NOW(3), 100000) AS after_add;
```

```
+-------------------------+----------------------------+
| current_time            | after_add                  |
+-------------------------+----------------------------+
| 2025-01-16 11:48:10.505 | 2025-01-16 11:48:10.605000 |
+-------------------------+----------------------------+
```

Note: - `NOW(3)` returns the current time with a precision of 3 decimal places. - After adding 100000 microseconds (0.1 seconds), the time increases by 0.1 seconds.

MICROSECONDS_DIFF

Description

Calculates the microsecond difference between two datetime values. The result is the number of microseconds from <start_date> subtracted from <end_date>.

Syntax

```
MICROSECONDS_DIFF(<enddate>, <startdate>)
```

Parameters

Parameter	Description
<end_date>	The end time, of type DATETIMEV2
<start_date>	The start time, of type DATETIMEV2

Return Value

Returns an INT type representing the microsecond difference between the two times. - Returns a positive number if <end_date> is greater than <start_date>. - Returns a negative number if <end_date> is less than <start_date>. - 1 second = 1,000,000 microseconds.

Example

```
SELECT MICROSECONDS_DIFF('2020-12-25 21:00:00.623000', '2020-12-25 21:00:00.123000');
```

```
+-----------------------------------------------------------------------------------------------------------------------+
| microseconds_diff(cast('2020-12-25 21:00:00.623000' as DATETIMEV2(3)), cast('2020-12-25 21:00:00.123000' as DATETIMEV2(3))) |
+-----------------------------------------------------------------------------------------------------------------------+
|                                                                                                                500000 |
+-----------------------------------------------------------------------------------------------------------------------+
```

MICROSECONDS_SUB

Description

Subtracts a specified number of microseconds from a datetime value and returns a new datetime value.

Syntax

```
MICROSECONDS_SUB(<basetime>, <delta>)
```

Parameters

Parameter	Description
<basetime>	The input datetime value, of type DATETIMEV2
<delta>	The number of microseconds to subtract, of type INT; 1 second = 1,000,000 microseconds

Return Value

Returns a value of type DATETIMEV2, representing the time value after subtracting the specified number of microseconds from the input datetime. The precision of the return value is the same as that of the input parameter basetime.

Example

```
SELECT NOW(3) AS current_time, MICROSECONDS_SUB(NOW(3), 100000) AS after_sub;
```

```
+-------------------------+----------------------------+
| current_time            | after_sub                  |
+-------------------------+----------------------------+
| 2025-01-16 11:52:22.296 | 2025-01-16 11:52:22.196000 |
+-------------------------+----------------------------+
```

Note: - `NOW(3)` returns the current time with a precision of 3 decimal places. - After subtracting 100000 microseconds (0.1 seconds), the time decreases by 0.1 seconds. - The function's result is dependent on the precision of the input time.

MILLISECOND_TIMESTAMP

Description

The `MILLISECOND_TIMESTAMP` function converts a `DATETIME` value into a Unix timestamp (in milliseconds) starting from `1970-01-01 00:00:00 UTC`.

Syntax

```
MILLISECOND_TIMESTAMP(<datetime>)
```

Parameters

Parameter	Description
`<datetime>`	Required. The DATETIME value to be converted into a Unix timestamp (in milliseconds).

Return Value

- Returns an integer representing the Unix timestamp (in milliseconds) corresponding to the input datetime value.
- If `<datetime>` is NULL, the function returns NULL.
- If `<datetime>` is out of valid range, the function may return an error or unexpected value.

Example

```
SELECT MILLISECOND_TIMESTAMP('2025-01-23 12:34:56');
```

```
+-----------------------------------------------------------------+
| millisecond_timestamp(cast('2025-01-23 12:34:56' as DATETIMEV2(0))) |
+-----------------------------------------------------------------+
|                                                   1737606896000 |
+-----------------------------------------------------------------+
```

MILLISECONDS_ADD

Description

Adds a specified number of milliseconds to a datetime value and returns a new datetime value.

Syntax

```
MILLISECONDS_ADD(<basetime>, <delta>)
```

Parameters

Parameter	Description
<basetime>	The input datetime value, of type DATETIMEV2
<delta>	The number of milliseconds to add, of type INT; 1 second = 1,000 milliseconds = 1,000,000 microseconds

Return Value

Returns a value of type DATETIMEV2, representing the time value after adding the specified number of milliseconds to the input datetime. The precision of the return value is the same as that of the input parameter basetime.

Example

```
SELECT MILLISECONDS_ADD('2023-09-08 16:02:08.435123', 1);
```

```
+----------------------------------------------------------------------------+
| milliseconds_add(cast('2023-09-08 16:02:08.435123' as DATETIMEV2(6)), 1) |
+----------------------------------------------------------------------------+
| 2023-09-08 16:02:08.436123                                                 |
+----------------------------------------------------------------------------+
```

Note: - In the example, after adding 1 millisecond, the time increases from .435123 to .436123. - 1 millisecond equals 1000 microseconds. - The function's result is dependent on the precision of the input time; the example uses a precision of 6 decimal places.

MILLISECONDS_DIFF

Description

Calculates the millisecond difference between two datetime values. The result is the number of milliseconds from <start_date> subtracted from <end_date>.

Syntax

```
MILLISECONDS_DIFF(<enddate>, <startdate>)
```

Parameters

Parameter	Description
<end_date>	The end time, of type DATETIMEV2
<start_date>	The start time, of type DATETIMEV2

Return Value

Returns an INT type representing the millisecond difference between the two times. - Returns a positive number if <end_date> is greater than <start_date>. - Returns a negative number if <end_date> is less than <start_date>. - 1 second = 1,000 milliseconds. - 1 millisecond = 1,000 microseconds.

Example

```
SELECT MILLISECONDS_DIFF('2020-12-25 21:00:00.623000', '2020-12-25 21:00:00.123000');
```

```
+-----------------------------------------------------------------------------------------------------------------------------+
| milliseconds_diff(cast('2020-12-25 21:00:00.623000' as DATETIMEV2(3)), cast('2020-12-25 21:00:00.123000' as DATETIMEV2(3))) |
+-----------------------------------------------------------------------------------------------------------------------------+
|                                                                                                                         500 |
+-----------------------------------------------------------------------------------------------------------------------------+
```

Note: - The time difference in the example is 0.5 seconds, which equals 500 milliseconds. - The function's result is dependent on the precision of the input time; the example uses a precision of 3 decimal places. - The result only returns the millisecond difference and does not include the microsecond part.

MILLISECONDS_SUB

Description

Subtracts a specified number of milliseconds from a datetime value and returns a new datetime value.

Syntax

```
MILLISECONDS_SUB(<basetime>, <delta>)
```

Parameters

Parameter	Description
<basetime>	The input datetime value, of type DATETIMEV2
<delta>	The number of milliseconds to subtract, of type INT; 1 second = 1,000 milliseconds = 1,000,000 microseconds

Return Value

Returns a value of type DATETIMEV2, representing the time value after subtracting the specified number of milliseconds from the input datetime. The precision of the return value is the same as that of the input parameter basetime.

Example

```
SELECT MILLISECONDS_SUB('2023-09-08 16:02:08.435123', 1);
```

```
+----------------------------------------------------------------------------+
| milliseconds_sub(cast('2023-09-08 16:02:08.435123' as DATETIMEV2(6)), 1) |
+----------------------------------------------------------------------------+
| 2023-09-08 16:02:08.434123                                                 |
+----------------------------------------------------------------------------+
1 row in set (0.11 sec)
```

Note: - In the example, after subtracting 1 millisecond, the time decreases from .435123 to .434123. - 1 millisecond equals 1000 microseconds. - The function's result is dependent on the precision of the input time; the example uses a precision of 6 decimal places. - The result retains microsecond-level precision.

MINUTE

Description

Extracts the minute part from a datetime value. The returned value ranges from 0 to 59.

Syntax

```
MINUTE(<datetime>)
```

Parameters

Parameter	Description
<datetime>	The input datetime value, which can be of type DATE, DATETIME, DATETIMEV2, or TIME

Return Value

Returns an INT type representing the minute value, with a range of 0-59.

Example

```
SELECT MINUTE('2018-12-31 23:59:59');
```

```
+-----------------------------------------------------+
| minute(cast('2018-12-31 23:59:59' as DATETIMEV2(0))) |
+-----------------------------------------------------+
|                                                  59 |
+-----------------------------------------------------+
```

Note: - The input parameter can be of various time-related types. - The returned value is always an integer between 0 and 59. - If the input parameter is NULL, the function returns NULL.

MINUTE_CEIL

Description

Rounds up a datetime value to the nearest specified minute interval. If a starting time (origin) is provided, it uses that time as the reference for calculating the interval.

Syntax

```
MINUTE_CEIL(<datetime>)
MINUTE_CEIL(<datetime>, <origin>)
MINUTE_CEIL(<datetime>, <period>)
MINUTE_CEIL(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	The datetime value to round up, of type DATETIME or DATETIMEV2
<period>	The minute interval value, of type INT, representing the number of minutes in each interval
<origin>	The starting point for the interval, of type DATETIME or DATETIMEV2; defaults to 0001-01-01 00:00:00

Return Value

Returns a value of type DATETIMEV2, representing the rounded-up datetime value based on the specified minute interval. The precision of the return value is the same as that of the input parameter datetime.

Example

```
SELECT MINUTE_CEIL("2023-07-13 22:28:18", 5);
```

```
+-----------------------------------------------------------+
| minute_ceil(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2023-07-13 22:30:00                                       |
+-----------------------------------------------------------+
```

Note: - If no period is specified, it defaults to a 1-minute interval. - The period must be a positive integer. - The result is always rounded up to a future time.

Best Practices

See also date_ceil

MINUTE_FLOOR

Description

Rounds down a datetime value to the nearest specified minute interval. If a starting time (origin) is provided, it uses that time as the reference for calculating the interval.

Syntax

```
MINUTE_FLOOR(<datetime>)
MINUTE_FLOOR(<datetime>, <origin>)
MINUTE_FLOOR(<datetime>, <period>)
MINUTE_FLOOR(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	The datetime value to round down, of type DATETIME or DATETIMEV2
<period>	The minute interval value, of type INT, representing the number of minutes in each interval
<origin>	The starting point for the interval, of type DATETIME or DATETIMEV2; defaults to 0001-01-01 00:00:00

Return Value

Returns a value of type DATETIMEV2, representing the rounded-down datetime value.

Example

```
SELECT MINUTE_FLOOR("2023-07-13 22:28:18", 5);
```

```
+------------------------------------------------------------+
| minute_floor(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+------------------------------------------------------------+
| 2023-07-13 22:25:00                                        |
+------------------------------------------------------------+
```

Note: - If no period is specified, it defaults to a 1-minute interval. - The period must be a positive integer. - The result is always rounded down to a past time. - Unlike MINUTE_CEIL, MINUTE_FLOOR always discards the portion that exceeds the interval.

Best Practices

See also date_floor

MINUTES_ADD

Description

Adds a specified number of minutes to a datetime value and returns a new datetime value.

Syntax

```
MINUTES_ADD(<datetime>, <minutes>)
```

Parameters

Parameter	Description
<datetime>	The input datetime value, which can be of type DATE, DATETIME, or DATETIMEV2
<minutes>	The number of minutes to add, of type INT; can be positive or negative

Return Value

Returns a value of type DATETIME, representing the datetime value after adding the specified number of minutes.

Example

```
SELECT MINUTES_ADD("2020-02-02", 1);
```

```
+-----------------------------------------------------+
| minutes_add(cast('2020-02-02' as DATETIMEV2(0)), 1) |
+-----------------------------------------------------+
| 2020-02-02 00:01:00                                 |
+-----------------------------------------------------+
```

Note: - When the number of minutes added is negative, it effectively subtracts the corresponding number of minutes. - The function automatically handles cases that cross hours and days. - If the input parameter is NULL, the function returns NULL.

MINUTES_DIFF

Description

Calculates the minute difference between two datetime values. The result is the number of minutes from <start_date> subtracted from <end_date>.

Syntax

```
MINUTES_DIFF(<enddate>, <startdate>)
```

Parameters

Parameter	Description
<end_date>	The end time, which can be of type DATE, DATETIME, or DATETIMEV2
<start_date>	The start time, which can be of type DATE, DATETIME, or DATETIMEV2

Return Value

Returns an INT type representing the minute difference between the two times. - Returns a positive number if <end_date> is greater than <start_date>. - Returns a negative number if <end_date> is less than <start_date>.

Example

```
SELECT MINUTES_DIFF('2020-12-25 22:00:00', '2020-12-25 21:00:00');
```

```
+-----------------------------------------------------------------------------------------------------+
| minutes_diff(cast('2020-12-25 22:00:00' as DATETIMEV2(0)), cast('2020-12-25 21:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------+
|                                                                                                  60 |
+-----------------------------------------------------------------------------------------------------+
```

Note: - The calculation only considers complete minutes; seconds and milliseconds are ignored. - If either input parameter is NULL, the function returns NULL. - It can handle time differences that span days, months, or years.

MINUTES_SUB

Description

Subtracts a specified number of minutes from a datetime value and returns a new datetime value.

Syntax

```
MINUTES_SUB(<date>, <minutes>)
```

Parameters

Parameter	Description
<datetime>	The input datetime value, which can be of type DATE, DATETIME, or DATETIMEV2
<minutes>	The number of minutes to subtract, of type INT; can be positive or negative

Return Value

Returns a value of type DATETIME, representing the datetime value after subtracting the specified number of minutes.

Example

```
SELECT MINUTES_SUB("2020-02-02 02:02:02", 1);
```

```
+-----------------------------------------------------------+
| minutes_sub(cast('2020-02-02 02:02:02' as DATETIMEV2(0)), 1) |
+-----------------------------------------------------------+
| 2020-02-02 02:01:02                                       |
+-----------------------------------------------------------+
```

Note: - When the number of minutes subtracted is negative, it effectively adds the corresponding number of minutes. - The function automatically handles cases that cross hours and days. - If the input parameter is NULL, the function returns NULL. - The result retains the seconds portion of the original time.

MONTH

Description

Extracts the month value from a datetime value. The returned value ranges from 1 to 12, representing the 12 months of the year.

Syntax

```
MONTH(<date>)
```

Parameters

Parameter	Description
<date>	The input datetime value, which can be of type DATE, DATETIME, or DATETIMEV2

Return Value

Returns an INT type representing the month value: - Range: 1 to 12 - 1 represents January, and 12 represents December. - If the input is NULL, the function returns NULL.

Example

```
SELECT MONTH('1987-01-01');
```

```
+-----------------------------------------+
| month(cast('1987-01-01' as DATETIMEV2(0))) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

MONTH_CEIL

Description

Rounds up a datetime value to the nearest specified month interval. If a starting time (origin) is provided, it uses that time as the reference for calculating the interval.

Syntax

```
MONTH_CEIL(<datetime>)
MONTH_CEIL(<datetime>, <origin>)
MONTH_CEIL(<datetime>, <period>)
MONTH_CEIL(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	The datetime value to round up, of type DATETIME or DATETIMEV2
<period>	The month interval value, of type INT, representing the number of months in each interval
<origin>	The starting point for the interval, of type DATETIME or DATETIMEV2; defaults to 0001-01-01 00:00:00

Return Value

Returns a value of type DATETIME, representing the rounded-up datetime value. The time portion of the result will be set to 00:00:00.

Example

```
SELECT MONTH_CEIL("2023-07-13 22:28:18", 5);
```

```
+-----------------------------------------------------------+
| month_ceil(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2023-10-01 00:00:00                                       |
+-----------------------------------------------------------+
```

Note: - If no period is specified, it defaults to a 1-month interval. - The period must be a positive integer. - The result is always rounded up to a future time. - The time portion of the returned value is always set to 00:00:00.

Best Practices

See also date_ceil

MONTH_FLOOR

Description

Rounds down a datetime value to the nearest specified month interval. If a starting time (origin) is provided, it uses that time as the reference for calculating the interval.

Syntax

```
MONTH_FLOOR(<datetime>)
MONTH_FLOOR(<datetime>, <origin>)
MONTH_FLOOR(<datetime>, <period>)
MONTH_FLOOR(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	The datetime value to round down, of type DATETIME or DATETIMEV2
<period>	The month interval value, of type INT, representing the number of months in each interval
<origin>	The starting point for the interval, of type DATETIME or DATETIMEV2; defaults to 0001-01-01 00:00:00

Return Value

Returns a value of type DATETIME, representing the rounded-down datetime value. The time portion of the result will be set to 00:00:00.

Example

```
SELECT MONTH_FLOOR("2023-07-13 22:28:18", 5);
```

```
+-----------------------------------------------------------+
| month_floor(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2023-05-01 00:00:00                                       |
+-----------------------------------------------------------+
```

Note: - If no period is specified, it defaults to a 1-month interval. - The period must be a positive integer. - The result is always rounded down to a past time. - The time portion of the returned value is always set to 00:00:00. - Unlike MONTH_CEIL, MONTH_FLOOR always discards the portion that exceeds the interval.

Best Practices

See also date_floor

MONTHNAME

Description

Returns the English name of the month corresponding to a given date. The returned value is the full English name of the month (from January to December).

Syntax

```
MONTHNAME(<date>)
```

Parameters

Parameter	Description
<date>	The input datetime value, which can be of type DATE, DATETIME, or DATETIMEV2

Return Value

Returns a value of type VARCHAR representing the English name of the month: - Possible return values: January, February, March, April, May, June, July, August, September, October, November, December - If the input is NULL, the function returns NULL. - The first letter of the return value is capitalized, and the remaining letters are in lowercase.

Example

```
SELECT MONTHNAME('2008-02-03 00:00:00');
```

```
+-----------------------------------------------------+
| monthname(cast('2008-02-03 00:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------+
| February                                            |
+-----------------------------------------------------+
```

MONTHS_ADD

Description

The MONTHS_ADD function is used to add or subtract a specified number of months to a given date and returns the resulting date.

Syntax

```
MONTHS_ADD(<datetime/date>, <nums>)
```

Parameters

Parameter	Description
<datetime/date>	The date value to which months will be added or subtracted
<nums>	The number of months to add or subtract

Return Value

The return value is of the same type as the input . Special cases: - If the input is 0000-00-00 or 0000-00-00 00:00:00, the function returns NULL. - If the input is NULL, the function returns NULL. - If the input is MONTHS_ADD(“9999-12-31”, 1), the function will return NULL.

Example

```
select months_add("2020-01-31 02:02:02", 1),months_add("2020-01-31", 1),months_add("2020-01-31", -1);
```

```
+-----------------------------------------------------------+-------------------------------------------+-----------------------
-----------------------+
| months_add(cast('2020-01-31 02:02:02' as DATETIMEV2(0)), 1) | months_add(cast('2020-01-31' as DATEV2), 1) | months_add(cast('2020-
01-31' as DATEV2), -1) |
+-----------------------------------------------------------+-------------------------------------------+-----------------------
-----------------------+
| 2020-02-29 02:02:02                                       | 2020-02-29                                | 2019-12-31
|
+-----------------------------------------------------------+-------------------------------------------+-----------------------
-----------------------+
```

MONTHS_DIFF

Description

The MONTHS_DIFF function calculates the number of complete months between two dates. It accepts two date arguments and returns the difference in months as an integer.

Syntax

```
MONTHS_DIFF(<enddate>, <startdate>)
```

Parameters

参数	说明
<enddate>	The ending date, representing the later date in the difference calculation. Supports DATE (e.g., YYYY-MM-DD) or DATETIME (e.g., YYYY-MM-DD HH:MM:SS) types.
<startdate>	The starting date, representing the earlier date in the difference calculation. Supports DATE (e.g., YYYY-MM-DD) or DATETIME (e.g., YYYY-MM-DD HH:MM:SS) types.

Return Value

returns the number of months resulting from <enddate> minus <startdate> - When either <enddate> or <startdate> is NULL, or both are NULL, it returns NULL

Example

```
select months_diff('2020-12-25','2020-10-25'),months_diff('2020-10-25 10:00:00','2020-12-25 11:00:00');
```

```
+-----------------------------------------------------------------------------------------+-----------------------------------------
-------------------------------------------------------------+
| months_diff(cast('2020-12-25' as DATETIMEV2(0)), cast('2020-10-25' as DATETIMEV2(0))) | months_diff(cast('2020-10-25 10:00:00' as
DATETIMEV2(0)), cast('2020-12-25 11:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------+-----------------------------------------
-------------------------------------------------------------+
|                                                                                       2 |
-2 |
+-----------------------------------------------------------------------------------------+-----------------------------------------
-------------------------------------------------------------+
```

MONTHS_SUB

Description

The MONTHS_SUB function is used to add or subtract a specified number of months to a given date and returns the resulting date.

Syntax

```
MONTHS_SUB(<datetime/date>,  <nums>)
```

Parameters

Parameter	Description
<datetime/date>	The date value to which months will be added or subtracted
<nums>	The number of months to add or subtract

Return Value

The return value is of the same type as the input . Special cases: - If the input is 0000-00-00 or 0000-00-00 00:00:00, the function returns NULL. - If the input is NULL, the function returns NULL. - If the input is MONTHS_SUB(“9999-12-31”, -1), the function will return NULL.

Example

```
select months_sub("2020-01-31 02:02:02", 1),months_sub("2020-01-31", 1),months_sub("2020-01-31", -1);
```

```
+-----------------------------------------------------------+-------------------------------------------+-----------------------
-----------------------+
| months_sub(cast('2020-01-31 02:02:02' as DATETIMEV2(0)), 1) | months_sub(cast('2020-01-31' as DATEV2), 1) | months_sub(cast('2020-
01-31' as DATEV2), -1) |
+-----------------------------------------------------------+-------------------------------------------+-----------------------
-----------------------+
| 2019-12-31 02:02:02                                       | 2019-12-31                                | 2020-02-29
|
+-----------------------------------------------------------+-------------------------------------------+-----------------------
-----------------------+
```

NOW

Description

The function retrieves the current system time and returns it as a datetime value (`DATETIME`). An optional precision can be specified to adjust the number of digits in the fractional seconds part of the return value.

Syntax

```
NOW([<precision>])
```

Parameters

Parameter	Description
<precision>	Optional parameter specifying the precision of the fractional seconds part in the return value. The range is 0 to 6, and the default is 0 (no fractional seconds). Limited by the JDK implementation: if FE is built with JDK8, the precision supports up to milliseconds (3 fractional digits), and higher precision digits will be filled with 0. If higher precision is required, please use JDK11.

Return Value

- Returns the current system time as a DATETIME type.
- If the specified `<precision>` is out of range (e.g., negative or greater than 6), the function will return an error.

Example

```
select NOW(), NOW(3), NOW(6);
```

```
+---------------------+-------------------------+----------------------------+
| now()               | now(3)                  | now(6)                     |
```

```
+---------------------+-------------------------+----------------------------+
| 2025-01-23 11:08:35 | 2025-01-23 11:08:35.561 | 2025-01-23 11:08:35.562000 |
+---------------------+-------------------------+----------------------------+
```

QUARTER

Description

The function returns the quarter (1 to 4) of the given date. Each quarter includes three months: - Q1: January to March - Q2: April to June - Q3: July to September - Q4: October to December

Syntax

```
QUARTER(<datetime>)
```

Parameters

Parameter	Description
<datetime>	A valid DATE or DATETIME value to determine the quarter.

Return Value

- Returns an integer representing the quarter of the input date, ranging from 1 to 4.
- If the input is NULL, the function returns NULL.
- If the input is an invalid date (e.g., 0000-00-00), the function returns NULL.

Example

```
SELECT QUARTER('2025-01-16'),QUARTER('2025-01-16 01:11:10');
```

```
+---------------------------------------+-----------------------------------------------+
| quarter(cast('2025-01-16' as DATETIME)) | quarter(cast('2025-01-16 01:11:10' as DATETIME)) |
+---------------------------------------+-----------------------------------------------+
|                                     1 |                                             1 |
+---------------------------------------+-----------------------------------------------+
```

SEC_TO_TIME

Description

The `SEC_TO_TIME` function converts a value in seconds into a `TIME` type, returning the result in the format `HH:MM:SS`. The input seconds represent the time elapsed since the start of a day (`00:00:00`).

Syntax

```
SEC_TO_TIME(<seconds>)
```

Parameters

Parameter	Description
`<seconds>`	Required. The input number of seconds, representing the time elapsed since the start of a day (00:00:00). Supports positive or negative integers.

Return Value

- Returns a TIME value in the format `HH:MM:SS`, representing the time calculated from the start of a day (00:00:00).
- If `<seconds>` is NULL, the function returns NULL.

Example

```
SELECT SEC_TO_TIME(59738);
```

```
+--------------------+
| sec_to_time(59738) |
+--------------------+
| 16:35:38           |
+--------------------+
```

SECOND

Description

The function returns the second part of the specified datetime value. The range of seconds is 0 to 59.

Syntax

```
SECOND(<datetime>)
```

Parameters

Parameter	Description
<datetime>	The input date or datetime value. Supports DATE or DATETIME types.

Return Value

- Returns an integer representing the second part of the input datetime value, ranging from 0 to 59.
- If the input is NULL, the function returns NULL.
- If the input is an invalid date (e.g., 0000-00-00 00:00:00), the function returns NULL.

Example

```
select second('2018-12-31 23:59:59');
```

```
+-----------------------------------------------------+
| second(cast('2018-12-31 23:59:59' as DATETIMEV2(0))) |
+-----------------------------------------------------+
|                                                  59 |
+-----------------------------------------------------+
```

SECOND_CEIL

Description

The function aligns the input datetime value upwards to the nearest second boundary based on the specified period and returns the aligned datetime value.

Syntax

```
SECOND_CEIL(<datetime>[, <period>][, <origin_datetime>])
```

Parameters

Parameter	Description
<datetime>	Required. The input datetime value. Supports the DATETIME type.
<period>	Optional. Specifies the number of seconds in each period. Supports positive integers (INT). Defaults to 1 second.
<origin_datetime>	Optional. The starting point for alignment. Supports the DATETIME type. Defaults to 0001-01-01T00:00:00 if not specified.

Return Value

- Returns a datetime value representing the input datetime aligned upwards to the nearest specified second boundary.
- If <datetime> is NULL, the function returns NULL.
- If <datetime> is an invalid date (e.g., 0000-00-00T00:00:00), the function returns NULL.

Example

Only specifying <datetime>

```
SELECT SECOND_CEIL('2025-01-23 12:34:56');
```

```
+-----------------------------------------------------------+
| second_ceil(cast('2025-01-23 12:34:56' as DATETIMEV2(0))) |
+-----------------------------------------------------------+
| 2025-01-23 12:34:56                                       |
+-----------------------------------------------------------+
```

Specifying <datetime> and <origin_datetime>

```
SELECT SECOND_CEIL('2025-01-23 12:34:56', '2025-01-01 00:00:00');
```

```
+-----------------------------------------------------------------------------------------------------------+
| second_ceil(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), cast('2025-01-01 00:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------------+
| 2025-01-23 12:34:56                                                                                       |
+-----------------------------------------------------------------------------------------------------------+
```

Specifying <datetime> and <period>

```
SELECT SECOND_CEIL('2025-01-23 12:34:56', 5)
```

```
+-------------------------------------------------------------+
| second_ceil(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), 5) |
+-------------------------------------------------------------+
| 2025-01-23 12:35:00                                         |
+-------------------------------------------------------------+
```

Specifying <datetime>, <period> and <origin_datetime>

```
SELECT SECOND_CEIL('2025-01-23 12:34:56', 10, '2025-01-23 12:00:00');
```

```
+-----------------------------------------------------------------------------------------------------------------+
| second_ceil(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), 10, cast('2025-01-23 12:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------------------+
| 2025-01-23 12:35:00                                                                                             |
+-----------------------------------------------------------------------------------------------------------------+
```

SECOND_FLOOR

Description

The function aligns the input datetime value upwards to the nearest second boundary based on the specified period and returns the aligned datetime value.

Syntax

```
SECOND_FLOOR(<datetime>[, <period>][, <origin_datetime>])
```

Parameters

Parameter	Description
<datetime>	Required. The input datetime value. Supports the DATETIME type.
<period>	Optional. Specifies the number of seconds in each period. Supports positive integers (INT). Defaults to 1 second.
<origin_datetime>	Optional. The starting point for alignment. Supports the DATETIME type. Defaults to 0001-01-01T00:00:00 if not specified.

Return Value

- Returns a datetime value representing the input datetime aligned upwards to the nearest specified second boundary.
- If <datetime> is NULL, the function returns NULL.
- If <datetime> is an invalid date (e.g., 0000-00-00T00:00:00), the function returns NULL.

Example

Only specifying <datetime>

```
SELECT SECOND_FLOOR('2025-01-23 12:34:56');
```

```
+-----------------------------------------------------------+
| second_floor(cast('2025-01-23 12:34:56' as DATETIMEV2(0))) |
+-----------------------------------------------------------+
| 2025-01-23 12:34:56                                       |
+-----------------------------------------------------------+
```

Specifying <datetime> and <origin_datetime>

```
SELECT SECOND_FLOOR('2025-01-23 12:34:56', '2025-01-01 00:00:00');
```

```
+-----------------------------------------------------------------------------------------------------------+
| second_floor(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), cast('2025-01-01 00:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------------+
| 2025-01-23 12:34:56                                                                                       |
+-----------------------------------------------------------------------------------------------------------+
```

Specifying <datetime> and <period>

```
SELECT SECOND_FLOOR('2025-01-23 12:34:56', 5)
```

```
+-------------------------------------------------------------+
| second_floor(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), 5) |
+-------------------------------------------------------------+
| 2025-01-23 12:34:55                                         |
+-------------------------------------------------------------+
```

Specifying <datetime>, <period> and <origin_datetime>

```
SELECT SECOND_FLOOR('2025-01-23 12:34:56', 10, '2025-01-23 12:00:00');
```

```
+-----------------------------------------------------------------------------------------------------------------+
| second_floor(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), 10, cast('2025-01-23 12:00:00' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------------------+
| 2025-01-23 12:34:50                                                                                             |
+-----------------------------------------------------------------------------------------------------------------+
```

SECONDS_ADD

Description

The function adds or subtracts a specified number of seconds to/from a given datetime value and returns the resulting datetime.

Syntax

```
SECONDS_ADD(<datetime>, <seconds>)
```

Parameters

Parameter	Description
<datetime>	Required. The input datetime value. Supports the DATETIME and DATE type.
<seconds>	Required. The number of seconds to add or subtract. Supports integers (INT). Positive numbers add seconds, while negative numbers subtract seconds.

Return Value

- Returns a datetime value of the same type as the input .
- If <datetime> is NULL, the function returns NULL.
- If <datetime> is an invalid date (e.g., 0000-00-00T00:00:00), the function returns NULL.

Examples

```
SELECT SECONDS_ADD('2025-01-23 12:34:56', 30),SECONDS_ADD('2025-01-23 12:34:56', -30);
```

```
+-----------------------------------------------------------+-----------------------------------------------------------+
| seconds_add(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), 30) | seconds_add(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), -30) |
+-----------------------------------------------------------+-----------------------------------------------------------+
| 2025-01-23 12:35:26                                       | 2025-01-23 12:34:26                                       |
+-----------------------------------------------------------+-----------------------------------------------------------+
```

SECONDS_DIFF

Description

The function calculates the time difference between two datetime values and returns the difference in seconds.

Syntax

```
SECONDS_DIFF(<end_datetime>, <start_datetime>)
```

Parameters

Parameter	Description
<end_datetime>	Required. The ending datetime value. Supports the DATETIME and DATE type.
<start_datetime>	Required. The starting datetime value. Supports the DATETIME and DATE type.

Return Value

- Returns an integer representing the difference in seconds between two datetime values:
  - If <end_datetime> is later than <start_datetime>, returns a positive value.
  - If <end_datetime> is earlier than <start_datetime>, returns a negative value.
  - If <end_datetime> and <start_datetime> are equal, returns 0.
- If either parameter is NULL, the function returns NULL.
- If the input datetime values are invalid (e.g., 0000-00-00T00:00:00), the function returns NULL.

Example

```
SELECT SECONDS_DIFF('2025-01-23 12:35:56', '2025-01-23 12:34:56');
```

```
+-----------------------------------------------------------------------------------------------------------+
| seconds_diff(cast('2025-01-23 12:35:56' as DATETIMEV2(0)), cast('2025-01-23 12:34:56' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------------+
|                                                                                                        60 |
+-----------------------------------------------------------------------------------------------------------+
```

SECONDS_SUB

Description

The function subtracts or adds a specified number of seconds to/from a given datetime value and returns the resulting datetime.

Syntax

```
SECONDS_SUB(<datetime>, <seconds>)
```

Parameters

Parameter	Description
<datetime>	Required. The input datetime value. Supports the DATETIME and DATE type.
<seconds>	Required. The number of seconds to subtract or add. Supports integers (INT). Positive numbers add seconds, while negative numbers subtract seconds.

Return Value

- Returns a datetime value of the same type as the input .
- If <datetime> is NULL, the function returns NULL.
- If <datetime> is an invalid date (e.g., 0000-00-00T00:00:00), the function returns NULL.

Examples

```
SELECT SECONDS_SUB('2025-01-23 12:34:56', 30),SECONDS_SUB('2025-01-23 12:34:56', -30);
```

```
+-----------------------------------------------------------+-----------------------------------------------------------+
| seconds_sub(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), 30) | seconds_sub(cast('2025-01-23 12:34:56' as DATETIMEV2(0)), -30) |
+-----------------------------------------------------------+-----------------------------------------------------------+
| 2025-01-23 12:34:26                                       | 2025-01-23 12:35:26                                       |
+-----------------------------------------------------------+-----------------------------------------------------------+
```

STR_TO_DATE

Description

The function converts the input datetime string into a DATETIME value based on the specified format.

Syntax

```
STR_TO_DATE(<datetime_str>, <format>)
```

Parameters

Parameter	Description
<datetime_str>	Required. The input datetime string to be converted.
<format>	Required. The specified datetime format string, such as %Y-%m-%d %H:%i:%s, etc. for specific format parameters, see the DATE_FORMAT documentation.

In addition, the <format> supports the following several alternative formats and interprets them to the regular format:

Alternative Input	Interpret as
yyyyMMdd	%Y%m%d
yyyy-MM-dd	%Y-%m-%d
yyyy-MM-dd HH:mm:ss	%Y-%m-%d %H:%i:%s

Return Value

- Returns a DATETIME value representing the converted datetime.
- If the input <datetime_str> or <format> is invalid, the function returns NULL.

Example

Convert common datetime strings to DATETIME

```
SELECT STR_TO_DATE('2025-01-23 12:34:56', '%Y-%m-%d %H:%i:%s'),STR_TO_DATE('2025-01-23 12:34:56', 'yyyy-MM-dd HH:mm:ss');
```

```
+-----------------------------------------------------+-----------------------------------------------------------+
| str_to_date('2025-01-23 12:34:56', '%Y-%m-%d %H:%i:%s') | str_to_date('2025-01-23 12:34:56', 'yyyy-MM-dd HH:mm:ss') |
+-----------------------------------------------------+-----------------------------------------------------------+
| 2025-01-23 12:34:56.000000                          | 2025-01-23 12:34:56.000000                                |
+-----------------------------------------------------+-----------------------------------------------------------+
```

Others

```
select STR_TO_DATE('200442 Monday', '%X%V %W'),STR_TO_DATE('2023','%Y');
```

```
+----------------------------------------+--------------------------+
| str_to_date('200442 Monday', '%X%V %W') | str_to_date('2023', '%Y') |
+----------------------------------------+--------------------------+
| 2004-10-18                             | 2023-01-01               |
+----------------------------------------+--------------------------+
```

TIME_TO_SEC

Description

The function converts an input `TIME` or `DATETIME` value into the total time in seconds. If the input is of `DATETIME` type, the function automatically extracts the time part (`HH:MM:SS`).

Syntax

```
TIME_TO_SEC(<time>)
```

Parameters

Parameter	Description
`<time>`	Required. Supports TIME or DATETIME values. If the input is DATETIME, the function extracts the time part for calculation.

Return Value

- Returns an integer representing the total seconds of the input time value.
- If `<time>` is NULL, the function returns NULL.

Example

```
SELECT TIME_TO_SEC('16:32:18'),TIME_TO_SEC('2025-01-01 16:32:18');
```

```
+--------------------------------------+-------------------------------------------------+
| time_to_sec(cast('16:32:18' as TIME)) | time_to_sec(cast('2025-01-01 16:32:18' as TIME)) |
+--------------------------------------+-------------------------------------------------+
|                                59538 |                                           59538 |
+--------------------------------------+-------------------------------------------------+
```

TIMEDIFF

Description

The TIMEDIFF function calculates the difference between two datetime values. This function accepts two arguments and returns the difference as a TIME type.

Syntax

```
TIMEDIFF(<end_datetime>, <start_datetime>)
```

Parameters

Parameter	Description
<end_datetime>	The ending datetime value.
<start_datetime>	The starting datetime value.

Return Value

Returns a TIME type value representing the difference between the two inputs: - If <end_datetime> is later than <start_datetime>, it returns a positive time difference. - If <end_datetime> is earlier than <start_datetime>, it returns a negative time difference. - If <end_datetime> and <start_datetime> are equal, it returns 00:00:00. - If <end_datetime> or <start_datetime> is NULL, the function returns NULL. - If <end_datetime> or <start_datetime> is an invalid datetime value (e.g., 0000-00-00 00:00:00), the function returns NULL.

Example

```
SELECT TIMEDIFF('2024-07-20 16:59:30','2024-07-11 16:35:21');
```

```
+-----------------------------------------------------------------------------------------------------+
| timediff(cast('2024-07-20 16:59:30' as DATETIMEV2(0)), cast('2024-07-11 16:35:21' as DATETIMEV2(0))) |
+-----------------------------------------------------------------------------------------------------+
| 216:24:09                                                                                           |
+-----------------------------------------------------------------------------------------------------+
```

TIMESTAMP

Description

The TIMESTAMP function has two uses:

1. Convert a datetime string to a DATETIME type
2. Combine two arguments into a DATETIME type

Syntax

```
TIMESTAMP(string)
TIMESTAMP(date, time)
```

Parameters

Parameter	Description
`string`	A datetime string
`date`	A date value, which can be a DATE type or a properly formatted date string
`time`	A time value, which can be a TIME type or a properly formatted time string

Return Value

Returns a value of type DATETIME.

Example

```
-- Convert a string to DATETIME
SELECT TIMESTAMP('2019-01-01 12:00:00');
```

```
+-----------------------------------+
| timestamp('2019-01-01 12:00:00')  |
+-----------------------------------+
| 2019-01-01 12:00:00               |
+-----------------------------------+
```

```
-- Combine date and time into DATETIME
SELECT TIMESTAMP('2019-01-01', '12:00:00');
```

```
+---------------------------------------+
| timestamp('2019-01-01', '12:00:00')   |
+---------------------------------------+
| 2019-01-01 12:00:00                   |
+---------------------------------------+
```

TIMESTAMPADD

Description

The `timestampadd` function is used to add a specified time unit (such as year, month, day, hour, minute, second, etc.) to a date. This function is commonly used for date and time calculations.

Syntax

```
TIMESTAMPADD(<unit>, <interval>, <datetime_expr>)
```

Parameters

Parameter	Description
unit	Time unit, specifies the time unit to add, common values include SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, YEAR
interval	The time interval to add, typically an integer, which can be positive or negative to add or subtract the time length
datetime_expr	A valid datetime data type

Return Value

The return value is the new date and time, representing the result of adding or subtracting the specified time interval to the given timestamp.

If the input parameters are invalid, `NULL` is returned.

Examples

```
SELECT TIMESTAMPADD(MINUTE,1,'2019-01-02');
```

```
+------------------------------------------------+
| timestampadd(MINUTE, 1, '2019-01-02 00:00:00') |
+------------------------------------------------+
| 2019-01-02 00:01:00                            |
+------------------------------------------------+
```

```
SELECT TIMESTAMPADD(WEEK,1,'2019-01-02');
```

```
+-----------------------------------------------+
| timestampadd(WEEK, 1, '2019-01-02 00:00:00') |
+-----------------------------------------------+
| 2019-01-09 00:00:00                           |
+-----------------------------------------------+
```

```
SELECT TIMESTAMPADD(WEEK,1,'1196440219');
```

```
+-----------------------------------------------------------+
| timestampadd(WEEK, 1, CAST('1196440219' AS datetimev2(6))) |
+-----------------------------------------------------------+
| NULL                                                      |
+-----------------------------------------------------------+
```

TIMESTAMPDIFF

Description

The `timestampdiff` function is used to calculate the difference between two dates and returns the time interval between them. The difference can be returned in the specified time unit (such as seconds, minutes, hours, days, months, years, etc.).

Syntax

```
TIMESTAMPDIFF(<unit>, <datetime_expr1>, <datetime_expr2>)
```

Parameters

Parameter	Description
unit	Time unit, specifies the unit in which to return the difference, common values include SECOND, MINUTE, HOUR, DAY, MONTH, YEAR
datetime_expr1	The first datetime, a valid target date

Parameter	Description
date_expr2	The second datetime, a valid target date

Return Value

The return value is the difference between the two date-times, with the unit determined by the unit parameter.

If the input parameters are invalid, `NULL` is returned.

Examples

```
SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
```

```
+-----------------------------------------------------------------+
| timestampdiff(MONTH, '2003-02-01 00:00:00', '2003-05-01 00:00:00') |
+-----------------------------------------------------------------+
|                                                               3 |
+-----------------------------------------------------------------+
```

```
SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
```

```
+----------------------------------------------------------------+
| timestampdiff(YEAR, '2002-05-01 00:00:00', '2001-01-01 00:00:00') |
+----------------------------------------------------------------+
|                                                             -1 |
+----------------------------------------------------------------+
```

```
SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
```

```
+---------------------------------------------------------------------+
| timestampdiff(MINUTE, '2003-02-01 00:00:00', '2003-05-01 12:05:55') |
+---------------------------------------------------------------------+
|                                                              128885 |
+---------------------------------------------------------------------+
```

```
SELECT  TIMESTAMPDIFF(MINUTE,'2003-02-01','1196440219');
```

```
+-------------------------------------------------------------------------------+
| timestampdiff(MINUTE, '2003-02-01 00:00:00', CAST('1196440219' AS datetimev2(6))) |
+-------------------------------------------------------------------------------+
|                                                                          NULL |
+-------------------------------------------------------------------------------+
```

TO_DATE

Description

Date conversion function, used to convert date time (DATETIME) to date type (DATE), that is, remove the time part and keep only the date (YYYY-MM-DD)

Syntax

```
TO_DATE(<datetime_value>)
```

Required parameter

Parameter	Description
datetime_value	DATETIME type date-time

Example

Convert `2020-02-02 00:00:00` to `2020-02-02`

```
select to_date("2020-02-02 00:00:00");
```

```
+--------------------------------+
| to_date('2020-02-02 00:00:00') |
+--------------------------------+
| 2020-02-02                     |
+--------------------------------+
```

TO_DAYS

Description

Date calculation function, which is used to convert a date into a day value, that is, to calculate the total number of days from December 31, 0 AD (the base date) to the specified date.

Syntax

```
TO_DAYS(<datetime_or_date_value>)
```

Required parameters

Parameter	Description
<datetime_or_date_value>	datetime or date type date-time

Example

Query how many days are there since October 7, 2007

```
select to_days('2007-10-07');
```

```
+---------------------------------------+
| to_days(cast('2007-10-07' as DATEV2)) |
+---------------------------------------+
|                                733321 |
+---------------------------------------+
```

```
select to_days('2007-10-07 10:03:09');
```

```
+-----------------------------------------------+
| to_days(cast('2007-10-07 10:03:09' as DATEV2)) |
+-----------------------------------------------+
|                                        733321 |
+-----------------------------------------------+
```

TO_ISO8601 | Date Time Functions

TO_ISO8601

Description

Converts a datetime value to an ISO8601 formatted string.

Syntax

```
TO_ISO8601(<dt>)
```

Parameters

Parameter	Description
<dt>	The input datetime value, which can be of type DATETIME or DATE

Return Value

Returns a value of type VARCHAR, representing the datetime in ISO8601 format.

Example

```
SELECT TO_ISO8601('2020-01-01 12:30:45');
```

```
+-----------------------------------+
| to_iso8601('2020-01-01 12:30:45.0') |
+-----------------------------------+
| 2020-01-01T12:30:45                |
+-----------------------------------+
```

TO_MONDAY

to_monday

Description

Syntax

```
DATE to_monday(DATETIME date)
```

Round a date or datetime down to the nearest Monday, return type is Date or DateV2. Specially, input 1970-01-01, 1970-01-02, 1970-01-03 and 1970-01-04 will return '1970-01-01'

example

```
MySQL [(none)]> select to_monday('2022-09-10');
+-----------------------------------+
| to_monday('2022-09-10 00:00:00') |
+-----------------------------------+
| 2022-09-05                        |
+-----------------------------------+
```

keywords

```
MONDAY
```

UNIX_TIMESTAMP

Description

Converts a Date or Datetime type to a UNIX timestamp.

If no argument is provided, it converts the current time to a timestamp.

The argument must be of Date or Datetime type.

For the format specification, refer to the format description of the date_format function.

This function is affected by the time zone.

Syntax

```
UNIX_TIMESTAMP([DATETIME date[, STRING fmt]])
```

Parameter

Paramters	Description
<date>	The datetime value to be converted is of type `datetime` or `date` type, with a convertible range from '1970-01-01 00:00:01.000000 UTC' to '3001-01-19 03:14:07.999999 UTC'.
<fmt>	The 'date' parameter refers to the specific part that needs to be converted into a timestamp, and it is a parameter of type string. If this parameter is provided, only the part matching the format will be converted into a timestamp.

Return value

Returns two types based on the input:

- If the input `date`(only `datetime` type have the scale not zero) scale is not 0 or a format parameter is provided, returns a timestamp of type Decimal with a maximum precision of six decimal places.
- If the input datetime scale is 0 and no format parameter is provided, returns a timestamp of type INT.
- For times before '1970-01-01 00:00:01.000000 UTC' or after '3001-01-19 03:14:07.999999 UTC', the function returns 0.

Returns NULL if any argument is NULL.

Examples

```
-- All the following results are returned in the UTC time zone

set time_zone= 'UTC';

------Displays the timestamp of the current time
mysql> select unix_timestamp();
+------------------+
| unix_timestamp() |
+------------------+
|       1753933330 |
+------------------+

---Input a datetime to display its timestamp
mysql> select unix_timestamp('2007-11-30 10:30:19');
+---------------------------------------+
| unix_timestamp('2007-11-30 10:30:19') |
+---------------------------------------+
|                            1196389819 |
+---------------------------------------+

---Matches the format to display the timestamp corresponding to the given datetime
mysql> select unix_timestamp('2007-11-30 10:30-19', '%Y-%m-%d %H:%i-%s');
+------------------------------------------------------------+
| unix_timestamp('2007-11-30 10:30-19', '%Y-%m-%d %H:%i-%s') |
+------------------------------------------------------------+
|                                         1196389819.000000 |
+------------------------------------------------------------+


---Only matches year, month, and day to display the timestamp
mysql> select unix_timestamp('2007-11-30 10:30%3A19', '%Y-%m-%d');
+-----------------------------------------------------+
| unix_timestamp('2007-11-30 10:30%3A19', '%Y-%m-%d') |
+-----------------------------------------------------+
|                                  1196352000.000000 |
+-----------------------------------------------------+


---Matching with other characters
mysql> select unix_timestamp('2007-11-30 10:30%3A19', '%Y-%m-%d %H:%i%%3A%s');
+------------------------------------------------------------------+
| unix_timestamp('2007-11-30 10:30%3A19', '%Y-%m-%d %H:%i%%3A%s') |
+------------------------------------------------------------------+
|                                              1196389819.000000 |
+------------------------------------------------------------------+


---Time beyond the minimum range returns 0
mysql> SELECT UNIX_TIMESTAMP('1970-01-01 00:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('1970-01-01 00:00:00') |
+---------------------------------------+
|                                     0 |
+---------------------------------------+


---Input time with non-zero scale
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19.123');
+-----------------------------------------+
| UNIX_TIMESTAMP('2015-11-13 10:20:19.123') |
+-----------------------------------------+
```

```
|                           1447381219.123 |
+------------------------------------------+

---Exceeding the maximum allowed time range

mysql> SELECT UNIX_TIMESTAMP('3001-01-19 03:14:07.999999');
+-----------------------------------------------+
| UNIX_TIMESTAMP('3001-01-19 03:14:07.999999') |
+-----------------------------------------------+
|                                      0.000000 |
+-----------------------------------------------+


---Returns NULL if any argument is NULL
mysql> select unix_timestamp(NULL);
+----------------------+
| unix_timestamp(NULL) |
+----------------------+
|                 NULL |
+----------------------+

mysql> select unix_timestamp('2038-01-19 11:14:08',null);
+--------------------------------------------+
| unix_timestamp('2038-01-19 11:14:08',null) |
+--------------------------------------------+
|                                       NULL |
+--------------------------------------------+
```

keywords

```
UNIX_TIMESTAMP,UNIX,TIMESTAMP
```

UTC_TIMESTAMP

utc_timestamp

Description

Syntax

```
DATETIME UTC_TIMESTAMP()
```

Returns the current UTC date and time in “YYYY-MM-DD HH: MM: SS” or

A Value of “YYYYMMDDHMMSS” Format

Depending on whether the function is used in a string or numeric context

example

```
mysql> select utc_timestamp(),utc_timestamp() + 1;
+---------------------+---------------------+
| utc_timestamp()     | utc_timestamp() + 1 |
+---------------------+---------------------+
| 2019-07-10 12:31:18 |      20190710123119 |
+---------------------+---------------------+
```

keywords

```
UTC_TIMESTAMP,UTC,TIMESTAMP
```

WEEK

week

Description

Syntax

```
INT WEEK(DATE date[, INT mode])
```

Returns the week number for date.The value of the mode argument defaults to 0. The following table describes how the mode argument works.

Mode	First day of week	Range	Week 1 is the first week …
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with 4 or more days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year
4	Sunday	0-53	with 4 or more days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year

The parameter is Date or Datetime type

example

```
mysql> select week('2020-1-1');
+------------------+
| week('2020-1-1') |
+------------------+
|                0 |
+------------------+
```

```
mysql> select week('2020-7-1',1);
+---------------------+
| week('2020-7-1', 1) |
+---------------------+
|                  27 |
+---------------------+
```

keywords

```
WEEK
```

WEEK_CEIL | Date Time Functions

WEEK_CEIL

Description

Rounds up a datetime value to the nearest specified week interval. If a starting time (origin) is provided, it uses that time as the reference for calculating the interval.

Syntax

```
WEEK_CEIL(<datetime>)
WEEK_CEIL(<datetime>, <origin>)
WEEK_CEIL(<datetime>, <period>)
WEEK_CEIL(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	The datetime value to round up, of type DATETIME or DATETIMEV2
<period>	The week interval value, of type INT, representing the number of weeks in each interval
<origin>	The starting point for the interval, of type DATETIME or DATETIMEV2; defaults to 0001-01-01 00:00:00

Return Value

Returns a value of type DATETIME, representing the rounded-up datetime value. The time portion of the result will be set to 00:00:00.

Note: - If no period is specified, it defaults to a 1-week interval. - The period must be a positive integer. - The result is always rounded up to a future time. - The time portion of the returned value is always set to 00:00:00.

Example

```
SELECT WEEK_CEIL('2023-07-13 22:28:18', 2);
```

```
+-----------------------------------------------------------+
| week_ceil(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 2) |
+-----------------------------------------------------------+
| 2023-07-24 00:00:00                                       |
+-----------------------------------------------------------+
```

WEEK_FLOOR | Date Time Functions

WEEK_FLOOR

Description

Rounds down a datetime value to the nearest specified week interval. If a starting time (origin) is provided, it uses that time as the reference for calculating the interval.

Syntax

```
WEEK_FLOOR(<datetime>)
WEEK_FLOOR(<datetime>, <origin>)
WEEK_FLOOR(<datetime>, <period>)
WEEK_FLOOR(<datetime>, <period>, <origin>)
```

Parameters

Parameter	Description
<datetime>	The datetime value to round down, of type DATETIME or DATETIMEV2
<period>	The week interval value, of type INT, representing the number of weeks in each interval
<origin>	The starting point for the interval, of type DATETIME or DATETIMEV2; defaults to 0001-01-01 00:00:00

Return Value

Returns a value of type DATETIME, representing the rounded-down datetime value. The time portion of the result will be set to 00:00:00.

Note: - If no period is specified, it defaults to a 1-week interval. - The period must be a positive integer. - The result is always rounded down to a past time. - The time portion of the returned value is always set to 00:00:00.

Example

```
SELECT WEEK_FLOOR('2023-07-13 22:28:18', 2);
```

```
+-----------------------------------------------------------+
| week_floor(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 2) |
+-----------------------------------------------------------+
| 2023-07-03 00:00:00                                       |
+-----------------------------------------------------------+
```

WEEKDAY

weekday

Description

Syntax

```
INT WEEKDAY (DATETIME date)
```

The WEEKDAY function returns the index value of the working day of the date, that is, 0 on Monday, 1 on Tuesday, and 6 on Sunday.

The parameter is Date or Datetime type

Notice the difference between WEEKDAY and DAYOFWEEK:

```
          +-----+-----+-----+-----+-----+-----+-----+
          | Sun | Mon | Tues| Wed | Thur| Fri | Sat |
          +-----+-----+-----+-----+-----+-----+-----+
  weekday |  6  |  0  |  1  |  2  |  3  |  4  |  5  |
          +-----+-----+-----+-----+-----+-----+-----+
dayofweek |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
          +-----+-----+-----+-----+-----+-----+-----+
```

example

```
mysql> select weekday('2019-06-25');
+---------------------------------+
| weekday('2019-06-25 00:00:00') |
+---------------------------------+
|                               1 |
+---------------------------------+

mysql> select weekday(cast(20190625 as date));
+----------------------------------+
| weekday(CAST(20190625 AS DATE)) |
+----------------------------------+
|                                1 |
+----------------------------------+
```

keywords

```
WEEKDAY
```

WEEKOFYEAR

weekofyear

Description

Syntax

```
INT WEEKOFYEAR (DATETIME DATE)
```

Get the Weeks of the Year

The parameter is Date or Datetime type

example

```
mysql> select weekofyear('2008-02-20 00:00:00');
+----------------------------------+
| weekofyear('2008-02-20 00:00:00') |
+----------------------------------+
|                                8 |
+----------------------------------+
```

keywords

```
WEEKOFYEAR
```

WEEKS_ADD

Description

This function is used to add (or subtract) a certain number of weeks from a specified date or time value.

Syntax

```
WEEKS_ADD(<datetime_or_date_value>, <weeks_value>)
```

Required parameters

Parameter	Description
<datetime_or_date_value>	DATETIME or DATE type date input value
<weeks_value>	Integer, indicating the number of weeks to increase or decrease (positive number indicates increase, negative number indicates decrease)

example

1. Add one week to the time 2020-02-02 02:02:02 sql select weeks_add("2020-02-02 02:02:02", 1); text +---------------------------------+ | weeks_add('2020-02-02 02:02:02', 1) | +---------------------------------+ | 2020-02-09 02:02:02 | +---------------------------------+
2. Subtract one week from the time 2020-02-02 02:02:02 sql select weeks_add("2020-02-02 02:02:02", -1); text +--+ | weeks_add(cast('2020-02-02 02:02:02' as DATETIMEV2(0)), -1) | +--+ | 2020-01-26 02:02:02 | +--+
3. Add one week to the date 2020-02-02 sql select weeks_add("2020-02-02", 1); text +---+ | weeks_add(cast('2020-02-02' as DATEV2), 1) | +---+ | 2020-02-09 | +---+

WEEKS_DIFF

Description

Calculates the number of complete weeks (in 7-day units) between two date or time values.

Syntax

```
WEEKS_DIFF(<end_date>, <start_date>)
```

Required parameters

Parameter Name	Data Type	Description
end_date	DATE, DATETIME	Later date or date-time
start_date	DATE, DATETIME	Earlier date or date-time

Example

1. How many weeks are there between 2020-12-25 and 2020-10-25 sql select weeks_diff('2020-12-25','2020-10-25'); text +--+ | weeks_diff('2020-12-25 00:00:00', '2020-10-25 00:00:00') | +--+ | 8 | +--+
2. How many weeks are there between 2020-12-25 10:10:02 and 2020-10-25 12:10:02 sql select weeks_diff('2020-12-25 10:10:02','2020-10-25 12:10:02'); text +---+ | weeks_diff(cast('2020-12-25 10:10:02' as DATETIMEV2(0)), cast('2020-10-25 12:10:02' as DATETIMEV2(0))) | +---+ | 8 | +---+
3. How many weeks are there between 2020-12-25 10:10:02 and 2020-10-25 sql select weeks_diff('2020-12-25 10:10:02','2020-10-25'); text +---+ | weeks_diff(cast('2020-12-25 10:10:02' as DATETIMEV2(0)), cast('2020-10-25' as DATEV2)) | +---+ | 8 | +---+

WEEKS_SUB

Description

Subtracts a specified number of weeks from a specified date or time value (i.e., subtracts weeks * 7 days).

Syntax

```
WEEKS_SUB(<date_value>, <week_period>)
```

Required parameters

Parameter	Description
date_value	DATE or DATETIME type input value.
week_period	Integer, representing the number of weeks to subtract (positive to decrease, negative to increase).

Example

1. Subtract one week from the datetime 2020-02-02 02:02:02 sql select weeks_sub("2020-02-02 02:02:02", 1); text +-------------------------------------+ | weeks_sub('2020-02-02 02:02:02', 1) | +-------------------------------------+ | 2020-01-26 02:02:02 | +-------------------------------------+

2. Subtract one week from the date 2020-02-02 sql select weeks_sub("2020-02-02", 1); text +---+ | weeks_sub(cast('2020-02-02' as DATEV2), 1) | +---+ | 2020-01-26 | +---+

YEAR

year

Description

Syntax

```
INT YEAR(DATETIME date)
```

Returns the year part of the date type, ranging from 1000 to 9999

The parameter is Date or Datetime type

example

```
mysql> select year('1987-01-01');
+-----------------------------+
| year('1987-01-01 00:00:00') |
+-----------------------------+
|                        1987 |
+-----------------------------+
```

keywords

```
YEAR
```

YEAR_CEIL

year_ceil

description

Syntax

```
DATETIME YEAR_CEIL(DATETIME datetime)
DATETIME YEAR_CEIL(DATETIME datetime, DATETIME origin)
DATETIME YEAR_CEIL(DATETIME datetime, INT period)
DATETIME YEAR_CEIL(DATETIME datetime, INT period, DATETIME origin)
```

Convert the date to the nearest rounding up time of the specified time interval period.

- datetime: a valid date expression.
- period: specifies how many years each cycle consists of.
- origin: starting from 0001-01-01T00:00:00.

example

```
mysql> select year_ceil("2023-07-13 22:28:18", 5);
+-----------------------------------------------------------+
| year_ceil(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5) |
+-----------------------------------------------------------+
| 2025-01-01 00:00:00                                       |
+-----------------------------------------------------------+
1 row in set (0.02 sec)
```

keywords

```
YEAR_CEIL, YEAR, CEIL
```

Best Practice

See also date_ceil

YEAR_FLOOR

Description

It is used to round the given date down to the specified year interval starting point. It supports multiple variants, which can specify the starting time (origin) and period (period) in different ways to round.

Syntax

```
YEAR_FLOOR(<date_value>, [<period> | <origin_date_value>])
YEAR_FLOOR(<date_value>, <period>, <origin_date_value>)
```

Parameters

Parameter	Type	Description
<date_value>	DATE, DATETIME	The DATE or DATETIME input value to be rounded.
<origin_date_value>	DATE, DATETIME	The DATE or DATETIME input value used as the reference point. If not provided, the default is 0001-01-01T00:00:00.
<period>	INT	The rounding interval, a positive integer indicating the number of years per cycle.

Example

1. Rounding to the whole year `sql SELECT YEAR_FLOOR('2023-07-13 22:28:18'); +---------- ---------- ----------

```
| 2023-03-15 00:00:00                                                                          |
+----------------------------------------------------------------------------------------------+
```

3. Rounding with period as unit

```
 SELECT YEAR_FLOOR('2023-07-13', 5);
```

```
+-------------------------------------------------+
| year_floor(cast('2023-07-13' as DATETIMEV2(0)), 5) |
+-------------------------------------------------+
| 2020-01-01 00:00:00                             |
+-------------------------------------------------+
```

4. Round origin and period `sql SELECT YEAR_FLOOR('2023-07-13 22:28:18', 5, '2018-06-01'); +---+ | year_floor(cast('2023-07-13 22:28:18' as DATETIMEV2(0)), 5, cast('2018-06-01' as DATETIMEV2(0))) | +---+ | 2023-06-01 00:00:00 | +---+`

```
 SELECT YEAR_FLOOR('2023-07-13', 5, '2016-01-01');
```

```
+---------------------------------------------------------------------------------------+
| year_floor(cast('2023-07-13' as DATETIMEV2(0)), 5, cast('2016-01-01' as DATETIMEV2(0))) |
+---------------------------------------------------------------------------------------+
| 2021-01-01 00:00:00                                                                   |
+---------------------------------------------------------------------------------------+
```

YEARWEEK

yearweek

Description

Syntax

```
INT YEARWEEK(DATE date[, INT mode])
```

Returns year and week for a date.The value of the mode argument defaults to 0. When the week of the date belongs to the previous year, the year and week of the previous year are returned; when the week of the date belongs to the next year, the year of the next year is returned and the week is 1.

The following table describes how the mode argument works.

Mode	First day of week	Range	Week 1 is the first week …
0	Sunday	1-53	with a Sunday in this year
1	Monday	1-53	with 4 or more days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year
4	Sunday	1-53	with 4 or more days this year
5	Monday	1-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year

The parameter is Date or Datetime type

example

```
mysql> select yearweek('2021-1-1');
+----------------------+
| yearweek('2021-1-1') |
+----------------------+
|               202052 |
+----------------------+
```

```
mysql> select yearweek('2020-7-1');
+----------------------+
| yearweek('2020-7-1') |
+----------------------+
|               202026 |
+----------------------+
```

```
mysql> select yearweek('2024-12-30',1);
+-----------------------------------+
| yearweek('2024-12-30 00:00:00', 1) |
+-----------------------------------+
|                            202501 |
+-----------------------------------+
```

keywords

```
YEARWEEK
```

YEARS_ADD

Description

Returns a new datetime value that is the result of adding a specified number of years to the input datetime.

Syntax

```
YEARS_ADD(<date>, <years>)
```

Parameters

Parameter	Description
<date>	The input datetime value, which can be of type DATETIME or DATE
<years>	The number of years to add, of type INT

Return Value

Returns a value with the same type as the input <date> (DATETIME or DATE), representing the time value after adding the specified number of years to the input datetime.

Example

```
SELECT YEARS_ADD('2020-01-31 02:02:02', 1);
```

```
+-----------------------------------+
| years_add('2020-01-31 02:02:02', 1) |
+-----------------------------------+
| 2021-01-31 02:02:02               |
+-----------------------------------+
```

YEARS_DIFF

Description

Calculates the difference in years between two datetime values.

Syntax

```
YEARS_DIFF(<enddate>, <startdate>)
```

Parameters

Parameter	Description
`<enddate>`	The end date, which can be of type DATETIME or DATE
`<startdate>`	The start date, which can be of type DATETIME or DATE

Return Value

Returns a value of type INT, representing the number of years between the two dates.

Example

```
SELECT YEARS_DIFF('2020-12-25', '2019-10-25');
```

```
+-----------------------------------------------------------+
| years_diff('2020-12-25 00:00:00', '2019-10-25 00:00:00') |
+-----------------------------------------------------------+
|                                                         1 |
+-----------------------------------------------------------+
```

YEARS_SUB

Description

Returns a new datetime value that is the result of subtracting a specified number of years from the input datetime.

Syntax

```
YEARS_SUB(<date>, <years>)
```

Parameters

Parameter	Description
<date>	The input datetime value, which can be of type DATETIME or DATE
<years>	The number of years to subtract, of type INT

Return Value

Returns a value with the same type as the input <date> (DATETIME or DATE), representing the time value after subtracting the specified number of years from the input datetime.

Example

```
SELECT YEARS_SUB('2020-02-02 02:02:02', 1);
```

```
+-----------------------------------+
| years_sub('2020-02-02 02:02:02', 1) |
+-----------------------------------+
| 2019-02-02 02:02:02               |
+-----------------------------------+
```

ST_ANGLE

Description

Enter three points that represent two intersecting lines. Return the Angle between these lines.

Syntax

```
ST_ANGLE( <point1>, <point2>, <point3>)
```

Parameters

Parameters	Instructions
<point1>	The first point of the first line
<point2>	The second point of the first line and the first point of the second line
<point3>	The second point of the second line

Return Value

The Angle between these lines is expressed in radians and ranges from [0, 2pi]. The Angle is measured clockwise from the first line to the second line.

ST_ANGLE has the following edge cases:

- If point 2 and point 3 are the same, NULL is returned
- If point 2 and point 1 are the same, NULL is returned
- NULL is returned if points 2 and 3 are perfect Antipodes
- NULL is returned if points 2 and 1 are perfect AntipodesL
- If any input geography is not a single point or is an empty geography, an error is thrown

Examples

```
SELECT ST_Angle(ST_Point(1, 0),ST_Point(0, 0),ST_Point(0, 1));
```

```
+-----------------------------------------------------------------------+
| st_angle(st_point(1.0, 0.0), st_point(0.0, 0.0), st_point(0.0, 1.0)) |
+-----------------------------------------------------------------------+
|                                                 4.71238898038469 |
+-----------------------------------------------------------------------+
```

```
SELECT ST_Angle(ST_Point(0, 0),ST_Point(1, 0),ST_Point(0, 1));
```

```
+----------------------------------------------------------------------+
| st_angle(st_point(0.0, 0.0), st_point(1.0, 0.0), st_point(0.0, 1.0)) |
+----------------------------------------------------------------------+
|                                              0.78547432161873854 |
+----------------------------------------------------------------------+
```

```
SELECT ST_Angle(ST_Point(1, 0),ST_Point(0, 0),ST_Point(1, 0));
```

```
+----------------------------------------------------------------------+
| st_angle(st_point(1.0, 0.0), st_point(0.0, 0.0), st_point(1.0, 0.0)) |
+----------------------------------------------------------------------+
|                                                                    0 |
+----------------------------------------------------------------------+
```

```
SELECT ST_Angle(ST_Point(1, 0),ST_Point(0, 0),ST_Point(0, 0));
```

```
+----------------------------------------------------------------------+
| st_angle(st_point(1.0, 0.0), st_point(0.0, 0.0), st_point(0.0, 0.0)) |
+----------------------------------------------------------------------+
|                                                                 NULL |
+----------------------------------------------------------------------+
```

```
SELECT ST_Angle(ST_Point(0, 0),ST_Point(-30, 0),ST_Point(150, 0));
```

```
+------------------------------------------------------------------------+
| st_angle(st_point(0.0, 0.0), st_point(-30.0, 0.0), st_point(150.0, 0.0)) |
+------------------------------------------------------------------------+
|                                                                   NULL |
+------------------------------------------------------------------------+
```

ST_ANGLE_SPHERE

Description

Calculate the center Angle between two points on the Earth's surface in degrees. The parameters passed in are the longitude of point X, latitude of point X, longitude of point Y, and latitude of point Y.

Syntax

```
ST_ANGLE_SPHERE( <x_lng>, <x_lat>, <y_lng>, <y_lat>)
```

Parameters

Parameters	Instructions
<x_lng>	Longitude data, reasonable value range is [-180, 180]
<y_lng>	Longitude data, reasonable value range is [-180, 180]
<x_lat>	Latitude data, reasonable value range is [-90, 90]
<y_lat>	Latitude data, reasonable value range is [-90, 90]

Return Value

The Angle of the center between two points

Examples

```
select ST_Angle_Sphere(116.35620117, 39.939093, 116.4274406433, 39.9020987219);
```

```
+-------------------------------------------------------------------------+
| st_angle_sphere(116.35620117, 39.939093, 116.4274406433, 39.9020987219) |
+-------------------------------------------------------------------------+
|                                                  0.0659823452409903 |
+-------------------------------------------------------------------------+
```

```
select ST_Angle_Sphere(0, 0, 45, 0);
```

```
+--------------------------------------+
| st_angle_sphere(0.0, 0.0, 45.0, 0.0) |
+--------------------------------------+
|                                   45 |
+--------------------------------------+
```

ST_AREA_SQUARE_KM

Description

Calculate the area of the region on the sphere of the Earth

Syntax

```
ST_AREA_SQUARE_KM( <geo>)
```

Parameters

Parameters	Instructions
<geo>	The spherical position of the earth

Return Value

ST_Area_Square_Km( ): the units returned are square kilometers

Examples

```
SELECT ST_Area_Square_Km(ST_Polygon("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))"));
```

```
+-----------------------------------------------------------------------+
| st_area_square_km(st_polygon('POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))')) |
+-----------------------------------------------------------------------+
|                                              12364.036567076409 |
+-----------------------------------------------------------------------+
```

ST_AREA_SQUARE_METERS

Description

Calculate the area of the region on the sphere of the Earth

Syntax

```
ST_AREA_SQUARE_METERS( <geo>)
```

Parameters

Parameters	Instructions
<geo>	The spherical position of the earth

Return Value

ST_Area_Square_Meters( ): the units returned are square meters

Examples

```
SELECT ST_Area_Square_Meters(ST_Circle(0, 0, 1));
```

```
+-------------------------------------------------+
| st_area_square_meters(st_circle(0.0, 0.0, 1.0)) |
+-------------------------------------------------+
|                            3.1415926535897869 |
+-------------------------------------------------+
```

```
SELECT ST_Area_Square_Meters(ST_Point(0, 1));
```

```
+-----------------------------------------+
| st_area_square_meters(st_point(0.0, 1.0)) |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+
```

```
SELECT ST_Area_Square_Meters(ST_LineFromText("LINESTRING (1 1, 2 2)"));
```

```
+-----------------------------------------------------------------+
| st_area_square_meters(st_linefromtext('LINESTRING (1 1, 2 2)')) |
+-----------------------------------------------------------------+
|                                                               0 |
+-----------------------------------------------------------------+
```

ST_ASBINARY

Description

Convert a geometric figure to a standard WKB (Well-known binary) representation.

At present, the supported geometry is: Point, LineString, Polygon.

Syntax

```
ST_ASBINARY( <geo>)
```

Parameters

Parameters	Instructions
<geo>	The graph that needs to be converted

Return Value

The WKB representation of the geometry:

Examples

```
select ST_AsBinary(st_point(24.7, 56.7));
```

```
+--------------------------------------------+
| st_asbinary(st_point(24.7, 56.7))          |
+--------------------------------------------+
| \x0101000000333333333b338409a99999999594c40 |
+--------------------------------------------+
```

```
select ST_AsBinary(ST_GeometryFromText("LINESTRING (1 1, 2 2)"));
```

```
+-------------------------------------------------------------------------------------+
| st_asbinary(st_geometryfromtext('LINESTRING (1 1, 2 2)'))                           |
+-------------------------------------------------------------------------------------+
| \x010200000002000000000000000000f03f000000000000f03f0000000000000040000000000000040 |
+-------------------------------------------------------------------------------------+
```

```
select ST_AsBinary(ST_Polygon("POLYGON ((114.104486 22.547119,114.093758 22.547753,114.096504 22.532057,114.104229
22.539826,114.106203 22.542680,114.104486 22.547119))"));
```

```
+-------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
```

```
| st_asbinary(st_polygon('POLYGON ((114.104486 22.547119,114.093758 22.547753,114.096504 22.532057,114.104229 22.539826,114.106203
22.542680,114.104486 22.547119))'))                                                                                                |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|
\x0103000000010000000600000f3380ce6af865c402d05a4fd0f8c364041ef8d2100865c403049658a398c3640b9fb1c1f2d865c409d9b36e334883640de921cb0
ab865c40cf876709328a36402cefaa07cc865c407b319413ed8a3640f3380ce6af865c402d05a4fd0f8c3640 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

ST_ASTEXT

Description

Convert a geometric figure to a representation of WKT (Well Known Text)

Alias

- ST_ASWKT

Syntax

```
ST_ASTEXT(GEOMETRY <geo>)
```

Parameters

Parameters	Instructions
<geo>	The graph that needs to be converted

Return Value

The WKT representation of the geometry:

Examples

```
SELECT ST_AsText(ST_Point(24.7, 56.7));
```

```
+---------------------------------+
| st_astext(st_point(24.7, 56.7)) |
+---------------------------------+
| POINT (24.7 56.7)               |
+---------------------------------+
```

ST_AZIMUTH

Description

Enter two points and return the azimuth of the line segment formed by points 1 and 2. Azimuth is the arc of the Angle between the true north line of point 1 and the line segment formed by points 1 and 2.

Syntax

```
ST_AZIMUTH( <point1>, <point2>)
```

Parameters

Parameters	Instructions
<point1>	The first point used to calculate the azimuth
<point2>	The second point used to calculate the azimuth

Return Value

Positive angles are measured clockwise on the sphere. For example, the azimuth of a line segment:

- North is 0
- East is PI/2
- The guide is PI
- The west is 3PI/2

ST_Azimuth has the following edge cases:

- Return NULL if both input points are the same.
- NULL is returned if the two input points are perfect mapping points.
- An error is thrown if any of the input geographies are not a single point or are empty geographies

Examples

```
SELECT st_azimuth(ST_Point(1, 0),ST_Point(0, 0));
```

```
+---------------------------------------------------+
| st_azimuth(st_point(1.0, 0.0), st_point(0.0, 0.0)) |
+---------------------------------------------------+
|                                 4.71238898038469 |
+---------------------------------------------------+
```

```
SELECT st_azimuth(ST_Point(0, 0),ST_Point(1, 0));
```

```
+-----------------------------------------------------+
| st_azimuth(st_point(0.0, 0.0), st_point(1.0, 0.0)) |
+-----------------------------------------------------+
|                                1.5707963267948966 |
+-----------------------------------------------------+
```

```
SELECT st_azimuth(ST_Point(0, 0),ST_Point(0, 1));
```

```
+-----------------------------------------------------+
| st_azimuth(st_point(0.0, 0.0), st_point(0.0, 1.0)) |
+-----------------------------------------------------+
|                                                   0 |
+-----------------------------------------------------+
```

```
SELECT st_azimuth(ST_Point(-30, 0),ST_Point(150, 0));
```

```
+-------------------------------------------------------+
| st_azimuth(st_point(-30.0, 0.0), st_point(150.0, 0.0)) |
+-------------------------------------------------------+
|                                                  NULL |
+-------------------------------------------------------+
```

ST_CIRCLE

Description

Convert a WKT (Well Known Text) to a circle on the sphere of the Earth.

Syntax

```
ST_CIRCLE( <center_lng>, <center_lat>, <radius>)
```

Parameters

Parameters	Instructions
<center_lng>	Longitude of the center of the circle
<center_lat>	The latitude of the center of the circle
<radius>	Radius of a circle

- The unit of radius is meters. A maximum of 9999999 RADIUS is supported

Return Value

A circle on a sphere based on basic information about the circle

Examples

```
SELECT ST_AsText(ST_Circle(111, 64, 10000));
```

```
+-----------------------------------------+
| st_astext(st_circle(111.0, 64.0, 10000.0)) |
+-----------------------------------------+
| CIRCLE ((111 64), 10000)                |
+-----------------------------------------+
```

ST_CONTAINS

Description

Determines whether the geometry shape1 is fully capable of containing the geometry shape2

Syntax

```
ST_CONTAINS( <shape1>, <shape2>)
```

Parameters

Parameters	Instructions
<shape1>	The passed geometry used to determine whether shape2 is included
<shape2>	The passed geometry is used to determine whether shape1 is included

Return Value

Return 1:shape1 The graph can contain the graph shape2

Return 0:shape1 Graph cannot contain graph shape2

Examples

```
SELECT ST_Contains(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(5, 5));
```

```
+-----------------------------------------------------------------------------------+
| st_contains(st_polygon('POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'), st_point(5.0, 5.0)) |
+-----------------------------------------------------------------------------------+
|                                                                                 1 |
+-----------------------------------------------------------------------------------+
```

```
SELECT ST_Contains(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(50, 50));
```

```
+-------------------------------------------------------------------------------------+
| st_contains(st_polygon('POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'), st_point(50.0, 50.0)) |
+-------------------------------------------------------------------------------------+
|                                                                                   0 |
+-------------------------------------------------------------------------------------+
```

ST_DISJOINT

Description

Determines whether the geometry shape1 does not intersect with the geometry shape2.

> **Note**
>
> Supported since Apache Doris 2.1.10.

Syntax

```
ST_DISJOINT( <shape1>, <shape2> )
```

Parameters

Parameters	Instructions
<shape1>	The passed geometry used to determine whether it does not intersect with shape2
<shape2>	The passed geometry used to determine whether it does not intersect with shape1

Return Value

Return 1: shape1 The graph does not intersect with the graph shape2

Return 0: shape1 The graph intersects with the graph shape2

Examples

```
SELECT ST_Disjoint(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(5, 5));
```

```
+-----------------------------------------------------------------------------------+
| ST_Disjoint(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(5, 5)) |
+-----------------------------------------------------------------------------------+
|                                                                                 0 |
+-----------------------------------------------------------------------------------+
```

```
SELECT ST_Disjoint(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(50, 50));
```

```
+-------------------------------------------------------------------------------------+
| ST_Disjoint(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(50, 50)) |
+-------------------------------------------------------------------------------------+
```

```
|                                                                                1 |
+----------------------------------------------------------------------------------+
```

ST_DISTANCE_SPHERE

Description

Calculate the spherical distance in meters between two points on the Earth. The parameters passed in are the longitude of point X, latitude of point X, longitude of point Y, and latitude of point Y.

Syntax

```
ST_DISTANCE_SPHERE( <x_lng>, <x_lat>, <y_lng>, <y_lat>)
```

Parameters

Parameters	Instructions
<x_lng>	Longitude data, reasonable value range is [-180, 180]
<y_lng>	Longitude data, reasonable value range is [-180, 180]
<x_lat>	Latitude data, reasonable value range is [-90, 90]
<y_lat>	Latitude data, reasonable value range is [-90, 90]

Return Value

The spherical distance between two points

Examples

```
select st_distance_sphere(116.35620117, 39.939093, 116.4274406433, 39.9020987219);
```

```
+-----------------------------------------------------------------------------+
| st_distance_sphere(116.35620117, 39.939093, 116.4274406433, 39.9020987219) |
+-----------------------------------------------------------------------------+
|                                                          7336.9135549995917 |
+-----------------------------------------------------------------------------+
```

ST_GEOMETRYFROMTEXT

Description

Convert a WKT (Well Known Text) to the corresponding memory geometry

Alias

- ST_GEOMFROMTEXT

Syntax

```
ST_GEOMETRYFROMTEXT( <wkt>)
```

Parameters

Parameters	Instructions
<wkt>	The memory form of the graph

Support WKT Formats

- `POINT` - A single point in space
- `LINESTRING` - A sequence of connected line segments
- `POLYGON` - A closed area defined by one or more rings, requiring at least three distinct points and closed ends.
- `MULTIPOLYGON` - A collection of polygons, requiring polygons in a multipolygon can only share discrete points.

> **Note**
>
> Supported MULTIPOLYGON format parsing since Apache Doris 2.1.10

Return Value

The corresponding geometric storage form of WKB

Returns NULL when the input WKT format does not conform to specifications or when the input is NULL.

Examples

```
-- POINT example
```

```
SELECT ST_AsText(ST_GeometryFromText("POINT (1 1)"));
```

```
+---------------------------------------------+
| ST_AsText(ST_GeometryFromText("POINT (1 1)")) |
+---------------------------------------------+
| POINT (1 1)                                 |
+---------------------------------------------+
```

```
-- POINT illegal example(too many points)
SELECT ST_AsText(ST_GeometryFromText("POINT (1 1, 2 2)"));
```

```
+-------------------------------------------------+
| ST_AsText(ST_GeometryFromText("POINT (1 1, 2 2)")) |
+-------------------------------------------------+
| NULL                                            |
+-------------------------------------------------+
```

```
-- LINESTRING example
SELECT ST_AsText(ST_GeometryFromText("LINESTRING (1 1, 2 2)"));
```

```
+---------------------------------------------------------+
| st_astext(st_geometryfromtext('LINESTRING (1 1, 2 2)')) |
+---------------------------------------------------------+
| LINESTRING (1 1, 2 2)                                   |
+---------------------------------------------------------+
```

```
-- LINESTRING illegal example(too few verteices)
SELECT ST_AsText(ST_GeometryFromText("LINESTRING (1 1)"));
```

```
+-------------------------------------------------+
| ST_AsText(ST_GeometryFromText("LINESTRING (1 1)")) |
+-------------------------------------------------+
| NULL                                            |
+-------------------------------------------------+
```

```
-- POLYGON example
SELECT ST_AsText(ST_GeometryFromText("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))"));
```

```
+-----------------------------------------------------------------------+
| ST_AsText(ST_GeometryFromText("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))")) |
+-----------------------------------------------------------------------+
| POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))                                   |
+-----------------------------------------------------------------------+
```

```
-- POLYGON illegal example(not closed end to end)
SELECT ST_AsText(ST_GeometryFromText("POLYGON ((0 0, 1 0, 1 1, 0 1))"));
```

```
+-------------------------------------------------------------+
| ST_AsText(ST_GeometryFromText("POLYGON ((0 0, 1 0, 1 1, 0 1))")) |
+-------------------------------------------------------------+
```

```
| NULL                                                          |
+---------------------------------------------------------------+
```

```
-- POLYGON illegal example(too few verteices)
SELECT ST_AsText(ST_GeometryFromText("POLYGON ((0 0, 1 0, 0 0))"));
```

```
+-----------------------------------------------------------+
| ST_AsText(ST_GeometryFromText("POLYGON ((0 0, 1 0, 0 0))")) |
+-----------------------------------------------------------+
| NULL                                                      |
+-----------------------------------------------------------+
```

```
-- MULTIPOLYGON example
SELECT ST_AsText(ST_GeometryFromText("MULTIPOLYGON (((0 0, 1 0, 1 1, 0 1, 0 0)), ((2 2, 3 2, 3 3, 2 3, 2 2)))"));
```

```
+-----------------------------------------------------------------------------------------------------+
| ST_AsText(ST_GeometryFromText("MULTIPOLYGON (((0 0, 1 0, 1 1, 0 1, 0 0)), ((2 2, 3 2, 3 3, 2 3, 2 2)))")) |
+-----------------------------------------------------------------------------------------------------+
| MULTIPOLYGON (((0 0, 1 0, 1 1, 0 1, 0 0)), ((2 2, 3 2, 3 3, 2 3, 2 2)))                             |
+-----------------------------------------------------------------------------------------------------+
```

```
-- MULTIPOLYGON example (polygons in multipolygon only share discrete points.)
SELECT ST_AsText(ST_GeometryFromText("MULTIPOLYGON (((0 0, 10 0, 10 10, 0 10, 0 0), (4 4, 6 4, 6 6, 4 6, 4 4)), ((4 5, 5 4, 6 5, 5
6, 4 5)))"));
```

```
+-----------------------------------------------------------------------------------------------------------------------------------
-------+
| ST_AsText(ST_GeometryFromText("MULTIPOLYGON (((0 0, 10 0, 10 10, 0 10, 0 0), (4 4, 6 4, 6 6, 4 6, 4 4)), ((4 5, 5 4, 6 5, 5 6, 4
5)))")) |
+-----------------------------------------------------------------------------------------------------------------------------------
-------+
| MULTIPOLYGON (((0 0, 10 0, 10 10, 0 10, 0 0), (4 4, 6 4, 6 6, 4 6, 4 4)), ((4 5, 5 4, 6 5, 5 6, 4 5)))
|
+-----------------------------------------------------------------------------------------------------------------------------------
-------+
```

```
-- MULTIPOLYGON illegal example(overlap exists)
SELECT ST_AsText(ST_GeometryFromText("MULTIPOLYGON (((0 0, 10 0, 10 10, 0 10, 0 0)), ((10 0, 20 0, 20 10, 10 10, 10 0)))"));
```

```
+-----------------------------------------------------------------------------------------------------------------------+
| ST_AsText(ST_GeometryFromText("MULTIPOLYGON (((0 0, 10 0, 10 10, 0 10, 0 0)), ((10 0, 20 0, 20 10, 10 10, 10 0)))")) |
+-----------------------------------------------------------------------------------------------------------------------+
| NULL                                                                                                                  |
+-----------------------------------------------------------------------------------------------------------------------+
```

```
-- input NULL
SELECT ST_AsText(ST_GeometryFromText(NULL));
```

```
+--------------------------------------+
| ST_AsText(ST_GeometryFromText(NULL)) |
+--------------------------------------+
```

```
| NULL                               |
+------------------------------------+
```

ST_GEOMETRYFROMWKB

Description

Converts a standard figure WKB (Well-known binary) to the corresponding memory geometry

Alias

- ST_GEOMFROMWKB

Syntax

```
ST_GEOMETRYFROMWKB( <wkb>)
```

Parameters

Parameters	Instructions
<wkb>	The memory form of the graph

Return Value

The corresponding geometric storage form of WKB

Examples

```
select ST_AsText(ST_GeometryFromWKB(ST_AsBinary(ST_Point(24.7, 56.7))));
```

```
+-------------------------------------------------------------------+
| st_astext(st_geometryfromwkb(st_asbinary(st_point(24.7, 56.7)))) |
+-------------------------------------------------------------------+
| POINT (24.7 56.7)                                                 |
+-------------------------------------------------------------------+
```

```
select ST_AsText(ST_GeomFromWKB(ST_AsBinary(ST_Point(24.7, 56.7))));
```

```
+-----------------------------------------------------------+
| st_astext(st_geomfromwkb(st_asbinary(st_point(24.7, 56.7)))) |
+-----------------------------------------------------------+
| POINT (24.7 56.7)                                         |
+-----------------------------------------------------------+
```

```
select ST_AsText(ST_GeometryFromWKB(ST_AsBinary(ST_GeometryFromText("LINESTRING (1 1, 2 2)"))));
```

```
+-------------------------------------------------------------------------------------+
| st_astext(st_geometryfromwkb(st_asbinary(st_geometryfromtext('LINESTRING (1 1, 2 2)')))) |
+-------------------------------------------------------------------------------------+
| LINESTRING (1 1, 2 2)                                                               |
+-------------------------------------------------------------------------------------+
```

```
select ST_AsText(ST_GeometryFromWKB(ST_AsBinary(ST_Polygon("POLYGON ((114.104486 22.547119,114.093758 22.547753,114.096504
22.532057,114.104229 22.539826,114.106203 22.542680,114.104486 22.547119))"))));
```

```
+-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| st_astext(st_geometryfromwkb(st_asbinary(st_polygon('POLYGON ((114.104486 22.547119,114.093758 22.547753,114.096504
22.532057,114.104229 22.539826,114.106203 22.542680,114.104486 22.547119))')))) |
+-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| POLYGON ((114.104486 22.547119, 114.093758 22.547753, 114.096504 22.532057, 114.104229 22.539826, 114.106203 22.54268, 114.104486
22.547119))                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
```

```
select ST_AsText(ST_GeomFromWKB(ST_AsBinary(ST_Polygon("POLYGON ((114.104486 22.547119,114.093758 22.547753,114.096504
22.532057,114.104229 22.539826,114.106203 22.542680,114.104486 22.547119))"))));
```

```
+-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| st_astext(st_geomfromwkb(st_asbinary(st_polygon('POLYGON ((114.104486 22.547119,114.093758 22.547753,114.096504
22.532057,114.104229 22.539826,114.106203 22.542680,114.104486 22.547119))')))) |
+-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| POLYGON ((114.104486 22.547119, 114.093758 22.547753, 114.096504 22.532057, 114.104229 22.539826, 114.106203 22.54268, 114.104486
22.547119))                                                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
```

ST_INTERSECTS

Description

Determines whether the geometry shape1 intersects with the geometry shape2.

> **Note**
>
> Supported since Apache Doris 2.1.10.

Syntax

```
ST_INTERSECTS( <shape1>, <shape2> )
```

Parameters

Parameters	Instructions
<shape1>	The passed geometry used to determine whether it intersects with shape2
<shape2>	The passed geometry used to determine whether it intersects with shape1

Return Value

Return 1: shape1 The graph intersects with the graph shape2

Return 0: shape1 The graph does not intersect with the graph shape2

Examples

```
SELECT ST_Intersects(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(5, 5));
```

```
+-----------------------------------------------------------------------------------+
| ST_Intersects(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(5, 5)) |
+-----------------------------------------------------------------------------------+
|                                                                                 1 |
+-----------------------------------------------------------------------------------+
```

```
SELECT ST_Intersects(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(50, 50));
```

```
+-------------------------------------------------------------------------------------+
| ST_Intersects(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_Point(50, 50)) |
+-------------------------------------------------------------------------------------+
```

```
|                                                                                      0 |
+----------------------------------------------------------------------------------------+
```

ST_LINEFROMTEXT

Description

Converts a WKT (Well Known Text) to a memory representation in the form of a Line

Alias

- ST_LINESTRINGFROMTEXT

Syntax

```
ST_LINEFROMTEXT( <wkt>)
```

Parameters

Parameters	Instructions
<wkt>	A line segment consisting of two coordinates

Return Value

The memory form of a line segment.

Examples

```
SELECT ST_AsText(ST_LineFromText("LINESTRING (1 1, 2 2)"));
```

```
+-----------------------------------------------------+
| st_astext(st_geometryfromtext('LINESTRING (1 1, 2 2)')) |
+-----------------------------------------------------+
| LINESTRING (1 1, 2 2)                               |
+-----------------------------------------------------+
```

ST_POINT

Description

With the given X coordinate value, the Y coordinate value returns the corresponding Point.

The current value is only meaningful in the sphere set, X/Y corresponds to longitude/latitude;

Syntax

```
ST_POINT( <x>, <y>)
```

Parameters

Parameters	Instructions
<x>	x-coordinate
<y>	y-coordinate

Return Value

Given horizontal coordinate and vertical coordinate corresponding position information

Examples

```
SELECT ST_AsText(ST_Point(24.7, 56.7));
```

```
+--------------------------------+
| st_astext(st_point(24.7, 56.7)) |
+--------------------------------+
| POINT (24.7 56.7)              |
+--------------------------------+
```

ST_POLYGON

Description

Convert a WKT (Well Known Text) to the corresponding polygonal memory form

Alias

- ST_POLYGONFROMTEXT
- ST_POLYFROMTEXT

Syntax

```
ST_POLYGON( <wkt>)
```

Parameters

Parameters	Instructions
<wkt>	A POLYGON generated by the polygon function

Return Value

Memory form of polygon

Examples

```
SELECT ST_AsText(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"));
```

```
+-----------------------------------------------------------------+
| st_astext(st_polygon('POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))')) |
+-----------------------------------------------------------------+
| POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))                         |
+-----------------------------------------------------------------+
```

ST_TOUCHES

Description

Determines whether the geometry shape1 touches the geometry shape2. (i.e., whether the boundaries of the two geometries intersect but their interiors do not)

> **Note**
>
> Supported since Apache Doris 2.1.10.

Syntax

```
ST_TOUCHES( <shape1>, <shape2> )
```

Parameters

Parameters	Instructions
<shape1>	The passed geometry used to determine whether it touches shape2
<shape2>	The passed geometry used to determine whether it touches shape1

Return Value

Return 1: shape1 The graph touches the graph shape2

Return 0: shape1 The graph does not touch the graph shape2

Examples

```
SELECT ST_Touches(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_LineStringFromText("LINESTRING (10 5, 15 5)"));
```

```
+-----------------------------------------------------------------------------------------------------------------+
| ST_Touches(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_LineStringFromText("LINESTRING (10 5, 15 5)")) |
+-----------------------------------------------------------------------------------------------------------------+
|                                                                                                               1 |
+-----------------------------------------------------------------------------------------------------------------+
```

```
SELECT ST_Touches(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_LineStringFromText("LINESTRING (5 5, 15 5)"));
```

```
+-----------------------------------------------------------------------------------------------------------------+
| ST_Touches(ST_Polygon("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"), ST_LineStringFromText("LINESTRING (5 5, 15 5)")) |
+-----------------------------------------------------------------------------------------------------------------+
```

```
|                                                
```

ST_X

Description

When point is a valid POINT type, return the corresponding x-coordinate value

Syntax

```
ST_X( <point>)
```

Parameters

Parameters	Instructions
<point>	The geometric coordinates of a two-dimensional point

Return Value

X value in geometric coordinates

Examples

```
SELECT ST_X(ST_Point(24.7, 56.7));
```

```
+----------------------------+
| st_x(st_point(24.7, 56.7)) |
+----------------------------+
|                       24.7 |
+----------------------------+
```

ST_Y

Description

When point is a valid POINT type, return the corresponding y-coordinate value

Syntax

```
ST_Y( <point>)
```

Parameters

Parameters	Instructions
<point>	The geometric coordinates of a two-dimensional point

Return Value

Y value in geometric coordinates

Examples

```
SELECT ST_Y(ST_Point(24.7, 56.7));
```

```
+---------------------------+
| ST_Y(ST_Point(24.7, 56.7)) |
+---------------------------+
| 56.7                      |
+---------------------------+
```

AES_DECRYPT

Description

AES decryption function. This function behaves the same as the `AES_DECRYPT` function in MySQL. The default algorithm is `AES_128_ECB`, and the padding mode is `PKCS7`.

> **Until 2.1.6, function with two arguments will ignore session variable `block_encryption_mode` and always use `AES_128_ECB` to do decryption. So it's not recommended to use it.**
>
> Since 2.1.7, it works as expected.

Syntax

```
AES_DECRYPT( <str>, <key_str>[, <init_vector>][, <encryption_mode>])
```

Parameters

parameter	description
`<str>`	The text to be decrypted
`<key_str>`	is the key. Note that this key is not a hexadecimal encoding, but an encoded string representation. For example, for 128-bit key encryption, the length of `key_str` should be 16. If the key length is insufficient, use **zero padding** to make it complete. If the length exceeds, use circular XOR to find the final key. For example, if the 128-bit key used by the algorithm is `key`, then `key[i] = key_str[i] ^ key_str[i+128] ^ key_str[i+256] ^ ...`
`<init_vector>`	It is the initial vector used in the algorithm. It is only effective under specific algorithms. If not specified, Doris uses the built-in vector
`<encryption_mode>`	For encryption algorithms, optional values are given in variables

Return Value

If decryption is successful: Returns the decrypted data, usually a binary representation of the plaintext.

If decryption fails: Returns NULL.

Examples

Decryption successful

Using the default algorithm

```
set block_encryption_mode='';
select aes_decrypt(from_base64('wr2JEDVXzL9+2XtRhgIloA=='),'F3229A0B371ED2D9441B830D21A390C3');
```

```
+--------------------------------------------------------------------------------+
| aes_decrypt(from_base64('wr2JEDVXzL9+2XtRhgIloA=='), '***', '', 'AES_128_ECB') |
+--------------------------------------------------------------------------------+
| text                                                                           |
+--------------------------------------------------------------------------------+
```

Use AES_256_CBC algorithm

```
set block_encryption_mode="AES_256_CBC";
select aes_decrypt(from_base64('3dym0E7/+1zbrLIaBVNHSw=='),'F3229A0B371ED2D9441B830D21A390C3');
```

```
+--------------------------------------------------------------------------------+
| aes_decrypt(from_base64('3dym0E7/+1zbrLIaBVNHSw=='), '***', '', 'AES_256_CBC') |
+--------------------------------------------------------------------------------+
| text                                                                           |
+--------------------------------------------------------------------------------+
```

Use AES_256_CBC algorithm and set initial vector

```
select AES_DECRYPT(FROM_BASE64('tsmK1HzbpnEdR2//WhO+MA=='),'F3229A0B371ED2D9441B830D21A390C3', '0123456789');
```

```
+-----------------------------------------------------------------------------------------+
| aes_decrypt(from_base64('tsmK1HzbpnEdR2//WhO+MA=='), '***', '0123456789', 'AES_256_CBC') |
+-----------------------------------------------------------------------------------------+
| text                                                                                    |
+-----------------------------------------------------------------------------------------+
```

Decryption failed

```
select AES_DECRYPT(FROM_BASE64('tsmK1H3zbpnEdR2//WhO+MA=='),'F3229A0B371ED2D9441B830D21A390C3', '0123456789');
```

```
+-----------------------------------------------------------------------------------------+
| aes_decrypt(from_base64('tsmK1H3zbpnEdR2//WhO+MA=='), '***', '0123456789', 'AES_256_CBC') |
+-----------------------------------------------------------------------------------------+
| NULL                                                                                    |
+-----------------------------------------------------------------------------------------+
```

AES_ENCRYPT

Description

AES encryption function. This function behaves the same as the `AES_ENCRYPT` function in MySQL. The default algorithm is `AES_128_ECB`, and the padding mode is `PKCS7`.

The AES_ENCRYPT function does not use the passed key directly, but further processes it. The specific steps are as follows:

1. Determine the number of bytes of the key according to the encryption algorithm used. For example, if the AES_128_ECB algorithm is used, the number of bytes of the key is `128 / 8 = 16` (if the AES_256_ECB algorithm is used, the number of bytes of the key is `256 / 8 = 32`);
2. Then, for the key input by the user, the `i`th byte and the `16*k+i`th byte are XORed. If the key input by the user is less than 16 bytes, 0 is added at the end.
3. Finally, use the newly generated key to encrypt;

> **Until 2.1.6, function with two arguments will ignore session variable `block_encryption_mode` and always use `AES_128_ECB` to do encryption. So it's not recommended to use it.**
>
> Since 2.1.7, it works as expected.

Syntax

```
AES_ENCRYPT( <str>, <key_str>[, <init_vector>][, <encryption_mode>])
```

Parameters

parameter	description
`<str>`	The text to be encrypted
`<key_str>`	is the key. Note that this key is not a hexadecimal encoding, but an encoded string representation. For example, for 128-bit key encryption, the length of `key_str` should be 16. If the key length is insufficient, use **zero padding** to make it complete. If the length exceeds, use circular XOR to find the final key. For example, if the 128-bit key used by the algorithm is `key`, then `key[i] = key_str[i] ^ key_str[i+128] ^ key_str[i+256] ^ ...`
`<init_vector>`	It is the initial vector used in the algorithm. It is only effective under specific algorithms. If not specified, Doris uses the built-in vector
`<encryption_mode>`	For encryption algorithms, optional values are given in variables

Return Value

Returns the encrypted binary data

Examples

Using AES_128_ECB algorithm

```
set block_encryption_mode='';
select to_base64(aes_encrypt('text','F3229A0B371ED2D9441B830D21A390C3'));
```

```
+-------------------------------------------------------------+
| to_base64(aes_encrypt('text', '***', '', 'AES_128_ECB')) |
+-------------------------------------------------------------+
| wr2JEDVXzL9+2XtRhgIloA==                                    |
+-------------------------------------------------------------+
```

Use AES_256_CBC algorithm

```
set block_encryption_mode="AES_256_CBC";
select to_base64(aes_encrypt('text','F3229A0B371ED2D9441B830D21A390C3'));
```

```
+-------------------------------------------------------------+
| to_base64(aes_encrypt('text', '***', '', 'AES_256_CBC')) |
+-------------------------------------------------------------+
| 3dymOE7/+1zbrLIaBVNHSw==                                    |
+-------------------------------------------------------------+
```

Use AES_256_CBC algorithm and set initial vector

```
select to_base64(aes_encrypt('text','F3229A0B371ED2D9441B830D21A390C3', '0123456789'));
```

```
+--------------------------------------------------------------------+
| to_base64(aes_encrypt('text', '***', '0123456789', 'AES_256_CBC')) |
+--------------------------------------------------------------------+
| tsmK1HzbpnEdR2//WhO+MA==                                           |
+--------------------------------------------------------------------+
```

CRC32

Description

Use CRC32 to compute the result.

Syntax

```
CRC32( <str> )
```

Parameters

parameter	description
<str>	The value to be used for CRC calculation

Return Value

Returns the Cyclic Redundancy Check value of this string.

Examples

```
select crc32("abc"),crc32("中国");
```

```
+--------------+-----------------+
| crc32('abc') | crc32('中国')   |
+--------------+-----------------+
|    891568578 |       737014929 |
+--------------+-----------------+
```

MD5

description

Calculates an MD5 128-bit checksum for the string

Syntax

```
MD5( <str> )
```

Parameters

parameter	description
<str>	The MD5 value to be calculated

Return Value

Returns the MD5 value of a string.

Examples

```
select md5("abc");
```

```
+----------------------------------+
| md5('abc')                       |
+----------------------------------+
| 900150983cd24fb0d6963f7d28e17f72 |
+----------------------------------+
```

MD5SUM

description

Calculates an MD5 128-bit checksum for the strings

Syntax

```
MD5SUM( <str> [ , <str> ... ] )
```

Parameters

parameter	description
<str>	The MD5 value to be calculated

Return Value

Returns the MD5 value of multiple strings

Examples

```
select md5("abcd"),md5sum("ab","cd");
```

```
+----------------------------------+----------------------------------+
| md5('abcd')                      | md5sum('ab', 'cd')               |
+----------------------------------+----------------------------------+
| e2fc714c4727ee9395f324cd2e7f331f | e2fc714c4727ee9395f324cd2e7f331f |
+----------------------------------+----------------------------------+
```

MURMUR_HASH3_32

Description

Calculate 32-bit murmur3 hash value

- Note: When calculating hash values, it is recommended to use `xxhash_32` instead of `murmur_hash3_32`.

Syntax

```
MURMUR_HASH3_32( <str> [ , <str> ... ] )
```

Parameters

parameter	description
<str>	The 32-bit murmur3 hash value to be calculated

Return Value

Returns the 32-bit murmur3 hash of the input string.

-When the parameter is NULL, it returns NULL

Examples

```
select murmur_hash3_32(null), murmur_hash3_32("hello"), murmur_hash3_32("hello", "world");
```

```
+----------------------+-------------------------+----------------------------------+
| murmur_hash3_32(NULL) | murmur_hash3_32('hello') | murmur_hash3_32('hello', 'world') |
+----------------------+-------------------------+----------------------------------+
|                 NULL |              1321743225 |                        984713481 |
+----------------------+-------------------------+----------------------------------+
```

MURMUR_HASH3_64

Description

Calculate 64-bit murmur3 hash value

- Note: After testing, the performance of `xxhash_64` is about twice that of `murmur_hash3_64`, so when calculating hash values, it is recommended to use `xxhash_64` instead of `murmur_hash3_64`.

Syntax

```
MURMUR_HASH3_64( <str> [ , <str> ... ] )
```

Parameters

parameter	description
<str>	The 64-bit murmur3 hash value to be calculated

Return Value

Returns the 64-bit murmur3 hash of the input string.

Examples

```
select murmur_hash3_64(null), murmur_hash3_64("hello"), murmur_hash3_64("hello", "world");
```

```
+----------------------+-------------------------+----------------------------------+
| murmur_hash3_64(NULL) | murmur_hash3_64('hello') | murmur_hash3_64('hello', 'world') |
+----------------------+-------------------------+----------------------------------+
|                 NULL |     -3215607508166160593 |               3583109472027628045 |
+----------------------+-------------------------+----------------------------------+
```

SHA1

Description

Use the SHA1 algorithm to digest the information.

Alias

SHA

Syntax

```
SHA1( <str> )
```

Parameters

parameter	description
<str>	The sha1 value to be calculated

Return Value

Returns the sha1 value of the input string

Examples

```
select sha("123"), sha1("123");
```

```
+------------------------------------------+------------------------------------------+
| sha1('123')                              | sha1('123')                              |
+------------------------------------------+------------------------------------------+
| 40bd001563085fc35165329ea1ff5c5ecbdbbeef | 40bd001563085fc35165329ea1ff5c5ecbdbbeef |
+------------------------------------------+------------------------------------------+
```

SHA2

Description

The information is digested using SHA2.

Syntax

```
SHA2( <str>, <digest_length>)
```

Parameters

parameter	description
<str>	Content to be encrypted
<digest_length>	Summary length, supports 224, 256, 384, 512

Return Value

Returns the sha2 value of the input string

Examples

```
select sha2('abc', 224), sha2('abc', 384), sha2(NULL, 512);
```

```
+----------------------------------------------------------+----------------------------------------------------------------------
-------------------------+-----------------+
| sha2('abc', 224)                                         | sha2('abc', 384)
| sha2(NULL, 512) |
+----------------------------------------------------------+----------------------------------------------------------------------
-------------------------+-----------------+
| 23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7 |
cb00753f45a35e8bb5a03d699ac65007272c32ab0eded1631a8b605a43ff5bed8086072ba1e7cc2358baeca134c825a7 | NULL            |
+----------------------------------------------------------+----------------------------------------------------------------------
-------------------------+-----------------+
```

SM3

Description

Calculation SM3 256-bit

Syntax

```
SM3( <str> )
```

Parameters

parameter	description
<str>	The value of sm3 that needs to be calculated

Return Value

Returns the sm3 value of the input string

Examples

```
select sm3("abcd");
```

```
+------------------------------------------------------------------+
| sm3('abcd')                                                      |
+------------------------------------------------------------------+
| 82ec580fe6d36ae4f81cae3c73f4a5b3b5a09c943172dc9053c69fd8e18dca1e |
+------------------------------------------------------------------+
```

SM3SUM

Description

Calculate multiple strings SM3 256-bit

Syntax

```
SM3SUM( <str> [ , <str> ... ] )
```

Parameters

parameter	description
<str>	The value of sm3 that needs to be calculated

Return Value

Returns the sm3 value of the input multiple strings

Examples

```
select sm3sum("ab","cd");
```

```
+------------------------------------------------------------------+
| sm3sum('ab', 'cd')                                               |
+------------------------------------------------------------------+
| 82ec580fe6d36ae4f81cae3c73f4a5b3b5a09c943172dc9053c69fd8e18dca1e |
+------------------------------------------------------------------+
```

SM4_DECRYPT

Description

SM4 is a national standard symmetric key encryption algorithm, widely used in finance, communications, e-commerce and other fields. The SM4_DECRYPT function is used to decrypt data using SM4. The default algorithm is `SM4_128_ECB`.

> **Until 2.1.6, function with two arguments will ignore session variable `block_encryption_mode` and always use `SM4_128_ECB` to do decryption.**
>
> Since 2.1.7, it works as expected.

Syntax

```
SM4_DECRYPT( <str>, <key_str>[, <init_vector>][, <encryption_mode>])
```

Parameters

parameter	description
`<str>`	The text to be decrypted
`<key_str>`	is the key. Note that this key is not a hexadecimal encoding, but an encoded string representation. For example, for 128-bit key encryption, the length of `key_str` should be 16. If the key length is insufficient, use **zero padding** to make it complete. If the length exceeds, use circular XOR to find the final key. For example, if the 128-bit key used by the algorithm is `key`, then `key[i] = key_str[i] ^ key_str[i+128] ^ key_str[i+256] ^ ...`
`<init_vector>`	It is the initial vector used in the algorithm. It is only effective under specific algorithms. If not specified, Doris uses the built-in vector
`<encryption_mode>`	For encryption algorithms, optional values are given in variables

Return Value

If decryption is successful: Returns the decrypted data, usually a binary representation of the plaintext.

If decryption fails: Returns NULL.

Examples

Decryption successful

Using the default algorithm

```
set block_encryption_mode='';
select SM4_DECRYPT(FROM_BASE64('aDjwRflBrDjhBZIOFNw3Tg=='),'F3229A0B371ED2D9441B830D21A390C3');
```

```
+-------------------------------------------------------------------------------+
| sm4_decrypt(from_base64('aDjwRflBrDjhBZIOFNw3Tg=='), '***', '', 'SM4_128_ECB') |
+-------------------------------------------------------------------------------+
| text                                                                          |
+-------------------------------------------------------------------------------+
```

Using SM4_128_CBC algorithm

```
set block_encryption_mode="SM4_128_CBC";
select SM4_DECRYPT(FROM_BASE64('FSYstvOmH2cXy7B/072Mug=='),'F3229A0B371ED2D9441B830D21A390C3');
```

```
+-------------------------------------------------------------------------------+
| sm4_decrypt(from_base64('FSYstvOmH2cXy7B/072Mug=='), '***', '', 'SM4_128_CBC') |
+-------------------------------------------------------------------------------+
| text                                                                          |
+-------------------------------------------------------------------------------+
```

Use SM4_128_CBC algorithm and initial vector

```
select SM4_DECRYPT(FROM_BASE64('1Y4NGIukSbv9OrkZnRD1bQ=='),'F3229A0B371ED2D9441B830D21A390C3', '0123456789');
```

```
+-----------------------------------------------------------------------------------------+
| sm4_decrypt(from_base64('1Y4NGIukSbv9OrkZnRD1bQ=='), '***', '0123456789', 'SM4_128_CBC') |
+-----------------------------------------------------------------------------------------+
| text                                                                                    |
+-----------------------------------------------------------------------------------------+
```

Decryption failed

```
set block_encryption_mode='';
select SM4_DECRYPT(FROM_BASE64('aDjwRflBrDjhBZIdOFNw3Tg=='),'F3229A0B371ED2D9441B830D21A390C3');
```

```
+-------------------------------------------------------------------------------+
| sm4_decrypt(from_base64('aDjwRflBrDjhBZIdOFNw3Tg=='), '***', '', 'SM4_128_ECB') |
+-------------------------------------------------------------------------------+
| NULL                                                                          |
+-------------------------------------------------------------------------------+
```

SM4_ENCRYPT

Description

SM4 is a national standard symmetric key encryption algorithm, widely used in finance, communications, e-commerce and other fields. The SM4_ENCRYPT function is used to encrypt data with SM4. The default algorithm is `SM4_128_ECB`.

> **Until 2.1.6, function with two arguments will ignore session variable `block_encryption_mode` and always use `SM4_128_ECB` to do encryption.**
>
> Since 2.1.7, it works as expected.

Syntax

```
SM4_ENCRYPT( <str>, <key_str>[, <init_vector>][, <encryption_mode>])
```

Parameters

parameter	description
`<str>`	The text to be encrypted
`<key_str>`	is the key. Note that this key is not a hexadecimal encoding, but an encoded string representation. For example, for 128-bit key encryption, the length of `key_str` should be 16. If the key length is insufficient, use **zero padding** to make it complete. If the length exceeds, use circular XOR to find the final key. For example, if the 128-bit key used by the algorithm is `key`, then `key[i] = key_str[i] ^ key_str[i+128] ^ key_str[i+256] ^ ...`
`<init_vector>`	It is the initial vector used in the algorithm. It is only effective under specific algorithms. If not specified, Doris uses the built-in vector
`<encryption_mode>`	For encryption algorithms, optional values are given in variables

Return Value

Returns the encrypted binary data

Examples

Using the default algorithm

```
set block_encryption_mode='';
select TO_BASE64(SM4_ENCRYPT('text','F3229A0B371ED2D9441B830D21A390C3'));
```

```
+------------------------------------------------------------+
| to_base64(sm4_encrypt('text', '***', '', 'SM4_128_ECB')) |
+------------------------------------------------------------+
| aDjwRflBrDjhBZIOFNw3Tg==                                   |
+------------------------------------------------------------+
```

Using SM4_128_CBC algorithm

```
set block_encryption_mode="SM4_128_CBC";
select TO_BASE64(SM4_ENCRYPT('text','F3229AOB371ED2D9441B830D21A390C3'));
```

```
+------------------------------------------------------------+
| to_base64(sm4_encrypt('text', '***', '', 'SM4_128_CBC')) |
+------------------------------------------------------------+
| FSYstvOmH2cXy7B/O72Mug==                                   |
+------------------------------------------------------------+
```

Use the SM4_128_CBC algorithm and set the initial vector

```
set block_encryption_mode="SM4_128_CBC";
select to_base64(SM4_ENCRYPT('text','F3229AOB371ED2D9441B830D21A390C3', '0123456789'));
```

```
+--------------------------------------------------------------------+
| to_base64(sm4_encrypt('text', '***', '0123456789', 'SM4_128_CBC')) |
+--------------------------------------------------------------------+
| 1Y4NGIukSbv9OrkZnRD1bQ==                                           |
+--------------------------------------------------------------------+
```

XXHASH_32

Description

Calculate the 32-bit xxhash value of the input string

Syntax

```
XXHASH_32( <str> [ , <str> ... ] )
```

Parameters

parameter	description
<str>	The 32-bit xxhash value to be calculated

Return Value

Returns the 32-bit xxhash value of the input string.

Examples

```
select xxhash_32(NULL), xxhash_32("hello"), xxhash_32("hello", "world");
```

```
+-----------------+-------------------+----------------------------+
| xxhash_32(NULL) | xxhash_32('hello') | xxhash_32('hello', 'world') |
+-----------------+-------------------+----------------------------+
|            NULL |          -83855367 |                 -920844969 |
+-----------------+-------------------+----------------------------+
```

XXHASH_64

Description

Calculates the 64-bit xxhash value of the input string

- Note: After testing, the performance of `xxhash_64` is about twice that of `murmur_hash3_64`, so when calculating hash values, it is recommended to use `xxhash_64` instead of `murmur_hash3_64`.

Syntax

```
XXHASH_64( <str> [ , <str> ... ] )
```

Parameters

parameter	description
<str>	The 64-bit xxhash value to be calculated

Return Value

Returns the 64-bit xxhash value of the input string.

Examples

```
select xxhash_64(NULL), xxhash_64("hello"), xxhash_64("hello", "world");
```

```
+-----------------+----------------------+----------------------------+
| xxhash_64(NULL) | xxhash_64('hello')   | xxhash_64('hello', 'world') |
+-----------------+----------------------+----------------------------+
|            NULL | -7685981735718036227 |         7001965798170371843 |
+-----------------+----------------------+----------------------------+
```

COSINE_DISTANCE

Description

Calculates the cosine distance between two vectors (vector values are coordinates)

Syntax

```
COSINE_DISTANCE(<array1>, <array2>)
```

Parameters

Parameter	Description
<array1>	The first vector (the vector value is the coordinate). The subtypes of the input array are: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE. The number of elements must be consistent with array2
<array2>	The second vector (the vector value is the coordinate), the subtype of the input array supports: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, the number of elements must be consistent with array1

Return Value

Returns the cosine distance between two vectors (where the vector values are coordinates). If the input array is NULL, or any element in the array is NULL, then NULL is returned.

Example

```
SELECT COSINE_DISTANCE([1, 2], [2, 3]),COSINE_DISTANCE([3, 6], [4, 7]);
```

```
+---------------------------------+---------------------------------+
| cosine_distance([1, 2], [2, 3]) | cosine_distance([3, 6], [4, 7]) |
+---------------------------------+---------------------------------+
|            0.007722123286332261 |           0.0015396467945875125 |
+---------------------------------+---------------------------------+
```

INNER_PRODUCT

Description

Computes the scalar product of two vectors of the same size

Syntax

```
INNER_PRODUCT(<array1>, <array2>)
```

Parameters

Parameter	Description
<array1>	The first vector, the subtype of the input array supports: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, the number of elements must be consistent with array2
<array2>	The second vector, the subtype of the input array supports: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, the number of elements must be consistent with array1

Return Value

Returns the scalar product of two vectors of the same size. If the input array is NULL, or any element in array is NULL, then NULL is returned.

Examples

```
SELECT INNER_PRODUCT([1, 2], [2, 3]),INNER_PRODUCT([3, 6], [4, 7]);
```

```
+-------------------------------+-------------------------------+
| inner_product([1, 2], [2, 3]) | inner_product([3, 6], [4, 7]) |
+-------------------------------+-------------------------------+
|                             8 |                            54 |
+-------------------------------+-------------------------------+
```

L1_DISTANCE

Description

Calculate the distance between two points in L1 space (vector values are coordinates)

Syntax

```
L1_DISTANCE(<array1>, <array2>)
```

Parameters

Parameter	Description
<array1>	The first vector (the vector value is the coordinate). The subtypes of the input array are: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE. The number of elements must be consistent with array2
<array2>	The second vector (the vector value is the coordinate), the subtype of the input array supports: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, the number of elements must be consistent with array1

Return Value

Returns the distance between two points (vector values are coordinates) in L1 space. If the input array is NULL, or any element in the array is NULL, then NULL is returned.

Example

```
SELECT L1_DISTANCE([4, 5], [6, 8]),L1_DISTANCE([3, 6], [4, 5]);
```

```
+----------------------------+----------------------------+
| l1_distance([4, 5], [6, 8]) | l1_distance([3, 6], [4, 5]) |
+----------------------------+----------------------------+
|                          5 |                          2 |
+----------------------------+----------------------------+
```

L2_DISTANCE

Description

Calculate the distance between two points (vector values are coordinates) in Euclidean space

Syntax

```
L2_DISTANCE(<array1>, <array2>)
```

Parameters

Parameter	Description
<array1>	The first vector (the vector value is the coordinate),The subtypes of the input array are: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, The number of elements must be consistent with array2
<array2>	The second vector (the vector value is the coordinate), the subtype of the input array supports: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, FLOAT, DOUBLE, the number of elements must be consistent with array1

Return Value

Returns the distance between two points (vector values are coordinates) in Euclidean space. If the input array is NULL, or any element in the array is NULL, then NULL is returned.

Example

```
SELECT L2_DISTANCE([4, 5], [6, 8]),L2_DISTANCE([3, 6], [4, 5]);
```

```
+----------------------------+----------------------------+
| l2_distance([4, 5], [6, 8]) | l2_distance([3, 6], [4, 5]) |
+----------------------------+----------------------------+
|           3.605551275463989 |          1.4142135623730951 |
+----------------------------+----------------------------+
```

BIT_LENGTH

Description

It is used to return the median of the binary representation of a string (that is, the total number of binary digits). It calculates the number of bits occupied by the binary encoding of the string.

Syntax

```
BIT_LENGTH( <str>)
```

Parameters

parameter	description
<str>	The string to be calculated

Return Value

Returns the number of bits occupied by <str> in the binary representation, including all 0 and 1.

Examples

```
select BIT_LENGTH("abc"), BIT_LENGTH("中国"), BIT_LENGTH(123);
```

```
+------------------+----------------------+---------------------------------------+
| bit_length('abc') | bit_length('中国')   | bit_length(cast(123 as VARCHAR(65533))) |
+------------------+----------------------+---------------------------------------+
|                24 |                   48 |                                    24 |
+------------------+----------------------+---------------------------------------+
```

BITAND

Description

Used to perform a bitwise AND operation. The bitwise AND operation compares each bit of two integers. The result is 1 only when both corresponding binary bits are 1, otherwise it is 0.

Integer range: TINYINT, SMALLINT, INT, BIGINT, LARGEINT

Syntax

```
BITAND( <lhs>, <rhs>)
```

Parameters

parameter	description
<lhs>	The first number involved in the bitwise AND operation
<rhs>	The second number to be included in the bitwise AND operation

Return Value

Returns the result of the AND operation on two integers.

Examples

```
select BITAND(3,5), BITAND(4,7);
```

```
+---------+---------+
| (3 & 5) | (4 & 7) |
+---------+---------+
|       1 |       4 |
+---------+---------+
```

BIT_COUNT

Description

Used to return the number of 1 bits in the binary representation of an integer value. This function can be used to quickly count the number of “active” bits of an integer in the binary representation, and is usually used to analyze data distribution or perform certain bit operations

Syntax

```
BIT_COUNT( <x>)
```

Parameters

parameter	description
<x>	Counts the number of 1s in the binary representation of integer x. Integer types can be: TINYINT, SMALLINT, INT, BIGINT, LARGEINT

Return Value

Returns the number of 1s in the binary representation of <x>

Examples

```
select BIT_COUNT(8), BIT_COUNT(-1);
```

```
+--------------+---------------+
| bit_count(8) | bit_count(-1) |
+--------------+---------------+
|            1 |             8 |
+--------------+---------------+
```

BITNOT

Description

Used to perform a bitwise inversion operation on an integer.

Integer range: TINYINT, SMALLINT, INT, BIGINT, LARGEINT

Syntax

```
BITNOT( <x>)
```

Parameters

parameter	description
<x>	Integer operations

Return Value

Returns the result of the NOT operation of one integer.

Examples

```
select BITNOT(7), BITNOT(-127);
```

```
+-------+----------+
| (~ 7) | (~ -127) |
+-------+----------+
|    -8 |      126 |
+-------+----------+
```

BITOR

Description

Used to perform a bitwise OR operation on two integers.

Integer range: TINYINT, SMALLINT, INT, BIGINT, LARGEINT

Syntax

```
BITOR( <lhs>, <rhs>)
```

Parameters

parameter	description
<lhs>	The first BOOLEAN value to be evaluated
<rhs>	The second BOOLEAN value to be evaluated

Return Value

Returns the result of the OR operation on two integers.

Examples

```
select BITOR(3,5), BITOR(4,7);
```

```
+---------+---------+
| (3 | 5) | (4 | 7) |
+---------+---------+
|       7 |       7 |
+---------+---------+
```

BIT_SHIFT_LEFT

Description

Functions for left shift operations are usually used to perform bit shift operations, which shift all bits of a binary number to the left by a specified number of bits. It is a form of bitwise operation that is often used to process binary data or perform efficient mathematical calculations.

For the maximum value of BIGINT type, 9223372036854775807, a one-bit left shift results in -2. ### Syntax

```
BIT_SHIFT_LEFT( <x>, <bits>)
```

Parameters

parameter	description
<x>	The number to be shifted
<bits>	The number of bits to shift left. It is an integer that determines how many bits <x> will be shifted left

Return Value

Returns an integer representing the result of a left shift operation.

Examples

```
select BIT_SHIFT_LEFT(5, 2), BIT_SHIFT_LEFT(-5, 2), BIT_SHIFT_LEFT(9223372036854775807, 1);
```

```
+---------------------+----------------------+--------------------------------------+
| bit_shift_left(5, 2) | bit_shift_left(-5, 2) | bit_shift_left(9223372036854775807, 1) |
+---------------------+----------------------+--------------------------------------+
|                  20 |                  -20 |                                   -2 |
+---------------------+----------------------+--------------------------------------+
```

BIT_SHIFT_RIGHT

Description

Used for right shift operations, usually used to shift all bits of a binary number to the right by a specified number of bits. This operation is usually used to process binary data, or for some mathematical calculations (such as efficient implementation of division).

The result of logically shifting -1 right by one position is BIGINT_MAX(9223372036854775807).

Shifting a number right by a negative amount always results in a result of 0.

Syntax

```
BIT_SHIFT_RIGHT( <x>, <bits>)
```

Parameters

parameter	description
<x>	The number to be shifted
<bits>	The number of bits to shift right. It is an integer that determines how many bits <x> will be shifted right.

Return Value

Returns an integer representing the result of a right shift operation.

Examples

```
select BIT_SHIFT_RIGHT(1024,3), BIT_SHIFT_RIGHT(-1,1), BIT_SHIFT_RIGHT(100, -1);
```

```
+--------------------------+-----------------------+--------------------------+
| bit_shift_right(1024, 3) | bit_shift_right(-1, 1) | bit_shift_right(100, -1) |
+--------------------------+-----------------------+--------------------------+
|                      128 |    9223372036854775807 |                        0 |
+--------------------------+-----------------------+--------------------------+
```

ARRAY | Array Functions

ARRAY

Description

construct an array with variadic elements and return it, T could be column or literal

Syntax

```
ARRAY([ <element> [, ...] ])
```

Parameters

Parameter	Description
<element>	The parameter can be multiple columns or constants

Return Value

Returns an array.

Example

```
-- setup
create table nested(qid int, creationDate datetime) distributed by hash(creationDate) buckets 1 properties ("replication_num"="1");
insert into nested values (null,'2009-06-16 07:40:56'),(null,'2009-06-16 07:50:05'),(null,'2009-06-16 08:09:18'),(null,'2009-06-16
08:15:45');
```

```
select array("1", 2, 1.1);
```

```
+-----------------------------------------------+
| array('1', cast(2 as TEXT), cast(1.1 as TEXT)) |
+-----------------------------------------------+
| ["1", "2", "1.1"]                             |
+-----------------------------------------------+
```

```
select array(null, 1);
```

```
+----------------+
| array(NULL, 1) |
+----------------+
| [NULL, 1]      |
+----------------+
```

```
select array(1, 2, 3);
```

```
+----------------+
| array(1, 2, 3) |
+----------------+
| [1, 2, 3]      |
+----------------+
```

```
select array(qid, creationDate, null) from nested  limit 4;
```

```
+-----------------------------------------------------+
| array(cast(qid as DATETIMEV2(0)), creationDate, NULL) |
+-----------------------------------------------------+
| [null, "2009-06-16 07:40:56", null]                  |
| [null, "2009-06-16 07:50:05", null]                  |
| [null, "2009-06-16 08:09:18", null]                  |
| [null, "2009-06-16 08:15:45", null]                  |
+-----------------------------------------------------+
```

ARRAY_APPLY

Description

Filter array to match specific binary condition

Syntax

```
ARRAY_APPLY(<arr>, <op>, <val>)
```

Parameters

Parameter	Description
<arr>	Input array
<op>	Filter condition, including =, >=, <=, >, <, !=
<val>	Filter value. If null, the result will be null. Only constant values are supported.

Return Value

The filtered array matched with condition.

Example

```
select array_apply([1, 2, 3, 4, 5], ">=", 2);
```

```
+-----------------------------------------+
| array_apply(ARRAY(1, 2, 3, 4, 5), '>=', 2) |
+-----------------------------------------+
| [2, 3, 4, 5]                            |
+-----------------------------------------+
```

```
select array_apply([1000000, 1000001, 1000002], "=", "1000002");
```

```
+-----------------------------------------------------------+
| array_apply(ARRAY(1000000, 1000001, 1000002), '=', 1000002) |
+-----------------------------------------------------------+
| [1000002]                                                 |
+-----------------------------------------------------------+
```

ARRAY_AVG

Description

Get the average of all elements in an array (NULL values are skipped). When the array is empty or all elements in the array are NULL values, the function returns NULL.

Syntax

```
ARRAY_AVG(<arr>)
```

Parameters

Parameter	Description
<arr>	The array to calculate the average values from

Return Value

Returns a constant. Special cases: - NULL values in the array will be skipped. - Strings and varchar in the array will be skipped.

Example

```
create table array_type_table(
    k1 INT,
    k2 Array<int>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
    'replication_num' = '1'
);
insert into array_type_table values (0, []), (1, [NULL]), (2, [1, 2, 3]), (3, [1, NULL, 3]);
select k2, array_avg(k2) from array_type_table;
```

```
+--------------+-----------------+
| k2           | array_avg(`k2`) |
+--------------+-----------------+
| []           |            NULL |
| [NULL]       |            NULL |
| [1, 2, 3]    |               2 |
| [1, NULL, 3] |               2 |
+--------------+-----------------+
```

```
select array_avg(['test',2,1,null]);
```

```
+-----------------------------------------------------------+
| array_avg(cast(['test', '2', '1', NULL] as ARRAY<DOUBLE>)) |
+-----------------------------------------------------------+
|                                                       1.5 |
+-----------------------------------------------------------+
```

ARRAY_COMPACT

Description

Removes consecutive duplicate elements from an array. The order of result values is determined by the order in the source array.

Syntax

```
ARRAY_COMPACT(<arr>)
```

Parameters

Parameter	Description
<arr>	The array to remove consecutive duplicate elements from

Return Value

An array without consecutive duplicate elements.

Example

```
select array_compact([1, 2, 3, 3, null, null, 4, 4]);
```

```
+--------------------------------------------------+
| array_compact(ARRAY(1, 2, 3, 3, NULL, NULL, 4, 4)) |
+--------------------------------------------------+
| [1, 2, 3, NULL, 4]                               |
+--------------------------------------------------+
```

```
select array_compact(['aaa','aaa','bbb','ccc','ccccc',null, null,'dddd']);
```

```
+-------------------------------------------------------------------------+
| array_compact(ARRAY('aaa', 'aaa', 'bbb', 'ccc', 'ccccc', NULL, NULL, 'dddd')) |
+-------------------------------------------------------------------------+
| ['aaa', 'bbb', 'ccc', 'ccccc', NULL, 'dddd']                            |
+-------------------------------------------------------------------------+
```

```
select array_compact(['2015-03-13','2015-03-13']);
```

```
+-----------------------------------------------+
| array_compact(ARRAY('2015-03-13', '2015-03-13')) |
+-----------------------------------------------+
```

```
| ['2015-03-13']                                  |
+-------------------------------------------------+
```

ARRAY_CONCAT

Description

Concat all arrays passed in the arguments

Syntax

```
ARRAY_CONCAT(<arr1> [,<arr2> , ...])
```

Parameters

Parameter	Description
<arr1>	The source array
<arr2>	The array to be appended to arr1

Return Value

The concatenated array. Special cases: - If an array is NULL (not `[NULL]`), the function returns NULL.

Example

```
select array_concat([1, 2], [7, 8], [5, 6]);
```

```
+-----------------------------------------------------+
| array_concat(ARRAY(1, 2), ARRAY(7, 8), ARRAY(5, 6)) |
+-----------------------------------------------------+
| [1, 2, 7, 8, 5, 6]                                  |
+-----------------------------------------------------+
```

```
select array_concat([1, 2], [7, 8], [5, 6], NULL);
```

```
+-------------------------------------------+
| array_concat([1, 2], [7, 8], [5, 6], NULL) |
+-------------------------------------------+
| NULL                                      |
+-------------------------------------------+
```

```
CREATE TABLE array_test (
    id int,
    col2 ARRAY<INT>,
    col3 ARRAY<INT>
)
duplicate key (id)
distributed by hash(id) buckets 1
```

```
properties(
  'replication_num' = '1'
);
INSERT INTO array_test (id, col2, col3) VALUES
(1,[1, 2, 3], [3, 4, 5]),
(2,[1, NULL, 2], [NULL]),
(3,[1, 2, 3], NULL),
(4,[], []);

select col2, col3, array_concat(col2, col3) from array_test;
```

```
+--------------+-----------+-----------------------------+
| col2         | col3      | array_concat(`col2`, `col3`) |
+--------------+-----------+-----------------------------+
| [1, 2, 3]    | [3, 4, 5] | [1, 2, 3, 3, 4, 5]          |
| [1, NULL, 2] | [NULL]    | [1, NULL, 2, NULL]          |
| [1, 2, 3]    | NULL      | NULL                        |
| []           | []        | []                          |
+--------------+-----------+-----------------------------+
```

ARRAY_CONTAINS

array_contains

array_contains

description

Syntax

```
BOOLEAN array_contains(ARRAY<T> arr, T value)
```

Check if a value presents in an array column. Return below values:

```
1    - if value presents in an array;
0    - if value does not present in an array;
NULL - when array is NULL;
```

example

```
mysql> SELECT id,c_array,array_contains(c_array, 5) FROM `array_test`;
+------+-----------------+-----------------------------+
| id   | c_array         | array_contains(`c_array`, 5) |
+------+-----------------+-----------------------------+
|    1 | [1, 2, 3, 4, 5] |                           1 |
|    2 | [6, 7, 8]       |                           0 |
|    3 | []              |                           0 |
|    4 | NULL            |                        NULL |
+------+-----------------+-----------------------------+

mysql> select array_contains([null, 1], null);
+--------------------------------------+
| array_contains(ARRAY(NULL, 1), NULL) |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
1 row in set (0.00 sec)
```

keywords

ARRAY,CONTAIN,CONTAINS,ARRAY_CONTAINS

ARRAY_COUNT

Description

Use lambda expressions as input parameters to perform corresponding expression calculations on the internal data of other input ARRAY parameters. Returns the number of elements such that the return value of `lambda(array1[i], ...)` is not 0. Returns 0 if no element is found that satisfies this condition.

There are one or more parameters are input in the lambda expression, which must be consistent with the number of input array columns later.The number of elements of all input arrays must be the same. Legal scalar functions can be executed in lambda, aggregate functions, etc. are not supported.

Syntax

```
ARRAY_COUNT(<arr>),
ARRAY_COUNT(<lambda>, <arr>[, ... ])
```

Parameters

Parameter	Description
<lambda>	A lambda expression where the input parameters must match the number of columns in the given array. The expression can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

After applying the lambda expression, returns the number of non-zero elements in the ARRAY. If no such elements are found, returns 0.

Example

```
-- setup
create table array_test(id int, c_array1 array<int>, c_array2 array<int>) distributed by hash(id) buckets 1 properties
("replication_num"="1");
insert into array_test values (1,[1,2,3,4,5],[10,20,-40,80,-100]),(2,[6,7,8],[10,12,13]),(3,[1],[-100]),(4,[1,null,2],[null,3,1]),
(5,[],[]),(6,null,null);
```

```
select array_count(x -> x, [0, 1, 2, 3]);
```

```
+-----------------------------------------------------+
| array_count(array_map([x] -> x(0), ARRAY(0, 1, 2, 3))) |
+-----------------------------------------------------+
```

```
|                                                     3 |
+-------------------------------------------------------+
```

```
select array_count(x -> x > 2, [0, 1, 2, 3]);
```

```
+-----------------------------------------------------------+
| array_count(array_map([x] -> x(0) > 2, ARRAY(0, 1, 2, 3))) |
+-----------------------------------------------------------+
|                                                         1 |
+-----------------------------------------------------------+
```

```
select array_count(x -> x is null, [null, null, null, 1, 2]);
```

```
+-------------------------------------------------------------------------+
| array_count(array_map([x] -> x(0) IS NULL, ARRAY(NULL, NULL, NULL, 1, 2))) |
+-------------------------------------------------------------------------+
|                                                                       3 |
+-------------------------------------------------------------------------+
```

```
select array_count(x -> power(x,2)>10, [1, 2, 3, 4, 5]);
```

```
+-----------------------------------------------------------------------------+
| array_count(array_map([x] -> power(x(0), 2.0) > 10.0, ARRAY(1, 2, 3, 4, 5))) |
+-----------------------------------------------------------------------------+
|                                                                           2 |
+-----------------------------------------------------------------------------+
```

```
select *, array_count((x, y) -> x>y, c_array1, c_array2) from array_test;
```

```
+------+-----------------+-------------------------+-----------------------------------------------------------------------------+
| id   | c_array1        | c_array2                | array_count(array_map([x, y] -> x(0) > y(1), `c_array1`, `c_array2`)) |
+------+-----------------+-------------------------+-----------------------------------------------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] |                                                                           2 |
|    2 | [6, 7, 8]       | [10, 12, 13]            |                                                                           0 |
|    3 | [1]             | [-100]                  |                                                                           1 |
|    4 | [1, NULL, 2]    | [NULL, 3, 1]            |                                                                           1 |
|    5 | []              | []                      |                                                                           0 |
|    6 | NULL            | NULL                    |                                                                           0 |
+------+-----------------+-------------------------+-----------------------------------------------------------------------------+
```

ARRAY_CUM_SUM

Description

Get the cumulative sum of an array (NULL values are skipped). If the array contains NULL values, then NULL is set at the same position in the result array.

Syntax

```
ARRAY_CUM_SUM(<arr>)
```

Parameters

Parameter	Description
<arr>	The array to calculate the cumulative sum from

Return Value

Returns an array. Special cases: - NULL values in the array are skipped, and NULL is set at the same position in the result array.

Example

```
create table array_type_table(
    k1 INT,
    k2 Array<int>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
    'replication_num' = '1'
);
insert into array_type_table values (0, []),
(1, [NULL]),
(2, [1, 2, 3, 4]),
(3, [1, NULL, 3, NULL, 5]);
select k2, array_cum_sum(k2) from array_type_table;
```

```
+----------------------+----------------------+
| k2                   | array_cum_sum(`k2`)  |
+----------------------+----------------------+
| []                   | []                   |
| [NULL]               | [NULL]               |
| [1, 2, 3, 4]         | [1, 3, 6, 10]        |
| [1, NULL, 3, NULL, 5] | [1, NULL, 4, NULL, 9] |
+----------------------+----------------------+
```

ARRAY_DIFFERENCE

Description

Calculates the difference between adjacent array elements. Returns an array where the first element will be 0, the second is the difference between a[1] - a[0]. need notice that NULL will be return NULL

Syntax

```
ARRAY_DIFFERENCE(<arr>)
```

Parameters

Parameter	Description
<arr>	The array to calculate the difference from

Return Value

Returns an array. Special cases: - If a NULL value exists in the input, the result will be NULL.

Example

```
CREATE TABLE array_type_table (
   k1 INT,
   k2 ARRAY<INT>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table (k1, k2) VALUES
(0, []),
(1, [NULL]),
(2, [1, 2, 3]),
(3, [1, NULL, 3]),
(4, [0, 1, 2, 3, NULL, 4, 6]),
(5, [1, 2, 3, 4, 5, 4, 3, 2, 1]),
(6, [6, 7, 8]);
select *,array_difference(k2) from array_type_table;
```

```
+------+----------------------------+--------------------------------+
| k1   | k2                         | array_difference(`k2`)         |
+------+----------------------------+--------------------------------+
|    0 | []                         | []                             |
|    1 | [NULL]                     | [NULL]                         |
|    2 | [1, 2, 3]                  | [0, 1, 1]                      |
|    3 | [1, NULL, 3]               | [0, NULL, NULL]                |
|    4 | [0, 1, 2, 3, NULL, 4, 6]   | [0, 1, 1, 1, NULL, NULL, 2]    |
```

```
|    5 | [1, 2, 3, 4, 5, 4, 3, 2, 1] | [0, 1, 1, 1, 1, -1, -1, -1, -1] |
|    6 | [6, 7, 8]                   | [0, 1, 1]                       |
+------+-----------------------------+---------------------------------+
```

ARRAY_DISTINCT

Description

Return the array which has been removed duplicate values. Return NULL for NULL input.

Syntax

```
ARRAY_DISTINCT(<arr> )
```

Parameters

Parameter	Description
<arr>	An array that might contain duplicate elements to be removed

Return Value

Return the array which has been removed duplicate values. Special case: - If the input array is NULL, the result is NULL.

Example

```
CREATE TABLE array_test (
    k1 INT,
    k2 ARRAY<INT>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_test VALUES
(1, [1, 2, 3, 4, 5]),
(2, [6, 7, 8]),
(3, []),
(4, NULL),
(5, [1, 2, 3, 4, 5, 4, 3, 2, 1]),
(6, [1, 2, 3, NULL]),
(7, [1, 2, 3, NULL, NULL]);
select k1, k2, array_distinct(k2) from array_test;
```

```
+------+---------------------------+--------------------------+
| k1   | k2                        | array_distinct(k2)       |
+------+---------------------------+--------------------------+
| 1    | [1, 2, 3, 4, 5]           | [1, 2, 3, 4, 5]          |
| 2    | [6, 7, 8]                 | [6, 7, 8]                |
| 3    | []                        | []                       |
| 4    | NULL                      | NULL                     |
| 5    | [1, 2, 3, 4, 5, 4, 3, 2, 1] | [1, 2, 3, 4, 5]          |
```

```
| 6    | [1, 2, 3, NULL]             | [1, 2, 3, NULL]           |
| 7    | [1, 2, 3, NULL, NULL]       | [1, 2, 3, NULL]           |
+------+-----------------------------+---------------------------+
```

```
CREATE TABLE array_test01 (
   k1 INT,
   k2 ARRAY<VARCHAR>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_test01 VALUES
(1, ['a', 'b', 'c', 'd', 'e']),
(2, ['f', 'g', 'h']),
(3, ['']),
(3, [NULL]),
(5, ['a', 'b', 'c', 'd', 'e', 'a', 'b', 'c']),
(6, NULL),
(7, ['a', 'b', NULL]),
(8, ['a', 'b', NULL, NULL]);
select k1, k2, array_distinct(k2) from array_test01;
```

```
+------+----------------------------------------+--------------------------+
| k1   | k2                                     | array_distinct(`k2`)     |
+------+----------------------------------------+--------------------------+
| 1    | ['a', 'b', 'c', 'd', 'e']              | ['a', 'b', 'c', 'd', 'e'] |
| 2    | ['f', 'g', 'h']                        | ['f', 'g', 'h']          |
| 3    | ['']                                   | ['']                     |
| 3    | [NULL]                                 | [NULL]                   |
| 5    | ['a', 'b', 'c', 'd', 'e', 'a', 'b', 'c'] | ['a', 'b', 'c', 'd', 'e'] |
| 6    | NULL                                   | NULL                     |
| 7    | ['a', 'b', NULL]                       | ['a', 'b', NULL]         |
| 8    | ['a', 'b', NULL, NULL]                 | ['a', 'b', NULL]         |
+------+----------------------------------------+--------------------------+
```

ARRAY_ENUMERATE

Description

Returns array sub item indexes e.g. [1, 2, 3, …, length (arr)]

Syntax

```
ARRAY_ENUMERATE(<arr>)
```

Parameters

Parameter	Description
<arr>	The array that returns array sub item indexes

Return Value

Returns an array containing the index of the array.

Example

```
create table array_type_table(
    k1 INT,
    k2 Array<STRING>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
    'replication_num' = '1'
);
insert into array_type_table values (0, []),
("1", [NULL]),
("2", ["1", "2", "3"]),
("3", ["1", NULL, "3"]),
("4", NULL);
select k2, array_enumerate(k2) from array_type_table;
```

```
+------------------+-----------------------+
| k2               | array_enumerate(`k2`) |
+------------------+-----------------------+
| []               | []                    |
| [NULL]           | [1]                   |
| ['1', '2', '3']  | [1, 2, 3]             |
| ['1', NULL, '3'] | [1, 2, 3]             |
| NULL             | NULL                  |
+------------------+-----------------------+
```

ARRAY_ENUMERATE_UNIQ

Description

Returns an array the same size as the source array, indicating for each element what its position is among elements with the same value. For example, array_enumerate_uniq([1, 2, 1, 4]) = [1, 1, 2, 1]. The array_enumerate_uniq function can take multiple arrays of the same size as arguments. In this case, uniqueness is considered for tuples of elements in the same positions in all the arrays. For example, array_enumerate_uniq([1, 2, 1, 1, 2], [2, 1, 2, 2, 1]) = [1, 1, 2, 3, 2].

Syntax

```
ARRAY_ENUMERATE_UNIQ(<arr1> [,<arr2> , ... ])
```

Parameters

Parameter	Description
<arr1>	Needed be computed array arr1
<arr2>	Needed be computed array arr2

Return Value

Returns an array the same size as the source array, indicating for each element what its position is among elements with the same value.

Example

```
select array_enumerate_uniq([1, 2, 3, 1, 2, 3]);
```

```
+-------------------------------------------------+
| array_enumerate_uniq(ARRAY(1, 2, 3, 1, 2, 3))   |
+-------------------------------------------------+
| [1, 1, 1, 2, 2, 2]                              |
+-------------------------------------------------+
```

```
select array_enumerate_uniq([1, 1, 1, 1, 1], [2, 1, 2, 1, 2], [3, 1, 3, 1, 3]);
```

```
+-------------------------------------------------------------------------------+
| array_enumerate_uniq(ARRAY(1, 1, 1, 1, 1), ARRAY(2, 1, 2, 1, 2), ARRAY(3, 1, 3, 1, 3)) |
+-------------------------------------------------------------------------------+
| [1, 1, 2, 2, 3]                                                               |
+-------------------------------------------------------------------------------+
```

ARRAY_EXCEPT

Description

Returns an array of the elements in array1 but not in array2, without duplicates. If the input parameter is null, null is returned.

Syntax

```
ARRAY_EXCEPT(<arr1> , <arr2> )
```

Parameters

Parameter	Description
<arr1>	The source array arr1
<arr2>	An array of elements to compare with arr1

Return Value

Returns an array, Special cases: - If the input parameter is null, null is returned.

Example

```
CREATE TABLE array_type_table (
    k1 INT,
    k2 ARRAY<INT>,
    k3 ARRAY<INT>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table VALUES
(1, [1, 2, 3], [2, 4, 5]),
(2, [2, 3], [1, 5]),
(3, [1, 1, 1], [2, 2, 2]);
select k1,k2,k3,array_except(k2,k3) from array_type_table;
```

```
+------+-----------------+--------------+--------------------------+
| k1   | k2              | k3           | array_except(`k2`, `k3`) |
+------+-----------------+--------------+--------------------------+
|    1 | [1, 2, 3]       | [2, 4, 5]    | [1, 3]                   |
|    2 | [2, 3]          | [1, 5]       | [2, 3]                   |
|    3 | [1, 1, 1]       | [2, 2, 2]    | [1]                      |
+------+-----------------+--------------+--------------------------+
```

```
CREATE TABLE array_type_table_nullable (
    k1 INT,
    k2 ARRAY<INT>,
    k3 ARRAY<INT>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table_nullable VALUES
(1, [1, NULL, 3], [1, 3, 5]),
(2, [NULL, NULL, 2], [2, NULL, 4]),
(3, NULL, [1, 2, 3]);
select k1,k2,k3,array_except(k2,k3) from array_type_table_nullable;
```

```
+------+-----------------+--------------+--------------------------+
| k1   | k2              | k3           | array_except(`k2`, `k3`) |
+------+-----------------+--------------+--------------------------+
|    1 | [1, NULL, 3]    | [1, 3, 5]    | [NULL]                   |
|    2 | [NULL, NULL, 2] | [2, NULL, 4] | []                       |
|    3 | NULL            | [1, 2, 3]    | NULL                     |
+------+-----------------+--------------+--------------------------+
```

```
CREATE TABLE array_type_table_varchar (
    k1 INT,
    k2 ARRAY<VARCHAR>,
    k3 ARRAY<VARCHAR>
)
    duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table_varchar VALUES
(1, ['hello', 'world', 'c++'], ['I', 'am', 'c++']),
(2, ['a1', 'equals', 'b1'], ['a2', 'equals', 'b2']),
(3, ['hasnull', NULL, 'value'], ['nohasnull', 'nonull', 'value']),
(3, ['hasnull', NULL, 'value'], ['hasnull', NULL, 'value']);
select k1,k2,k3,array_except(k2,k3) from array_type_table_varchar;
```

```
+------+---------------------------+---------------------------------+----------------------+
| k1   | k2                        | k3                              | array_except(k2, k3) |
+------+---------------------------+---------------------------------+----------------------+
|    1 | ["hello", "world", "c++"]  | ["I", "am", "c++"]              | ["hello", "world"]   |
|    2 | ["a1", "equals", "b1"]     | ["a2", "equals", "b2"]          | ["a1", "b1"]         |
|    3 | ["hasnull", null, "value"] | ["hasnull", null, "value"]      | []                   |
|    3 | ["hasnull", null, "value"] | ["nohasnull", "nonull", "value"] | ["hasnull", null]    |
+------+---------------------------+---------------------------------+----------------------+
```

```
CREATE TABLE array_type_table_decimal (
    k1 INT,
    k2 ARRAY<DECIMAL(10, 2)>,
    k3 ARRAY<DECIMAL(10, 2)>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
```

```
);
INSERT INTO array_type_table_decimal VALUES
(1, [1.1, 2.1, 3.44], [2.1, 3.4, 5.4]),
(2, [NULL, 2, 5], [NULL, NULL, 5.4]),
(1, [1, NULL, 2, 5], [1, 3.1, 5.4]);
select k1,k2,k3,array_except(k2,k3) from array_type_table_decimal;
```

```
+------+--------------------------+-------------------+----------------------+
| k1   | k2                       | k3                | array_except(k2, k3) |
+------+--------------------------+-------------------+----------------------+
|    1 | [1.00, null, 2.00, 5.00] | [1.00, 3.10, 5.40] | [null, 2.00, 5.00]   |
|    1 | [1.10, 2.10, 3.44]       | [2.10, 3.40, 5.40] | [1.10, 3.44]         |
|    2 | [null, 2.00, 5.00]       | [null, null, 5.40] | [2.00, 5.00]         |
+------+--------------------------+-------------------+----------------------+
```

ARRAY_EXISTS

Description

Use an optional lambda expression as an input parameter to perform corresponding expression calculations on the internal data of other input ARRAY parameters. Returns 1 when the calculation returns something other than 0; otherwise returns 0. There are one or more parameters input in the lambda expression, which must be consistent with the number of input array columns later. Legal scalar functions can be executed in lambda, aggregate functions, etc. are not supported.

When lambda expression is not used as a parameter, array1 is used as the calculation result.

Syntax

```
ARRAY_EXISTS([ <lambda>, ] <arr1> [, <arr2> , ...] )
```

Parameters

Parameter	Description
<lambda>	A lambda expression that can execute valid scalar functions but does not support aggregate functions.
<arr1>	Needed be computed array arr1
<arr2>	Needed be computed array arr2

Return Value

Returns the array computed using the expression. Special cases: - If the array contains `NULL` values or is itself `NULL`, the result is `NULL`.

Example

```
CREATE TABLE array_test2 (
    id INT,
    c_array1 ARRAY<INT>,
    c_array2 ARRAY<INT>
)
duplicate key (id)
distributed by hash(id) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_test2 VALUES
(1, [1, 2, 3, 4, 5], [10, 20, -40, 80, -100]),
(2, [6, 7, 8], [10, 12, 13]),
(3, [1], [-100]),
```

```
(4, NULL, NULL);
select *, array_exists(x->x>1,[1,2,3]) from array_test2 order by id;
```

```
+------+-----------------+-------------------------+-----------------------------------------------+
| id   | c_array1        | c_array2                | array_exists([x] -> x(0) > 1, ARRAY(1, 2, 3)) |
+------+-----------------+-------------------------+-----------------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [0, 1, 1]                                     |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [0, 1, 1]                                     |
|    3 | [1]             | [-100]                  | [0, 1, 1]                                     |
|    4 | NULL            | NULL                    | [0, 1, 1]                                     |
+------+-----------------+-------------------------+-----------------------------------------------+
```

```
select c_array1, c_array2, array_exists(x->x%2=0,[1,2,3]) from array_test2 order by id;
```

```
+-----------------+-------------------------+-------------------------------------------------+
| c_array1        | c_array2                | array_exists([x] -> x(0) % 2 = 0, ARRAY(1, 2, 3)) |
+-----------------+-------------------------+-------------------------------------------------+
| [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [0, 1, 0]                                       |
| [6, 7, 8]       | [10, 12, 13]            | [0, 1, 0]                                       |
| [1]             | [-100]                  | [0, 1, 0]                                       |
| NULL            | NULL                    | [0, 1, 0]                                       |
+-----------------+-------------------------+-------------------------------------------------+
```

```
select c_array1, c_array2, array_exists(x->abs(x)-1,[1,2,3]) from array_test2 order by id;
```

```
+-----------------+-------------------------+-----------------------------------------------------------------------------------+
| c_array1        | c_array2                | array_exists(cast(array_map(x -> (abs(x) - 1), [1, 2, 3]) as ARRAY<BOOLEAN>)) |
+-----------------+-------------------------+-----------------------------------------------------------------------------------+
| [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [0, 1, 1]                                                                         |
| [6, 7, 8]       | [10, 12, 13]            | [0, 1, 1]                                                                         |
| [1]             | [-100]                  | [0, 1, 1]                                                                         |
| NULL            | NULL                    | [0, 1, 1]                                                                         |
+-----------------+-------------------------+-----------------------------------------------------------------------------------+
```

```
select c_array1, c_array2, array_exists((x,y)->x>y,c_array1,c_array2) from array_test2 order by id;
```

```
+-----------------+-------------------------+-----------------------------------------------------------------+
| c_array1        | c_array2                | array_exists([x, y] -> x(0) > y(1), `c_array1`, `c_array2`) |
+-----------------+-------------------------+-----------------------------------------------------------------+
| [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [0, 0, 1, 0, 1]                                                 |
| [6, 7, 8]       | [10, 12, 13]            | [0, 0, 0]                                                       |
| [1]             | [-100]                  | [1]                                                             |
| NULL            | NULL                    | NULL                                                            |
+-----------------+-------------------------+-----------------------------------------------------------------+
```

```
select *, array_exists(c_array1) from array_test2 order by id;
```

```
+------+-----------------+-------------------------+--------------------------+
| id   | c_array1        | c_array2                | array_exists(`c_array1`) |
+------+-----------------+-------------------------+--------------------------+
|    1 | [1, 2, 3, 0, 5] | [10, 20, -40, 80, -100] | [1, 1, 1, 0, 1]          |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [1, 1, 1]                |
|    3 | [1]             | [-100]                  | [1]                      |
```

```
|    4 | NULL            | NULL                    | NULL                    |
+------+-----------------+-------------------------+-------------------------+
```

ARRAY_FILTER

Description

Use the lambda expression as the input parameter to calculate and filter the data of the ARRAY column of the other input parameter. And filter out the values of 0 and NULL in the result.

Syntax

```
ARRAY_FILTER(<lambda>, <arr>)
ARRAY_FILTER(<arr>, <filter_column>)
```

Parameters

Parameter	Description
<lambda>	A lambda expression where the input parameters must match the number of columns in the given array. The expression can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

Performs the specified expression calculation on the internal data of the input ARRAY parameter, filtering out 0 and NULL values from the result.

Example

```
-- setup
create table array_test(id int, c_array array<int>) distributed by hash(id) buckets 1 properties ("replication_num"="1");
insert into array_test values (1,[1,2,3,4,5]),(2,[6,7,8]),(3,[]),(4,null);
create table array_test2(id int, c_array1 array<int>, c_array2 array<int>) distributed by hash(id) buckets 1 properties
("replication_num"="1");
insert into array_test2 values (1,[1,2,3,4,5],[10,20,-40,80,-100]),(2,[6,7,8],[10,12,13]),(3,[1],[-100]),(4,null,null);
```

```
select c_array,array_filter(c_array,[0,1,0]) from array_test;
```

```
+-----------------+-----------------------------------------------------+
| c_array         | array_filter(`c_array`, ARRAY(FALSE, TRUE, FALSE)) |
+-----------------+-----------------------------------------------------+
| [1, 2, 3, 4, 5] | [2]                                                 |
| [6, 7, 8]       | [7]                                                 |
| []              | []                                                  |
| NULL            | NULL                                                |
+-----------------+-----------------------------------------------------+
```

```
select array_filter(x->(x > 1),[1,2,3,0,null]);
```

```
+-----------------------------------------------------------------------------------------+
| array_filter(ARRAY(1, 2, 3, 0, NULL), array_map([x] -> (x(0) > 1), ARRAY(1, 2, 3, 0, NULL))) |
+-----------------------------------------------------------------------------------------+
| [2, 3]                                                                                  |
+-----------------------------------------------------------------------------------------+
```

```
select *, array_filter(x->x>0,c_array2) from array_test2;
```

```
+------+-----------------+------------------------+-----------------------------------------------------------------------+
| id   | c_array1        | c_array2               | array_filter(`c_array2`, array_map([x] -> x(0) > 0, `c_array2`)) |
+------+-----------------+------------------------+-----------------------------------------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [10, 20, 80]                                                          |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [10, 12, 13]                                                          |
|    3 | [1]             | [-100]                  | []                                                                    |
|    4 | NULL            | NULL                    | NULL                                                                  |
+------+-----------------+------------------------+-----------------------------------------------------------------------+
```

```
select *, array_filter(x->x%2=0,c_array2) from array_test2;
```

```
+------+-----------------+------------------------+-----------------------------------------------------------------------+
| id   | c_array1        | c_array2               | array_filter(`c_array2`, array_map([x] -> x(0) % 2 = 0, `c_array2`)) |
+------+-----------------+------------------------+-----------------------------------------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [10, 20, -40, 80, -100]                                               |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [10, 12]                                                              |
|    3 | [1]             | [-100]                  | [-100]                                                                |
|    4 | NULL            | NULL                    | NULL                                                                  |
+------+-----------------+------------------------+-----------------------------------------------------------------------+
```

```
select *, array_filter(x->(x*(-10)>0),c_array2) from array_test2;
```

```
+------+-----------------+------------------------+-----------------------------------------------------------------------------+
| id   | c_array1        | c_array2               | array_filter(`c_array2`, array_map([x] -> (x(0) * (-10) > 0), `c_array2`)) |
+------+-----------------+------------------------+-----------------------------------------------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [-40, -100]                                                                 |
|    2 | [6, 7, 8]       | [10, 12, 13]            | []                                                                          |
|    3 | [1]             | [-100]                  | [-100]                                                                      |
|    4 | NULL            | NULL                    | NULL                                                                        |
+------+-----------------+------------------------+-----------------------------------------------------------------------------+
```

```
select *, array_filter(x->x>0, array_map((x,y)->(x>y), c_array1,c_array2)) as res from array_test2;
```

```
+------+-----------------+------------------------+--------+
| id   | c_array1        | c_array2               | res    |
+------+-----------------+------------------------+--------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [1, 1] |
|    2 | [6, 7, 8]       | [10, 12, 13]            | []     |
|    3 | [1]             | [-100]                  | [1]    |
|    4 | NULL            | NULL                    | NULL   |
+------+-----------------+------------------------+--------+
```

ARRAY_FIRST

Description

Use a lambda bool expression and an array as the input parameters, the lambda expression is used to evaluate the internal data of other input ARRAY parameters. Returns the first element in the array for which lambda(arr1[i]) returns something other than 0.

Syntax

```
ARRAY_FIRST(<lambda>, <arr>)
```

Parameters

Parameter	Description
<lambda>	A lambda expression where the input parameters must match the number of columns in the given array. The expression can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

Returns the index of the first non-zero value. If no such index is found, returns NULL.

Example

```
select array_first(x->x>2, [1,2,3,0]);
```

```
+-----------------------------------------------------------------------------------------+
| array_first(array_filter(ARRAY(1, 2, 3, 0), array_map([x] -> x(0) > 2, ARRAY(1, 2, 3, 0))), -1)|
+-----------------------------------------------------------------------------------------+
|                                                                                       3 |
+-----------------------------------------------------------------------------------------+
```

```
select array_first(x->x>4, [1,2,3,0]);
```

```
+-----------------------------------------------------------------------------------------+
| array_first(array_filter(ARRAY(1, 2, 3, 0), array_map([x] -> x(0) > 4, ARRAY(1, 2, 3, 0))), -1)|
+-----------------------------------------------------------------------------------------+
|                                                                                    NULL |
+-----------------------------------------------------------------------------------------+
```

```
select array_first(x->x>1, [1,2,3,0]);
```

```
+-----------------------------------------------------------------------------------------+
| array_first(array_filter(ARRAY(1, 2, 3, 0), array_map([x] -> x > 1, ARRAY(1, 2, 3, 0))), 1) |
+-----------------------------------------------------------------------------------------+
|                                                                                       2 |
+-----------------------------------------------------------------------------------------+
```

ARRAY_FIRST_INDEX

Description

Use an lambda expression as an input parameter to perform corresponding expression calculations on the internal data of other input ARRAY parameters. Returns the first index such that the return value of `lambda(array1[i], ...)` is not 0. Return 0 if such index is not found.

There are one or more parameters input in the lambda expression, and the number of elements of all input arrays must be the same. Legal scalar functions can be executed in lambda, aggregate functions, etc. are not supported.

Syntax

```
ARRAY_FIRST_INDEX(<lambda>, <arr> [, ...])
```

Parameters

Parameter	Description
<lambda>	A lambda expression where the input parameters must match the number of columns in the given array. The expression can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

Returns the index of the first non-zero value. If no such index is found, returns 0.

Example

```
-- setup
create table array_test(id int, col2 array<int>, col3 array<int>) distributed by hash(id) buckets 1 properties
("replication_num"="1");
insert into array_test values (1,[1,2,3],[3,4,5]),(2,[1,null,2],[null,3,1]),(3,[1,2,3],[9,8,7]),(4,null,null);
```

```
select array_first_index(x->x+1>3, [2, 3, 4]);
```

```
+-----------------------------------------------------------------+
| array_first_index(array_map([x] -> x(0) + 1 > 3, ARRAY(2, 3, 4))) |
+-----------------------------------------------------------------+
|                                                               2 |
+-----------------------------------------------------------------+
```

```
select array_first_index(x -> x is null, [null, 1, 2]);
```

```
+-------------------------------------------------------------------+
| array_first_index(array_map([x] -> x(0) IS NULL, ARRAY(NULL, 1, 2))) |
+-------------------------------------------------------------------+
|                                                                 1 |
+-------------------------------------------------------------------+
```

```
select array_first_index(x->power(x,2)>10, [1, 2, 3, 4]);
```

```
+-----------------------------------------------------------------------------+
| array_first_index(array_map([x] -> power(x(0), 2.0) > 10.0, ARRAY(1, 2, 3, 4))) |
+-----------------------------------------------------------------------------+
|                                                                           4 |
+-----------------------------------------------------------------------------+
```

```
select col2, col3, array_first_index((x,y)->x>y, col2, col3) from array_test;
```

```
+--------------+--------------+-----------------------------------------------------------------------+
| col2         | col3         | array_first_index(array_map([x, y] -> x(0) > y(1), `col2`, `col3`)) |
+--------------+--------------+-----------------------------------------------------------------------+
| [1, 2, 3]    | [3, 4, 5]    |                                                                     0 |
| [1, NULL, 2] | [NULL, 3, 1] |                                                                     3 |
| [1, 2, 3]    | [9, 8, 7]    |                                                                     0 |
| NULL         | NULL         |                                                                     0 |
+--------------+--------------+-----------------------------------------------------------------------+
```

ARRAY_INTERSECT

Description

Returns an array of the elements in the intersection of array1 and array2, without duplicates. If the input parameter is null, null is returned.

Syntax

```
ARRAY_INTERSECT(<arr1> , <arr2> )
```

Parameters

Parameter	Description
<arr1>	The source array arr1
<arr2>	Array of intersections with arr1

Return Value

Returns an array containing all elements of the intersection of arr1 and arr2. Special cases: - If the input parameter is NULL, the function returns NULL.

Notes

This function is supported only in the vectorized engine.

Example

```
CREATE TABLE array_type_table (
    k1 INT,
    k2 ARRAY<INT>,
    k3 ARRAY<INT>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table VALUES
(1, [1, 2, 3], [2, 4, 5]),
(2, [2, 3], [1, 5]),
(3, [1, 1, 1], [2, 2, 2]);
select k1,k2,k3,array_intersect(k2,k3) from array_type_table;
```

```
+------+----------------+-------------+----------------------------+
| k1   | k2             | k3          | array_intersect(`k2`, `k3`) |
```

```
+------+-----------------+--------------+-----------------------------+
|    1 | [1, 2, 3]       | [2, 4, 5]    | [2]                         |
|    2 | [2, 3]          | [1, 5]       | []                          |
|    3 | [1, 1, 1]       | [2, 2, 2]    | []                          |
+------+-----------------+--------------+-----------------------------+
```

```
CREATE TABLE array_type_table_nullable (
    k1 INT,
    k2 ARRAY<INT>,
    k3 ARRAY<INT>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table_nullable VALUES
(1, [1, NULL, 3], [1, 3, 5]),
(2, [NULL, NULL, 2], [2, NULL, 4]),
(3, NULL, [1, 2, 3]);
select k1,k2,k3,array_intersect(k2,k3) from array_type_table_nullable;
```

```
+------+-----------------+--------------+-----------------------------+
| k1   | k2              | k3           | array_intersect(`k2`, `k3`) |
+------+-----------------+--------------+-----------------------------+
|    1 | [1, NULL, 3]    | [1, 3, 5]    | [1, 3]                      |
|    2 | [NULL, NULL, 2] | [2, NULL, 4] | [NULL, 2]                   |
|    3 | NULL            | [1, 2, 3]    | NULL                        |
+------+-----------------+--------------+-----------------------------+
```

```
CREATE TABLE array_type_table_varchar (
    k1 INT,
    k2 ARRAY<VARCHAR>,
    k3 ARRAY<VARCHAR>
)
    duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table_varchar VALUES
(1, ['hello', 'world', 'c++'], ['I', 'am', 'c++']),
(2, ['a1', 'equals', 'b1'], ['a2', 'equals', 'b2']),
(3, ['hasnull', NULL, 'value'], ['nohasnull', 'nonull', 'value']),
(3, ['hasnull', NULL, 'value'], ['hasnull', NULL, 'value']);
select k1,k2,k3,array_intersect(k2,k3) from array_type_table_varchar;
```

```
+------+---------------------------+---------------------------------+---------------------------+
| k1   | k2                        | k3                              | array_intersect(k2, k3)   |
+------+---------------------------+---------------------------------+---------------------------+
|    1 | ["hello", "world", "c++"]  | ["I", "am", "c++"]              | ["c++"]                   |
|    2 | ["a1", "equals", "b1"]     | ["a2", "equals", "b2"]          | ["equals"]                |
|    3 | ["hasnull", null, "value"] | ["hasnull", null, "value"]      | [null, "value", "hasnull"] |
|    3 | ["hasnull", null, "value"] | ["nohasnull", "nonull", "value"] | ["value"]                 |
+------+---------------------------+---------------------------------+---------------------------+```
```

```
CREATE TABLE array_type_table_decimal (
    k1 INT,
    k2 ARRAY<DECIMAL(10, 2)>,
```

```
    k3 ARRAY<DECIMAL(10, 2)>
)
    duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_type_table_decimal VALUES
(1, [1.1, 2.1, 3.44], [2.1, 3.4, 5.4]),
(2, [NULL, 2, 5], [NULL, NULL, 5.4]),
(3, [1, NULL, 2, 5], [1, 3.1, 5.4]);
select k1,k2,k3,array_intersect(k2,k3) from array_type_table_decimal;
```

```
+------+--------------------------+-------------------+------------------------+
| k1   | k2                       | k3                | array_intersect(k2, k3) |
+------+--------------------------+-------------------+------------------------+
|    1 | [1.10, 2.10, 3.44]       | [2.10, 3.40, 5.40] | [2.10]                 |
|    2 | [null, 2.00, 5.00]       | [null, null, 5.40] | [null]                 |
|    3 | [1.00, null, 2.00, 5.00] | [1.00, 3.10, 5.40] | [1.00]                 |
+------+--------------------------+-------------------+------------------------+
```

ARRAY_JOIN

Description

Combine all elements in the array into a new string based on the separator (sep) and the string to replace NULL values (null_replace).

Syntax

```
ARRAY_JOIN(<arr> , <sep> [, <null_replace>])
```

Parameters

Parameter	Description
<arr>	An array to join
<sep>	Separator string
<null_replace>	String to replace NULL values

Return Value

Returns a new string with the following special cases: - If <sep> is NULL, the function returns NULL. - If <null_replace> is NULL, the function also returns NULL. - If <sep> is an empty string, no separator is applied. - If <null_replace> is an empty string or not specified, NULL elements in the array are discarded directly.

Example

```
CREATE TABLE array_test (
                            k1 INT,
                            k2 ARRAY<INT>
)
    duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);

INSERT INTO array_test VALUES
                           (1, [1, 2, 3, 4, 5]),
                           (2, [6, 7, 8]),
                           (3, []),
                           (4, NULL),
                           (5, [1, 2, 3, 4, 5, 4, 3, 2, 1]),
                           (6, [1, 2, 3, NULL]),
                           (7, [4, 5, 6, NULL, NULL]);
select k1, k2, array_join(k2, '_', 'null') from array_test order by k1;
```

```
+------+-----------------------------+-----------------------------------+
| k1   | k2                          | array_join(`k2`, '_', 'null')     |
+------+-----------------------------+-----------------------------------+
|  1   | [1, 2, 3, 4, 5]             | 1_2_3_4_5                         |
|  2   | [6, 7, 8]                   | 6_7_8                             |
|  3   | []                          |                                   |
|  4   | NULL                        | NULL                              |
|  5   | [1, 2, 3, 4, 5, 4, 3, 2, 1] | 1_2_3_4_5_4_3_2_1                 |
|  6   | [1, 2, 3, NULL]             | 1_2_3_null                        |
|  7   | [4, 5, 6, NULL, NULL]       | 4_5_6_null_null                   |
+------+-----------------------------+-----------------------------------+
```

```
select k1, k2, array_join(k2, '_') from array_test order by k1;
```

```
+------+-----------------------------+---------------------------+
| k1   | k2                          | array_join(`k2`, '_')     |
+------+-----------------------------+---------------------------+
|  1   | [1, 2, 3, 4, 5]             | 1_2_3_4_5                 |
|  2   | [6, 7, 8]                   | 6_7_8                     |
|  3   | []                          |                           |
|  4   | NULL                        | NULL                      |
|  5   | [1, 2, 3, 4, 5, 4, 3, 2, 1] | 1_2_3_4_5_4_3_2_1         |
|  6   | [1, 2, 3, NULL]             | 1_2_3                     |
|  7   | [4, 5, 6, NULL, NULL]       | 4_5_6                     |
+------+-----------------------------+---------------------------+
```

```
CREATE TABLE array_test01 (
    k1 INT,
    k2 ARRAY<STRING>
)
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
  'replication_num' = '1'
);

INSERT INTO array_test01 VALUES
(1, ['a', 'b', 'c', 'd']),
(2, ['e', 'f', 'g', 'h']),
(3, [NULL, 'a', NULL, 'b', NULL, 'c']),
(4, ['d', 'e', NULL, ' ']),
(5, [' ', NULL, 'f', 'g']);
select k1, k2, array_join(k2, '_', 'null') from array_test01 order by k1;
```

```
+------+-----------------------------------+-----------------------------------+
| k1   | k2                                | array_join(`k2`, '_', 'null')     |
+------+-----------------------------------+-----------------------------------+
|  1   | ['a', 'b', 'c', 'd']              | a_b_c_d                           |
|  2   | ['e', 'f', 'g', 'h']              | e_f_g_h                           |
|  3   | [NULL, 'a', NULL, 'b', NULL, 'c'] | null_a_null_b_null_c              |
|  4   | ['d', 'e', NULL, ' ']             | d_e_null_                         |
|  5   | [' ', NULL, 'f', 'g']             |  _null_f_g                        |
+------+-----------------------------------+-----------------------------------+
```

```
select k1, k2, array_join(k2, '_') from array_test01 order by k1;
```

```
+------+----------------------------------+---------------------------+
| k1   | k2                               | array_join(`k2`, '_')     |
+------+----------------------------------+---------------------------+
|  1   | ['a', 'b', 'c', 'd']             | a_b_c_d                   |
|  2   | ['e', 'f', 'g', 'h']             | e_f_g_h                   |
|  3   | [NULL, 'a', NULL, 'b', NULL, 'c'] | a_b_c                     |
|  4   | ['d', 'e', NULL, ' ']            | d_e_                      |
|  5   | [' ', NULL, 'f', 'g']            |  _f_g                     |
+------+----------------------------------+---------------------------+
```

ARRAY_LAST

Description

Use a lambda bool expression and an array as the input parameters, the lambda expression is used to evaluate the internal data of other input ARRAY parameters. Returns the last element in the array for which lambda(arr1[i]) returns something other than 0.

Syntax

```
ARRAY_LAST(<lambda>, <arr>)
```

Parameters

Parameter	Description
<lambda>	A lambda expression where the input parameters must match the number of columns in the given array. The expression can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

Returns the index of the last non-zero value. If no such index is found, returns NULL.

Example

```
select array_last(x->x>2, [1,2,3,0]) ;
```

```
+-----------------------------------------------------------------------------------------+
| array_last(array_filter(ARRAY(1, 2, 3, 0), array_map([x] -> x(0) > 2, ARRAY(1, 2, 3, 0))), -1) |
+-----------------------------------------------------------------------------------------+
|                                                                                       3 |
+-----------------------------------------------------------------------------------------+
```

```
select array_last(x->x>4, [1,2,3,0]) ;
```

```
+-----------------------------------------------------------------------------------------+
| array_last(array_filter(ARRAY(1, 2, 3, 0), array_map([x] -> x(0) > 4, ARRAY(1, 2, 3, 0))), -1) |
+-----------------------------------------------------------------------------------------+
|                                                                                    NULL |
+-----------------------------------------------------------------------------------------+
```

ARRAY_LAST_INDEX

Description

Use an lambda expression as an input parameter to perform corresponding expression calculations on the internal data of other input ARRAY parameters. Returns the last index such that the return value of `lambda(array1[i], ...)` is not 0. Return 0 if such index is not found.

There are one or more parameters input in the lambda expression, and the number of elements of all input arrays must be the same. Legal scalar functions can be executed in lambda, aggregate functions, etc. are not supported.

Syntax

```
ARRAY_LAST_INDEX(<lambda>, <arr> [, ...])
```

Parameters

Parameter	Description
<lambda>	A lambda expression where the input parameters must match the number of columns in the given array. The expression can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

Returns the index of the last non-zero value. If no such index is found, returns 0.

Example

```
-- setup
create table array_test(id int, col2 array<int>, col3 array<int>) distributed by hash(id) buckets 1 properties
("replication_num"="1");
insert into array_test values (1,[1,2,3],[3,4,5]),(2,[1,null,2],[null,3,1]),(3,[1,2,3],[9,8,7]),(4,null,null);
```

```
select array_last_index(x -> x is null, [null, null, 1, 2]);
```

```
+-----------------------------------------------------------------------+
| array_last_index(array_map([x] -> x IS NULL, ARRAY(NULL, NULL, 1, 2))) |
+-----------------------------------------------------------------------+
|                                                                     2 |
+-----------------------------------------------------------------------+
```

```
select array_last_index(x->x='s', ['a', 's', 's', 's', 'b']);
```

```
+-----------------------------------------------------------------------------+
| array_last_index(array_map([x] -> x = 's', ARRAY('a', 's', 's', 's', 'b'))) |
+-----------------------------------------------------------------------------+
|                                                                           4 |
+-----------------------------------------------------------------------------+
```

```
select array_last_index(x->power(x,2)>10, [1, 4, 3, 4]);
```

```
+-----------------------------------------------------------------------------+
| array_last_index(array_map([x] -> power(x, 2.0) > 10.0, ARRAY(1, 4, 3, 4))) |
+-----------------------------------------------------------------------------+
|                                                                           4 |
+-----------------------------------------------------------------------------+
```

```
select col2, col3, array_last_index((x,y)->x>y, col2, col3) from array_test;
```

```
+--------------+--------------+-----------------------------------------------------------------------+
| col2         | col3         | array_last_index(array_map([x, y] -> x(0) > y(1), `col2`, `col3`)) |
+--------------+--------------+-----------------------------------------------------------------------+
| [1, 2, 3]    | [3, 4, 5]    |                                                                     0 |
| [1, NULL, 2] | [NULL, 3, 1] |                                                                     3 |
| [1, 2, 3]    | [9, 8, 7]    |                                                                     0 |
| NULL         | NULL         |                                                                     0 |
+--------------+--------------+-----------------------------------------------------------------------+
```

ARRAY_MAP

description

Use a lambda expression as the input parameter to calculate the corresponding expression for the internal data of other input ARRAY parameters. The number of parameters entered the lambda expression is 1 or more, which must be consistent with the number of input array columns. The scalar functions can be executed in lambda, and aggregate functions are not supported.

Syntax

```
ARRAY_MAP(lambda, <arr> [ , <arr> ... ] )
```

Parameters

Parameter	Description
lambda	A lambda expression with one or more input parameters, which must match the number of input arrays. The lambda can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

An ARRAY processed through the lambda expression.

example

```
CREATE TABLE array_test2 (
                            id INT,
                            c_array1 ARRAY<INT>,
                            c_array2 ARRAY<INT>
)
    duplicate key (id)
distributed by hash(id) buckets 1
properties(
    'replication_num' = '1'
    );
INSERT INTO array_test2 (id, c_array1, c_array2) VALUES
                                                    (1, [1, 2, 3, 4, 5], [10, 20, -40, 80, -100]),
                                                    (2, [6, 7, 8], [10, 12, 13]),
                                                    (3, [1], [-100]),
                                                    (4, NULL, NULL);
select *, array_map(x->x,[1,2,3]) from array_test2 order by id;
```

```
+------+----------------+-----------------------+--------------------------------------+
| id   | c_array1       | c_array2              | array_map([x] -> x(0), ARRAY(1, 2, 3)) |
+------+----------------+-----------------------+--------------------------------------+
```

```
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [1, 2, 3]                              |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [1, 2, 3]                              |
|    3 | [1]             | [-100]                  | [1, 2, 3]                              |
|    4 | NULL            | NULL                    | [1, 2, 3]                              |
+------+-----------------+-------------------------+----------------------------------------+
```

```
select *, array_map(x->x+2,[1,2,3]) from array_test2 order by id;
```

```
+------+-----------------+-------------------------+----------------------------------------------+
| id   | c_array1        | c_array2                | array_map([x] -> x(0) + 2, ARRAY(1, 2, 3)) |
+------+-----------------+-------------------------+----------------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [3, 4, 5]                                    |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [3, 4, 5]                                    |
|    3 | [1]             | [-100]                  | [3, 4, 5]                                    |
|    4 | NULL            | NULL                    | [3, 4, 5]                                    |
+------+-----------------+-------------------------+----------------------------------------------+
```

```
select c_array1, c_array2, array_map(x->x,[1,2,3]) from array_test2 order by id;
```

```
+-----------------+-------------------------+---------------------------------------+
| c_array1        | c_array2                | array_map([x] -> x(0), ARRAY(1, 2, 3)) |
+-----------------+-------------------------+---------------------------------------+
| [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [1, 2, 3]                             |
| [6, 7, 8]       | [10, 12, 13]            | [1, 2, 3]                             |
| [1]             | [-100]                  | [1, 2, 3]                             |
| NULL            | NULL                    | [1, 2, 3]                             |
+-----------------+-------------------------+---------------------------------------+
```

```
select c_array1, c_array2, array_map(x->power(x,2),[1,2,3]) from array_test2 order by id;
```

```
+-----------------+-------------------------+-------------------------------------------------+
| c_array1        | c_array2                | array_map([x] -> power(x(0), 2.0), ARRAY(1, 2, 3)) |
+-----------------+-------------------------+-------------------------------------------------+
| [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [1, 4, 9]                                       |
| [6, 7, 8]       | [10, 12, 13]            | [1, 4, 9]                                       |
| [1]             | [-100]                  | [1, 4, 9]                                       |
| NULL            | NULL                    | [1, 4, 9]                                       |
+-----------------+-------------------------+-------------------------------------------------+
```

```
select c_array1, c_array2, array_map((x,y)->x+y,c_array1,c_array2) from array_test2 order by id;
```

```
+-----------------+-------------------------+-----------------------------------------------------------+
| c_array1        | c_array2                | array_map([x, y] -> x(0) + y(1), `c_array1`, `c_array2`) |
+-----------------+-------------------------+-----------------------------------------------------------+
| [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [11, 22, -37, 84, -95]                                    |
| [6, 7, 8]       | [10, 12, 13]            | [16, 19, 21]                                              |
| [1]             | [-100]                  | [-99]                                                     |
| NULL            | NULL                    | NULL                                                      |
+-----------------+-------------------------+-----------------------------------------------------------+
```

```
select c_array1, c_array2, array_map((x,y)->power(x,2)+y,c_array1, c_array2) from array_test2 order by id;
```

```
+-----------------+------------------------+-------------------------------------------------------------------------+
| c_array1        | c_array2               | array_map([x, y] -> power(x(0), 2.0) + y(1), `c_array1`, `c_array2`) |
+-----------------+------------------------+-------------------------------------------------------------------------+
| [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [11, 24, -31, 96, -75]                                                  |
| [6, 7, 8]       | [10, 12, 13]            | [46, 61, 77]                                                            |
| [1]             | [-100]                  | [-99]                                                                   |
| NULL            | NULL                    | NULL                                                                    |
+-----------------+------------------------+-------------------------------------------------------------------------+
```

```
select *,array_map(x->x=3,c_array1) from array_test2 order by id;
```

```
+------+-----------------+------------------------+---------------------------------------+
| id   | c_array1        | c_array2               | array_map([x] -> x(0) = 3, `c_array1`) |
+------+-----------------+------------------------+---------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [0, 0, 1, 0, 0]                       |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [0, 0, 0]                             |
|    3 | [1]             | [-100]                  | [0]                                   |
|    4 | NULL            | NULL                    | NULL                                  |
+------+-----------------+------------------------+---------------------------------------+
```

```
select *,array_map(x->x>3,c_array1) from array_test2 order by id;
```

```
+------+-----------------+------------------------+---------------------------------------+
| id   | c_array1        | c_array2               | array_map([x] -> x(0) > 3, `c_array1`) |
+------+-----------------+------------------------+---------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [0, 0, 0, 1, 1]                       |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [1, 1, 1]                             |
|    3 | [1]             | [-100]                  | [0]                                   |
|    4 | NULL            | NULL                    | NULL                                  |
+------+-----------------+------------------------+---------------------------------------+
```

```
select *,array_map((x,y)->x>y,c_array1,c_array2) from array_test2 order by id;
```

```
+------+-----------------+------------------------+-----------------------------------------------------------+
| id   | c_array1        | c_array2               | array_map([x, y] -> x(0) > y(1), `c_array1`, `c_array2`) |
+------+-----------------+------------------------+-----------------------------------------------------------+
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [0, 0, 1, 0, 1]                                           |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [0, 0, 0]                                                 |
|    3 | [1]             | [-100]                  | [1]                                                       |
|    4 | NULL            | NULL                    | NULL                                                      |
+------+-----------------+------------------------+-----------------------------------------------------------+
```

```
select array_map(x->cast(x as string), c_array1) from test_array_map_function;
```

```
+-----------------+-----------------------------------------------------+
| c_array1        | array_map([x] -> CAST(x(0) AS CHARACTER), `c_array1`) |
+-----------------+-----------------------------------------------------+
| [1, 2, 3, 4, 5] | ['1', '2', '3', '4', '5']                           |
| [6, 7, 8]       | ['6', '7', '8']                                     |
| []              | []                                                  |
| NULL            | NULL                                                |
+-----------------+-----------------------------------------------------+
```

ARRAY_MAX

description

Get the maximum element in an array (NULL values are skipped). When the array is empty or all elements in the array are NULL values, the function returns NULL.

Syntax

```
ARRAY_MAX(<arr>)
```

Parameters

Parameter	Description
<arr>	ARRAY array

Return Value

Returns the largest element in the array. Special cases: - NULL values in the array are skipped. - For an empty array or an array where all elements are NULL, the result is NULL.

example

```
create table array_type_table(
    k1 INT,
    k2 Array<int>
        )
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
    'replication_num' = '1'
    );
insert into array_type_table values (0, []), (1, [NULL]), (2, [1, 2, 3]), (3, [1, NULL, 3]);
select k2, array_max(k2) from array_type_table;
```

```
+--------------+-----------------+
| k2           | array_max(`k2`) |
+--------------+-----------------+
| []           |            NULL |
| [NULL]       |            NULL |
| [1, 2, 3]    |               3 |
| [1, NULL, 3] |               3 |
+--------------+-----------------+
```

ARRAY_MIN

description

Get the minimum element in an array (NULL values are skipped). When the array is empty or all elements in the array are NULL values, the function returns NULL.

Syntax

```
ARRAY_MIN(<arr>)
```

Parameters

Parameter	Description
<arr>	ARRAY array

Return Value

Returns the smallest element in the array. Special cases: - NULL values in the array are skipped. - For an empty array or an array where all elements are NULL, the result is NULL.

example

```
create table array_type_table(
    k1 INT,
    k2 Array<int>
    )
duplicate key (k1)
distributed by hash(k1) buckets 1
properties(
    'replication_num' = '1'
    );
insert into array_type_table values (0, []), (1, [NULL]), (2, [1, 2, 3]), (3, [1, NULL, 3]);
select k2, array_min(k2) from array_type_table;
```

```
+--------------+-----------------+
| k2           | array_min(`k2`) |
+--------------+-----------------+
| []           |            NULL |
| [NULL]       |            NULL |
| [1, 2, 3]    |               1 |
| [1, NULL, 3] |               1 |
+--------------+-----------------+
```

ARRAY_POPBACK

description

Remove the last element from array.

Syntax

```
ARRAY_POPBACK(<arr>)
```

Parameters

Parameter	Description
<arr>	ARRAY array

Return Value

Returns the array after removing the last element. Special cases: - If the input parameter is NULL, returns NULL.

example

```
select array_popback(['test', NULL, 'value']);
```

```
+-----------------------------------------------------+
| array_popback(ARRAY('test', NULL, 'value'))         |
+-----------------------------------------------------+
| ["test", NULL]                                       |
+-----------------------------------------------------+
```

ARRAY_POPFRONT

description

Remove the first element from array.

Syntax

```
ARRAY_POPFRONT(<arr>)
```

Parameters

Parameter	Description
<arr>	ARRAY array

Return Value

Returns the array after removing the first element. Special cases: - If the input parameter is NULL, returns NULL.

example

```
select array_popfront(['test', NULL, 'value']);
```

```
+-----------------------------------------------------+
| array_popfront(ARRAY('test', NULL, 'value'))        |
+-----------------------------------------------------+
| [NULL, "value"]                                      |
+-----------------------------------------------------+
```

ARRAY_POSITION

description

Returns a position/index of first occurrence of the `value` in the given array.

Syntax

```
ARRAY_POSITION(<arr>, <value>)
```

Parameters

Parameter	Description
<arr>	ARRAY array
<value>	Element to search for

Returns Value

The position of the value in the array (starting from 1). Special cases: - 0, if the value does not exist in the array. - NULL, if the array is NULL.

example

```
CREATE TABLE array_test (
                            id INT,
                            c_array ARRAY<INT>,
                            array_position INT
)
    duplicate key (id)
distributed by hash(id) buckets 1
properties(
  'replication_num' = '1'
);
INSERT INTO array_test (id, c_array, array_position) VALUES
                                                        (1, [1, 2, 3, 4, 5], 5),
                                                        (2, [6, 7, 8], 0),
                                                        (3, [], 0),
                                                        (4, NULL, NULL);
SELECT id,c_array,array_position(c_array, 5) FROM `array_test`;
```

```
+------+-----------------+-----------------------------+
| id   | c_array         | array_position(`c_array`, 5) |
+------+-----------------+-----------------------------+
|    1 | [1, 2, 3, 4, 5] |                           5 |
|    2 | [6, 7, 8]       |                           0 |
|    3 | []              |                           0 |
```

```
|    4 | NULL            |                          NULL |
+------+-----------------+-------------------------------+
```

```
select array_position([1, null], null);
```

```
+-------------------------------------+
| array_position(ARRAY(1, NULL), NULL) |
+-------------------------------------+
|                                   2 |
+-------------------------------------+
```

ARRAY_PRODUCT

Description

Calculates the product of all elements in an array

Syntax

```
ARRAY_PRODUCT(<arr>)
```

Parameters

Parameter	Description
<arr>	Corresponding array

Return Value

Returns the product of all elements in the array. NULL values in the array will be skipped. For an empty array or an array with all NULL values, the result returns a NULL value.

Example

```
SELECT ARRAY_PRODUCT([1, 2, 3]),ARRAY_PRODUCT([1, NULL, 3]),ARRAY_PRODUCT([NULL]);
```

```
+-------------------------+---------------------------+-----------------------------------------+
| array_product([1, 2, 3]) | array_product([1, NULL, 3]) | array_product(cast([NULL] as ARRAY<DOUBLE>)) |
+-------------------------+---------------------------+-----------------------------------------+
|                       6 |                         3 |                                    NULL |
+-------------------------+---------------------------+-----------------------------------------+
```

ARRAY_PUSHBACK

Description

Add value to the end of the array

Syntax

```
ARRAY_PUSHBACK(<arr>, <value>)
```

Parameters

Parameter	Description
<arr>	Corresponding array
<value>	Value to be added

Return Value

Returns the array after adding value

Example

```
SELECT ARRAY_PUSHBACK([1, 2], 3),ARRAY_PUSHBACK([3, 4], 6);
```

```
+--------------------------+--------------------------+
| array_pushback([1, 2], 3) | array_pushback([3, 4], 6) |
+--------------------------+--------------------------+
| [1, 2, 3]                | [3, 4, 6]                |
+--------------------------+--------------------------+
```

ARRAY_PUSHFRONT

Description

Add value to the beginning of the array

Syntax

```
ARRAY_PUSHFRONT(<arr>, <value>)
```

Parameters

Parameter	Description
<arr>	Corresponding array
<value>	Value to be added

Return Value

Returns the array after adding value

Example

```
SELECT ARRAY_PUSHFRONT([1, 2], 3),ARRAY_PUSHFRONT([3, 4], 6);
```

```
+----------------------------+----------------------------+
| array_pushfront([1, 2], 3) | array_pushfront([3, 4], 6) |
+----------------------------+----------------------------+
| [3, 1, 2]                  | [6, 3, 4]                  |
+----------------------------+----------------------------+
```

ARRAY_RANGE

Description

1. Generate int array
2. Generate date and time array

Aliases

- SEQUENCE

Syntax

```
ARRAY_RANGE(<end>)
ARRAY_RANGE(<start>, <end>)
ARRAY_RANGE(<start>, <end>, <step>)
ARRAY_RANGE(<start_datetime>, <end_datetime>)
ARRAY_RANGE(<start_datetime>, <end_datetime>, INTERVAL <interval_step> <unit>)
```

Parameters

Parameter	Description
<start>	The starting value is a positive integer, the default value is 0
<end>	End value, a positive integer
<step>	Step size, a positive integer, default is 1
<start_datetime>	Start date, datetimev2 type
<end_datetime>	End date, datetimev2 type
<interval_step>	Interval value, default is 1
<unit>	Interval unit, supports year/month/week/day/hour/minute/second, default is day

Return Value

1. Returns an array from start to end - 1, with a step length of step. If the third parameter step is negative or zero, the function result will be NULL
2. Returns an array of datetimev2 between start_datetime and the closest end_datetime (calculated by Interval_step UNIT). If the third argument interval_step is negative or zero, the function result will be NULL

Example

```
SELECT ARRAY_RANGE(0,20,2),ARRAY_RANGE(cast('2019-05-15 12:00:00' as datetimev2(0)), cast('2022-05-17 12:00:00' as datetimev2(0)),
interval 2 year);
```

```
+-----------------------------------+-----------------------------------------------------------------------------------------------
------------------------+
| array_range(0, 20, 2)             | array_range_year_unit(cast('2019-05-15 12:00:00' as DATETIMEV2(0)), cast('2022-05-17
12:00:00' as DATETIMEV2(0)), 2) |
+-----------------------------------+-----------------------------------------------------------------------------------------------
------------------------+
| [0, 2, 4, 6, 8, 10, 12, 14, 16, 18] | ["2019-05-15 12:00:00", "2021-05-15 12:00:00"]
|
+-----------------------------------+-----------------------------------------------------------------------------------------------
------------------------+
```

ARRAY_REMOVE

Description

Removes all specified elements from an array

Syntax

```
ARRAY_REMOVE(<arr>, <val>)
```

Parameters

Parameter	Description
<arr>	Corresponding array
<val>	Specifying Elements

Return Value

Returns the array after removing all specified elements. If the input parameter is NULL, it returns NULL

Example

```
SELECT ARRAY_REMOVE(['test', NULL, 'value'], 'value');
```

```
+-------------------------------------------------+
| array_remove(['test', NULL, 'value'], 'value') |
+-------------------------------------------------+
| ["test", null]                                  |
+-------------------------------------------------+
```

ARRAY_REPEAT

Description

Generates an array containing n repeated elements

Syntax

```
ARRAY_REPEAT(<element>, <n>)
```

Parameters

Parameter	Description
<n>	Number of digits
<element>	Specifying Elements

Return Value

Returns an array containing n repeated elements. array_with_constant has the same function as array_repeat and is used to be compatible with the hive syntax format.

Example

```
SELECT ARRAY_REPEAT("hello", 2),ARRAY_REPEAT(12345, 3);
```

```
+--------------------------+-----------------------+
| array_repeat('hello', 2) | array_repeat(12345, 3) |
+--------------------------+-----------------------+
| ["hello", "hello"]       | [12345, 12345, 12345]  |
+--------------------------+-----------------------+
```

ARRAY_REVERSE_SORT

Description

Sort the elements in an array in descending order

Syntax

```
ARRAY_REVERSE_SORT(<arr>)
```

Parameters

Parameter	Description
<arr>	Corresponding array

Return Value

Returns an array sorted in descending order. If the input array is NULL, it returns NULL. If the array element contains NULL, the output sorted array will put NULL at the end.

Example

```
SELECT ARRAY_REVERSE_SORT([1, 2, 3, 6]),ARRAY_REVERSE_SORT([1, 4, 3, 5, NULL]),ARRAY_REVERSE_SORT([NULL]);
```

```
+---------------------------------+---------------------------------------+---------------------------+
| array_reverse_sort([1, 2, 3, 6]) | array_reverse_sort([1, 4, 3, 5, NULL]) | array_reverse_sort([NULL]) |
+---------------------------------+---------------------------------------+---------------------------+
| [6, 3, 2, 1]                    | [5, 4, 3, 1, null]                    | [null]                    |
+---------------------------------+---------------------------------------+---------------------------+
```

ARRAY_SHUFFLE

Description

Randomly arrange the elements in an array

Aliases

- SHUFFLE

Syntax

```
ARRAY_SHUFFLE(<array>, <seed>)
```

Parameters

Parameter	Description
<array>	The array to be randomly permuted
<seed>	An optional parameter that sets the initial value of the pseudo-random number generator used to generate pseudo-random numbers

Return Value

Randomize the elements in an array. The parameter array1 is the array to be randomly arranged, and the optional parameter seed is the initial value used by the pseudo-random number generator to generate pseudo-random numbers. shuffle has the same function as array_shuffle.

Example

```
SELECT ARRAY_SHUFFLE([1, 2, 3, 6]),ARRAY_SHUFFLE([1, 4, 3, 5, NULL],1);
```

```
+----------------------------+-------------------------------------+
| array_shuffle([1, 2, 3, 6]) | array_shuffle([1, 4, 3, 5, NULL], 1) |
+----------------------------+-------------------------------------+
| [2, 6, 3, 1]               | [4, 1, 3, 5, null]                  |
+----------------------------+-------------------------------------+
```

ARRAY_SIZE

Description

Count the number of elements in an array

Aliases

- SIZE
- CARDINALITY

Syntax

```
ARRAY_SIZE(<arr>)
```

Parameters

Parameter	Description
<arr>	The array to be calculated

Return Value

Returns the number of elements in the array. If the input array is NULL, it returns NULL.

Example

```
SELECT ARRAY_SIZE(['a', 'b', 'c']),ARRAY_SIZE([NULL]),ARRAY_SIZE([]);
```

```
+-----------------------------+--------------------+-----------------+
| cardinality(['a', 'b', 'c']) | cardinality([NULL]) | cardinality([]) |
+-----------------------------+--------------------+-----------------+
|                           3 |               NULL |               0 |
+-----------------------------+--------------------+-----------------+
```

ARRAY_SLICE

Description

Specify the starting position and length to extract a portion of elements from an array to form a new sub-array

Syntax

```
ARRAY_SLICE(<arr>, <off>, <len>)
```

Parameters

Parameter	Description
<arr>	Corresponding array
<off>	Starting position. If off is a positive number, it indicates the offset from the left. If off is a negative number, it indicates the offset from the right. When the specified off is not within the actual range of the array, an empty array is returned.
<len>	If len is a negative number, it means the length is 0.

Return Value

Returns a subarray containing all elements of the specified length starting from the specified position. If the input parameter is NULL, it returns NULL.

Example

```
SELECT ARRAY_SLICE([1, 2, 3, 6],2,3),ARRAY_SLICE([1, 4, 3, 5, NULL],-2,1),ARRAY_SLICE([1, 3, 5],0);
```

```
+--------------------------------+---------------------------------------+--------------------------+
| array_slice([1, 2, 3, 6], 2, 3) | array_slice([1, 4, 3, 5, NULL], -2, 1) | array_slice([1, 3, 5], 0) |
+--------------------------------+---------------------------------------+--------------------------+
| [2, 3, 6]                      | [5]                                   | []                       |
+--------------------------------+---------------------------------------+--------------------------+
```

ARRAY_SORT

Description

Sort the elements in an array in ascending order

Syntax

```
ARRAY_SORT(<arr>)
```

Parameters

Parameter	Description
<arr>	Corresponding array

Return Value

Returns an array sorted in ascending order. If the input array is NULL, it returns NULL. If the array elements contain NULL, the output sorted array will put NULL first.

Example

```
SELECT ARRAY_SORT([1, 2, 3, 6]),ARRAY_SORT([1, 4, 3, 5, NULL]),ARRAY_SORT([NULL]);
```

```
+-------------------------+-------------------------------+--------------------+
| array_sort([1, 2, 3, 6]) | array_sort([1, 4, 3, 5, NULL]) | array_sort([NULL]) |
+-------------------------+-------------------------------+--------------------+
| [1, 2, 3, 6]             | [null, 1, 3, 4, 5]             | [null]             |
+-------------------------+-------------------------------+--------------------+
```

ARRAY_SORTBY

Description

First, arrange the key column in ascending order, and then return the corresponding column of the src column sorted in this order as the result; Returns NULL if the input array src is NULL. If the input array key is NULL, the order in which src is returned remains unchanged. If the input array key element contains NULL, the output sorted array will place NULL first.

Syntax

```
ARRAY_SORTBY(<src>, <key>)
ARRAY_SORTBY(<lambda>, <arr> [, ...])
```

Parameters

Parameter	Description
<lambda>	A lambda expression where the input parameters must match the number of columns in the given array. The expression can execute valid scalar functions but does not support aggregate functions.
<arr>	ARRAY array

Return Value

Returns a sorted ARRAY type result.

Example

```
-- setup
create table test_array_sortby(id int, c_array1 array<int>, c_array2 array<int>) distributed by hash(id) buckets 1 properties
("replication_num"="1");
insert into test_array_sortby values (0,null,[2]),(1,[1,2,3,4,5],[10,20,-40,80,-100]),(2,[6,7,8],[10,12,13]),(3,[1],[-100]),
(4,null,null),(5,[3],null),(6,[1,2],[2,1]),(7,[null],[null]),(8,[1,2,3],[3,2,1]);
create table array_test2(id int, c_array1 array<int>, c_array2 array<int>) distributed by hash(id) buckets 1 properties
("replication_num"="1");
insert into array_test2 values (1,[1,2,3],[10,11,12]),(2,[4,3,5],[10,20,30]),(3,[-40,30,-100],[30,10,20]);
```

```
select array_sortby(['a','b','c'],[3,2,1]);
```

```
+-------------------------------------------------+
| array_sortby(ARRAY('a', 'b', 'c'), ARRAY(3, 2, 1)) |
+-------------------------------------------------+
| ['c', 'b', 'a']                                 |
+-------------------------------------------------+
```

```
select array_sortby([1,2,3,4,5],[10,5,1,20,80]);
```

```
+-----------------------------------------------------------+
| array_sortby(ARRAY(1, 2, 3, 4, 5), ARRAY(10, 5, 1, 20, 80)) |
+-----------------------------------------------------------+
| [3, 2, 1, 4, 5]                                           |
+-----------------------------------------------------------+
```

```
select *,array_sortby(c_array1,c_array2) from test_array_sortby order by id;
```

```
+------+-----------------+-------------------------+-------------------------------------+
| id   | c_array1        | c_array2                | array_sortby(`c_array1`, `c_array2`) |
+------+-----------------+-------------------------+-------------------------------------+
|    0 | NULL            | [2]                     | NULL                                |
|    1 | [1, 2, 3, 4, 5] | [10, 20, -40, 80, -100] | [5, 3, 1, 2, 4]                     |
|    2 | [6, 7, 8]       | [10, 12, 13]            | [6, 7, 8]                           |
|    3 | [1]             | [-100]                  | [1]                                 |
|    4 | NULL            | NULL                    | NULL                                |
|    5 | [3]             | NULL                    | [3]                                 |
|    6 | [1, 2]          | [2, 1]                  | [2, 1]                              |
|    7 | [NULL]          | [NULL]                  | [NULL]                              |
|    8 | [1, 2, 3]       | [3, 2, 1]               | [3, 2, 1]                           |
+------+-----------------+-------------------------+-------------------------------------+
```

```
select *, array_map((x,y)->(y+x),c_array1,c_array2) as arr_sum,array_sortby((x,y)->(y+x),c_array1,c_array2) as arr_sort from
array_test2;
```

```
+------+-----------------+--------------+----------------+-----------------+
| id   | c_array1        | c_array2     | arr_sum        | arr_sort        |
+------+-----------------+--------------+----------------+-----------------+
|    1 | [1, 2, 3]       | [10, 11, 12] | [11, 13, 15]   | [1, 2, 3]       |
|    2 | [4, 3, 5]       | [10, 20, 30] | [14, 23, 35]   | [4, 3, 5]       |
|    3 | [-40, 30, -100] | [30, 10, 20] | [-10, 40, -80] | [-100, -40, 30] |
+------+-----------------+--------------+----------------+-----------------+
```

ARRAY_SUM

Description

Calculates the sum of all elements in an array

Syntax

```
ARRAY_SUM(<src>)
```

Parameters

Parameter	Description
<src>	Corresponding array

Return Value

Returns the sum of all elements in the array. NULL values in the array will be skipped. For an empty array or an array with all NULL values, the result returns a NULL value.

Example

```
SELECT ARRAY_SUM([1, 2, 3, 6]),ARRAY_SUM([1, 4, 3, 5, NULL]),ARRAY_SUM([NULL]);
```

```
+------------------------+-----------------------------+-----------------------------------------+
| array_sum([1, 2, 3, 6]) | array_sum([1, 4, 3, 5, NULL]) | array_sum(cast([NULL] as ARRAY<BOOLEAN>)) |
+------------------------+-----------------------------+-----------------------------------------+
|                     12 |                          13 |                                    NULL |
+------------------------+-----------------------------+-----------------------------------------+
```

ARRAY_UNION

Description

Merge multiple arrays without duplicate elements to generate a new array

Syntax

```
ARRAY_UNION(<array>, <array> [, ... ])
```

Parameters

Parameter	Description
<array>	The array to be merged

Return Value

Returns an array containing all elements in the union of all arrays, excluding duplicates. If the input parameter is NULL, it returns NULL.

Example

```
SELECT ARRAY_UNION([1, 2, 3, 6],[1, 2, 5]),ARRAY_UNION([1, 4, 3, 5, NULL],[1,6,10]);
```

```
+-------------------------------------+-----------------------------------------+
| array_union([1, 2, 3, 6], [1, 2, 5]) | array_union([1, 4, 3, 5, NULL], [1, 6, 10]) |
+-------------------------------------+-----------------------------------------+
| [3, 2, 1, 6, 5]                     | [null, 10, 3, 1, 6, 4, 5]                 |
+-------------------------------------+-----------------------------------------+
```

ARRAY_WITH_CONSTANT

Description

Generates an array containing n repeated elements

Syntax

```
ARRAY_WITH_CONSTANT(<n>, <element>)
```

Parameters

Parameter	Description
<n>	Number of digits
<element>	Specifying Elements

Return Value

Returns an array containing n repeated elements. array_repeat has the same function as array_with_constant and is used to be compatible with the hive syntax format.

Example

```
SELECT ARRAY_WITH_CONSTANT(2, "hello"),ARRAY_WITH_CONSTANT(3, 12345);
```

```
+--------------------------------+-------------------------------+
| array_with_constant(2, 'hello') | array_with_constant(3, 12345) |
+--------------------------------+-------------------------------+
| ["hello", "hello"]              | [12345, 12345, 12345]         |
+--------------------------------+-------------------------------+
```

ARRAY_ZIP

Description

Merges all arrays into a single array. The resulting array contains corresponding elements from the source arrays, grouped in the order of the argument list.

Syntax

```
ARRAY_ZIP(<array>[, <array> ])
```

Parameters

Parameter	Description
<array>	Input array

Return Value

Returns an array with the elements from the source array grouped into a structure. The data types in the structure are the same as the input array and are in the order in which the array was passed.

Example

```
SELECT ARRAY_ZIP(['a', 'b', 'c'], [1, 2, 3]);
```

```
+-----------------------------------------------------+
| array_zip(['a', 'b', 'c'], [1, 2, 3])               |
+-----------------------------------------------------+
| [{"1":"a", "2":1}, {"1":"b", "2":2}, {"1":"c", "2":3}] |
+-----------------------------------------------------+
```

ARRAYS_OVERLAP

Description

Determine whether the left and right arrays contain common elements

Syntax

```
ARRAYS_OVERLAP(<left>, <right>)
```

Parameters

Parameter	Description
<left>	The array to be judged
<right>	The array to be judged

Return Value

Returns the judgment result: 1: left and right arrays have common elements; 0: left and right arrays do not have common elements; NULL: left or right array is NULL; or any element in left and right array is NULL

Example

```
SELECT ARRAYS_OVERLAP(['a', 'b', 'c'], [1, 2, 'b']);
```

```
+-------------------------------------------------+
| arrays_overlap(['a', 'b', 'c'], ['1', '2', 'b']) |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
```

COUNTEQUAL

Description

Determine the number of value elements in the array

Syntax

```
COUNTEQUAL(<arr>, <value>)
```

Parameters

Parameter	Description
<arr>	Input arrayd
<value>	Judging elements

Return Value

The returned judgment results are as follows: num: the number of value in array; 0: value does not exist in array arr; NULL: if the array is NULL.

Example

```
SELECT COUNTEQUAL(NULL,1),COUNTEQUAL([1, 2, 3, 'c'],2),COUNTEQUAL([],'b');
```

```
+--------------------+-----------------------------------------------------+-----------------------------------------+
| countequal(NULL, 1) | countequal(['1', '2', '3', 'c'], cast(2 as TEXT)) | countequal(cast([] as ARRAY<TEXT>), 'b') |
+--------------------+-----------------------------------------------------+-----------------------------------------+
|               NULL |                                                   1 |                                       0 |
+--------------------+-----------------------------------------------------+-----------------------------------------+
```

MAP | Map Functions

MAP

Description

Constructs a Map<K, V> of a specific type using some set of key-value pairs

Syntax

```
MAP( <key1> , <value1> [, <key2>,<value2> ... ])
```

Parameters

Parameter	Description
<key>	Constructing the key of the map, case sensitive
<value>	Constructing the value of the map

Return Value

Returns a specific type Map<K, V> constructed from a number of key-value pairs

Example

```
select map(1, "100", 0.1, 2),map(1, "100", 0.1, 2)[1];
```

```
+-----------------------------------------------------------------------------------------+-----------------------------------------
-------------------------------------------------------------------------------+
| map(cast(1 as DECIMALV3(2, 1)), '100', cast(0.1 as DECIMALV3(2, 1)), cast(2 as TEXT)) | element_at(map(cast(1 as DECIMALV3(2, 1)),
'100', cast(0.1 as DECIMALV3(2, 1)), cast(2 as TEXT)), cast(1 as DECIMALV3(2, 1))) |
+-----------------------------------------------------------------------------------------+-----------------------------------------
-------------------------------------------------------------------------------+
| {1.0:"100", 0.1:"2"}                                                                    | 100
|
+-----------------------------------------------------------------------------------------+-----------------------------------------
-------------------------------------------------------------------------------+
```

MAP_CONTAINS_KEY

Description

Determines whether the given `map` contains a specific key `key`

Syntax

```
MAP_CONTAINS_KEY(<map>, <key>)
```

Parameters

Parameter	Description
<map>	Input map content
<key>	The key to be retrieved

Return Value

Determines whether the given `map` contains a specific key `key`, and returns 1 if it exists, otherwise returns 0.

Example

```
select map_contains_key(map(null, 1, 2, null), null),map_contains_key(map(1, "100", 0.1, 2), 0.11);
```

```
+-------------------------------------------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| map_contains_key(map(NULL, 1, 2, NULL), NULL) | map_contains_key(cast(map(cast(1 as DECIMALV3(2, 1)), '100', cast(0.1 as
DECIMALV3(2, 1)), cast(2 as TEXT)) as MAP<DECIMALV3(3, 2),VARCHAR(3)>), cast(0.11 as DECIMALV3(3, 2))) |
+-------------------------------------------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
|                                         1 |
0 |
+-------------------------------------------+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
```

MAP_CONTAINS_VALUE

Description

Determines whether the given `map` contains a specific value `value`

Syntax

```
MAP_CONTAINS_VALUE(<map>, <value>)
```

Parameters

Parameter	Description
`<map>`	Input map content
`<value>`	The value to be retrieved

Return Value

Determines whether the given `map` contains a specific value `value`, and returns 1 if it exists, otherwise returns 0.

Example

```
select map_contains_value(map(null, 1, 2, null), null),map_contains_value(map(1, "100", 0.1, 2), 101);
```

```
+-------------------------------------------------+-------------------------------------------------------------------------------------------------
-------------------------------------------------+
| map_contains_value(map(NULL, 1, 2, NULL), NULL) | map_contains_value(map(cast(1 as DECIMALV3(2, 1)), '100', cast(0.1 as
DECIMALV3(2, 1)), cast(2 as TEXT)), cast(101 as VARCHAR(3))) |
+-------------------------------------------------+-------------------------------------------------------------------------------------------------
-------------------------------------------------+
|                                               1 |
0 |
+-------------------------------------------------+-------------------------------------------------------------------------------------------------
-------------------------------------------------+
```

MAP_KEYS

Description

Extract the keys of the given `map` into an `ARRAY` of the corresponding type

Syntax

```
MAP_KEYS(<map>)
```

Parameters

Parameter	Description
<map>	Input map content

Return Value

Extract the keys of the given `map` into an `ARRAY` of the corresponding type

Example

```
select map_keys(map()),map_keys(map(1, "100", 0.1, 2));
```

```
+-----------------+-----------------------------------------------------------------------------------------------+
| map_keys(map()) | map_keys(map(cast(1 as DECIMALV3(2, 1)), '100', cast(0.1 as DECIMALV3(2, 1)), cast(2 as TEXT))) |
+-----------------+-----------------------------------------------------------------------------------------------+
| []              | [1.0, 0.1]                                                                                    |
+-----------------+-----------------------------------------------------------------------------------------------+
```

MAP_SIZE

Description

Count the number of elements in a Map

Syntax

```
MAP_SIZE(<map>)
```

Parameters

Parameter	Description
<map>	Input map content

Return Value

Returns the number of elements in the Map

Example

```
select map_size(map()),map_size(map(1, "100", 0.1, 2));
```

```
+-----------------+-----------------------------------------------------------------------------------------------+
| map_size(map()) | map_size(map(cast(1 as DECIMALV3(2, 1)), '100', cast(0.1 as DECIMALV3(2, 1)), cast(2 as TEXT))) |
+-----------------+-----------------------------------------------------------------------------------------------+
|               0 |                                                                                             2 |
+-----------------+-----------------------------------------------------------------------------------------------+
```

MAP_VALUES

Description

Extract the values of the given `map` into an `ARRAY` of the corresponding type

Syntax

```
MAP_VALUES(<map>)
```

Parameters

Parameter	Description
<map>	Input map content

Return Value

Extract the values of the given `map` into an `ARRAY` of the corresponding type

Example

```
select map_values(map()),map_values(map(1, "100", 0.1, 2));
```

```
+------------------+-----------------------------------------------------------------------------------------------+
| map_values(map()) | map_values(map(cast(1 as DECIMALV3(2, 1)), '100', cast(0.1 as DECIMALV3(2, 1)), cast(2 as TEXT))) |
+------------------+-----------------------------------------------------------------------------------------------+
| []               | ["100", "2"]                                                                                  |
+------------------+-----------------------------------------------------------------------------------------------+
```

NAMED_STRUCT

Description

Construct and return a struct based on the given strings and values. Notes:

- The number of parameters must be a non-zero even number.The odd-indexed elements are the names of the fields, which must be constant strings.The even-indexed elements are the values of the fields, which can be either multiple columns or constants.

Syntax

```
NAMED_STRUCT( <field_name> , <filed_value> [ , <field_name> , <filed_value> ... ] )
```

Parameters

Parameter	Description
<field_name>	The odd-indexed elements in constructing the struct are the field names, which must be constant strings
<filed_value>	The even-indexed elements in constructing the struct represent the field values, which can be either multiple columns or constants

Return Value

Construct and return a struct based on the given strings and values.

Example

```
select named_struct('f1', 1, 'f2', 'a', 'f3', "abc"),named_struct('a', null, 'b', "v");
```

```
+-----------------------------------------------+----------------------------------+
| named_struct('f1', 1, 'f2', 'a', 'f3', 'abc') | named_struct('a', NULL, 'b', 'v') |
+-----------------------------------------------+----------------------------------+
| {"f1":1, "f2":"a", "f3":"abc"}                | {"a":null, "b":"v"}              |
+-----------------------------------------------+----------------------------------+
```

STRUCT | Struct Functions

STRUCT

Description

construct an struct with variadic elements and return it, Tn could be column or literal

Syntax

```
STRUCT( <expr1> [ , <expr2> ... ] )
```

Parameters

Parameter	Description
<expr>	Construct the input content for the struct

Return Value

construct an struct with variadic elements and return it, Tn could be column or literal

Example

```
select struct(1, 'a', "abc"),struct(null, 1, null),struct(cast('2023-03-16' as datetime));
```

```
+-------------------------------------+-------------------------------------+-----------------------------------------+
| struct(1, 'a', 'abc')               | struct(NULL, 1, NULL)               | struct(cast('2023-03-16' as DATETIMEV2(0))) |
+-------------------------------------+-------------------------------------+-----------------------------------------+
| {"col1":1, "col2":"a", "col3":"abc"} | {"col1":null, "col2":1, "col3":null} | {"col1":"2023-03-16 00:00:00"}              |
+-------------------------------------+-------------------------------------+-----------------------------------------+
```

STRUCT_ELEMENT

Description

Return a specific field within a data column of a struct.

Syntax

```
STRUCT_ELEMENT( <struct>, '<filed_location>/<filed_name>')
```

Parameters

Parameter	Description
<struct>	If the input struct column is null, return null
<filed_location>	The position of the field starts from 1, and only constants are supported
<filed_name>	The field name must be a constant, case sensitive

Return Value

Return the specified field column, with the type being any type

Example

```
select struct_element(named_struct('f1', 1, 'f2', 'a'), 'f2'),struct_element(named_struct('f1', 1, 'f2', 'a'), 1);
```

```
+-----------------------------------------------------+-----------------------------------------------------+
| struct_element(named_struct('f1', 1, 'f2', 'a'), 'f2') | struct_element(named_struct('f1', 1, 'f2', 'a'), 1) |
+-----------------------------------------------------+-----------------------------------------------------+
| a                                                   |                                                   1 |
+-----------------------------------------------------+-----------------------------------------------------+
```

```
select struct_col, struct_element(struct_col, 'f1') from test_struct;
```

```
+-----------------------------------------------+-----------------------------------+
| struct_col                                    | struct_element(`struct_col `, 'f1') |
+-----------------------------------------------+-----------------------------------+
| {1, 2, 3, 4, 5}                               |                                 1 |
| {1, 1000, 10000000, 100000000000, 100000000000} |                                 1 |
| {5, 4, 3, 2, 1}                               |                                 5 |
| NULL                                          |                              NULL |
| {1, NULL, 3, NULL, 5}                         |                                 1 |
+-----------------------------------------------+-----------------------------------+
```

GET_JSON_BIGINT

Description

Parses and obtains the integer (BIGINT) content of the specified path within the json string.

Syntax

```
GET_JSON_BIGINT( <json_str>, <json_path>)
```

Required Parameters

parameters	described
<json_str>	The JSON string from which to extract data is needed.
<json_path>	JSON path, specifying the location of the field. Paths can be denoted in dot notation.

Return Value

- Returns the BIGINT value of the field pointed to by the path.
- Returns NULL if no corresponding field is found in the specified path, or the field value cannot be converted to type BIGINT.

Usage Notes

Where <json_str> `must start with the $symbol and use. As a path splitter. If the path contains. , you can use double quotes to enclose it. Use [ ] to represent the array index, starting from 0. The content of path cannot contain “, [and]. Returns NULL if the format is incorrect, or the format is incorrect, or a match cannot be found. In addition, it is recommended to use the jsonb type and the jsonb_extract_XXX function to achieve the same functionality. Special circumstances will be handled as follows: - Returns NULL if the specified field does not exist in JSON - If the actual type of the specified field in JSON is inconsistent with the type specified by json_extract_t, the specified type t will be returned if it can be losslessly converted to the specified type, and NULL will be returned if it cannot.

Examples

1. Get the value with key as “k1”

```
SELECT get_json_bigint('{"k1":1, "k2":"2"}', "$.k1");
```

```
+-----------------------------------------------+
| get_json_bigint('{"k1":1, "k2":"2"}', '$.k1') |
+-----------------------------------------------+
```

```
|                                          1 |
+--------------------------------------------+
```

2. Gets the second element in the array with key “my.key”

```
SELECT get_json_bigint('{"k1":"v1", "my.key":[1, 1678708107000, 3]}', '$. "my.key"[1]');
```

```
+-------------------------------------------------------------------------------+
| get_json_bigint('{"k1":"v1", "my.key":[1, 1678708107000, 3]}', '$. "my.key"[1]') |
+-------------------------------------------------------------------------------+
|                                                                 1678708107000 |
+-------------------------------------------------------------------------------+
```

3. Gets the first element in an array with secondary path k1.key -> k2

```
SELECT get_json_bigint('{"k1.key":{"k2":[1678708107000, 2]}}', '$. "k1.key".k2[0]');
```

```
+-----------------------------------------------------------------------------+
| get_json_bigint('{"k1.key":{"k2":[1678708107000, 2]}}', '$."k1.key".k2[0]') |
+-----------------------------------------------------------------------------+
|                                                               1678708107000 |
+-----------------------------------------------------------------------------+
```

GET_JSON_DOUBLE

Description

Function is used to extract the value of a field from a JSON document and convert it to the DOUBLE type. This function returns the value of the field on the specified path, or returns NULL if the value cannot be converted to type DOUBLE or the field pointed to by the path does not exist.

Syntax

```
GET_JSON_DOUBLE( <json_str>, <json_path>)
```

Required Parameters

parameters	described
<json_str>	The JSON string from which to extract data is needed.
<json_path>	JSON path, specifying the location of the field. Paths can be denoted in dot notation.

Return Value

- Returns the DOUBLE value of the field pointed to by the path.
- Returns NULL if no corresponding field is found in the specified path, or the field value cannot be converted to type DOUBLE.

Usage Notes

Parses and obtains the floating point content of the specified path within the json string. Where <json_path> must start with the $symbol and use. As a path splitter. If the path contains. , you can use double quotes to enclose it. Use [ ] to represent the array index, starting from 0. The content of path cannot contain ", [and]. Returns NULL if` is incorrectly formatted, or json_path is incorrectly formatted, or a match cannot be found. In addition, it is recommended to use the jsonb type and the jsonb_extract_XXX function to achieve the same functionality. Special circumstances will be handled as follows: - Returns NULL if the field specified by json_path does not exist in JSON - If the actual type of the field specified by json_path in JSON is inconsistent with the type specified by json_extract_t, the specified type t is returned if it can be losslessly converted to the specified type, and NULL is returned if it cannot.

Examples

1. Get the value with key as “k1”

```
SELECT get_json_double('{"k1":1.3, "k2":"2"}', "$.k1");
```

```
+-------------------------------------------------+
| get_json_double('{"k1":1.3, "k2":"2"}', '$.k1') |
+-------------------------------------------------+
|                                             1.3 |
+-------------------------------------------------+
```

2. Gets the second element in the array with key “my.key”

```
SELECT get_json_double('{"k1":"v1", "my.key":[1.1, 2.2, 3.3]}', '$. "my.key"[1]');
```

```
+-------------------------------------------------------------------------+
| get_json_double('{"k1":"v1", "my.key":[1.1, 2.2, 3.3]}', '$."my.key"[1]') |
+-------------------------------------------------------------------------+
|                                                                     2.2 |
+-------------------------------------------------------------------------+
```

3. Gets the first element in an array with secondary path k1.key -> k2

```
SELECT get_json_double('{"k1.key":{"k2":[1.1, 2.2]}}', '$."k1.key".k2[0]');
```

```
+---------------------------------------------------------------+
| get_json_double('{"k1.key":{"k2":[1.1, 2.2]}}', '$."k1.key".k2[0]') |
+---------------------------------------------------------------+
|                                                           1.1 |
+---------------------------------------------------------------+
```

GET_JSON_INT

Description

Function is used to extract the value of a field from a JSON document and convert it to type INT. This function returns the value of the field on the specified path, or returns NULL if the value cannot be converted to type INT or the field pointed to by the path does not exist.

Syntax

```
GET_JSON_INT( <json_str>, <json_path>)
```

Required Parameters

parameters	described
<json_str>	The JSON string from which to extract data is needed.
<json_path>	JSON path, specifying the location of the field. Paths can be denoted in dot notation.

Usage Notes

Parses and obtains the integer content of the specified path within the json string. Where <json_path> `must start with the $symbol and use. As a path splitter. If the path contains. , you can use double quotes to enclose it. Use [ ] to represent the array index, starting from 0. The content of path cannot contain “, [and]. Returns NULL if the format is incorrect, or the format is incorrect, or a match cannot be found. In addition, it is recommended to use the jsonb type and the jsonb_extract_XXX function to achieve the same functionality. Special circumstances will be handled as follows: - Returns NULL if the specified field does not exist in JSON - If the actual type of the specified field in JSON is inconsistent with the type specified by json_extract_t, the specified type t will be returned if it can be losslessly converted to the specified type, and NULL will be returned if it cannot.

Examples

1. Get the value with key as “k1”

```
SELECT get_json_int('{"k1":1, "k2":"2"}', "$.k1");
```

```
+-----------------------------------------+
| get_json_int('{"k1":1, "k2":"2"}', '$.k1') |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

2. Gets the second element in the array with key “my.key”

```
SELECT get_json_int('{"k1":"v1", "my.key":[1, 2, 3]}', '$."my.key"[1]');
```

```
+-------------------------------------------------------------------+
| get_json_int('{"k1":"v1", "my.key":[1, 2, 3]}', '$."my.key"[1]') |
+-------------------------------------------------------------------+
|                                                                 2 |
+-------------------------------------------------------------------+

3. Gets the first element in an array with secondary path k1.key -> k2

```sql
 SELECT get_json_int('{"k1.key":{"k2":[1, 2]}}', '$."k1.key".k2[0]');
```

```
+-----------------------------------------------------------+
| get_json_int('{"k1.key":{"k2":[1, 2]}}', '$."k1.key".k2[0]') |
+-----------------------------------------------------------+
|                                                         1 |
+-----------------------------------------------------------+
```
```

GET_JSON_STRING

Description

This function is used to extract a field's value from a JSON document and convert it to `STRING` type. It returns the field value at the specified path. If the value cannot be converted to a string, or if the field at the specified path does not exist, it returns `NULL`.

Syntax

```
GET_JSON_STRING( <json_str>, <json_path>)
```

Required Parameters

Parameter	Description
`<json_str>`	The JSON string from which data needs to be extracted.
`<json_path>`	JSON path that specifies the field's location. The path can use dot notation.

Return Value

It returns the `STRING` value of the field at the specified path. If the specified path does not point to a valid field or the field value cannot be converted to a `STRING` type, it returns `NULL`.

Usage Notes

Parses and retrieves the string content of the specified path in the JSON string. The `<json_path>` must start with the `$` symbol, using `.` as the path delimiter. If the path contains a `.`, it should be enclosed in double quotes. Use `[ ]` to indicate array indices, starting from 0. The path should not contain `"`, `[`, and `]`. If the `<json_str>` format is incorrect, or if the `<json_path>` format is invalid, or if no matching field is found, `NULL` is returned.

Additionally, it is recommended to use the `jsonb` type and `jsonb_extract_XXX` functions to achieve the same functionality.

Special case handling as follows: - If the field specified by `<json_path>` does not exist in the JSON, return `NULL`. - If the actual type of the field specified by `<json_path>` differs from the type expected by `json_extract_t`, if it can be losslessly converted to the expected type, it will return the specified type. Otherwise, it will return `NULL`.

Examples

1. Get the value for key "k1"

```
SELECT get_json_string('{"k1":"v1", "k2":"v2"}', "$.k1");
```

```
+-------------------------------------------------+
| get_json_string('{"k1":"v1", "k2":"v2"}', '$.k1') |
+-------------------------------------------------+
| v1                                              |
+-------------------------------------------------+
```

2. Get the second element of the array for key “my.key”

```
SELECT get_json_string('{"k1":"v1", "my.key":["e1", "e2", "e3"]}', '$."my.key"[1]');
```

```
+-----------------------------------------------------------------------------+
| get_json_string('{"k1":"v1", "my.key":["e1", "e2", "e3"]}', '$."my.key"[1]') |
+-----------------------------------------------------------------------------+
| e2                                                                          |
+-----------------------------------------------------------------------------+
```

3. Get the first element of the array in the secondary path k1.key -> k2

```
 SELECT get_json_string('{"k1.key":{"k2":["v1", "v2"]}}', '$."k1.key".k2[0]');
```

```
+-------------------------------------------------------------------+
| get_json_string('{"k1.key":{"k2":["v1", "v2"]}}', '$."k1.key".k2[0]') |
+-------------------------------------------------------------------+
| v1                                                                |
+-------------------------------------------------------------------+
```

JSON_ARRAY

Description

Generate a json array containing the specified values, return empty if no values

Syntax

```
JSON_ARRAY (<a>, ...)
```

Parameters

Parameter	Description
<a>, ...	Elements to be included in the JSON array. It can be a single or multiple values of any type, including NULL.

Return Values

Returns a JSON array containing the specified values. If no values are specified, an empty JSON array is returned.

Examples

```
select json_array();
```

```
+--------------+
| json_array() |
+--------------+
| []           |
+--------------+
```

```
select json_array(null);
```

```
+--------------------+
| json_array('NULL') |
+--------------------+
| [NULL]             |
+--------------------+
```

```
SELECT json_array(1, "abc", NULL, TRUE, CURTIME());
```

```
+-----------------------------------------------+
| json_array(1, 'abc', 'NULL', TRUE, curtime()) |
+-----------------------------------------------+
| [1, "abc", NULL, TRUE, "10:41:15"]            |
+-----------------------------------------------+
```

```
select json_array("a", null, "c");
```

```
+------------------------------+
| json_array('a', 'NULL', 'c') |
+------------------------------+
| ["a", NULL, "c"]             |
+------------------------------+
```

JSON_CONTAINS

Description

This function is used to check whether a JSON document contains a specified JSON element. If the specified element exists in the JSON document, it returns 1; otherwise, it returns 0. If the JSON document or the queried element is invalid, it returns `NULL`.

Syntax

```
JSON_CONTAINS(<json_str>, <candidate> [, <json_path>])
```

Required Parameters

Parameter	Description
`<json_str>`	The JSON string to be checked.
`<candidate>`	The JSON element to check for inclusion.

Optional Parameters

Parameter	Description
`<json_path>`	An optional JSON path to specify the subdocument to check. If not provided, the root document is used by default.

Return Value

- If `<json_path>` exists in `json_doc`, it returns 1.
- If `<json_path>` does not exist in `json_doc`, it returns 0.
- If any parameter is invalid or the JSON document format is incorrect, it returns `NULL`.

Examples

```
SELECT JSON_CONTAINS('{"a": 1, "b": 2, "c": {"d": 4}}', '1', '$.a');
```

```
+-----------------------------------------------------------------------------------------+
| json_contains(cast('{"a": 1, "b": 2, "c": {"d": 4}}' as JSON), cast('1' as JSON), '$.a') |
+-----------------------------------------------------------------------------------------+
|                                                                                       1 |
+-----------------------------------------------------------------------------------------+
```

```
SELECT json_contains('[1, 2, {"x": 3}]', '1');
```

```
+-------------------------------------------------------------------------+
| json_contains(cast('[1, 2, {"x": 3}]' as JSON), cast('1' as JSON), '$') |
+-------------------------------------------------------------------------+
|                                                                       1 |
+-------------------------------------------------------------------------+
```

JSON_EXISTS_PATH

Description

It is used to judge whether the field specified by json_path exists in the JSON data. If it exists, it returns TRUE, and if it does not exist, it returns FALSE

Syntax

```
JSON_EXISTS_PATH (<json_str>,  <path>)
```

Alias

- JSONB_EXISTS_PATH

Parameters

Parameter	Description
<json_str>	The element to be included in the JSON array. It can be a value of any type, including NULL. If no element is specified, an empty array is returned.
<path>	The JSON path to be judged. If it is NULL, then return NULL.

Return Values

If it exists, return TRUE; if it does not exist, return FALSE.

Examples

```
SELECT JSON_EXISTS_PATH('{"id": 123, "name": "doris"}', '$.name');
```

```
+-----------------------------------------------------------------------+
| jsonb_exists_path(cast('{"id": 123, "name": "doris"}' as JSON), '$.name') |
+-----------------------------------------------------------------------+
|                                                                     1 |
+-----------------------------------------------------------------------+
```

```
SELECT JSON_EXISTS_PATH('{"id": 123, "name": "doris"}', '$.age');
```

```
+-----------------------------------------------------------------------+
| jsonb_exists_path(cast('{"id": 123, "name": "doris"}' as JSON), '$.age') |
+-----------------------------------------------------------------------+
|                                                                     0 |
+-----------------------------------------------------------------------+
```

```
SELECT JSONB_EXISTS_PATH('{"id": 123, "name": "doris"}', '$.age');
```

```
+-----------------------------------------------------------------------------+
| jsonb_exists_path(cast('{"id": 123, "name": "doris"}' as JSON), '$.age') |
+-----------------------------------------------------------------------------+
|                                                                           0 |
+-----------------------------------------------------------------------------+
```

JSON_EXTRACT

Description

JSON_EXTRACT is a series of functions that extract the field specified by json_path from JSON data and provide different series of functions according to the type of the field to be extracted.

- JSON_EXTRACT returns the VARCHAR type for a json string of the VARCHAR type.
- JSON_EXTRACT_ISNULL returns the BOOLEAN type indicating whether it is a json null.
- JSON_EXTRACT_BOOL returns the BOOLEAN type.
- JSON_EXTRACT_INT returns the INT type.
- JSON_EXTRACT_BIGINT returns the BIGINT type.
- JSON_EXTRACT_LARGEINT returns the LARGEINT type.
- JSON_EXTRACT_DOUBLE returns the DOUBLE type.
- JSON_EXTRACT_STRING returns the STRING type.

Alias

- JSONB_EXTRACT is the same as JSON_EXTRACT.
- JSONB_EXTRACT_ISNULL is the same as JSON_EXTRACT_ISNULL.
- JSONB_EXTRACT_BOOL is the same as JSON_EXTRACT_BOOL.
- JSONB_EXTRACT_INT is the same as JSON_EXTRACT_INT.
- JSONB_EXTRACT_BIGINT is the same as JSON_EXTRACT_BIGINT.
- JSONB_EXTRACT_LARGEINT is the same as JSON_EXTRACT_LARGEINT.
- JSONB_EXTRACT_DOUBLE is the same as JSON_EXTRACT_DOUBLE.
- JSONB_EXTRACT_STRING is the same as JSON_EXTRACT_STRING.

Syntax

```
JSON_EXTRACT (<json_str>, <path>[, path] ...)
```

```
JSON_EXTRACT_ISNULL (<json_str>, <path>)
```

```
JSON_EXTRACT_BOOL (<json_str>, <path>)
```

```
JSON_EXTRACT_INT (<json_str>, <path>)
```

```
JSON_EXTRACT_BIGINT (<json_str>, <path>)
```

```
JSON_EXTRACT_LARGEINT (<json_str>, <path>)
```

```
JSON_EXTRACT_DOUBLE (<json_str>, <path>)
```

```
JSON_EXTRACT_STRING (<json_str>, <path>)
```

Alias functions have the same syntax and usage as the above functions, except for the function names.

Parameters

Parameter	Description
<json_str>	The JSON-type parameter or field to be extracted.
<path>	The JSON path to extract the target element from the target JSON.

json path syntax: - '$' for json document root - '.k1' for element of json object with key 'k1' - If the key column value contains ".", double quotes are required in json_path, For example: SELECT json_extract('{"k1.a":"abc","k2":300}', '$."k1.a"'); - '[i]' for element of json array at index i - Use '[last]' to get the last element of json_array, and '[last-1]' to get the penultimate element, and so on.

<path> does not auto-broadcast over arrays: if the JSON value is an array and <path> is `$.k`, the result is NULL because `$.k` only traverses object members. To target an element by index use `$[i].k`. Array-wildcard broadcasting via `$[*].k` was introduced in Doris 4.0; on 2.1 and 3.x, extract per-element values via `LATERAL VIEW EXPLODE` patterns instead.

Return Values

According to the type of the field to be extracted, return the data type of the specified JSON_PATH in the target JSON. Special case handling is as follows: * If the field specified by json_path does not exist in the JSON, return NULL. * If the actual type of the field specified by json_path in the JSON is inconsistent with the type specified by json_extract_t. * if it can be losslessly converted to the specified type, return the specified type t; if not, return NULL.

Examples

```
SELECT json_extract('{"id": 123, "name": "doris"}', '$.id');
```

```
+-----------------------------------------------------+
| json_extract('{"id": 123, "name": "doris"}', '$.id') |
+-----------------------------------------------------+
| 123                                                 |
+-----------------------------------------------------+
```

```
SELECT json_extract('[1, 2, 3]', '$.[1]');
```

```
+----------------------------------+
| json_extract('[1, 2, 3]', '$.[1]') |
+----------------------------------+
| 2                                |
+----------------------------------+
```

```
SELECT json_extract('{"k1": "v1", "k2": { "k21": 6.6, "k22": [1, 2] } }', '$.k1', '$.k2.k21', '$.k2.k22', '$.k2.k22[1]');
```

```
+-----------------------------------------------------------------------------------------------------------------+
| json_extract('{"k1": "v1", "k2": { "k21": 6.6, "k22": [1, 2] } }', '$.k1', '$.k2.k21', '$.k2.k22', '$.k2.k22[1]') |
+-----------------------------------------------------------------------------------------------------------------+
| ["v1",6.6,[1,2],2]                                                                                              |
+-----------------------------------------------------------------------------------------------------------------+
```

```
SELECT json_extract('{"id": 123, "name": "doris"}', '$.aaa', '$.name');
```

```
+-----------------------------------------------------------------+
| json_extract('{"id": 123, "name": "doris"}', '$.aaa', '$.name') |
+-----------------------------------------------------------------+
| [null,"doris"]                                                  |
+-----------------------------------------------------------------+
```

```
SELECT json_extract_no_quotes('{"id": 123, "name": "doris"}', '$.name');
```

```
+--------------------------------------------------------------------+
| json_extract_no_quotes('{"id": 123, "name": "doris"}', '$.name') |
+--------------------------------------------------------------------+
| doris                                                              |
+--------------------------------------------------------------------+
```

```
SELECT JSON_EXTRACT_ISNULL('{"id": 123, "name": "doris"}', '$.id');
```

```
+-----------------------------------------------------------------------------+
| jsonb_extract_isnull(cast('{"id": 123, "name": "doris"}' as JSON), '$.id') |
+-----------------------------------------------------------------------------+
|                                                                           0 |
+-----------------------------------------------------------------------------+
```

```
SELECT JSON_EXTRACT_BOOL('{"id": 123, "name": "NULL"}', '$.id');
```

```
+---------------------------------------------------------------------------+
| jsonb_extract_bool(cast('{"id": 123, "name": "NULL"}' as JSON), '$.id') |
+---------------------------------------------------------------------------+
|                                                                      NULL |
+---------------------------------------------------------------------------+
```

```
SELECT JSON_EXTRACT_INT('{"id": 123, "name": "NULL"}', '$.id');
```

```
+---------------------------------------------------------------------+
| jsonb_extract_int(cast('{"id": 123, "name": "NULL"}' as JSON), '$.id') |
+---------------------------------------------------------------------+
|                                                                 123 |
+---------------------------------------------------------------------+
```

```
SELECT JSON_EXTRACT_INT('{"id": 123, "name": "doris"}', '$.name');
```

```
+-----------------------------------------------------------------------+
| jsonb_extract_int(cast('{"id": 123, "name": "doris"}' as JSON), '$.name') |
+-----------------------------------------------------------------------+
|                                                                  NULL |
+-----------------------------------------------------------------------+
```

```
SELECT JSON_EXTRACT_STRING('{"id": 123, "name": "doris"}', '$.name');
```

```
+--------------------------------------------------------------------------+
| jsonb_extract_string(cast('{"id": 123, "name": "doris"}' as JSON), '$.name') |
+--------------------------------------------------------------------------+
| doris                                                                    |
+--------------------------------------------------------------------------+
```

JSON_EXTRACT_BIGINT

JSON_EXTRACT_BOOL

JSON_EXTRACT_DOUBLE

JSON_EXTRACT_INT

JSON_EXTRACT_ISNULL

JSON_EXTRACT_LARGEINT

JSON_EXTRACT_STRING

JSON_INSERT

Description

The JSON_INSERT function is used to insert data into JSON and return the result.

Syntax

```
JSON_INSERT (<json_str>, <path>,  <val>[, <path>,  <val>, ...])
```

Parameters

Parameter	Description
<json_str>	The JSON object to be inserted. It can be a JSON object with elements of any type, including NULL. If no elements are specified, an empty array is returned. If json_str is not a valid JSON or any path parameter is not a valid path expression or contains a * wildcard, an error is returned.
<path>	The JSON path to be inserted. If it is NULL, then return NULL.
<val>	The value to be inserted into the JSON. If it is NULL, then a NULL value will be inserted at the corresponding position.

`json_insert` function inserts data in a JSON and returns the result.Returns NULL if `json_str` or `path` is NULL. Otherwise, an error occurs if the `json_str` argument is not a valid JSON or any path argument is not a valid path expression or contains a * wildcard.

The path-value pairs are evaluated left to right.

A path-value pair for a nonexisting path in the json adds the value to the json if the path identifies one of these types of values:

- A member not present in an existing object. The member is added to the object and associated with the new value.
- A position past the end of an existing array. The array is extended with the new value. If the existing value is not an array, it is autowrapped as an array, then extended with the new value.

Otherwise, a path-value pair for a nonexisting path in the json is ignored and has no effect.

Return Values

Returns a JSON value.

Examples

```
select json_insert(null, null, null);
```

```
+--------------------------------+
| json_insert(NULL, NULL, 'NULL') |
+--------------------------------+
| NULL                           |
+--------------------------------+
```

```
select json_insert('{"k": 1}', "$.k", 2);
```

```
+--------------------------------------+
| json_insert('{\"k\": 1}', '$.k', '2') |
+--------------------------------------+
| {"k":1}                              |
+--------------------------------------+
```

```
select json_insert('{"k": 1}', "$.j", 2);
```

```
+--------------------------------------+
| json_insert('{\"k\": 1}', '$.j', '2') |
+--------------------------------------+
| {"k":1,"j":2}                        |
+--------------------------------------+
```

```
select json_insert('{"k": 1}', "$.j", null);
```

```
+-----------------------------------------------+
| json_insert('{"k": 1}', '$.j', 'NULL', '660') |
+-----------------------------------------------+
| {"k":1,"j":null}                              |
+-----------------------------------------------+
```

JSON_LENGTH

Description

The JSON_LENGTH function returns the length or number of elements of a given JSON document. If the JSON document is an array, the number of elements in the array is returned; if the JSON document is an object, the number of key-value pairs in the object is returned. Returns NULL if the JSON document is empty or invalid.

Syntax

```
JSON_LENGTH(<json_str> [ , <json_path> ])
```

Required Parameters

parameters	described
<json_str>	The length of the JSON string needs to be checked.

Optional Parameters

parameters	described
<json_path>	If a path is specified, the JSON_LENGTH() function returns the length of the data that matches the path in the JSON document, otherwise it returns the length of the JSON document

Usage Notes

This function calculates the length of a JSON document based on the following rules: - The length of the scalar is 1. For example: ‘1’,‘“x”’,‘true’,’ false’, and ‘null’ are all of length 1. - The length of an array is the number of array elements. For example: ‘[1,2]’ has length 2. - The length of an object is the number of object members. For example: ‘{“x”: 1}’ has length 1

Return Value

- For a JSON array, returns the number of elements in the array.
- For JSON objects, returns the number of key-value pairs in the object.
- Returns NULL for invalid JSON strings.
- For other types (such as strings, numbers, booleans, null, etc.), NULL is returned.

Examples

```
SELECT json_length('{"k1":"v31","k2":300}');
```

```
+-------------------------------------+
| json_length('{"k1":"v31","k2":300}') |
+-------------------------------------+
| 2                                   |
+-------------------------------------+
```

```
SELECT json_length('"abc"');
```

```
+----------------------+
| json_length('"abc"') |
+----------------------+
| 1                    |
+----------------------+
```

```
SELECT json_length('{"x": 1, "y": [1, 2]}', '$.y');
```

```
+-------------------------------------------+
| json_length('{"x": 1, "y": [1, 2]}', '$.y') |
+-------------------------------------------+
| 2                                         |
+-------------------------------------------+
```

JSON_OBJECT

Description

Generate a json object containing the specified Key-Value, an exception error is returned when Key is NULL or the number of parameters are odd.

Syntax

```
JSON_OBJECT (<key>, <value>[,<key>, <value>, ...])
```

Parameters

Parameter	Description
<key>	The Key value in the Key-Value of the generated json object.
<value>	The Value value in the Key-Value of the generated json object.

Usage Notes

- By convention, the argument list consists of alternating keys and values.
- Key arguments are coerced to text.
- Value arguments are converted as per can convert to json, now we support array/struct/map/json as value.

Return Values

Return a json object. Special cases are as follows: * If no parameters are passed, return an empty json object. * If the number of parameters passed is odd, return an exception error. * If the passed Key is NULL, return an exception error. * If the passed Value is NULL, the Value value of the Key-Value pair in the returned json object is NULL.

Examples

```
select json_object();
```

```
+---------------+
| json_object() |
+---------------+
| {}            |
+---------------+
```

```
select json_object('time',curtime());
```

```
+--------------------------------+
| json_object('time', curtime()) |
+--------------------------------+
| {"time": "10:49:18"}           |
+--------------------------------+
```

```
SELECT json_object('id', 87, 'name', 'carrot');
```

```
+---------------------------------------+
| json_object('id', 87, 'name', 'carrot') |
+---------------------------------------+
| {"id": 87, "name": "carrot"}            |
+---------------------------------------+
```

```
select json_object('username',null);
```

```
+---------------------------------+
| json_object('username', 'NULL') |
+---------------------------------+
| {"username": NULL}              |
+---------------------------------+
```

```
select json_object(null,null);
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = json_object key can't be NULL: json_object(NULL)
```

```
-- support array as object value
SELECT json_object('id', 1, 'level', array('"aaa"','"bbb"'));
```

```
+-----------------------------------------------------------------------------------------------------+
| json_object('id', cast(1 as VARCHAR(65533)), 'level', cast(array('"aaa"', '"bbb"') as JSON), '6267') |
+-----------------------------------------------------------------------------------------------------+
| {"id":1,"level":["\"aaa\"","\"bbb\""]}                                                              |
+-----------------------------------------------------------------------------------------------------+
```

```
-- support map as object value
SELECT json_object('id', 1, 'level', map('a', 'b', 'c', 'd'));
```

```
+-----------------------------------------------------------------------------------------------+
| json_object('id', cast(1 as VARCHAR(65533)), 'level', cast(map('a', 'b', 'c', 'd') as JSON), '6267') |
+-----------------------------------------------------------------------------------------------+
| {"id":1,"level":{"a":"b","c":"d"}}                                                            |
+-----------------------------------------------------------------------------------------------+
```

```
-- support struct as object value
SELECT json_object('id', 1, 'level', named_struct('name', 'a', 'age', 1));
```

```
+-------------------------------------------------------------------------------------------------------+
| json_object('id', cast(1 as VARCHAR(65533)), 'level', cast(named_struct('name', 'a', 'age', 1) as JSON), '6267') |
```

```
+-----------------------------------------------------------------------------------------------------------------+
| {"id":1,"level":{"name":"a","age":1}}                                                                           |
+-----------------------------------------------------------------------------------------------------------------+
```

```
-- support json as object value
SELECT json_object('id', 1, 'level', cast('{\"a\":\"b\"}' as JSON));
```

```
+-----------------------------------------------------------------------------------------+
| json_object('id', cast(1 as VARCHAR(65533)), 'level', cast('{"a":"b"}' as JSON), '6267') |
+-----------------------------------------------------------------------------------------+
| {"id":1,"level":{"a":"b"}}                                                              |
+-----------------------------------------------------------------------------------------+
```

JSON_PARSE

Description

Parse the original JSON string into JSON binary format. To meet the needs of different abnormal data processing, different JSON_PARSE series functions are provided as follows: * JSON_PARSE: Parse the JSON string, and report an error when the input string is not a valid JSON string. * JSON_PARSE_ERROR_TO_INVALID: Parse the JSON string, and return NULL when the input string is not a valid JSON string. * JSON_PARSE_ERROR_TO_NULL: Parse the JSON string, and return NULL when the input string is not a valid JSON string. * JSON_PARSE_ERROR_TO_VALUE: Parse the JSON string, and return the default value specified by the parameter default_json_str when the input string is not a valid JSON string. * JSON_PARSE_NOTNULL: Parse the JSON string, and return NULL when the input string is not a valid JSON string.

Alias

- JSONB_PARSE is the same as JSON_PARSE
- JSONB_PARSE_ERROR_TO_INVALID is the same as JSON_PARSE_ERROR_TO_INVALID
- JSONB_PARSE_ERROR_TO_NULL is the same as JSON_PARSE_ERROR_TO_NULL
- JSONB_PARSE_ERROR_TO_VALUE is the same as JSON_PARSE_ERROR_TO_VALUE
- JSONB_PARSE_NOTNULL is the same as JSON_PARSE_NOTNULL

Syntax

```
JSON_PARSE (<json_str>)
```

```
JSON_PARSE_ERROR_TO_INVALID (<json_str>)
```

```
JSON_PARSE_ERROR_TO_NULL (<json_str>)
```

```
JSON_PARSE_ERROR_TO_VALUE (<json_str>, <default_json_str>)
```

```
JSONB_PARSE_NOTNULL (<json_str>)
```

Parameters

Parameter	Description
<json_str>	The JSON type parameter or field to be extracted.
<default_json_str>	When the input string is not a valid JSON string, return the default value specified by the parameter default_json_str.

Return Values

json_parse functions parse JSON string to binary format. A series of functions are provided to satisfy different demand for exception handling. - all return NULL if json_str is NULL - if json_str is not valid - json_parse will report error - json_parse_error_to_invalid will return NULL - json_parse_error_to_null will return NULL - json_parse_error_to_value will return the value specified by default_json_str - json_parse_notnull will return NULL

Examples

1. Parse valid JSON string

```
SELECT json_parse('{"k1":"v31","k2":300}');
```

```
+-------------------------------------+
| json_parse('{"k1":"v31","k2":300}') |
+-------------------------------------+
| {"k1":"v31","k2":300}               |
+-------------------------------------+
```

```
SELECT json_parse_error_to_invalid('{"k1":"v31","k2":300}');
```

```
+-----------------------------------------------------+
| jsonb_parse_error_to_invalid('{"k1":"v31","k2":300}') |
+-----------------------------------------------------+
| {"k1":"v31","k2":300}                               |
+-----------------------------------------------------+
```

```
SELECT json_parse_notnull('{"a":"b"}');
```

```
+---------------------------------+
| jsonb_parse_notnull('{"a":"b"}') |
+---------------------------------+
| {"a":"b"}                       |
+---------------------------------+
```

```
SELECT json_parse_error_to_value('{"k1":"v31","k2":300}','{}');
```

```
+-----------------------------------------------------------+
| jsonb_parse_error_to_value('{"k1":"v31","k2":300}', '{}') |
+-----------------------------------------------------------+
| {"k1":"v31","k2":300}                                     |
+-----------------------------------------------------------+
```

2. Parse invalid JSON string

```
SELECT json_parse('invalid json');
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = json parse error: Invalid document: document must be an object or an array for
value: invalid json
```

```
SELECT json_parse_error_to_invalid('invalid json');
```

```
+-----------------------------------------------+
| jsonb_parse_error_to_invalid('invalid json') |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+
```

```
SELECT json_parse_notnull('invalid json');
```

```
+----------------------------------------+
| jsonb_parse_error_to_null('invalid json') |
+----------------------------------------+
| NULL                                   |
+----------------------------------------+
```

```
SELECT json_parse_error_to_value('invalid json', '{}');
```

```
+--------------------------------------------------+
| json_parse_error_to_value('invalid json', '{}') |
+--------------------------------------------------+
| {}                                               |
+--------------------------------------------------+
```

JSON_PARSE_ERROR_TO_INVALID

Description

A variant of the `JSON_PARSE` function.

JSON_PARSE_ERROR_TO_NULL

Description

A variant of the `JSON_PARSE` function.

JSON_PARSE_ERROR_TO_VALUE

Description

A variant of the `JSON_PARSE` function.

JSON_PARSE_NOTNULL

Description

A variant of the `JSON_PARSE` function.

JSON_PARSE_NOTNULL_ERROR_TO_INVALID

Description

This function is used to parse a JSON string. If the JSON string is malformed or a parsing error occurs, the function will return an invalid JSON object (usually {}). The main purpose of this function is to ensure that when a JSON format error occurs, a safe default value is returned, preventing query failures due to parsing errors.

Aliases

- JSONB_PARSE_NOTNULL_ERROR_TO_INVALID

Syntax

```
JSON_PARSE_NOTNULL_ERROR_TO_INVALID( <str> )
```

Required Parameters

parameters	described
<str>	The JSON string to be parsed. This parameter should be a valid string containing JSON-formatted data. If the JSON format is invalid, the function will return an invalid JSON object.

Return Value

Returns an invalid JSON object (usually {}).

Examples

```

SELECT JSON_PARSE_NOTNULL_ERROR_TO_INVALID('{"name": "Alice", "age": 30}') AS parsed_json;
```

```
+---------------------------+
| parsed_json               |
+---------------------------+
| {"name":"Alice","age":30} |
+---------------------------+
```

JSON_PARSE_NOTNULL_ERROR_TO_VALUE

Description

Function to parse JSON strings. If the JSON string format is invalid or a parsing error occurs, the function returns the default value specified by the user instead of returning an invalid JSON object. The main purpose of this function is to provide a default value that can be used to replace invalid results in case of parsing errors, ensuring that the query returns a reasonable value.

Syntax

```
JSON_PARSE_NOTNULL_ERROR_TO_VALUE(< str >, <default_value>)
```

Alias

- JSONB_PARSE_NOTNULL_ERROR_TO_VALUE

Required Parameters

parameters	described
<str>	The JSON string to parse. This parameter should be a valid JSON string. If the JSON format is invalid, the function returns default_value.
<default_value>	The default value returned when parsing the error. This parameter can be of any type and is used to replace invalid JSON-formatted data.

Return Value

Return a JSON object. If the input JSON string is valid, the parsed JSON object is returned. If invalid, return the user-specified default_value.

Examples

```
SELECT JSON_PARSE_NOTNULL_ERROR_TO_VALUE('{"name": "Alice", "age": 30}', '{"name": "Unknown", "age":  0}') AS parsed_json;
```

```
+-----------------------------------------+
| parsed_json                             |
+-----------------------------------------+
| {"name":"Alice","age":30}               |
+-----------------------------------------+
```

JSON_PARSE_NULLABLE

Description

The JSON_PARSE_NULLABLE function is used to parse a JSON string into a valid JSON object. If the input string is invalid or NULL, it returns NULL without throwing an error.

Syntax

```
JSON_PARSE_NULLABLE( <str> )
```

Alias

- JSONB_PARSE_NULLABLE

Required Parameters

Parameter	Description
<str>	The input string in JSON format to be parsed.

Return Value

- If the input string is valid JSON, it returns the corresponding JSON object.
- If the input string is invalid or NULL, it returns NULL.

Examples

1.Valid JSON string:

```
SELECT JSON_PARSE_NULLABLE('{"name": "John", "age": 30}');
```

```
+-----------------------------------------------------+
| JSON_PARSE_NULLABLE('{"name": "John", "age": 30}')  |
+-----------------------------------------------------+
| {"name": "John", "age": 30}                         |
+-----------------------------------------------------+
```

2.Invalid JSON string:

```
SELECT JSON_PARSE_NULLABLE('{"name": "John", "age": }');
```

```
+-----------------------------------------------------+
| JSON_PARSE_NULLABLE('{"name": "John", "age": }')    |
+-----------------------------------------------------+
```

```
| NULL                                                |
+-----------------------------------------------------+
```

JSON_PARSE_NULLABLE_ERROR_TO_INVALID

Description

The `JSON_PARSE_NULLABLE_ERROR_TO_INVALID` function is used to parse a JSON string into a valid JSON object. If the input JSON string is invalid, it will return an "invalid JSON" marker (typically `INVALID_JSON`), without throwing an error. If the input is `NULL`, it will also return the `INVALID_JSON` marker.

Syntax

```
JSON_PARSE_NULLABLE_ERROR_TO_INVALID( <str> )
```

Alias

- JSONB_PARSE_NULLABLE_ERROR_TO_INVALID

Required Parameters

Parameter	Description
`<str>`	The input string in JSON format to be parsed.

Return Value

Condition	Return Value
If the input string is a valid JSON	It returns the corresponding JSON object.
If the input string is invalid or NULL	It returns the `INVALID_JSON` marker.

Examples

1. Valid JSON string:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_INVALID('{"name": "John", "age": 30}');
```

```
+-----------------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_INVALID('{"name": "John", "age": 30}')  |
+-----------------------------------------------------------------+
| {"name": "John", "age": 30}                                     |
+-----------------------------------------------------------------+
```

2. Invalid JSON string:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_INVALID('{"name": "John", "age": }');
```

```
+-------------------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_INVALID('{"name": "John", "age": }') |
+-------------------------------------------------------------------+
| INVALID_JSON                                                      |
+-------------------------------------------------------------------+
```

3. Input is NULL:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_INVALID(NULL);
```

```
+------------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_INVALID(NULL)                 |
+------------------------------------------------------------+
| INVALID_JSON                                               |
+------------------------------------------------------------+
```

JSON_PARSE_NULLABLE_ERROR_TO_NULL

Description

The `JSON_PARSE_NULLABLE_ERROR_TO_NULL` function is used to parse a JSON string into a valid JSON object. If the input JSON string is invalid, it will return `NULL` without throwing an error. If the input is `NULL`, it will directly return `NULL`.

Syntax

```
JSON_PARSE_NULLABLE_ERROR_TO_NULL( <str> )
```

Aliases

- JSONB_PARSE_NULLABLE_ERROR_TO_NULL

Required Parameters

Parameter	Description
`<str>`	The input string in JSON format to be parsed.

Return Value

If the input string is a valid JSON, it returns the corresponding JSON object. If the input string is invalid or NULL, it returns NULL.

Examples

1. Valid JSON string:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_NULL('{"name": "John", "age": 30}');
```

```
+-----------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_NULL('{"name": "John", "age": 30}') |
+-----------------------------------------------------------+
| {"name": "John", "age": 30}                               |
+-----------------------------------------------------------+
```

2. Invalid JSON string:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_NULL('{"name": "John", "age": }');
```

```
+------------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_NULL('{"name": "John", "age": }') |
+------------------------------------------------------------+
| NULL                                                       |
+------------------------------------------------------------+
```

3. Input is NULL:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_NULL(NULL);
```

```
+------------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_NULL(NULL)                     |
+------------------------------------------------------------+
| NULL                                                       |
+------------------------------------------------------------+
```

JSON_PARSE_NULLABLE_ERROR_TO_VALUE

Description

The JSON_PARSE_NULLABLE_ERROR_TO_VALUE function is used to parse a JSON string into a valid JSON object. If the input JSON string is invalid, it will return the default value specified by the user, instead of throwing an error. If the input is NULL, it will return the default value.

Syntax

```
JSON_PARSE_NULLABLE_ERROR_TO_VALUE( <str> , <default_value>)
```

Aliases

- JSONB_PARSE_NULLABLE_ERROR_TO_VALUE

Required Parameters

parameters	described
<str>	The input string in JSON format to be parsed.
<default_value>	The default value returned when parsing fails.

Return Value

If the input string is a valid JSON, it returns the corresponding JSON object. If the input string is invalid or NULL, it returns the default value specified by the default_value parameter.

Examples

1. Valid JSON string:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_VALUE('{"name": "John", "age": 30}', 'default');
```

```
+-------------------------------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_VALUE('{"name": "John", "age": 30}', 'default') |
+-------------------------------------------------------------------------------+
| {"name": "John", "age": 30}                                                   |
+-------------------------------------------------------------------------------+
```

2. Invalid JSON string:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_VALUE('{"name": "John", "age": }', 'default');
```

```
+-------------------------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_VALUE('{"name": "John", "age": }', 'default') |
+-------------------------------------------------------------------------+
| default                                                                 |
+-------------------------------------------------------------------------+
```

3. Input is NULL:

```
SELECT JSON_PARSE_NULLABLE_ERROR_TO_VALUE(NULL, 'default');
```

```
+-----------------------------------------------------------+
| JSON_PARSE_NULLABLE_ERROR_TO_VALUE(NULL, 'default')       |
+-----------------------------------------------------------+
| default                                                   |
+-----------------------------------------------------------+
```

JSON_QUOTE

Description

Enclose json_value in double quotes (“), escape special characters contained.

Syntax

```
JSON_QUOTE (<a>)
```

Parameters

Parameter	Description
<a>	The value of the json_value to be enclosed.

Return Values

Return a json_value. Special cases are as follows: * If the passed parameter is NULL, return NULL.

Examples

```
SELECT json_quote('null'), json_quote('"null"');
```

```
+-------------------+----------------------+
| json_quote('null') | json_quote('"null"') |
+-------------------+----------------------+
| "null"             | "\"null\""           |
+-------------------+----------------------+
```

```
SELECT json_quote('[1, 2, 3]');
```

```
+-------------------------+
| json_quote('[1, 2, 3]') |
+-------------------------+
| "[1, 2, 3]"             |
+-------------------------+
```

```
SELECT json_quote(null);
```

```
+-----------------+
| json_quote(null) |
+-----------------+
| NULL             |
+-----------------+
```

```
select json_quote("\n\b\r\t");
```

```
+------------------------+
| json_quote('\n\b\r\t') |
+------------------------+
| "\n\b\r\t"             |
+------------------------+
```

JSON_REPLACE

Description

The JSON_REPLACE function is used to update data in a JSON and return the result.

Syntax

```
JSON_REPLACE (<json_str>, <path>, <val>[, <jsonPath>, <val>, ...])
```

Parameters

Parameter	Description
<json_str>	The JSON data to be replaced. It can be a JSON object with elements of any type, including NULL. If no elements are specified, an empty array is returned. If json_str is not a valid JSON, an error will be returned.
<path>	The JSON path to be replaced.
<val>	The value to replace the value corresponding to the JSON_PATH Key. If it is NULL, then a NULL value will be inserted at the corresponding position.

Return Values

`json_replace` function updates data in a JSON and returns the result.Returns NULL if `json_str` or `path` is NULL. Otherwise, an error occurs if the `json_str` argument is not a valid JSON or any path argument is not a valid path expression or contains a * wildcard.

The path-value pairs are evaluated left to right.

A path-value pair for an existing path in the json overwrites the existing json value with the new value.

Otherwise, a path-value pair for a nonexisting path in the json is ignored and has no effect.

Examples

```
select json_replace(null, null, null);
```

```
+---------------------------------+
| json_replace(NULL, NULL, 'NULL') |
+---------------------------------+
| NULL                            |
+---------------------------------+
```

```
select json_replace('{"k": 1}', "$.k", 2);
```

```
+--------------------------------------+
| json_replace('{\"k\": 1}', '$.k', '2') |
+--------------------------------------+
| {"k":2}                              |
+--------------------------------------+
```

```
select json_replace('{"k": 1}', "$.j", 2);
```

```
+--------------------------------------+
| json_replace('{\"k\": 1}', '$.j', '2') |
+--------------------------------------+
| {"k":1}                              |
+--------------------------------------+
```

```
select json_replace(null, null, 's');
```

```
+-------------------------------------+
| json_replace(NULL, NULL, 's', '006') |
+-------------------------------------+
| NULL                                |
+-------------------------------------+
```

JSON_SEARCH

Description

The `JSON_SEARCH` function is used to search for a specified value within a JSON document. If the value is found, it returns the path to the value. If the value is not found, it returns `NULL`. This function can recursively search through the JSON data structure.

Syntax

```
JSON_SEARCH(<str>, <one_or_all>, <search_value> [, <start_path> [, <escape_char>]])
```

Required Parameters

parameters	described
`<str>`	The JSON document to search (can be a JSON string or JSON object).
`<one_or_all>`	Specifies whether to find all matching values. It can be ‘one’ or ‘all’.
`<search_value>`	The value to search for.

Optional Parameters

parameters	described
`<start_path>`	Specifies the path to start searching from. If not provided, the search starts from the entire JSON document.
`<escape_char>`	Specifies the character to escape special characters in the path.

Return Value

- If a matching value is found, it returns a JSON path (as a string) pointing to the matched value.
- If no matching value is found, it returns NULL.

Usage Notes

The one_or_all parameter determines whether to find all matching values. ‘one’ returns the first matching path, while ‘all’ returns all matching paths. If no matching values are found, the function returns NULL. The start_path parameter can limit the search range, making the query more efficient.

Examples

1. Search for a value (one):

```
SELECT JSON_SEARCH('{"name": "John", "age": 30}', 'one', 'John');
```

```
+-----------------------------------------------+
| JSON_SEARCH('{"name": "John", "age": 30}', 'one', 'John') |
+-----------------------------------------------+
| $.name                                         |
+-----------------------------------------------+
```

2. Search for multiple matching values (all):

```
SELECT JSON_SEARCH('{"person": {"name": "John", "age": 30}, "name": "John"}', 'all', 'John');
```

```
+-------------------------------------------------------------+
| JSON_SEARCH('{"person": {"name": "John", "age": 30}, "name": "John"}', 'all', 'John') |
+-------------------------------------------------------------+
| $.name                                                      |
| $.person.name                                               |
+-------------------------------------------------------------+
```

3. No matching value found:

```
SELECT JSON_SEARCH('{"name": "John", "age": 30}', 'one', 'Alice');
```

```
+-----------------------------------------------+
| JSON_SEARCH('{"name": "John", "age": 30}', 'one', 'Alice') |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+
```

4. Specify the starting search path:

```
SELECT JSON_SEARCH('{"person": {"name": "John", "age": 30}}', 'one', 'John', '$.person');
```

```
+-------------------------------------------------------------+
| JSON_SEARCH('{"person": {"name": "John", "age": 30}}', 'one', 'John', '$.person') |
+-------------------------------------------------------------+
| $.name                                                       |
+-------------------------------------------------------------+
```

JSON_SET

Description

The json_set function inserts or updates data in a JSON and returns the result.

Syntax

```
JSON_SET (<json_str>, <path>, <val> [, <path>, <val>, ...])
```

Parameters

Parameter	Description
<jsonStr>	The JSON object to be inserted. It can be a JSON object with elements of any type, including NULL. If no elements are specified, an empty array is returned. If json_str is not a valid JSON or any path parameter is not a valid path expression or contains a * wildcard, an error is returned.
<jsonPath>	The JSON path to be inserted. If it is NULL, then return NULL.
<val>	The value to be inserted into the JSON. If it is NULL, then a NULL value will be inserted at the corresponding position.

Return Values

Returns a JSON value.

`json_set` function inserts or updates data in a JSON and returns the result.Returns NULL if `json_str` or `path` is NULL. Otherwise, an error occurs if the `json_str` argument is not a valid JSON or any path argument is not a valid path expression or contains a * wildcard.

The path-value pairs are evaluated left to right.

A path-value pair for an existing path in the json overwrites the existing json value with the new value. A path-value pair for a nonexisting path in the json adds the value to the json if the path identifies one of these types of values:

- A member not present in an existing object. The member is added to the object and associated with the new value.
- A position past the end of an existing array. The array is extended with the new value. If the existing value is not an array, it is autowrapped as an array, then extended with the new value.

Otherwise, a path-value pair for a nonexisting path in the json is ignored and has no effect.

Examples

```
select json_set(null, null, null);
```

```
+------------------------------+
| json_set(NULL, NULL, 'NULL') |
+------------------------------+
| NULL                         |
+------------------------------+
```

```
select json_set('{"k": 1}', "$.k", 2);
```

```
+-----------------------------------+
| json_set('{\"k\": 1}', '$.k', '2') |
+-----------------------------------+
| {"k":2}                           |
+-----------------------------------+
```

```
select json_set('{"k": 1}', "$.j", 2);
```

```
+-----------------------------------+
| json_set('{\"k\": 1}', '$.j', '2') |
+-----------------------------------+
| {"k":1,"j":2}                     |
+-----------------------------------+
```

JSON_TYPE

Description

Used to determine the type of the field specified by `json_path` in the JSONB data. If the field does not exist, it returns NULL. If the field exists, it returns one of the following types:

- object
- array
- null
- bool
- int
- bigint
- largeint
- double
- string

Syntax

```
JSON_TYPE( <json>, <json_path> )
```

Alias

- JSONB_TYPE

Required Parameters

Parameter	Description
`<json>`	The JSON string to check the type of.
`<json_path>`	JSON path, which specifies the location of the field in JSON. The path is usually given in $. At the beginning, use. to represent the hierarchical structure.

Return Value

Returns the type of the JSON string. Possible values include: - “NULL”: Indicates that the value in the JSON document is null. - “BOOLEAN”: Indicates that the value in the JSON document is of boolean type (true or false). - “NUMBER”: Indicates that the value in the JSON document is a number. - “STRING”: Indicates that the value in the JSON document is a string. - “OBJECT”: Indicates that the value in the JSON document is a JSON object. - “ARRAY”: Indicates that the value in the JSON document is a JSON array.

Usage Notes

JSON_TYPE returns the type of the outermost value in the JSON document. If the JSON document contains multiple different types of values, it will return the type of the outermost value. For invalid JSON strings, JSON_TYPE returns NULL. Refer to json tutorial

Examples

1. JSON is of string type:

```
SELECT JSON_TYPE('{"name": "John", "age": 30}', '$.name');
```

```
+-----------------------------------------------------------------+
| jsonb_type(cast('{"name": "John", "age": 30}' as JSON), '$.name') |
+-----------------------------------------------------------------+
| string                                                          |
+-----------------------------------------------------------------+
```

2. JSON is of number type:

```
SELECT JSON_TYPE('{"name": "John", "age": 30}', '$.age');
```

```
+-----------------------------------------------------------------+
| jsonb_type(cast('{"name": "John", "age": 30}' as JSON), '$.age') |
+-----------------------------------------------------------------+
| int                                                             |
+-----------------------------------------------------------------+
```

JSON_UNQUOTE

Description

This function unquotes a JSON value and returns the result as a utf8mb4 string. If the argument is NULL, it will return NULL.

Syntax

```
JSON_UNQUOTE (<a>)
```

Parameters

Parameters	Description
<a>	The element to be unquoted.

Return Values

Returns a utf8mb4 string. Special cases are as follows: * If the passed parameter is NULL, return NULL. * If the passed parameter is not a value with double quotes, the value itself will be returned. * If the passed parameter is not a string, it will be automatically converted to a string and then the value itself will be returned.

Escape sequences within a string as shown in the following table will be recognized. Backslashes will be ignored for all other escape sequences.

Escape Sequence	Character Represented by Sequence
"	A double quote (“) character
A backspace character	
A formfeed character	
	A newline (linefeed) character
A carriage return character	
A tab character	
\	A backslash () character
	UTF-8 bytes for Unicode value XXXX

Examples

```
SELECT json_unquote('"doris"');
```

```
+-------------------------+
| json_unquote('"doris"') |
+-------------------------+
| doris                   |
+-------------------------+
```

```
SELECT json_unquote('[1, 2, 3]');
```

```
+---------------------------+
| json_unquote('[1, 2, 3]') |
+---------------------------+
| [1, 2, 3]                 |
+---------------------------+
```

```
SELECT json_unquote(null);
```

```
+--------------------+
| json_unquote(NULL) |
+--------------------+
| NULL               |
+--------------------+
```

```
SELECT json_unquote('"\\ttest"');
```

```
+--------------------------+
| json_unquote('"\ttest"') |
+--------------------------+
|       test                   |
+--------------------------+
```

```
select json_unquote('"doris');
```

```
+------------------------+
| json_unquote('"doris') |
+------------------------+
| "doris                 |
+------------------------+
```

```
select json_unquote('doris');
```

```
+-----------------------+
| json_unquote('doris') |
+-----------------------+
| doris                 |
+-----------------------+
```

```
select json_unquote(1);
```

```
+----------------------------------------+
| json_unquote(cast(1 as VARCHAR(65533))) |
+----------------------------------------+
| 1                                      |
+----------------------------------------+
```

JSON_VALID

Description

The JSON_VALID function returns 0 or 1 to indicate whether the input is a valid JSON. If the input is NULL, it returns NULL.

Syntax

```
JSON_VALID( <str> )
```

Required Parameters | Parameter | Description | |——|——| | `<str>` | The input string in JSON format to be parsed. |

Alias

- JSONB_VALID

Examples

1. Valid JSON string

```
SELECT json_valid('{"k1":"v31","k2":300}');
+-------------------------------------+
| json_valid('{"k1":"v31","k2":300}') |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

2. Invalid JSON string

```
SELECT json_valid('invalid json');
+----------------------------+
| json_valid('invalid json') |
+----------------------------+
|                          0 |
+----------------------------+
```

3. NULL 参数

```
SELECT json_valid(NULL);
+-----------------+
| json_valid(NULL) |
+-----------------+
|            NULL |
+-----------------+
```

IPV4_CIDR_TO_RANGE

Description

Receive an IPv4 and an Int16 value containing CIDR. Returns a struct that contains two IPv4 fields representing the lower range (min) and higher range (max) of the subnet, respectively.

Syntax

```
IPV4_CIDR_TO_RANGE(<ip_v4>, <cidr>)
```

Parameters

Parameter	Description
<ip_v4>	An IPv4 address of type String
<cidr>	The cidr value

Return Value

Returns a struct that contains two IPv4 fields representing the lower range (min) and higher range (max) of the subnet, respectively. - If input is NULL, the function returns NULL.

Example

```
SELECT ipv4_cidr_to_range(ipv4_string_to_num('192.168.5.2'), 16) as re1, ipv4_cidr_to_range(to_ipv4('192.168.5.2'), 16) as re2,
ipv4_cidr_to_range(NULL, NULL) as re3;
```

```
+-----------------------------------------------+-----------------------------------------------+------+
| re1                                           | re2                                           | re3  |
+-----------------------------------------------+-----------------------------------------------+------+
| {"min":"192.168.0.0", "max":"192.168.255.255"} | {"min":"192.168.0.0", "max":"192.168.255.255"} | NULL |
+-----------------------------------------------+-----------------------------------------------+------+
```

IPV4_NUM_TO_STRING

Description

Takes a Int16, Int32, Int64 number. Interprets it as an IPv4 address in big endian. Returns a string containing the corresponding IPv4 address in the format A.B.C.D (dot-separated numbers in decimal form)

Alias

- INET_NTOA

Syntax

```
IPV4_NUM_TO_STRING(<ipv4_num>)
```

Parameters

Parameter	Description
<ipv4_num>	Int type converted from ipv4

Return Value

Returns a string containing the corresponding IPv4 address in the format A.B.C.D (dot-separated numbers in decimal form), special case: - Will return `NULL` if the input parameter is negative or larger than `4294967295`(num value of `'255.255.255.255'`)

Example

```
select ipv4_num_to_string(3232235521);
```

```
+--------------------------------+
| ipv4_num_to_string(3232235521) |
+--------------------------------+
| 192.168.0.1                    |
+--------------------------------+
```

```
select num,ipv4_num_to_string(num) from ipv4_bi;
```

```
+------------+--------------------------+
| num        | ipv4_num_to_string(`num`) |
+------------+--------------------------+
|         -1 | NULL                     |
|          0 | 0.0.0.0                  |
| 2130706433 | 127.0.0.1                |
```

```
| 4294967295 | 255.255.255.255           |
| 4294967296 | NULL                      |
+------------+---------------------------+
```

IPV4_STRING_TO_NUM

Description

Takes a string containing an IPv4 address in the format A.B.C.D (dot-separated numbers in decimal form). Returns a BIGINT number representing the corresponding IPv4 address in big endian.

Syntax

```
IPV4_STRING_TO_NUM(<ipv4_string>)
```

Parameters

Parameter	Description
<ipv4_string>	String type of ipv4, like 'A.B.C.D'

Return Value

Returns a BIGINT number representing the corresponding IPv4 address in big endian. - If the input string is not a valid IPv4 address or NULL, an error is returned

Example

```
select ipv4_string_to_num('192.168.0.1');
```

```
+-----------------------------------+
| ipv4_string_to_num('192.168.0.1') |
+-----------------------------------+
| 3232235521                        |
+-----------------------------------+
```

```
select ipv4_string_to_num('invalid');
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (172.17.0.2)[CANCELLED][INVALID_ARGUMENT][E33] Invalid IPv4 value
```

IPV4_STRING_TO_NUM_OR_DEFAULT

Description

Takes a string containing an IPv4 address in the format A.B.C.D (dot-separated numbers in decimal form). Returns a BIGINT number representing the corresponding IPv4 address in big endian.

Syntax

```
IPV4_STRING_TO_NUM_OR_DEFAULT(<ipv4_string>)
```

Parameters

Parameter	Description
<ipv4_string>	String type of ipv4, like 'A.B.C.D'

Return Value

Returns a BIGINT number representing the corresponding IPv4 address in big endian. - If the input string is not a valid IPv4 address or `NULL`, `0` is returned.

Example

```
select ipv4_string_to_num_or_default('192.168.0.1');
```

```
+-----------------------------------------+
| ipv4_string_to_num_or_default('192.168.0.1') |
+-----------------------------------------+
| 3232235521                              |
+-----------------------------------------+
```

```
create table ipv4_str(
    str varchar(20)
) properties('replication_num' = '1');
insert into ipv4_str values('0.0.0.0'),('127.0.0.1'),('255.255.255.255'),('invalid');
```

```
select str, ipv4_string_to_num_or_default(str) from ipv4_str;
```

```
+-----------------+-----------------------------------+
|str              | ipv4_string_to_num_or_default(str) |
+-----------------+-----------------------------------+
| 0.0.0.0         | 0                                 |
| 127.0.0.1       | 2130706433                        |
| 255.255.255.255 | 4294967295                        |
```

```
| invalid         | 0                                  |
+-----------------+------------------------------------+
```

IPV4_STRING_TO_NUM_OR_NULL

Description

Takes a string containing an IPv4 address in the format A.B.C.D (dot-separated numbers in decimal form). Returns a BIGINT number representing the corresponding IPv4 address in big endian.

Alias

- INET_ATON

Syntax

```
IPV4_STRING_TO_NUM_OR_NULL(<ipv4_string>)
```

Parameters

Parameter	Description
<ipv4_string>	String type of ipv4, like 'A.B.C.D'

Return Value

Returns a BIGINT number representing the corresponding IPv4 address in big endian. - If the input string is not a valid IPv4 address, NULL is returned.

Example

```
select ipv4_string_to_num_or_null('192.168.0.1');
```

```
+-------------------------------------------+
| ipv4_string_to_num_or_null('192.168.0.1') |
+-------------------------------------------+
| 3232235521                                |
+-------------------------------------------+
```

```
select str, ipv4_string_to_num_or_null(str) from ipv4_str;
```

```
+-----------------+---------------------------------+
|str              | ipv4_string_to_num_or_null(str) |
+-----------------+---------------------------------+
| 0.0.0.0         | 0                               |
| 127.0.0.1       | 2130706433                      |
| 255.255.255.255 | 4294967295                      |
```

```
| invalid         | NULL                           |
+-----------------+--------------------------------+
```

IPV6_CIDR_TO_RANGE

Description

Receive an IPv6 and an Int16 value containing CIDR. Returns a struct that contains two IPv6 fields representing the lower range (min) and higher range (max) of the subnet, respectively.

Syntax

```
IPV6_CIDR_TO_RANGE(ip_v6, cidr)
```

Parameters

Parameter	Description
<ip_v6>	An IPv6 address of type String
<cidr>	The cidr value

Return Value

Returns a struct that contains two IPv6 fields representing the lower range (min) and higher range (max) of the subnet, respectively. - If input is NULL, the function returns NULL.

Example

```
SELECT ipv6_cidr_to_range(to_ipv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32), ipv6_cidr_to_range(NULL, NULL);
```

```
+-------------------------------------------------------------------------+--------------------------------+
| ipv6_cidr_to_range(to_ipv6('2001:0db8:0000:85a3:0000:0000:ac1f:8001'), 32) | ipv6_cidr_to_range(NULL, NULL) |
+-------------------------------------------------------------------------+--------------------------------+
| {"min":"2001:db8::", "max":"2001:db8:ffff:ffff:ffff:ffff:ffff:ffff"}       | NULL                           |
+-------------------------------------------------------------------------+--------------------------------+
```

IPV6_NUM_TO_STRING

Description

Takes an IPv6 address in binary format of type String. Returns the string of this address in text format. - The IPv4 address mapped by IPv6 starts with ::ffff:111.222.33.44.

Alias

- INET6_NTOA

Syntax

```
IPV6_NUM_TO_STRING(<ipv6_num>)
```

Parameters

Parameter	Description
<ipv6_num>	An IPv6 address in binary format of type String

Return Value

Returns the string of the ipv6 address in text format. - If the input string is not the binary encoding of a valid IPv6 address, NULL is returned.

Example

```
select ipv6_num_to_string(unhex('2A0206B8000000000000000000000011')) as addr, ipv6_num_to_string("-23vno12i34nlfwlsj");
```

```
+--------------+-----------------------------------------+
| addr         | ipv6_num_to_string('-23vno12i34nlfwlsj') |
+--------------+-----------------------------------------+
| 2a02:6b8::11 | NULL                                    |
+--------------+-----------------------------------------+
```

IPV6_STRING_TO_NUM

Description

The reverse function of IPV6_NUM_TO_STRING, it takes an IP address String and returns an IPv6 address in binary format.

Syntax

```
IPV6_STRING_TO_NUM(<ipv6_string>)
```

Parameters

Parameter	Description
<ipv6_string>	An IPv6 address of type String

Return Value

Returns an IPv6 address in binary format. - Will return an error if the input string is not a valid IP address or `NULL` - If the input string contains a valid IPv4 address, returns its IPv6 equivalent.

Example

```
select hex(ipv6_string_to_num('1111::ffff')), hex(ipv6_string_to_num('192.168.0.1'));
```

```
+--------------------------------------+--------------------------------------+
| hex(ipv6_string_to_num('1111::ffff')) | hex(ipv6_string_to_num('192.168.0.1')) |
+--------------------------------------+--------------------------------------+
| 1111000000000000000000000000FFFF      | 00000000000000000000FFFFC0A80001       |
+--------------------------------------+--------------------------------------+
```

```
select hex(ipv6_string_to_num('notaaddress'));
```

```
ERROR 1105 (HY000): errCode = 2, detailMessage = (172.17.0.2)[CANCELLED][E33] Invalid IPv6 value
```

IPV6_STRING_TO_NUM_OR_DEFAULT

Description

The reverse function of IPV6_NUM_TO_STRING, it takes an IP address String and returns an IPv6 address in binary format.

Syntax

```
IPV6_STRING_TO_NUM_OR_DEFAULT(<ipv6_string>)
```

Parameters

Parameter	Description
<ipv6_string>	An IPv6 address of type String

Return Value

Returns an IPv6 address in binary format. - will return `0` if the input string is not a valid IP address - If the input string contains a valid IPv4 address, returns its IPv6 equivalent.

Example

```
select hex(ipv6_string_to_num_or_default('1111::ffff')) as r1,hex(ipv6_string_to_num_or_default('192.168.0.1')) as r2,
hex(ipv6_string_to_num_or_default('notaaddress')) as r3;
```

```
+----------------------------------+----------------------------------+----------------------------------+
| r1                               | r2                               | r3                               |
+----------------------------------+----------------------------------+----------------------------------+
| 1111000000000000000000000000FFFF | 00000000000000000000FFFFC0A80001 | 00000000000000000000000000000000 |
+----------------------------------+----------------------------------+----------------------------------+
```

IPV6_STRING_TO_NUM_OR_NULL

Description

The reverse function of IPV6_NUM_TO_STRING, it takes an IP address String and returns an IPv6 address in binary format.

Alias

- INET6_ATON

Syntax

```
IPV6_STRING_TO_NUM_OR_NULL(<ipv6_string>)
```

Parameters

Parameter	Description
<ipv6_string>	An IPv6 address of type String

Return Value

Returns an IPv6 address in binary format. - If an illegal IP address is entered, NULL is returned. - If the input string contains a valid IPv4 address, returns its IPv6 equivalent.

Example

```
select hex(ipv6_string_to_num_or_null('1111::ffff')) as r1, hex(ipv6_string_to_num_or_null('192.168.0.1')) as r2,
hex(ipv6_string_to_num_or_null('notaaddress')) as r3;
```

```
+----------------------------------+----------------------------------+------+
| r1                               | r2                               | r3   |
+----------------------------------+----------------------------------+------+
| 11110000000000000000000000000FFFF | 00000000000000000000FFFFC0A80001 | NULL |
+----------------------------------+----------------------------------+------+
```

IS_IP_ADDRESS_IN_RANGE

Description

Determine whether the IP (IPv4 or IPv6) address is included in the network represented by CIDR notation.

Syntax

```
IS_IP_ADDRESS_IN_RANGE(ip_str, cidr_prefix)
```

Parameters

Parameter	Description
<ip_str>	An IPv4 or IPv6 address of type String
<cidr_prefix>	The cidr prefix

Return Value

If the address is included in the network represented by CIDR notation, returns true; otherwise, return false. - If input is NULL, the function returns NULL.

Example

```
SELECT is_ip_address_in_range('127.0.0.1', '127.0.0.0/8') as v4, is_ip_address_in_range('::ffff:192.168.0.1',
'::ffff:192.168.0.4/128') as v6, is_ip_address_in_range('127.0.0.1', NULL) as nil;
```

```
+------+------+------+
| v4   | v6   | nil  |
+------+------+------+
|    1 |    0 | NULL |
+------+------+------+
```

IS_IPV4_COMPAT

Description

This function takes an IPv6 address represented in numeric form as a binary string, as returned by INET6_ATON().INET6_ATON is also named IPV6_STRING_TO_NUM_OR_NULL. - IPv4-compatible addresses have the form `::ipv4_address`

Syntax

```
IS_IPV4_COMPAT(INET6_ATON(<ipv4_addr>))
```

Parameters

Parameter	Description
`<ipv4_addr>`	An IPv4-compatible addresses, it is like '::ipv4_address'

Return Value

Returns 1 if the argument is a valid IPv4-compatible IPv6 address, 0 otherwise. - If input is NULL, the function returns NULL.

Example

```
SELECT IS_IPV4_COMPAT(INET6_ATON('::ffff:10.0.5.9')) AS re1, IS_IPV4_COMPAT(INET6_ATON('::10.0.5.9')) AS re2;
```

```
+------+------+
| re1  | re2  |
+------+------+
|    0 |    1 |
+------+------+
```

IS_IPV4_MAPPED

Description

This function takes an IPv6 address represented in numeric form as a binary string, as returned by INET6_ATON().INET6_ATON is also named IPV6_STRING_TO_NUM_OR_NULL. - IPv4-mapped addresses have the form `::ffff:ipv4_address`.

Syntax

```
IS_IPV4_MAPPED(INET6_ATON(<ipv4_addr>))
```

Parameters

Parameter	Description
`<ipv4_addr>`	An IPv4-compatible addresses, it is like '::ipv4_address'

Return Value

Returns 1 if the argument is a valid IPv4-mapped IPv6 address, 0 otherwise. - If input is NULL, the function returns NULL.

Example

```
SELECT IS_IPV4_MAPPED(INET6_ATON('::ffff:10.0.5.9')) AS re1, IS_IPV4_MAPPED(INET6_ATON('::10.0.5.9')) AS re2;
```

```
+------+------+
| re1  | re2  |
+------+------+
|    1 |    0 |
+------+------+
```

IS_IPV4_STRING

Description

Determine whether an IPv4-type address is a valid IPv4 address.

Syntax

```
IS_IPV4_STRING(<ipv4_str>)
```

Parameters

Parameter	Description
<ipv4_str>	An IPv4 address of type String

Return Value

If it is a correctly formatted and valid IPv4 address, return true; On the contrary, return false. - If input is NULL, the function returns NULL.

Example

```
CREATE TABLE `test_is_ipv4_string` (
      `id` int,
      `ip_v4` string
    ) ENGINE=OLAP
    DISTRIBUTED BY HASH(`id`) BUCKETS 4
    PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
    );

insert into test_is_ipv4_string values(0, NULL), (1, '0.0.0.'), (2, ''), (3, '.'), (4, '255.255.255.255');

select id, ip_v4, is_ipv4_string(ip_v4) from test_is_ipv4_string order by id;
```

```
+------+-----------------+-----------------------+
| id   | ip_v4           | is_ipv4_string(ip_v4) |
+------+-----------------+-----------------------+
|    0 | NULL            |                  NULL |
|    1 | 0.0.0.          |                     0 |
|    2 |                 |                     0 |
|    3 | .               |                     0 |
|    4 | 255.255.255.255 |                     1 |
+------+-----------------+-----------------------+
```

IS_IPV6_STRING

Description

Determine whether an IPv6-type address is a valid IPv6 address.

Syntax

```
IS_IPV6_STRING(<ipv6_str>)
```

Parameters

Parameter	Description
<ipv6_str>	An IPv6 address of type String

Return Value

If it is a correctly formatted and valid IPv6 address, return true; On the contrary, return false. - If input is NULL, the function returns NULL.

Example

```
CREATE TABLE `test_is_ipv6_string` (
      `id` int,
      `ip_v6` string
    ) ENGINE=OLAP
    DISTRIBUTED BY HASH(`id`) BUCKETS 4
    PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
    );

insert into test_is_ipv6_string values(0, NULL), (1, '::'), (2, ''), (3, '2001:1b70:a1:610::b102:2'), (4,
'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff');

select id, ip_v6, is_ipv6_string(ip_v6) from test_is_ipv6_string order by id;
```

```
+------+-----------------------------------------+----------------------+
| id   | ip_v6                                   | is_ipv6_string(ip_v6) |
+------+-----------------------------------------+----------------------+
|    0 | NULL                                    |                 NULL |
|    1 | ::                                      |                    1 |
|    2 |                                         |                    0 |
|    3 | 2001:1b70:a1:610::b102:2                |                    1 |
|    4 | ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff |                    0 |
+------+-----------------------------------------+----------------------+
```

TO_IPV4

Description

This function like ipv4_string_to_num that takes a string form of IPv4 address and returns value of IPv4 type, which is binary equal to value returned by ipv4_string_to_num.

Syntax

```
TO_IPV4(<ipv4_str>)
```

Parameters

Parameter	Description
<ipv4_str>	An IPv4 address of type String

Return Value

Returns value of IPv4 type, which is binary equal to value returned by ipv4_string_to_num. - If the IPv4 address has an invalid format, throw an exception

Example

```
SELECT to_ipv4('255.255.255.255');
```

```
+---------------------------+
| to_ipv4('255.255.255.255') |
+---------------------------+
| 255.255.255.255           |
+---------------------------+
```

TO_IPV4_OR_DEFAULT

Description

This function like ipv4_string_to_num that takes a string form of IPv4 address and returns value of IPv4 type, which is binary equal to value returned by ipv4_string_to_num.

Syntax

```
TO_IPV4_OR_DEFAULT(<ipv4_str>)
```

Parameters

Parameter	Description
<ipv4_str>	An IPv4 address of type String

Return Value

Returns value of IPv4 type. - If the IPv4 address has an invalid format, returns 0.0.0.0 (0 as IPv4).

Example

```
SELECT to_ipv4_or_default('255.255.255.255'), to_ipv4_or_default('.'), to_ipv4_or_default(NULL);
```

```
+--------------------------------------+------------------------+-------------------------+
| to_ipv4_or_default('255.255.255.255') | to_ipv4_or_default('.') | to_ipv4_or_default(NULL) |
+--------------------------------------+------------------------+-------------------------+
| 255.255.255.255                      | 0.0.0.0                | 0.0.0.0                 |
+--------------------------------------+------------------------+-------------------------+
```

TO_IPV4_OR_NULL

Description

This function like ipv4_string_to_num that takes a string form of IPv4 address and returns value of IPv4 type, which is binary equal to value returned by ipv4_string_to_num.

Syntax

```
TO_IPV4_OR_NULL(<ipv4_str>)
```

Parameters

Parameter	Description
<ipv4_str>	An IPv4 address of type String

Return Value

Returns value of IPv4 type. - If the IPv4 address has an invalid format, returns NULL.

Example

```
SELECT to_ipv4_or_null('255.255.255.255'), to_ipv4_or_null('.'), to_ipv4_or_null(NULL);
```

```
+-----------------------------------+---------------------+----------------------+
| to_ipv4_or_null('255.255.255.255') | to_ipv4_or_null('.') | to_ipv4_or_null(NULL) |
+-----------------------------------+---------------------+----------------------+
| 255.255.255.255                   | NULL                | NULL                 |
+-----------------------------------+---------------------+----------------------+
```

TO_IPV6

Description

Convert a string form of IPv6 address to IPv6 type, which is binary equal to value returned by ipv6_string_to_num.

Syntax

```
TO_IPV6(<ipv6_str>)
```

Parameters

Parameter	Description
<ipv6_str>	An IPv6 address of type String

Return Value

Returns value of IPv6 type. - If the IPv6 address has an invalid format, throw an exception

Example

```
SELECT to_ipv6('::'),to_ipv6('2001:1b70:a1:610::b102:2');
```

```
+--------------+-----------------------------------+
| to_ipv6('::') | to_ipv6('2001:1b70:a1:610::b102:2') |
+--------------+-----------------------------------+
| ::           | 2001:1b70:a1:610::b102:2          |
+--------------+-----------------------------------+
```

TO_IPV6_OR_DEFAULT

Description

Convert a string form of IPv6 address to IPv6 type.

Syntax

```
TO_IPV6_OR_DEFAULT(<ipv6_str>)
```

Parameters

Parameter	Description
<ipv6_str>	An IPv6 address of type String

Return Value

Returns value of IPv6 type. - If the IPv6 address has an invalid format, returns :: (0 as IPv6).

Example

```
SELECT to_ipv6_or_default('.'), to_ipv6_or_default('2001:1b70:a1:610::b102:2');
```

```
+-------------------------+-----------------------------------------------+
| to_ipv6_or_default('.') | to_ipv6_or_default('2001:1b70:a1:610::b102:2') |
+-------------------------+-----------------------------------------------+
| ::                      | 2001:1b70:a1:610::b102:2                      |
+-------------------------+-----------------------------------------------+
```

TO_IPV6_OR_NULL

Description

Convert a string form of IPv6 address to IPv6 type.

Syntax

```
TO_IPV6_OR_NULL(<ipv6_str>)
```

Parameters

Parameter	Description
<ipv6_str>	An IPv6 address of type String

Return Value

Returns value of IPv6 type. - If the IPv6 address has an invalid format, returns NULL.

Example

```
SELECT to_ipv6_or_null('.'), to_ipv6_or_null('2001:1b70:a1:610::b102:2');
```

```
+----------------------+-------------------------------------------+
| to_ipv6_or_null('.') | to_ipv6_or_null('2001:1b70:a1:610::b102:2') |
+----------------------+-------------------------------------------+
| NULL                 | 2001:1b70:a1:610::b102:2                  |
+----------------------+-------------------------------------------+
```

BITMAP_AND

Description

Computes the intersection of two or more input BITMAPs and returns a new BITMAP.

Syntax

```
BITMAP_AND(<bitmap>, <bitmap>,[, <bitmap>...])
```

Parameters

Parameter	Description
<bitmap>	One of the original BITMAPs whose intersection is being sought

Return Value

Returns a BITMAP - If the parameter has a null value, it returns NULL

Examples

```
select bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5')));
```

```
+-----------------------------------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'))) |
+-----------------------------------------------------------------------------------------------------------------+
| 1,2                                                                                                             |
+-----------------------------------------------------------------------------------------------------------------+
```

```
select bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'),
bitmap_from_string('1,2,3,4,5'),bitmap_empty()));
```

```
+-----------------------------------------------------------------------------------------------------------------------------------------
----+
| bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'),
bitmap_empty())) |
+-----------------------------------------------------------------------------------------------------------------------------------------
----+
|
|
+-----------------------------------------------------------------------------------------------------------------------------------------
----+
```

```
select bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'),NULL));
```

```
+-----------------------------------------------------------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_and(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'), NULL)) |
+-----------------------------------------------------------------------------------------------------------------------------------------+
| NULL                                                                                                                                    |
+-----------------------------------------------------------------------------------------------------------------------------------------+
```

BITMAP_AND_COUNT

Description

Computes the intersection of two or more input BITMAPs and returns the number of intersections.

Syntax

```
BITMAP_AND_COUNT(<bitmap>, <bitmap>,[, <bitmap>...])
```

Parameters

Parameter	Description
<bitmap>	One of the original BITMAPs whose intersection is being sought

Return Value

Returns an integer - If the parameter has a null value, it returns NULL

Examples

```
select bitmap_and_count(bitmap_from_string('1,2,3'),bitmap_from_string('3,4,5'));
```

```
+-----------------------------------------------------------------------+
| bitmap_and_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5')) |
+-----------------------------------------------------------------------+
|                                                                     1 |
+-----------------------------------------------------------------------+
```

```
select bitmap_and_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'));
```

```
+-------------------------------------------------------------------------------------------------------------+
| (bitmap_and_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'))) |
+-------------------------------------------------------------------------------------------------------------+
|                                                                                                           2 |
+-------------------------------------------------------------------------------------------------------------+
```

```
select bitmap_and_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'),bitmap_empty());
```

```
+-----------------------------------------------------------------------------------------------------------------------------+
| (bitmap_and_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'), bitmap_empty())) |
+-----------------------------------------------------------------------------------------------------------------------------+
|                                                                                                                           0 |
+-----------------------------------------------------------------------------------------------------------------------------+
```

```
select bitmap_and_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'), NULL);
```

```
+-----------------------------------------------------------------------------------------------------------------+
| (bitmap_and_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'), NULL)) |
+-----------------------------------------------------------------------------------------------------------------+
|                                                                                                            NULL |
+-----------------------------------------------------------------------------------------------------------------+
```

BITMAP_AND_NOT,BITMAP_ANDNOT

Description

Perform a NOT operation on two BITMAPs and return the result. The first input parameter is called `base BITMAP` and the second is called `exclude BITMAP`.

Alias

- BITMAP_ANDNOT

Syntax

```
BITMAP_AND_NOT(<bitmap1>, <bitmap2>)
```

Parameters

Parameter	Description
<bitmap1>	`Base BITMAP` to be negated
<bitmap2>	`Exclusion BITMAP` to be negated

Return Value

Returns a BITMAP. - If the parameter has a null value, returns NULL

Examples

```
select bitmap_count(bitmap_and_not(bitmap_from_string('1,2,3'),bitmap_from_string('3,4,5'))) cnt;
```

```
+------+
| cnt  |
+------+
|    2 |
+------+
```

```
select bitmap_to_string(bitmap_and_not(bitmap_from_string('1,2,3'),bitmap_from_string('3,4,5')));
```

```
+-------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_and_not(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5'))) |
+-------------------------------------------------------------------------------------+
| 1,2                                                                                 |
+-------------------------------------------------------------------------------------+
```

```
select bitmap_to_string(bitmap_and_not(bitmap_from_string('1,2,3'),bitmap_empty()));
```

```
+-----------------------------------------------------------------------------+
| bitmap_to_string(bitmap_and_not(bitmap_from_string('1,2,3'), bitmap_empty())) |
+-----------------------------------------------------------------------------+
| 1,2,3                                                                       |
+-----------------------------------------------------------------------------+
```

```
select bitmap_to_string(bitmap_and_not(bitmap_from_string('1,2,3'),NULL));
```

```
+--------------------------------------------------------------------+
| bitmap_to_string(bitmap_and_not(bitmap_from_string('1,2,3'), NULL)) |
+--------------------------------------------------------------------+
| NULL                                                               |
+--------------------------------------------------------------------+
```

BITMAP_AND_NOT_COUNT,BITMAP_ANDNOT_COUNT

Description

Performs a NOT operation on two BITMAPs and returns the number of elements in the result set. The first input parameter is called `base BITMAP` and the second is called `exclusion BITMAP`.

Alias

- BITMAP_ANDNOT_COUNT

Syntax

```
BITMAP_AND_NOT_COUNT(<bitmap1>, <bitmap2>)
```

Parameters

Parameter	Description
`<bitmap1>`	`Base BITMAP` to be negated
`<bitmap2>`	`Exclusion BITMAP` to be negated

Return Value

Returns an integer. - If the parameter has a null value, returns NULL

Examples

```
select bitmap_and_not_count(null, bitmap_from_string('1,2,3')) banc1, bitmap_and_not_count(bitmap_from_string('1,2,3') ,null) banc2;
```

```
+-------+-------+
| banc1 | banc2 |
+-------+-------+
|     0 |     0 |
+-------+-------+
```

```
select bitmap_and_not_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5')) banc;
```

```
+------+
| banc |
+------+
```

```
|    2 |
+------+
```

BITMAP_CONTAINS

Description

Calculates whether the input value is in the BITMAP and returns a boolean value.

Syntax

```
BITMAP_CONTAINS(<bitmap>, <bigint>)
```

Parameters

Parameter	Description
<bitmap>	BITMAP collection
<bitint>	The integer to be checked for existence

Return Value

Returns a boolean - If the parameter is empty, returns NULL

Examples

```
select bitmap_contains(to_bitmap(1),2) cnt1, bitmap_contains(to_bitmap(1),1) cnt2;
```

```
+------+------+
| cnt1 | cnt2 |
+------+------+
|    0 |    1 |
+------+------+
```

BITMAP_COUNT

Description

Count the number of elements in the input BITMAP

Syntax

```
BITMAP_COUNT(<bitmap>)
```

Parameters

Parameter	Description
<bitmap>	a BITMAP

Return Value

Returns an integer

Examples

```
select bitmap_count(to_bitmap(1)) cnt;
```

```
+------+
| cnt  |
+------+
|    1 |
+------+
```

BITMAP_EMPTY

Description

Construct an empty BITMAP. Mainly used to fill default values during insert or stream load. For example:

```
cat data | curl --location-trusted -u user:passwd -T - -H "columns: dt,page,v1,v2=bitmap_empty()"
http://127.0.0.1:8040/api/test_database/test_table/_stream_load
```

Syntax

```
BITMAP_EMPTY()
```

Return Value

Returns an empty array with no elements

Examples

```
select bitmap_to_string(bitmap_empty());
```

```
+----------------------------------+
| bitmap_to_string(bitmap_empty()) |
+----------------------------------+
|                                  |
+----------------------------------+
```

```
select bitmap_count(bitmap_empty());
```

```
+-----------------------------+
| bitmap_count(bitmap_empty()) |
+-----------------------------+
|                           0 |
+-----------------------------+
```

BITMAP_FROM_ARRAY

Description

Converts an array of TINYINT/SMALLINT/INT/BIGINT type to a BITMAP. When the input field is illegal, the result returns NULL.

Syntax

```
BITMAP_FROM_ARRAY(<arr>)
```

Parameters

Parameter	Description
<arr>	integer array

Return Value

Returns a BITMAP - When the input field is invalid, the result is NULL

Examples

```
SELECT bitmap_to_string(bitmap_from_array(array(1, 0, 1, 1, 0, 1, 0))) AS bs;
```

```
+------+
| bs   |
+------+
| 0,1  |
+------+
```

BITMAP_FROM_BASE64

Description

Convert a base64 string (which can be converted by `bitmap_to_base64` function) to a BITMAP. Returns NULL when the input string is invalid.

Syntax

```
BITMAP_FROM_BASE64(<base64_str>)
```

Parameters

Parameter	Description
<base64_str>	base64 string (can be converted by `bitmap_to_base64` function)

Return Value

Returns a BITMAP - When the input field is invalid, the result is NULL

Examples

```
select bitmap_to_string(bitmap_from_base64("AA==")) bts;
```

```
+------+
| bts  |
+------+
|      |
+------+
```

```
select bitmap_to_string(bitmap_from_base64("AQEAAAA=")) bts;
```

```
+------+
| bts  |
+------+
| 1    |
+------+
```

```
select bitmap_to_string(bitmap_from_base64("AjowAAACAAAAAAAAAJgAAAAYAAAAGgAAAAEAf5Y=")) bts;
```

```
+-----------+
| bts       |
+-----------+
```

```
| 1,9999999 |
+-----------+
```

BITMAP_FROM_STRING

Description

Convert a string into a BITMAP. The string consists of a group of unsigned bigint numbers separated by commas. (The number values are between: 0 ~ 18446744073709551615) For example, the string “0, 1, 2” will be converted into a Bitmap, where the 0th, 1st, and 2nd bits are set. When the input field is invalid, NULL is returned

Syntax

```
BITMAP_FROM_STRING(<str>)
```

Parameters

Parameter	Description
<str>	Array string, for example “0, 1, 2” string will be converted to a Bitmap with bits 0, 1, 2 set

Return Value

Returns a BITMAP - When the input field is invalid, the result is NULL

Examples

```
select bitmap_to_string(bitmap_from_string("0, 1, 2")) bts;
```

```
+-------+
| bts   |
+-------+
| 0,1,2 |
+-------+
```

```
select bitmap_from_string("-1, 0, 1, 2") bfs;
```

```
+------+
| bfs  |
+------+
| NULL |
+------+
```

```
select bitmap_to_string(bitmap_from_string("0, 1, 18446744073709551615")) bts;
```

```
+--------------------------+
| bts                      |
+--------------------------+
| 0,1,18446744073709551615 |
+--------------------------+
```

BITMAP_HAS_ALL

Description

Determines whether one Bitmap contains all the elements of another Bitmap.

Syntax

```
BITMAP_HAS_ALL(<bitmap1>, <bitmap2>)
```

Parameters

Parameter	Description
<bitmap1>	The first Bitmap
<bitmap2>	The second Bitmap

Return Value

Returns true if <bitmap1> contains all the elements of <bitmap2>;
Returns true if <bitmap2> contains no elements;
Otherwise, returns false.

Examples

To check if one Bitmap contains all elements of another Bitmap:

```
select bitmap_has_all(bitmap_from_string('0, 1, 2'), bitmap_from_string('1, 2'));
```

The result will be:

```
+-----------------------------------------------------------------------+
| bitmap_has_all(bitmap_from_string('0, 1, 2'), bitmap_from_string('1, 2')) |
+-----------------------------------------------------------------------+
|                                                                     1 |
+-----------------------------------------------------------------------+
```

To check if an empty Bitmap contains all elements of another Bitmap:

```
select bitmap_has_all(bitmap_empty(), bitmap_from_string('1, 2'));
```

The result will be:

```
+-----------------------------------------------------------+
| bitmap_has_all(bitmap_empty(), bitmap_from_string('1, 2')) |
```

```
+-----------------------------------------------------------+
|                                                         0 |
+-----------------------------------------------------------+
```

BITMAP_HAS_ANY

Description

Determines whether two Bitmaps share any common elements.

Syntax

```
BITMAP_HAS_ANY(<bitmap1>, <bitmap2>)
```

Parameters

Parameter	Description
<bitmap1>	The first Bitmap
<bitmap2>	The second Bitmap

Return Value

Returns true if the two Bitmaps have any common elements;
Returns false if the two Bitmaps do not have any common elements.

Examples

```
mysql> select bitmap_has_any(to_bitmap(1), to_bitmap(2));
```

```
+-------------------------------------------+
| bitmap_has_any(to_bitmap(1), to_bitmap(2)) |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
```

```
mysql> select bitmap_has_any(to_bitmap(1), to_bitmap(1));
```

```
+-------------------------------------------+
| bitmap_has_any(to_bitmap(1), to_bitmap(1)) |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
```

BITMAP_HASH

Description

Computes the 32-bit hash value of any input type and returns a Bitmap containing that hash value.

Syntax

```
BITMAP_HASH(<expr>)
```

Parameters

Parameter	Description
<expr>	Any value or field expression

Return Value

Returns a Bitmap containing the 32-bit hash value of the parameter <expr>.

The hash algorithm used is MurMur3.
MurMur3 is a high-performance, low-collision hashing algorithm that produces values close to a random distribution and can pass chi-squared distribution tests. Note that the hash values computed may differ across different hardware platforms and seed values.
For more details on the performance of this algorithm, see the Smhasher benchmark.

Examples

To compute the MurMur3 hash of a value, you can use:

```
select bitmap_to_array(bitmap_hash('hello'))[1];
```

The result will be:

```
+-----------------------------------------------------------+
| %element_extract%(bitmap_to_array(bitmap_hash('hello')), 1) |
+-----------------------------------------------------------+
|                                                1321743225 |
+-----------------------------------------------------------+
```

To count the distinct values in a column using bitmaps, which can be more efficient than `count distinct` in some scenarios. The example below populates a `words` table with 6 rows containing 4 distinct values; at the doc-corpus scale a real query of this shape can return numbers in the millions:

```
CREATE TABLE `words` (`word` VARCHAR(64))
DISTRIBUTED BY HASH(`word`) BUCKETS 1
PROPERTIES ("replication_num" = "1");

INSERT INTO `words` VALUES ('apple'), ('banana'), ('cherry'), ('apple'), ('date'), ('banana');

select bitmap_count(bitmap_union(bitmap_hash(`word`))) from `words`;
```

```
+-----------------------------------------------+
| bitmap_count(bitmap_union(bitmap_hash(`word`))) |
+-----------------------------------------------+
|                                             4 |
+-----------------------------------------------+
```

```
select bitmap_to_string(bitmap_hash(NULL)) AS res;
```

The result will be:

```
+------+
| res  |
+------+
|      |
+------+
```

BITMAP_HASH64

Description

Computes the 64-bit hash value of any input type and returns a Bitmap containing that hash value.

Syntax

```
BITMAP_HASH64(<expr>)
```

Parameters

Parameter	Description
<expr>	Any value or field expression

Return Value

Returns a Bitmap containing the 64-bit hash value of the parameter <expr>.

Examples

To compute the 64-bit hash of a value, you can use:

```
select bitmap_to_string(bitmap_hash64('hello'));
```

The result will be:

```
+----------------------------------------+
| bitmap_to_string(bitmap_hash64('hello')) |
+----------------------------------------+
| 15231136565543391023                   |
+----------------------------------------+
```

BITMAP_MAX

Description

Computes and returns the maximum value in a Bitmap.

Syntax

```
BITMAP_MAX(<bitmap>)
```

Parameters

Parameter	Description
<bitmap>	A Bitmap type column or expression

Return Value

The maximum value in the Bitmap.
Returns NULL if the Bitmap is empty.

Examples

To compute the maximum value in an empty Bitmap:

```
select bitmap_max(bitmap_from_string('')) value;
```

The result will be:

```
+-------+
| value |
+-------+
|  NULL |
+-------+
```

To compute the maximum value in a Bitmap with multiple elements:

```
select bitmap_max(bitmap_from_string('1,9999999999')) value;
```

The result will be:

```
+------------+
| value      |
+------------+
```

```
| 9999999999 |
+------------+
```

BITMAP_MIN

Description

Computes and returns the minimum value in a Bitmap.

Syntax

```
BITMAP_MIN(<bitmap>)
```

Parameters

Parameter	Description
<bitmap>	A Bitmap type column or expression

Return Value

The minimum value in the Bitmap.
Returns NULL if the Bitmap is empty.

Examples

To compute the minimum value in an empty Bitmap:

```
select bitmap_min(bitmap_from_string('')) value;
```

The result will be:

```
+-------+
| value |
+-------+
|  NULL |
+-------+
```

To compute the minimum value in a Bitmap with multiple elements:

```
select bitmap_min(bitmap_from_string('1,9999999999')) value;
```

The result will be:

```
+-------+
| value |
+-------+
```

```
|     1 |
+-------+
```

BITMAP_NOT

Description

Computes the difference between the first Bitmap and the second Bitmap, and returns the result as a new Bitmap.

Syntax

```
BITMAP_NOT(<bitmap1>, <bitmap2>)
```

Parameters

Parameter	Description
<bitmap1>	The first Bitmap
<bitmap2>	The second Bitmap

Return Value

A Bitmap representing the elements in <bitmap1> that are not in <bitmap2>.

Examples

To compute the difference between two Bitmaps:

```
select bitmap_to_string(bitmap_not(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4')));
```

The result will be an empty Bitmap, as all elements in <bitmap1> are also in <bitmap2>:

```
+-----------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_not(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'))) |
+-----------------------------------------------------------------------------------+
|                                                                                   |
+-----------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

To compute the difference where <bitmap1> has elements not present in <bitmap2>:

```
select bitmap_to_string(bitmap_not(bitmap_from_string('2,3,5'), bitmap_from_string('1,2,3,4')));
```

The result will be a Bitmap containing the element 5:

```
+-----------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_not(bitmap_from_string('2,3,5'), bitmap_from_string('1,2,3,4'))) |
```

```
+-------------------------------------------------------------------------------------+
| 5                                                                                   |
+-------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

BITMAP_OR

Description

Computes the union of two or more Bitmaps and returns a new Bitmap.

Syntax

```
BITMAP_OR(<bitmap1>, <bitmap2>, ..., <bitmapN>)
```

Parameters

Parameter	Description
<bitmap1>	The first Bitmap
<bitmap2>	The second Bitmap
...	...
<bitmapN>	The N-th Bitmap

Return Value

A Bitmap that represents the union of multiple Bitmaps.

Examples

To compute the union of two identical Bitmaps:

```
select bitmap_count(bitmap_or(to_bitmap(1), to_bitmap(1))) cnt;
```

The result will be:

```
+------+
| cnt  |
+------+
|    1 |
+------+
```

To convert the union of two identical Bitmaps to a string:

```
select bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(1)));
```

The result will be:

```
+-----------------------------------------------------+
| bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(1))) |
+-----------------------------------------------------+
| 1                                                   |
+-----------------------------------------------------+
```

To compute the union of two different Bitmaps:

```
select bitmap_count(bitmap_or(to_bitmap(1), to_bitmap(2))) cnt;
```

The result will be:

```
+------+
| cnt  |
+------+
|    2 |
+------+
```

To convert the union of two different Bitmaps to a string:

```
select bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2)));
```

The result will be:

```
+-----------------------------------------------------+
| bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2))) |
+-----------------------------------------------------+
| 1,2                                                 |
+-----------------------------------------------------+
```

To compute the union of multiple Bitmaps, including a `NULL` value:

```
select bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2), to_bitmap(10), to_bitmap(0), NULL));
```

The result will be:

```
+-------------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2), to_bitmap(10), to_bitmap(0), NULL)) |
+-------------------------------------------------------------------------------------------+
| 0,1,2,10                                                                                  |
+-------------------------------------------------------------------------------------------+
```

To compute the union of multiple Bitmaps, including an empty Bitmap:

```
select bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2), to_bitmap(10), to_bitmap(0), bitmap_empty()));
```

The result will be:

```
+-------------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2), to_bitmap(10), to_bitmap(0), bitmap_empty())) |
```

```
+-----------------------------------------------------------------------------------------------------+
| 0,1,2,10                                                                                            |
+-----------------------------------------------------------------------------------------------------+
```

To compute the union of Bitmaps created from strings and individual values:

```
select bitmap_to_string(bitmap_or(to_bitmap(10), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5')));
```

The result will be:

```
+-----------------------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_or(to_bitmap(10), bitmap_from_string('1,2'), bitmap_from_string('1,2,3,4,5'))) |
+-----------------------------------------------------------------------------------------------------+
| 1,2,3,4,5,10                                                                                        |
+-----------------------------------------------------------------------------------------------------+
```

BITMAP_OR_COUNT

Description

Computes the union of two or more input Bitmaps and returns the count of elements in the union.

Syntax

```
BITMAP_OR_COUNT(<bitmap1>, <bitmap2>, ..., <bitmapN>)
```

Parameters

Parameter	Description
<bitmap1>	The first Bitmap
<bitmap2>	The second Bitmap
...	...
<bitmapN>	The N-th Bitmap

Return Value

The count of elements in the union of multiple Bitmaps.
Returns `NULL` if any of the Bitmaps is `NULL`.

Examples

To compute the count of elements in the union of a non-empty Bitmap and an empty Bitmap:

```
select bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_empty());
```

The result will be:

```
+-----------------------------------------------------------+
| bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_empty()) |
+-----------------------------------------------------------+
|                                                         3 |
+-----------------------------------------------------------+
```

To compute the count of elements in the union of two identical Bitmaps:

```
select bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2,3'));
```

The result will be:

```
+-----------------------------------------------------------------------+
| bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2,3')) |
+-----------------------------------------------------------------------+
|                                                                     3 |
+-----------------------------------------------------------------------+
```

To compute the count of elements in the union of two different Bitmaps:

```
select bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5'));
```

The result will be:

```
+-----------------------------------------------------------------------+
| bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5')) |
+-----------------------------------------------------------------------+
|                                                                     5 |
+-----------------------------------------------------------------------+
```

To compute the count of elements in the union of multiple Bitmaps, including an empty Bitmap:

```
select bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5'), to_bitmap(100), bitmap_empty());
```

The result will be:

```
+-----------------------------------------------------------------------------------------------------+
| bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5'), to_bitmap(100), bitmap_empty()) |
+-----------------------------------------------------------------------------------------------------+
|                                                                                                   6 |
+-----------------------------------------------------------------------------------------------------+
```

To compute the count of elements in the union of multiple Bitmaps, including a `NULL` value:

```
select bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5'), to_bitmap(100), NULL);
```

The result will be:

```
+-----------------------------------------------------------------------------------------+
| bitmap_or_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5'), to_bitmap(100), NULL) |
+-----------------------------------------------------------------------------------------+
|                                                                                    NULL |
+-----------------------------------------------------------------------------------------+
```

BITMAP_REMOVE

Description

Removes a specified value from a Bitmap column.

Syntax

```
BITMAP_REMOVE(<bitmap>, <value>)
```

Parameters

Parameter	Description
<bitmap>	The Bitmap value
<value>	The value to remove

Return Value

Returns the Bitmap after removing the specified value.

Returns the original Bitmap if the value to be removed does not exist;
Returns NULL if the value to be removed is NULL.

Examples

To remove a value from a Bitmap:

```
select bitmap_to_string(bitmap_remove(bitmap_from_string('1, 2, 3'), 3)) res;
```

The result will be:

```
+------+
| res  |
+------+
| 1,2  |
+------+
```

To remove a NULL value from a Bitmap:

```
select bitmap_to_string(bitmap_remove(bitmap_from_string('1, 2, 3'), null)) res;
```

The result will be:

```
+------+
| res  |
+------+
| NULL |
+------+
```

BITMAP_SUBSET_IN_RANGE

Description

Returns a subset of the Bitmap within the specified range (excluding the range end).

Syntax

```
BITMAP_SUBSET_IN_RANGE(<bitmap>, <range_start_include>, <range_end_exclude>)
```

Parameters

Parameter	Description
<bitmap>	The Bitmap value
<range_start_include>	The start of the range (inclusive)
<range_end_exclude>	The end of the range (exclusive)

Return Value

A subset Bitmap within the specified range.

Examples

To get a subset of a Bitmap within the range 0 to 9:

```
select bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,2,3,4,5'), 0, 9)) value;
```

The result will be:

```
+-----------+
| value     |
+-----------+
| 1,2,3,4,5 |
+-----------+
```

To get a subset of a Bitmap within the range 2 to 3:

```
select bitmap_to_string(bitmap_subset_in_range(bitmap_from_string('1,2,3,4,5'), 2, 3)) value;
```

The result will be:

```
+-------+
| value |
+-------+
```

```
| 2     |
+-------+
```

BITMAP_SUBSET_LIMIT

Description

Extracts a subset of Bitmap elements starting from a specified position, with a limit on the number of elements specified by the cardinality limit, and returns the subset as a new Bitmap.

Syntax

```
BITMAP_SUBSET_LIMIT(<bitmap>, <position>, <cardinality_limit>)
```

Parameters

Parameter	Description
<bitmap>	The Bitmap value
<position>	The starting position (inclusive)
<cardinality_limit>	The maximum number of elements

Return Value

A subset Bitmap within the specified range and limit.

Examples

To get a subset of a Bitmap starting from position 0 with a cardinality limit of 3:

```
select bitmap_to_string(bitmap_subset_limit(bitmap_from_string('1,2,3,4,5'), 0, 3)) value;
```

The result will be:

```
+-----------+
| value     |
+-----------+
| 1,2,3     |
+-----------+
```

To get a subset of a Bitmap starting from position 4 with a cardinality limit of 3:

```
select bitmap_to_string(bitmap_subset_limit(bitmap_from_string('1,2,3,4,5'), 4, 3)) value;
```

The result will be:

```
+-------+
| value |
+-------+
| 4,5   |
+-------+
```

BITMAP_TO_ARRAY

Description

Converts a Bitmap into an Array.

Syntax

```
BITMAP_TO_ARRAY(<bitmap>)
```

Parameters

Parameter	Description
<bitmap>	A Bitmap type column or expression

Return Value

An array containing all the bit positions in the Bitmap.
Returns NULL if the Bitmap is NULL.

Examples

To convert a NULL Bitmap to an array:

```
select bitmap_to_array(null);
```

The result will be:

```
+------------------------+
| bitmap_to_array(NULL)  |
+------------------------+
| NULL                   |
+------------------------+
```

To convert an empty Bitmap to an array:

```
select bitmap_to_array(bitmap_empty());
```

The result will be:

```
+---------------------------------+
| bitmap_to_array(bitmap_empty()) |
+---------------------------------+
```

```
| []                             |
+--------------------------------+
```

To convert a Bitmap with a single element to an array:

```
select bitmap_to_array(to_bitmap(1));
```

The result will be:

```
+------------------------------+
| bitmap_to_array(to_bitmap(1)) |
+------------------------------+
| [1]                          |
+------------------------------+
```

To convert a Bitmap with multiple elements to an array:

```
select bitmap_to_array(bitmap_from_string('1,2,3,4,5'));
```

The result will be:

```
+-------------------------------------------------+
| bitmap_to_array(bitmap_from_string('1,2,3,4,5')) |
+-------------------------------------------------+
| [1, 2, 3, 4, 5]                                 |
+-------------------------------------------------+
```

BITMAP_TO_BASE64

Description

Converts a Bitmap into a Base64 encoded string.

Syntax

```
BITMAP_TO_BASE64(<bitmap>)
```

Parameters

Parameter	Description
<bitmap>	A Bitmap type column or expression

Return Value

A Base64 encoded string of the Bitmap.
Returns NULL if the Bitmap is NULL.

The BE configuration option enable_set_in_bitmap_value changes the specific format of bitmap values in memory, which affects the result of this function.
Due to the non-guaranteed order of elements in a bitmap, the generated Base64 string may not always be the same for the same content. However, the decoded bitmap from bitmap_from_base64 will be the same.

Examples

To convert a NULL Bitmap to a Base64 string:

```
select bitmap_to_base64(null);
```

The result will be:

```
+------------------------+
| bitmap_to_base64(NULL) |
+------------------------+
| NULL                   |
+------------------------+
```

To convert an empty Bitmap to a Base64 string:

```
select bitmap_to_base64(bitmap_empty());
```

The result will be:

```
+----------------------------------+
| bitmap_to_base64(bitmap_empty()) |
+----------------------------------+
| AA==                             |
+----------------------------------+
```

To convert a Bitmap with a single element to a Base64 string:

```
select bitmap_to_base64(to_bitmap(1));
```

The result will be:

```
+--------------------------------+
| bitmap_to_base64(to_bitmap(1)) |
+--------------------------------+
| AQEAAAA=                       |
+--------------------------------+
```

To convert a Bitmap with multiple elements to a Base64 string:

```
select bitmap_to_base64(bitmap_from_string("1,9999999"));
```

The result will be:

```
+----------------------------------------------------------+
| bitmap_to_base64(bitmap_from_string("1,9999999"))        |
+----------------------------------------------------------+
| AjowAAACAAAAAAAAAJgAAAAYAAAAGgAAAAEAf5Y=                 |
+----------------------------------------------------------+
```

BITMAP_TO_STRING

Description

Converts a Bitmap into a comma-separated string containing all the set bit positions.

Syntax

```
BITMAP_TO_STRING(<bitmap>)
```

Parameters

Parameter	Description
<bitmap>	A Bitmap type column or expression

Return Value

A string containing all the set bit positions in the Bitmap, separated by commas.
Returns NULL if the Bitmap is NULL.

Examples

To convert a NULL Bitmap to a string:

```
select bitmap_to_string(null);
```

The result will be:

```
+------------------------+
| bitmap_to_string(NULL) |
+------------------------+
| NULL                   |
+------------------------+
```

To convert an empty Bitmap to a string:

```
select bitmap_to_string(bitmap_empty());
```

The result will be:

```
+---------------------------------+
| bitmap_to_string(bitmap_empty()) |
+---------------------------------+
```

```
|                                 |
+---------------------------------+
```

To convert a Bitmap with a single element to a string:

```
select bitmap_to_string(to_bitmap(1));
```

The result will be:

```
+--------------------------------+
| bitmap_to_string(to_bitmap(1)) |
+--------------------------------+
| 1                              |
+--------------------------------+
```

To convert a Bitmap with multiple elements to a string:

```
select bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2)));
```

The result will be:

```
+-------------------------------------------------------+
| bitmap_to_string(bitmap_or(to_bitmap(1), to_bitmap(2))) |
+-------------------------------------------------------+
| 1,2                                                   |
+-------------------------------------------------------+
```

BITMAP_XOR

Description

Computes the symmetric difference (XOR operation) of two or more input Bitmaps and returns a new Bitmap.

Syntax

```
BITMAP_XOR(<bitmap1>, <bitmap2>, ..., <bitmapN>)
```

Parameters

Parameter	Description
<bitmap1>	The first Bitmap
<bitmap2>	The second Bitmap
...	...
<bitmapN>	The N-th Bitmap

Return Value

A Bitmap representing the symmetric difference of multiple Bitmaps.

Examples

To compute the symmetric difference of two Bitmaps:

```
select bitmap_count(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'))) cnt;
```

The result will be:

```
+------+
| cnt  |
+------+
|    2 |
+------+
```

To convert the symmetric difference of two Bitmaps to a string:

```
select bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4')));
```

The result will be:

```
+-------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'))) |
+-------------------------------------------------------------------------------+
| 1,4                                                                           |
+-------------------------------------------------------------------------------+
```

To compute the symmetric difference of three Bitmaps:

```
select bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5')));
```

The result will be:

```
+-------------------------------------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'))) |
+-------------------------------------------------------------------------------------------------------------------+
| 1,3,5                                                                                                             |
+-------------------------------------------------------------------------------------------------------------------+
```

To compute the symmetric difference of multiple Bitmaps, including an empty Bitmap:

```
select bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'),
bitmap_empty()));
```

The result will be:

```
+-------------------------------------------------------------------------------------------------------------------------------------
--+
| bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'),
bitmap_empty())) |
+-------------------------------------------------------------------------------------------------------------------------------------
--+
| 1,3,5
|
+-------------------------------------------------------------------------------------------------------------------------------------
--+
```

To compute the symmetric difference of multiple Bitmaps, including a `NULL` value:

```
select bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'), NULL));
```

The result will be:

```
+-------------------------------------------------------------------------------------------------------------------+
| bitmap_to_string(bitmap_xor(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'), NULL)) |
+-------------------------------------------------------------------------------------------------------------------+
| NULL                                                                                                              |
+-------------------------------------------------------------------------------------------------------------------+
```

BITMAP_XOR_COUNT

Description

Computes the symmetric difference (XOR operation) of two or more Bitmap sets and returns the count of elements in the result set.

Syntax

```
BITMAP_XOR_COUNT(<bitmap1>, <bitmap2>, ..., <bitmapN>)
```

Parameters

Parameter	Description
<bitmap1>	The first Bitmap
<bitmap2>	The second Bitmap
...	...
<bitmapN>	The N-th Bitmap

Return Value

The count of elements in the Bitmap resulting from the XOR operation of multiple Bitmaps.
Returns `NULL` if any of the input Bitmap parameters is `NULL`.

Examples

To compute the count of elements in the symmetric difference of two Bitmaps:

```
select bitmap_xor_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5'));
```

The result will be:

```
+-----------------------------------------------------------------------+
| bitmap_xor_count(bitmap_from_string('1,2,3'), bitmap_from_string('3,4,5')) |
+-----------------------------------------------------------------------+
|                                                                     4 |
+-----------------------------------------------------------------------+
```

To compute the count of elements in the symmetric difference of two identical Bitmaps:

```
select bitmap_xor_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2,3'));
```

The result will be:

```
+-----------------------------------------------------------------------+
| bitmap_xor_count(bitmap_from_string('1,2,3'), bitmap_from_string('1,2,3')) |
+-----------------------------------------------------------------------+
|                                                                     0 |
+-----------------------------------------------------------------------+
```

To compute the count of elements in the symmetric difference of two different Bitmaps:

```
select bitmap_xor_count(bitmap_from_string('1,2,3'), bitmap_from_string('4,5,6'));
```

The result will be:

```
+-----------------------------------------------------------------------+
| bitmap_xor_count(bitmap_from_string('1,2,3'), bitmap_from_string('4,5,6')) |
+-----------------------------------------------------------------------+
|                                                                     6 |
+-----------------------------------------------------------------------+
```

To compute the count of elements in the symmetric difference of three Bitmaps:

```
select bitmap_xor_count(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'));
```

The result will be:

```
+-----------------------------------------------------------------------------------------------------+
| bitmap_xor_count(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'))    |
+-----------------------------------------------------------------------------------------------------+
|                                                                                                   3 |
+-----------------------------------------------------------------------------------------------------+
```

To compute the count of elements in the symmetric difference of multiple Bitmaps, including an empty Bitmap:

```
select bitmap_xor_count(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'), bitmap_empty());
```

The result will be:

```
+-----------------------------------------------------------------------------------------------------------------+
| bitmap_xor_count(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'), bitmap_empty())    |
+-----------------------------------------------------------------------------------------------------------------+
|                                                                                                               3 |
+-----------------------------------------------------------------------------------------------------------------+
```

To compute the count of elements in the symmetric difference of multiple Bitmaps, including a `NULL` value:

```
select bitmap_xor_count(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'), NULL);
```

The result will be:

```
+-----------------------------------------------------------------------------------------------------+
| bitmap_xor_count(bitmap_from_string('2,3'), bitmap_from_string('1,2,3,4'), bitmap_from_string('3,4,5'), NULL)   |
```

```
+-------------------------------------------------------------------------------------------------------------------+
|                                                                                                              NULL |
+-------------------------------------------------------------------------------------------------------------------+
```

SUB_BITMAP

Description

Extracts a subset of Bitmap elements starting from a specified position and limited by a specified cardinality limit, returning the subset as a new Bitmap.

Syntax

```
SUB_BITMAP(<bitmap>, <position>, <cardinality_limit>)
```

Parameters

Parameter	Description
<bitmap>	The Bitmap value
<position>	The starting position (inclusive)
<cardinality_limit>	The maximum number of elements

Return Value

A subset Bitmap within the specified range and limit.

Examples

To get a subset of a Bitmap starting from position 0 with a cardinality limit of 3:

```
select bitmap_to_string(sub_bitmap(bitmap_from_string('1,0,1,2,3,1,5'), 0, 3)) value;
```

The result will be:

```
+-------+
| value |
+-------+
| 0,1,2 |
+-------+
```

To get a subset of a Bitmap starting from position -3 with a cardinality limit of 2:

```
select bitmap_to_string(sub_bitmap(bitmap_from_string('1,0,1,2,3,1,5'), -3, 2)) value;
```

The result will be:

```
+-------+
| value |
+-------+
| 2,3   |
+-------+
```

To get a subset of a Bitmap starting from position 2 with a cardinality limit of 100:

```
select bitmap_to_string(sub_bitmap(bitmap_from_string('1,0,1,2,3,1,5'), 2, 100)) value;
```

The result will be:

```
+-------+
| value |
+-------+
| 2,3,5 |
+-------+
```

TO_BITMAP

Description

Converts an unsigned bigint to a Bitmap.

The input is an unsigned bigint with a value in the range 0 to 18446744073709551615, and the output is a Bitmap containing that element.

Syntax

```
TO_BITMAP(<expr>)
```

Parameters

Parameter	Description
<expr>	An unsigned bigint with a range of 0 to 18446744073709551615

Return Value

A Bitmap containing the corresponding bigint.
Returns `NULL` if the input value is not within the specified range.

Examples

To convert an integer to a Bitmap and count the number of elements in the Bitmap:

```
select bitmap_count(to_bitmap(10));
```

The result will be:

```
+----------------------------+
| bitmap_count(to_bitmap(10)) |
+----------------------------+
|                          1 |
+----------------------------+
```

To convert a negative integer to a Bitmap, which is outside the valid range, and convert it to a string:

```
select bitmap_to_string(to_bitmap(-1));
```

The result will be:

```
+--------------------------------+
| bitmap_to_string(to_bitmap(-1)) |
+--------------------------------+
|                                |
+--------------------------------+
```

HLL_CARDINALITY

Description

HLL_CARDINALITY calculates the cardinality of a HyperLogLog (HLL) type value. It is an approximate counting algorithm suitable for estimating the number of distinct elements in large datasets.

Syntax

```
HLL_CARDINALITY(<hll>)
```

Parameters

Parameter	Description
<hll>	The HLL type value representing the dataset whose cardinality needs to be estimated.

Return Value

Returns the estimated cardinality of the HLL type value, representing the number of distinct elements in the dataset.

Example

```
select HLL_CARDINALITY(uv_set) from test_uv;
```

```
+--------------------------+
| hll_cardinality(`uv_set`) |
+--------------------------+
|                        3 |
+--------------------------+
```

HLL_EMPTY

Description

`HLL_EMPTY` returns an empty HLL (HyperLogLog) value, representing a data set with no elements.

Syntax

```
HLL_EMPTY()
```

Return Value

Returns an empty HLL type value, representing a data set with no elements.

Example

```
select hll_cardinality(hll_empty());
```

```
+-----------------------------+
| hll_cardinality(hll_empty()) |
+-----------------------------+
|                           0 |
+-----------------------------+
```

HLL_FROM_BASE64

Description

Converts a base64-encoded string (usually the result of the `HLL_TO_BASE64` function) into an HLL. If the input string is invalid or NULL, the function returns NULL.

Syntax

```
HLL_FROM_BASE64(<input>)
```

Parameters

Parameter	Description
`<input>`	A base64-encoded string, typically generated by the `HLL_TO_BASE64` function. If the input string is invalid or NULL, the function returns NULL.

Examples

```
select hll_union_agg(hll_from_base64(hll_to_base64(pv))), hll_union_agg(pv) from test_hll;
```

```
+---------------------------------------------------+------------------+
| hll_union_agg(hll_from_base64(hll_to_base64(pv))) | hll_union_agg(pv) |
+---------------------------------------------------+------------------+
|                                                 3 |                3 |
+---------------------------------------------------+------------------+
```

```
select hll_cardinality(hll_from_base64(hll_to_base64(hll_hash('abc'))));
```

```
+-----------------------------------------------------------------+
| hll_cardinality(hll_from_base64(hll_to_base64(hll_hash('abc')))) |
+-----------------------------------------------------------------+
|                                                               1 |
+-----------------------------------------------------------------+
```

```
select hll_cardinality(hll_from_base64(hll_to_base64(hll_hash(''))));
```

```
+-----------------------------------------------------------------+
| hll_cardinality(hll_from_base64(hll_to_base64(hll_hash(''')))) |
+-----------------------------------------------------------------+
|                                                               1 |
+-----------------------------------------------------------------+
```

```
select hll_cardinality(hll_from_base64(hll_to_base64(hll_hash(NULL))));
```

```
+------------------------------------------------------------------+
| hll_cardinality(hll_from_base64(hll_to_base64(hll_hash(NULL)))) |
+------------------------------------------------------------------+
|                                                                0 |
+------------------------------------------------------------------+
```

HLL_HASH

Description

Converts a given value to the HLL (HyperLogLog) type. This function is typically used during data loading to create an HLL from raw data.

Syntax

```
HLL_HASH(<value>)
```

Parameters

Parameter	Description
<value>	The value to be converted to HLL type. This can be a string, number, or any data type.

Examples

```
SELECT HLL_CARDINALITY(HLL_HASH('abc'));
```

```
+----------------------------------+
| hll_cardinality(HLL_HASH('abc')) |
+----------------------------------+
|                                1 |
+----------------------------------+
```

HLL_TO_BASE64

Description

Converts an input HLL to a base64-encoded string. If the input is <null>, the function returns <null>.

Syntax

```
HLL_TO_BASE64(<hll_input>)
```

Parameters

Parameter	Description
<hll_input>	The HyperLogLog (HLL) data to be converted to a base64-encoded string. If the input is <null>, the function returns <null>.

Examples

```
select hll_to_base64(NULL);
```

```
+---------------------+
| hll_to_base64(NULL) |
+---------------------+
| NULL                |
+---------------------+
```

```
select hll_to_base64(hll_empty());
```

```
+----------------------------+
| hll_to_base64(hll_empty()) |
+----------------------------+
| AA==                       |
+----------------------------+
```

```
select hll_to_base64(hll_hash('abc'));
```

```
+--------------------------------+
| hll_to_base64(hll_hash('abc')) |
+--------------------------------+
| AQEC5XSzrpDsdw==               |
+--------------------------------+
```

```
select hll_union_agg(hll_from_base64(hll_to_base64(pv))), hll_union_agg(pv) from test_hll;
```

```
+-------------------------------------------------+------------------+
| hll_union_agg(hll_from_base64(hll_to_base64(pv))) | hll_union_agg(pv) |
+-------------------------------------------------+------------------+
|                                               3 |                3 |
+-------------------------------------------------+------------------+
```

```
select hll_cardinality(hll_from_base64(hll_to_base64(hll_hash('abc'))));
```

```
+-----------------------------------------------------------------+
| hll_cardinality(hll_from_base64(hll_to_base64(hll_hash('abc')))) |
+-----------------------------------------------------------------+
|                                                               1 |
+-----------------------------------------------------------------+
```

CONNECTION_ID

Description

Get the connection number of the current sql client.

Syntax

```
CONNECTION_ID()
```

Return Value

The connection number of the current sql client.

Examples

```
select connection_id();
```

```
+-----------------+
| connection_id() |
+-----------------+
|             549 |
+-----------------+
```

CURRENT_CATALOG

Description

Get the catalog of the current sql client connection.

Syntax

```
CURRENT_CATALOG()
```

Return Value

The catalog name of the current sql client connection.

Examples

```
select current_catalog();
```

```
+------------------+
| current_catalog() |
+------------------+
| internal          |
+------------------+
```

CURRENT_USER

Description

Get the current username and its IP rule whitelist.

Syntax

```
CURRENT_USER()
```

Return Value

Returns the current username and its IP whitelist.

Format:`<user_name>@<ip_white_list>`

Examples

- root user, no IP restrictions

```
select current_user();
```

```
+----------------+
| current_user() |
+----------------+
| 'root'@'%'     |
+----------------+
```

- doris user, IP whitelist is 192.168.*

```
select current_user();
```

```
+---------------------+
| current_user()      |
+---------------------+
| 'doris'@'192.168.%' |
+---------------------+
```

DATABASE

Description

Get the database of the current sql client connection.

Alias

- SCHEMA

Syntax

```
DATABASE()
```

or

```
SCHEMA()
```

Return Value

The name of the database connected to the current sql client.

Examples

```
select database(),schema();
```

```
+------------+------------+
| database() | database() |
+------------+------------+
| test       | test       |
+------------+------------+
```

LAST_QUERY_ID

Description

Get the query id of the current user's last query.

Syntax

```
LAST_QUERY_ID()
```

Return Value

Returns the query id of the current user's last query;

Examples

```
select last_query_id();
```

```
+-----------------------------------+
| last_query_id()                   |
+-----------------------------------+
| 128f913c3ec84a1e-b834bd0262cc090a |
+-----------------------------------+
```

SESSION_USER

Description

Get the current username and IP of Doris connection, compatible with MySQL protocol.

Syntax

```
SESSION_USER()
```

Return Value

Returns the current username and IP Doris is connected to. Format:<user_name>@<ip>

Examples

```
select session_user();
```

```
+----------------------+
| session_user()       |
+----------------------+
| 'root'@'10.244.2.10' |
+----------------------+
```

USER

Description

Get the current username and IP that Doris is connected to.

Syntax

```
USER()
```

Return Value

Returns the current username and IP Doris is connected to. format:<user_name>@<ip>

Examples

```
select user();
```

```
+---------------------+
| user()              |
+---------------------+
| 'root'@'10.244.2.5' |
+---------------------+
```

VERSION

Description

No practical meaning, for compatibility with MySQL protocol.

Syntax

```
VERSION()
```

Return Value

Compatible with MySQL protocol, fixed return value is “5.7.99”.

Examples

```
select version();
```

```
+-----------+
| version() |
+-----------+
| 5.7.99    |
+-----------+
```

CONVERT_TO

Description

Converts the character encoding of a column to a specified target character set. This function is typically used in the ORDER BY clause to ensure that columns containing Chinese characters are sorted according to their pinyin order. Currently, only conversion to `'gbk'` is supported.

Syntax

```
CONVERT_TO(<column>, <character>)
```

Parameters

Parameters	Description
`<column>`	The VARCHAR column whose encoding is to be converted.
`<character>`	The target character set. Currently, only `'gbk'` is supported.

Return Value

Returns a VARCHAR value with the converted encoding, allowing proper pinyin-based ordering when used in the ORDER BY clause.

Examples

```
SELECT * FROM class_test ORDER BY class_name;
```

```
+----------+------------+-------------+
| class_id | class_name | student_ids |
+----------+------------+-------------+
|        6 | asd        | [6]         |
|        7 | qwe        | [7]         |
|        8 | z          | [8]         |
|        2 | 哈         | [2]         |
|        3 | 哦         | [3]         |
|        1 | 啊         | [1]         |
|        4 | 张         | [4]         |
|        5 | 我         | [5]         |
+----------+------------+-------------+
```

```
SELECT * FROM class_test ORDER BY CONVERT_TO(class_name, 'gbk');
```

```
+----------+------------+-------------+
| class_id | class_name | student_ids |
+----------+------------+-------------+
```

```
|        6 | asd        | [6]         |
|        7 | qwe        | [7]         |
|        8 | z          | [8]         |
|        1 | 啊         | [1]         |
|        2 | 哈         | [2]         |
|        3 | 哦         | [3]         |
|        5 | 我         | [5]         |
|        4 | 张         | [4]         |
+----------+------------+-------------+
```

ESQUERY

Description

Use the ESQUERY(<field>, <query_dsl>) function to push down queries that cannot be expressed in SQL to Elasticsearch for filtering. The first parameter, <field>, is used to associate indexes, while the second parameter, <query_dsl>, is a JSON expression representing the basic Elasticsearch Query DSL. The JSON must be enclosed in curly brackets {} and contain exactly one root key (e.g., match_phrase, geo_shape, bool).

Syntax

```
ESQUERY(<field>, <query_dsl>)
```

Parameters

Parameter	Description
<field>	The field to be queried; used to associate indexes.
<query_dsl>	A JSON expression representing the Elasticsearch Query DSL. It must be enclosed in {} and contain exactly one root key (e.g., match_phrase, geo_shape, bool).

Return Value

Returns a boolean value indicating whether the document matches the provided Elasticsearch query DSL.

Examples

```
-- match_phrase SQL:
SELECT * FROM es_table
WHERE ESQUERY(k4, '{
    "match_phrase": {
       "k4": "doris on es"
    }
}');
```

```
-- geo_shape SQL:
SELECT * FROM es_table
WHERE ESQUERY(k4, '{
  "geo_shape": {
     "location": {
        "shape": {
           "type": "envelope",
           "coordinates": [
              [13, 53],
              [14, 52]
           ]
        },
        "relation": "within"
```

```
    }
  }
}');
```

FIELD

Description

Returns the position of the first occurrence of <expr> in the list of values <param> [, ...].
If <expr> is not found, the function returns 0. This function is commonly used in ORDER BY to implement custom sorting.

Syntax

```
FIELD(<expr>, <param> [, ...])
```

Parameters

Parameter	Description
<expr>	The value to be searched in the list of parameters.
<param>	A sequence of values to compare against <expr>.

Return Value

- Returns the position (1-based index) of <expr> in the list of <param> values.
- If <expr> is not found, returns 0.
- If <expr> is NULL, returns 0.

Examples

```
-- setup
CREATE TABLE baseall (k1 INT, k7 VARCHAR(64))
DISTRIBUTED BY HASH(k1) BUCKETS 1
PROPERTIES ("replication_num" = "1");

INSERT INTO baseall VALUES
    (1, 'wangjing04'),
    (2, 'wangyu14'),
    (3, 'yuanyuan06');

CREATE TABLE class_test (class_name VARCHAR(64))
DISTRIBUTED BY HASH(class_name) BUCKETS 1
PROPERTIES ("replication_num" = "1");

INSERT INTO class_test VALUES
    ('Suzi'), ('Suzi'),
    ('Ben'), ('Ben'),
    ('Henry'), ('Henry'),
    (NULL);
```

```
SELECT FIELD(2, 3, 1, 2, 5);
```

```
+----------------------+
| FIELD(2, 3, 1, 2, 5) |
+----------------------+
|                    3 |
+----------------------+
```

```
SELECT k1, k7 FROM baseall WHERE k1 IN (1,2,3) ORDER BY FIELD(k1, 2, 1, 3);
```

```
+------+------------+
| k1   | k7         |
+------+------------+
|    2 | wangyu14   |
|    1 | wangjing04 |
|    3 | yuanyuan06 |
+------+------------+
```

Custom string ordering. `FIELD` returns `0` for values not in the list — including `NULL`, which is why the `NULL` row sorts to the top in the ascending default:

```
SELECT class_name FROM class_test ORDER BY FIELD(class_name, 'Suzi', 'Ben', 'Henry');
```

```
+------------+
| class_name |
+------------+
| NULL       |
| Suzi       |
| Suzi       |
| Ben        |
| Ben        |
| Henry      |
| Henry      |
+------------+
```

Descending — `NULL` (FIELD = 0) is now last:

```
SELECT class_name FROM class_test ORDER BY FIELD(class_name, 'Suzi', 'Ben', 'Henry') DESC;
```

```
+------------+
| class_name |
+------------+
| Henry      |
| Henry      |
| Ben        |
| Ben        |
| Suzi       |
| Suzi       |
| NULL       |
+------------+
```

You can also use `NULLS FIRST` / `NULLS LAST` to control `NULL` placement independently of the sort direction:

```
SELECT class_name FROM class_test ORDER BY FIELD(class_name, 'Suzi', 'Ben', 'Henry') NULLS FIRST;
```

```
+------------+
| class_name |
+------------+
| NULL       |
| Suzi       |
| Suzi       |
| Ben        |
| Ben        |
| Henry      |
| Henry      |
+------------+
```

G

QUANTILE_PERCENT

Description

The QUANTILE_PERCENT function is used to calculate the quantile value for a given percentage. It takes two parameters: a quantile_state column and a constant floating-point number representing the percentage. The function returns a floating-point number that represents the quantile value at the given percentage position.

Syntax

```
QUANTILE_PERCENT(<quantile_state>, <percent>)
```

Parameters

Parameter	Description
<quantile_state>	The target column.
<percent>	Target percent.

Return value

A Double type to represent quantile.

Example

```
CREATE TABLE IF NOT EXISTS quantile_state_agg_test (
         `dt` int(11) NULL COMMENT "",
         `id` int(11) NULL COMMENT "",
         `price` quantile_state QUANTILE_UNION NOT NULL COMMENT ""
        ) ENGINE=OLAP
        AGGREGATE KEY(`dt`, `id`)
        COMMENT "OLAP"
        DISTRIBUTED BY HASH(`dt`) BUCKETS 1
        PROPERTIES ("replication_num" = "1");

INSERT INTO quantile_state_agg_test VALUES(20220201,0, to_quantile_state(1, 2048));

INSERT INTO quantile_state_agg_test VALUES(20220201,1, to_quantile_state(-1, 2048)),
            (20220201,1, to_quantile_state(0, 2048)),(20220201,1, to_quantile_state(1, 2048)),
            (20220201,1, to_quantile_state(2, 2048)),(20220201,1, to_quantile_state(3, 2048));

SELECT dt, id, quantile_percent(quantile_union(price), 0) FROM quantile_state_agg_test GROUP BY dt, id ORDER BY dt, id
```

Result is

```
+----------+------+-----------------------------------------+
| dt       | id   | quantile_percent(quantile_union(price), 0) |
+----------+------+-----------------------------------------+
```

```
| 20220201 |    0 |                                       1 |
| 20220201 |    1 |                                      -1 |
+----------+------+-----------------------------------------+
```

QUANTILE_STATE_EMPTY

Description

Return an empty `quantile_state` type column.

Syntax

```
QUANTILE_STATE_EMPTY()
```

Return value

An empty `quantile_state` type column.

Example

```
select quantile_percent(quantile_union(quantile_state_empty()), 0)
```

Result is

```
+-----------------------------------------------------------+
| quantile_percent(quantile_union(quantile_state_empty()), 0) |
+-----------------------------------------------------------+
|                                                      NULL |
+-----------------------------------------------------------+
```

TO_QUANTILE_STATE

Description

This function converts numeric types to `QUANTILE_STATE` type. The compression parameter is optional and can be set in the range [2048, 10000]. The larger the value, the higher the accuracy of subsequent quantile approximation calculations, the greater the memory consumption, and the longer the calculation time. If the compression parameter is not specified or the value is set outside the range [2048, 10000], it runs with the default value of 2048.

Syntax

```
TO_QUANTILE_STATE(<raw_data>, <compression>)
```

Parameters

Parameter	Description
`<raw_data>`	The target column.
`<compression>`	Compression threshold.

Return value

The converted column of `QUANTILE_STATE` type.

Example

```
CREATE TABLE IF NOT EXISTS quantile_state_agg_test (
         `dt` int(11) NULL COMMENT "",
         `id` int(11) NULL COMMENT "",
         `price` quantile_state QUANTILE_UNION NOT NULL COMMENT ""
        ) ENGINE=OLAP
        AGGREGATE KEY(`dt`, `id`)
        COMMENT "OLAP"
        DISTRIBUTED BY HASH(`dt`) BUCKETS 1
        PROPERTIES ("replication_num" = "1");

INSERT INTO quantile_state_agg_test VALUES(20220201,0, to_quantile_state(1, 2048));

INSERT INTO quantile_state_agg_test VALUES(20220201,1, to_quantile_state(-1, 2048)),
            (20220201,1, to_quantile_state(0, 2048)),(20220201,1, to_quantile_state(1, 2048)),
            (20220201,1, to_quantile_state(2, 2048)),(20220201,1, to_quantile_state(3, 2048));

SELECT dt, id, quantile_percent(quantile_union(price), 0) FROM quantile_state_agg_test GROUP BY dt, id ORDER BY dt, id
```

Result is

```
+----------+------+-------------------------------------------+
| dt       | id   | quantile_percent(quantile_union(price), 0) |
+----------+------+-------------------------------------------+
| 20220201 |    0 |                                         1 |
| 20220201 |    1 |                                        -1 |
+----------+------+-------------------------------------------+
```

COALESCE

Description

Returns the first non-null expression from left to right in the parameter list. If all arguments are NULL, returns NULL.

Syntax

```
COALESCE(<expr> [, ...])
```

Parameters

Parameter	Description
<expr>	A sequence of expressions to evaluate. All expressions must be compatible data types.

Return Value

The first non-null expression in the parameter list. Returns NULL if all arguments are NULL.

Examples

```
SELECT COALESCE(NULL, '1111', '0000');
```

```
+--------------------------------+
| coalesce(NULL, '1111', '0000') |
+--------------------------------+
| 1111                           |
+--------------------------------+
```

GREATEST

Description

Compares multiple expressions and returns the greatest value among them. If any argument is `NULL`, the function returns `NULL`.

Syntax

```
GREATEST(<expr> [, ...])
```

Parameters

Parameter	Description
`<expr>`	A list of expressions to compare. Supports `TINYINT`, `SMALLINT`, `INT`, `BIGINT`, `LARGEINT`, `FLOAT`, `DOUBLE`, `STRING`, `DATETIME`, and `DECIMAL` types.

Return Value

- Returns the largest value among the given expressions.
- If any argument is `NULL`, returns `NULL`.

Examples

```
SELECT GREATEST(-1, 0, 5, 8);
```

```
+----------------------+
| GREATEST(-1, 0, 5, 8) |
+----------------------+
|                    8 |
+----------------------+
```

```
SELECT GREATEST(-1, 0, 5, NULL);
```

```
+-------------------------+
| GREATEST(-1, 0, 5, NULL) |
+-------------------------+
| NULL                    |
+-------------------------+
```

```
SELECT GREATEST(6.3, 4.29, 7.6876);
```

```
+----------------------------+
| GREATEST(6.3, 4.29, 7.6876) |
```

```
+-----------------------------+
|                      7.6876 |
+-----------------------------+
```

```
SELECT GREATEST('2022-02-26 20:02:11', '2020-01-23 20:02:11', '2020-06-22 20:02:11');
```

```
+-------------------------------------------------------------------------------+
| GREATEST('2022-02-26 20:02:11', '2020-01-23 20:02:11', '2020-06-22 20:02:11') |
+-------------------------------------------------------------------------------+
| 2022-02-26 20:02:11                                                           |
+-------------------------------------------------------------------------------+
```

IF

Description

Returns `valueTrue` when the condition is true, and returns `valueFalseOrNull` otherwise. The return type is determined by the result of the `valueTrue`/`valueFalseOrNull` expression.

Syntax

```
IF(<condition>, <value_true>, <value_false_or_null>)
```

Parameters

Parameter	Description
`<condition>`	The boolean condition to evaluate.
`<value_true>`	The value to return if `<condition>` evaluates to true.
`<value_false_or_null>`	The value to return if `<condition>` evaluates to false.

Return Value

The result of the IF expression: - Returns `valueTrue` when the condition is true. - Returns `valueFalseOrNull` when the condition is false.

Examples

```
create table test(
    user_id int
) properties('replication_num' = '1');
insert into test values(1),(2);
```

```
SELECT user_id, IF(user_id = 1, 'true', 'false') AS test_if FROM test;
```

```
+---------+---------+
| user_id | test_if |
+---------+---------+
| 1       | true    |
| 2       | false   |
+---------+---------+
```

IFNULL

Description

Returns <expr1> if it is not NULL; otherwise, returns <expr2>.

Alias

- NVL

Syntax

```
IFNULL(<expr1>, <expr2>)
```

Parameters

Parameter	Description
<expr1>	The first expression to check for NULL.
<expr2>	The value to return if <expr1> is NULL.

Return Value

- Returns <expr1> if it is not NULL.
- Otherwise, returns <expr2>.

Examples

```
SELECT IFNULL(1, 0);
```

```
+--------------+
| IFNULL(1, 0) |
+--------------+
|            1 |
+--------------+
```

```
SELECT IFNULL(NULL, 10);
```

```
+------------------+
| IFNULL(NULL, 10) |
+------------------+
|               10 |
+------------------+
```

LEAST

Description

Compares multiple expressions and returns the smallest value among them. If any parameter is `NULL`, the function returns `NULL`.

Syntax

```
LEAST(<expr> [, ...])
```

Parameters

Parameter	Description
`<expr>`	The expressions to be compared. Supported types include `TINYINT`, `SMALLINT`, `INT`, `BIGINT`, `LARGEINT`, `FLOAT`, `DOUBLE`, `STRING`, `DATETIME`, and `DECIMAL`.

Return Value

- Returns the smallest value among the given expressions.
- If any parameter is `NULL`, the function returns `NULL`.

Examples

```
SELECT LEAST(-1, 0, 5, 8);
```

```
+--------------------+
| LEAST(-1, 0, 5, 8) |
+--------------------+
|                 -1 |
+--------------------+
```

```
SELECT LEAST(-1, 0, 5, NULL);
```

```
+-----------------------+
| LEAST(-1, 0, 5, NULL) |
+-----------------------+
| NULL                  |
+-----------------------+
```

```
SELECT LEAST(6.3, 4.29, 7.6876);
```

```
+--------------------------+
| LEAST(6.3, 4.29, 7.6876) |
+--------------------------+
|                     4.29 |
+--------------------------+
```

```
SELECT LEAST('2022-02-26 20:02:11', '2020-01-23 20:02:11', '2020-06-22 20:02:11');
```

```
+---------------------------------------------------------------------------+
| LEAST('2022-02-26 20:02:11', '2020-01-23 20:02:11', '2020-06-22 20:02:11') |
+---------------------------------------------------------------------------+
| 2020-01-23 20:02:11                                                       |
+---------------------------------------------------------------------------+
```

NOT_NULL_OR_EMPTY

Description

The `not_null_or_empty` function checks if the given value is neither NULL nor empty. It returns true if the input value is not NULL and not an empty string, otherwise, it returns false.

Syntax

```
NOT_NULL_OR_EMPTY (<str>)
```

Parameters

Parameter	Description
<str>	The string to check

Return Value

Returns false if the string is an empty string or NULL, otherwise returns true.

Examples

```
select not_null_or_empty(null);
```

```
+-------------------------+
| not_null_or_empty(NULL) |
+-------------------------+
|                       0 |
+-------------------------+
```

```
select not_null_or_empty("");
```

```
+-----------------------+
| not_null_or_empty('') |
+-----------------------+
|                     0 |
+-----------------------+
```

```
select not_null_or_empty("a");
```

```
+------------------------+
| not_null_or_empty('a') |
+------------------------+
```

```
|                    1 |
+----------------------+
```

NULL_OR_EMPTY

Description

The `null_or_empty` function checks if the given value is NULL or an empty string. It returns true if the input value is NULL or an empty string, otherwise, it returns false.

Syntax

```
NULL_OR_EMPTY (<str>)
```

Parameters

Parameter	Description
`<str>`	The string to check.

Return Value

Returns true if the string is an empty string or NULL, otherwise returns false.

Examples

```
select null_or_empty(null);
```

```
+---------------------+
| null_or_empty(NULL) |
+---------------------+
|                   1 |
+---------------------+
```

```
select null_or_empty("");
```

```
+-------------------+
| null_or_empty('') |
+-------------------+
|                 1 |
+-------------------+
```

```
select null_or_empty("a");
```

```
+--------------------+
| null_or_empty('a') |
+--------------------+
```

```
|                   0 |
+---------------------+
```

NULLIF

Description

Returns `NULL` if the two input values are equal; otherwise, returns the first input value. This function is equivalent to the following `CASE WHEN` expression:

```
CASE
    WHEN <expr1> = <expr2> THEN NULL
    ELSE <expr1>
END
```

Syntax

```
NULLIF(<expr1>, <expr2>)
```

Parameters

Parameter	Description
`<expr1>`	The first input value to compare.
`<expr2>`	The second input value to compare against the first.

Return Value

- Returns `NULL` if `<expr1>` is equal to `<expr2>`.
- Otherwise, returns the value of `<expr1>`.

Examples

```
SELECT NULLIF(1, 1);
```

```
+--------------+
| NULLIF(1, 1) |
+--------------+
|         NULL |
+--------------+
```

```
SELECT NULLIF(1, 0);
```

```
+--------------+
| NULLIF(1, 0) |
+--------------+
```

```
|           1 |
+-------------+
```

ANY_VALUE

Description

Returns any value from the expression or column in the group. If there is a non-NULL value, it returns any non-NULL value; otherwise, it returns NULL.

Alias

- ANY

Syntax

```
ANY_VALUE(<expr>)
```

Parameters

Parameter	Description
<expr>	The column or expression to be aggregated.

Return Value

Returns any non-NULL value if a non-NULL value exists, otherwise returns NULL.

Example

```
-- setup
create table cost2(id int, name varchar(20)) distributed by hash(id) buckets 1 properties ("replication_num"="1");
insert into cost2 values (2,'jack'),(3,'jack');
```

```
select id, any_value(name) from cost2 group by id;
```

```
+------+------------------+
| id   | any_value(`name`) |
+------+------------------+
|    3 | jack             |
|    2 | jack             |
+------+------------------+
```

APPROX_COUNT_DISTINCT

Description

The APPROX_COUNT_DISTINCT function is implemented based on the HyperLogLog algorithm, which uses a fixed size of memory to estimate the column base. The algorithm is based on the assumption of a null distribution in the tails, and the accuracy depends on the data distribution. Based on the fixed bucket size used by Doris, the relative standard error of the algorithm is 0.8125%. For a more detailed and specific analysis, see related paper

Syntax

```
APPROX_COUNT_DISTINCT(<expr>)
```

Parameters

Parameters	Description
<expr>	The expression needs to be obtained

Return Value

Returns a value of type BIGINT.

Example

```
select approx_count_distinct(query_id) from log_statis group by datetime;
```

```
+-----------------+
| approx_count_distinct(`query_id`) |
+-----------------+
| 17721           |
+-----------------+
```

ARRAY_AGG

Description

Concatenates the values (including null values) in a column into an array, which can be used for pivoting rows into columns.

Syntax

```
ARRAY_AGG(<col>)
```

Parameters

Parameter	Description
<col>	An expression that determines the values to be placed into the array (usually column names).

Return Value

Returns a value of ARRAY type.Special cases:

- The order of elements in the array is not guaranteed.
- Returns the array generated by the conversion. The element type in the array is consistent with the type of col.

Example

```
-- setup
create table test_doris_array_agg(c1 int, c2 varchar(20)) distributed by hash(c1) buckets 1 properties ("replication_num"="1");
insert into test_doris_array_agg values (1,'a'),(1,'b'),(2,'c'),(2,null),(3,null);
```

```
select * from test_doris_array_agg;
```

```
+------+------+

| c1   | c2   |

+------+------+

|    1 | a    |

|    1 | b    |

|    2 | c    |

|    2 | NULL |
```

```
|    3 | NULL |
+------+------+
```

```
select c1, array_agg(c2) from test_doris_array_agg group by c1;
```

```
+------+-----------------+
| c1   | array_agg(`c2`) |
+------+-----------------+
|    1 | ["a","b"]       |
|    2 | [NULL,"c"]      |
|    3 | [NULL]          |
+------+-----------------+
```

AVG

Description

Calculates the average of all non-NULL values in a specified column or expression.

Syntax

```
AVG([DISTINCT] <expr>)
```

Parameters

Parameter	Description
<expr>	It is an expression or column, typically a numeric column or an expression that can be converted to a numeric value.
[DISTINCT]	It is an optional keyword that indicates the calculation of the average value after removing duplicate values from expr.

Return Value

Returns the average value of the selected column or expression. If all records in the group are NULL, the function returns NULL.

Example

```
SELECT datetime, AVG(cost_time) FROM log_statis group by datetime;
```

```
+---------------------+--------------------+
| datetime            | avg(`cost_time`)   |
+---------------------+--------------------+
| 2019-07-03 21:01:20 | 25.827794561933533 |
+---------------------+--------------------+
```

```
SELECT datetime, AVG(distinct cost_time) FROM log_statis group by datetime;
```

```
+---------------------+---------------------------+
| datetime            | avg(DISTINCT `cost_time`) |
+---------------------+---------------------------+
| 2019-07-04 02:23:24 |        20.666666666666668 |
+---------------------+---------------------------+
```

AVG_WEIGHTED

Description

Calculates the weighted arithmetic mean, i.e., the result is the sum of the products of corresponding values and weights, divided by the total sum of weights. If the total sum of weights is 0, it will return NaN.

Syntax

```
AVG_WEIGHTED(<x>, <weight>)
```

Parameters

Parameter	Description
<x>	It is the numeric expression for which the average needs to be calculated, and can be a column name, constant, or a complex numeric expression.
<weight>	It is a numeric expression, typically a column name, constant, or the result of another numeric calculation.

Return Value

The sum of the products of corresponding values and weights is accumulated, divided by the total sum of weights. If the total sum of weights equals 0, NaN will be returned.

Example

```
-- setup
create table test_doris_avg_weighted(k1 int, k2 int) distributed by hash(k1) buckets 1 properties ("replication_num"="1");
insert into test_doris_avg_weighted values (10,100),(20,200),(30,300),(40,400);
```

```
select k1,k2 from test_doris_avg_weighted;
```

```
+------+------+
| k1   | k2   |
+------+------+
|   10 |  100 |
|   20 |  200 |
|   30 |  300 |
|   40 |  400 |
+------+------+
```

```
select avg_weighted(k2,k1) from test_doris_avg_weighted;
```

```
+-------------------------------------+
| avg_weighted(k2, cast(k1 as DOUBLE)) |
+-------------------------------------+
|                                 300 |
+-------------------------------------+
```

BITMAP_AGG

Description

Aggregate the values of a column (excluding any NULL values) and return a single row bitmap value, i.e., convert multiple rows into one.

Syntax

```
BITMAP_AGG(<expr>)
```

Parameters

Parameter	Description
<expr>	The column or expression of values to be aggregated. The type of expr must be TINYINT, SMALLINT, INT, LARGEINT, or BIGINT, and it also supports VARCHAR that can be converted to one of these types.

Return Value

Returns a BITMAP type value. Special cases:

- If a value is less than 0 or greater than 18446744073709551615, the value will be ignored and will not be merged into the Bitmap.

Example

```
create table test_bitmap_agg(
    id int,
    k0 int,
    k1 varchar(32),
    k2 int,
    k3 int,
    k4 bigint,
    k5 varchar(32)
) properties('replication_num' = '1');
insert into test_bitmap_agg values
  (1, 10, '110', 11, 300, 10000000000, '0'),
  (2, 20, '120', 21, 400, 20000000000, '200000000000000'),
  (3, 30, '130', 31, 350, 30000000000, '300000000000000'),
  (4, 40, '140', 41, 500, 40000000000, '18446744073709551616'),
  (5, 50, '150', 51, 250, 50000000000, '18446744073709551615'),
  (6, 60, '160', 61, 600, 60000000000, '-1'),
  (7, 60, '160', 120, 600, 60000000000, NULL);
```

```
select * from test_bitmap_agg;
```

```
+------+------+------+------+------+-------------+----------------------+
| id   | k0   | k1   | k2   | k3   | k4          | k5                   |
+------+------+------+------+------+-------------+----------------------+
|    1 |   10 | 110  |   11 |  300 | 10000000000 | 0                    |
|    2 |   20 | 120  |   21 |  400 | 20000000000 | 200000000000000      |
|    3 |   30 | 130  |   31 |  350 | 30000000000 | 300000000000000      |
|    4 |   40 | 140  |   41 |  500 | 40000000000 | 18446744073709551616 |
|    5 |   50 | 150  |   51 |  250 | 50000000000 | 18446744073709551615 |
|    6 |   60 | 160  |   61 |  600 | 60000000000 | -1                   |
|    7 |   60 | 160  |  120 |  600 | 60000000000 | NULL                 |
+------+------+------+------+------+-------------+----------------------+
```

```
select bitmap_to_string(bitmap_agg(k0)) from test_bitmap_agg;
```

```
+----------------------------------+
| bitmap_to_string(bitmap_agg(k0)) |
+----------------------------------+
| 10,20,30,40,50,60                |
+----------------------------------+
```

```
select bitmap_to_string(bitmap_agg(k5)) from test_bitmap_agg;
```

```
+-----------------------------------------------------------+
| bitmap_to_string(bitmap_agg(cast(k5 as BIGINT)))          |
+-----------------------------------------------------------+
| 0,200000000000000,300000000000000,18446744073709551615 |
+-----------------------------------------------------------+
```

BITMAP_INTERSECT

Description

Aggregation function, used to calculate the bitmap intersection after grouping. Common usage scenarios such as: calculating user retention rate.

Syntax

```
BITMAP_INTERSECT(BITMAP <value>)
```

Parameters

Parameter	Description
<value>	Supported bitmap data types

Return Value

The data type of the return value is BITMAP.

Example

```
-- setup
CREATE TABLE user_tags (
    tag VARCHAR(20),
    date DATETIME,
    user_id BITMAP bitmap_union
) AGGREGATE KEY(tag, date) DISTRIBUTED BY HASH(tag) BUCKETS 1
PROPERTIES ("replication_num" = "1");
INSERT INTO user_tags VALUES
    ('A', '2020-05-18', to_bitmap(1)),
    ('A', '2020-05-18', to_bitmap(2)),
    ('A', '2020-05-19', to_bitmap(2)),
    ('A', '2020-05-19', to_bitmap(3)),
    ('B', '2020-05-18', to_bitmap(4)),
    ('B', '2020-05-19', to_bitmap(4)),
    ('B', '2020-05-19', to_bitmap(5));
```

Table schema

```
KeysType: AGG_KEY
Columns: tag varchar, date datetime, user_id bitmap bitmap_union
```

```
Find the retention of users between 2020-05-18 and 2020-05-19 under different tags.
mysql> select tag, bitmap_intersect(user_id) from (select tag, date, bitmap_union(user_id) user_id from table where date in ('2020-
05-18', '2020-05-19') group by tag, date) a group by tag;
```

Used in combination with the bitmap_to_string function to obtain the specific data of the intersection

```
Who are the users retained under different tags between 2020-05-18 and 2020-05-19?
mysql> select tag, bitmap_to_string(bitmap_intersect(user_id)) from (select tag, date, bitmap_union(user_id) user_id from table
where date in ('2020-05-18', '2020-05-19') group by tag, date) a group by tag;
```

BITMAP_UNION

Description

Calculate the union of input Bitmaps and return a new bitmap

Syntax

```
BITMAP_UNION(<expr>)
```

Parameters

Parameter	Description
<expr>	Supported data types of BITMAP

Return Value

The data type of the return value is BITMAP.

Example

```
-- setup
CREATE TABLE pv_bitmap (
  dt INT,
  page VARCHAR(10),
  user_id BITMAP BITMAP_UNION
) AGGREGATE KEY(dt, page)
DISTRIBUTED BY HASH(dt) BUCKETS 1
PROPERTIES ("replication_num" = "1");
INSERT INTO pv_bitmap VALUES
  (1, '100', to_bitmap(100)),
  (1, '100', to_bitmap(200)),
  (1, '100', to_bitmap(300)),
  (2, '200', to_bitmap(300));
```

```
select dt,page,bitmap_to_string(user_id) from pv_bitmap;
```

```
+------+------+---------------------------+
| dt   | page | bitmap_to_string(user_id) |
+------+------+---------------------------+
|    1 | 100  | 100,200,300               |
|    2 | 200  | 300                       |
+------+------+---------------------------+
```

Calculate the deduplication value of user_id:

```
select bitmap_count(bitmap_union(user_id)) from pv_bitmap;
```

```
+-------------------------------------+
| bitmap_count(bitmap_union(user_id)) |
+-------------------------------------+
|                                   3 |
+-------------------------------------+
```

Create table

The aggregation model needs to be used when creating the table. The data type is bitmap and the aggregation function is bitmap_union.

```
CREATE TABLE `pv_bitmap` (
  `dt` int (11) NULL COMMENT" ",
  `page` varchar (10) NULL COMMENT" ",
  `user_id` bitmap BITMAP_UNION NULL COMMENT" "
) ENGINE = OLAP
AGGREGATE KEY (`dt`,` page`)
COMMENT "OLAP"
DISTRIBUTED BY HASH (`dt`) BUCKETS 2;
```

Note: When the amount of data is large, it is best to create a corresponding rollup table for high-frequency bitmap_union queries

```
ALTER TABLE pv_bitmap ADD ROLLUP pv (page, user_id);
```

Data Load

`TO_BITMAP (expr)`: Convert 0 ~ 18446744073709551615 unsigned bigint to bitmap

`BITMAP_EMPTY ()`: Generate empty bitmap columns, used for insert or import to fill the default value

`BITMAP_HASH (expr)` or `BITMAP_HASH64 (expr)`: Convert any type of column to a bitmap by hashing

Stream Load

```
cat data | curl --location-trusted -u user: passwd -T--H "columns: dt, page, user_id, user_id = to_bitmap (user_id)" http: // host: 8410 / api / test / testDb / _stream_load
```

```
cat data | curl --location-trusted -u user: passwd -T--H "columns: dt, page, user_id, user_id = bitmap_hash (user_id)" http: // host: 8410 / api / test / testDb / _stream_load
```

```
cat data | curl --location-trusted -u user: passwd -T--H "columns: dt, page, user_id, user_id = bitmap_empty ()" http: // host: 8410 / api / test / testDb / _stream_load
```

Insert Into

id2’s column type is bitmap

```
insert into bitmap_table1 select id, id2 from bitmap_table2;
```

id2’s column type is bitmap

```
INSERT INTO bitmap_table1 (id, id2) VALUES (1001, to_bitmap (1000)), (1001, to_bitmap (2000));
```

id2’s column type is bitmap

```
insert into bitmap_table1 select id, bitmap_union (id2) from bitmap_table2 group by id;
```

id2’s column type is int

```
insert into bitmap_table1 select id, to_bitmap (id2) from table;
```

id2’s column type is String

```
insert into bitmap_table1 select id, bitmap_hash (id_string) from table;
```

BITMAP_UNION_COUNT

Description

Computes the union of input Bitmaps and returns their cardinality.

Syntax

```
BITMAP_UNION_COUNT(<expr>)
```

Parameters

Parameter	Description
<expr>	Supported data types of BITMAP

Return Value

Returns the size of the Bitmap union, that is, the number of elements after deduplication

Example

```
-- setup
create table pv_bitmap(dt int, page int, user_id bitmap bitmap_union) aggregate key(dt,page) distributed by hash(dt) buckets 1
properties ("replication_num"="1");
insert into pv_bitmap values (1,100,bitmap_from_string('100,200,300')),(2,200,bitmap_from_string('300'));
```

```
select dt,page,bitmap_to_string(user_id) from pv_bitmap;
```

```
+------+------+---------------------------+
| dt   | page | bitmap_to_string(user_id) |
+------+------+---------------------------+
|    1 | 100  | 100,200,300               |
|    2 | 200  | 300                       |
+------+------+---------------------------+
```

Calculate the deduplication value of user_id:

```
select bitmap_union_count(user_id) from pv_bitmap;
```

```
+-------------------------------------+
| bitmap_count(bitmap_union(user_id)) |
+-------------------------------------+
|                                   3 |
+-------------------------------------+
```

BITMAP-UNION-INT

Description

Counts the number of distinct values in columns of type TINYINT, SMALLINT and INT. The return value is the same as COUNT(DISTINCT expr)

Syntax

```
BITMAP_UNION_INT(<expr>)
```

Parameters

Parameter	Description
<expr>	Supports columns or column expressions of type TINYINT, SMALLINT and INT

Return Value

Returns the number of distinct values in a column.

Example

```
-- setup
create table pv_bitmap(dt int, page int, user_id bitmap bitmap_union) aggregate key(dt,page) distributed by hash(dt) buckets 1
properties ("replication_num"="1");
insert into pv_bitmap values (1,100,bitmap_from_string('100,200,300')),(1,300,bitmap_from_string('300')),
(2,200,bitmap_from_string('300'));
```

```
select dt,page,bitmap_to_string(user_id) from pv_bitmap;
```

```
+------+------+---------------------------+
| dt   | page | bitmap_to_string(user_id) |
+------+------+---------------------------+
|    1 | 100  | 100,200,300               |
|    1 | 300  | 300                       |
|    2 | 200  | 300                       |
+------+------+---------------------------+
```

```
select bitmap_union_int(dt) from pv_bitmap;
```

```
+----------------------+
| bitmap_union_int(dt) |
+----------------------+
```

```
|                    2 |
+----------------------+
```

COLLECT_LIST

Description

Aggregation function, used to aggregate all values of a column into an array.

Alias

- GROUP_ARRAY

Syntax

```
COLLECT_LIST(<expr> [,<max_size>])
```

Parameters

Parameter	Description
<expr>	Column or expression to aggregate
<max_size>	Optional parameter that can be set to limit the size of the resulting array to max_size elements

Return Value

The return type is ARRAY, which contains all values. Special circumstances:

- If the value is NULL, it will filter

Example

```
select k1,k2,k3 from collect_list_test order by k1;
```

```
+------+------------+-------+
| k1   | k2         | k3    |
+------+------------+-------+
|    1 | 2023-01-01 | hello |
|    2 | 2023-01-02 | NULL  |
|    2 | 2023-01-02 | hello |
|    3 | NULL       | world |
|    3 | 2023-01-02 | hello |
|    4 | 2023-01-02 | sql   |
|    4 | 2023-01-03 | sql   |
+------+------------+-------+
```

```
select collect_list(k1),collect_list(k1,3) from collect_list_test;
```

```
+-------------------------+-------------------------+
| collect_list(`k1`)      | collect_list(`k1`,3)    |
+-------------------------+-------------------------+
| [1,2,2,3,3,4,4]         | [1,2,2]                 |
+-------------------------+-------------------------+
```

```
select k1,collect_list(k2),collect_list(k3,1) from collect_list_test group by k1 order by k1;
```

```
+------+-------------------------+-------------------------+
| k1   | collect_list(`k2`)      | collect_list(`k3`,1)    |
+------+-------------------------+-------------------------+
|    1 | [2023-01-01]            | [hello]                 |
|    2 | [2023-01-02,2023-01-02] | [hello]                 |
|    3 | [2023-01-02]            | [world]                 |
|    4 | [2023-01-02,2023-01-03] | [sql]                   |
+------+-------------------------+-------------------------+
```

COLLECT_SET

Description

Aggregation function aggregates all unique values of the specified column, removes duplicate elements, and returns a set type result.

Alias

- GROUP_UNIQ_ARRAY

Syntax

```
COLLECT_SET(<expr> [,<max_size>])
```

Parameters

Parameter	Description
<expr>	Column or expression to aggregate
<max_size>	Optional parameter that can be set to limit the size of the resulting array to max_size elements

Return Value

The return type is ARRAY. This array contains all values after deduplication. Special case:

- If the value is NULL, it will filter

Example

```
select k1,k2,k3 from collect_set_test order by k1;
```

```
+------+------------+-------+
| k1   | k2         | k3    |
+------+------------+-------+
|    1 | 2023-01-01 | hello |
|    2 | 2023-01-01 | NULL  |
|    2 | 2023-01-02 | hello |
|    3 | NULL       | world |
|    3 | 2023-01-02 | hello |
|    4 | 2023-01-02 | doris |
|    4 | 2023-01-03 | sql   |
+------+------------+-------+
```

```
select collect_set(k1),collect_set(k1,2) from collect_set_test;
```

```
+-------------------------+-------------------------+
| collect_set(`k1`)       | collect_set(`k1`,2)     |
+-------------------------+-------------------------+
| [4,3,2,1]               | [1,2]                   |
+-------------------------------------------------+
```

```
select k1,collect_set(k2),collect_set(k3,1) from collect_set_test group by k1 order by k1;
```

```
+------+-------------------------+-------------------------+
| k1   | collect_set(`k2`)       | collect_set(`k3`,1)     |
+------+-------------------------+-------------------------+
|    1 | [2023-01-01]            | [hello]                 |
|    2 | [2023-01-01,2023-01-02] | [hello]                 |
|    3 | [2023-01-02]            | [world]                 |
|    4 | [2023-01-02,2023-01-03] | [sql]                   |
+------+-------------------------+-------------------------+
```

CORR

Description

Calculate the Pearson coefficient of two random variables.

Syntax

```
CORR(<expr1>, <expr2>)
```

Parameters

Parameter	Description
<expr1>	Numeric expression (column)
<expr2>	Numeric expression (column)

Return Value

The return value is of type DOUBLE, the covariance of expr1 and expr2, except the product of the standard deviation of expr1 and expr2, special case:

- If the standard deviation of expr1 or expr2 is 0, 0 will be returned.
- If a column of expr1 or expr2 is NULL, the row data will not be counted in the final result.

Example

```
-- setup
create table test_corr(
    id int,
    k1 double,
    k2 double
) distributed by hash (id) buckets 1
properties ("replication_num"="1");

insert into test_corr values
    (1, 20, 22),
    (1, 10, 20),
    (2, 36, 21),
    (2, 30, 22),
    (2, 25, 20),
    (3, 25, NULL),
    (4, 25, 21),
    (4, 25, 22),
    (4, 25, 20);
```

```
select * from test_corr;
```

```
+------+------+------+
| id   | k1   | k2   |
+------+------+------+
|    1 |   20 |   22 |
|    1 |   10 |   20 |
|    2 |   36 |   21 |
|    2 |   30 |   22 |
|    2 |   25 |   20 |
|    3 |   25 | NULL |
|    4 |   25 |   21 |
|    4 |   25 |   22 |
|    4 |   25 |   20 |
+------+------+------+
```

```
select id,corr(k1,k2) from test_corr group by id;
```

```
+------+--------------------+
| id   | corr(k1, k2)       |
+------+--------------------+
|    4 |                  0 |
|    1 |                  1 |
|    3 |               NULL |
|    2 | 0.4539206495016019 |
+------+--------------------+
```

COUNT

Description

Returns the number of non-NULL records in the specified column, or the total number of records.

Syntax

```
COUNT(DISTINCT <expr> [,<expr>,...])
COUNT(*)
COUNT(<expr>)
```

Parameters

Parameter	Description
<expr>	Conditional expression (column name)

Return Value

The return value is of numeric type. If expr is NULL, there will be no parameter statistics.

Example

```
-- setup
create table test_count(
    id int,
    name varchar(20),
    sex int
) distributed by hash(id) buckets 1
properties ("replication_num"="1");

insert into test_count values
    (1, '1', 1),
    (2, '2', 1),
    (3, '3', 1),
    (4, '0', 1),
    (4, '4', 1),
    (5, NULL, 1);

create table test_insert(
    id int,
    name varchar(20),
    sex int
) distributed by hash(id) buckets 1
properties ("replication_num"="1");

insert into test_insert values
    (1, '1', 1),
    (2, '2', 1),
    (3, '3', 1),
    (4, '0', 1),
```

```
    (4, '4', 1),
    (5, NULL, 1);
```

```
select * from test_count;
```

```
+------+------+------+
| id   | name | sex  |
+------+------+------+
|    1 | 1    |    1 |
|    2 | 2    |    1 |
|    3 | 3    |    1 |
|    4 | 0    |    1 |
|    4 | 4    |    1 |
|    5 | NULL |    1 |
+------+------+------+
```

```
select count(*) from test_count;
```

```
+----------+
| count(*) |
+----------+
|        6 |
+----------+
```

```
select count(name) from test_insert;
```

```
+-------------+
| count(name) |
+-------------+
|           5 |
+-------------+
```

```
select count(distinct sex) from test_insert;
```

```
+---------------------+
| count(DISTINCT sex) |
+---------------------+
|                   1 |
+---------------------+
```

```
select count(distinct id,sex) from test_insert;
```

```
+-------------------------+
| count(DISTINCT id, sex) |
+-------------------------+
|                       5 |
+-------------------------+
```

COUNT_BY_ENUM

Description

Treat the data in the column as enumeration values and count the number of each enumeration value. Returns the number of enumeration values for each column, as well as the number of non-null values and the number of null values.

Syntax

```
COUNT_BY_ENUM(<expr1>, <expr2>, ... , <exprN>)
```

Parameters

Parameter	Description
<expr1>	Fill in at least one input. Column whose value is of type STRING

Returned value

Returns a JSONArray string.

For example:

```
[{
  "cbe": {
    "F": 100,
    "M": 99
  },
  "notnull": 199,
  "null": 1,
  "all": 200
}, {
  "cbe": {
    "20": 10,
    "30": 5,
    "35": 1
  },
  "notnull": 16,
  "null": 184,
  "all": 200
}, {
  "cbe": {
    "China": 10,
    "United States": 9,
    "England": 20,
    "Germany": 30
  },
  "notnull": 69,
  "null": 131,
```

```
  "all": 200
}]
```

Description: The return value is a JSON array string and the order of the internal objects is the order of the input parameters. * cbe: count of non-null values based on enumeration values * notnull: number of non-null values. * null: number of null values * all: total number, including both null and non-null values.

Example

```
CREATE TABLE count_by_enum_test(
                                  `id` varchar(1024) NULL,
                                  `f1` text REPLACE_IF_NOT_NULL NULL,
                                  `f2` text REPLACE_IF_NOT_NULL NULL,
                                  `f3` text REPLACE_IF_NOT_NULL NULL
)
AGGREGATE KEY(`id`)
DISTRIBUTED BY HASH(id) BUCKETS 3
PROPERTIES (
    "replication_num" = "1"
);
```

```
INSERT into count_by_enum_test (id, f1, f2, f3) values
                                                  (1, "F", "10", "China"),
                                                  (2, "F", "20", "China"),
                                                  (3, "M", NULL, "United States"),
                                                  (4, "M", NULL, "United States"),
                                                  (5, "M", NULL, "England");
```

```
SELECT * from count_by_enum_test;
```

```
+------+------+------+---------------+
| id   | f1   | f2   | f3            |
+------+------+------+---------------+
| 1    | F    | 10   | China         |
| 2    | F    | 20   | China         |
| 3    | M    | NULL | United States |
| 4    | M    | NULL | United States |
| 5    | M    | NULL | England       |
+------+------+------+---------------+
```

```
select count_by_enum(f1) from count_by_enum_test;
```

```
+-----------------------------------------------------+
| count_by_enum(`f1`)                                 |
+-----------------------------------------------------+
| [{"cbe":{"M":3,"F":2},"notnull":5,"null":0,"all":5}] |
+-----------------------------------------------------+
```

```
select count_by_enum(f2) from count_by_enum_test;
```

```
+-----------------------------------------------------+
| count_by_enum(`f2`)                                 |
```

```
+-------------------------------------------------------+
| [{"cbe":{"10":1,"20":1},"notnull":2,"null":3,"all":5}] |
+-------------------------------------------------------+
```

```
select count_by_enum(f1,f2,f3) from count_by_enum_test;
```

```
+-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| count_by_enum(`f1`, `f2`, `f3`)
|
+-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
| [{"cbe":{"M":3,"F":2},"notnull":5,"null":0,"all":5},{"cbe":{"20":1,"10":1},"notnull":2,"null":3,"all":5},{"cbe":
{"England":1,"United States":2,"China":2},"notnull":5,"null":0,"all":5}] |
+-------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------+
```

COVAR,COVAR_POP

Description

Calculate the covariance between two numeric variables.

Alias

- COVAR_POP

Syntax

```
COVAR(<expr1>, <expr2>)
```

Parameters

Parameter	Description
<expr1>	Numeric expression or column
<expr2>	Numeric expression or column

Return Value

Returns the covariance value of expr1 and expr2, special case:

- If a column of expr1 or expr2 is NULL, the row data will not be counted in the final result.

Example

```
-- setup
create table baseall(
    id int,
    x double,
    y double
) distributed by hash(id) buckets 1
properties ("replication_num"="1");

insert into baseall values
    (1, 1.0, 2.0),
    (2, 2.0, 3.0),
    (3, 3.0, 4.0),
    (4, 4.0, NULL),
    (5, NULL, 5.0);
```

```
select covar(x,y) from baseall;
```

```
+--------------------+
| covar(x, y)          |
+--------------------+
| 0.89442719099991586 |
+--------------------+
```

COVAR_SAMP

Description

Computes the sample covariance between two numeric variables.

Syntax

```
COVAR_SAMP(<expr1>, <expr2>)
```

Parameters

Parameter	Description
<expr1>	Numeric expression or column
<expr2>	Numeric expression or column

Return Value

Returns the sample covariance of expr1 and expr2, special case:

- If a column of expr1 or expr2 is NULL, the row data will not be counted in the final result.

Example

```
-- setup
create table baseall(
    id int,
    x double,
    y double
) distributed by hash(id) buckets 1
properties ("replication_num"="1");

insert into baseall values
    (1, 1.0, 2.0),
    (2, 2.0, 3.0),
    (3, 3.0, 4.0),
    (4, 4.0, NULL),
    (5, NULL, 5.0);
```

```
select covar_samp(x,y) from baseall;
```

```
+--------------------+
| covar_samp(x, y)   |
+--------------------+
| 0.89442719099991586 |
+--------------------+
```

GROUP_ARRAY_INTERSECT

Description

Calculate the intersection elements of the input array across all rows and return a new array.

Syntax

```
GROUP_ARRAY_INTERSECT(<expr>)
```

Parameters

Parameter	Description
<expr>	Array columns or array values that require intersection

Return Value

Returns an array containing the intersection results

Example

```
select c_array_string from group_array_intersect_test where id in (18, 20);
```

```
+------+---------------------------+
| id   | col                       |
+------+---------------------------+
|    1 | ["a", "b", "c", "d", "e"] |
|    2 | ["a", "b"]                |
|    3 | ["a", null]               |
+------+---------------------------+
```

```
select group_array_intersect(col) from group_array_intersect_test;
```

```
+---------------------------+
| group_array_intersect(col) |
+---------------------------+
| ["a"]                     |
+---------------------------+
```

GROUP_BIT_AND

Description

Performs a bitwise AND operation on all values in a single integer column or expression.

Syntax

```
GROUP_BIT_AND(<expr>)
```

Parameters

Parameter	Description
<expr>	Supports all INT types

Return Value

Returns an integer value.

Example

```
-- setup
create table group_bit(value int) distributed by hash(value) buckets 1 properties ("replication_num"="1");
insert into group_bit values (3),(1),(2),(4);
```

```
select * from group_bit;
```

```
+-------+
| value |
+-------+
|     3 |
|     1 |
|     2 |
|     4 |
+-------+
```

```
select group_bit_and(value) from group_bit;
```

```
+------------------------+
| group_bit_and(`value`) |
+------------------------+
|                      0 |
+------------------------+
```

GROUP_BIT_OR

Description

Performs a bitwise OR operation on all values in a single integer column or expression.

Syntax

```
GROUP_BIT_OR(<expr>)
```

Parameters

Parameter	Description
<expr>	Supports all INT types

Return Value

Returns an integer value

Example

```
-- setup
create table group_bit(value int) distributed by hash(value) buckets 1 properties ("replication_num"="1");
insert into group_bit values (3),(1),(2),(4);
```

```
select * from group_bit;
```

```
+-------+
| value |
+-------+
|     3 |
|     1 |
|     2 |
|     4 |
+-------+
```

```
mysql> select group_bit_or(value) from group_bit;
```

```
+-----------------------+
| group_bit_or(`value`) |
+-----------------------+
|                     7 |
+-----------------------+
```

GROUP_BIT_XOR

Description

Performs a bitwise xor operation on all values in a single integer column or expression.

Syntax

```
GROUP_BIT_XOR(<expr>)
```

Parameters

Parameter	Description
<expr>	Supports all INT types

Return Value

Returns an integer value

Example

```
-- setup
create table group_bit(value int) distributed by hash(value) buckets 1 properties ("replication_num"="1");
insert into group_bit values (3),(1),(2),(4);
```

```
select * from group_bit;
```

```
+-------+
| value |
+-------+
|     3 |
|     1 |
|     2 |
|     4 |
+-------+
```

```
select group_bit_xor(value) from group_bit;
```

```
+------------------------+
| group_bit_xor(`value`) |
+------------------------+
|                      4 |
+------------------------+
```

GROUP_BITMAP_XOR

Description

Mainly used to merge the values of multiple bitmaps and perform bitwise xor calculations on the results.

Syntax

```
GROUP_BITMAP_XOR(<expr>)
```

Parameters

Parameter	Description
<expr>	Supported bitmap data types

Return Value

The data type of the return value is BITMAP.

Example

```
-- setup
create table pv_bitmap(id int, page varchar(10), user_id bitmap bitmap_union) aggregate key(id,page) distributed by hash(id) buckets
1 properties ("replication_num"="1");
insert into pv_bitmap values (1,'m',bitmap_from_string('4,7,8')),(2,'m',bitmap_from_string('1,3,6,15')),
(3,'m',bitmap_from_string('4,7'));
```

```
 select page, bitmap_to_string(user_id) from pv_bitmap;
```

```
+------+-----------------------------+
| page | bitmap_to_string(`user_id`) |
+------+-----------------------------+
| m    | 4,7,8                       |
| m    | 1,3,6,15                    |
| m    | 4,7                         |
+------+-----------------------------+
```

```
select page, bitmap_to_string(group_bitmap_xor(user_id)) from pv_bitmap group by page;
```

```
+------+-----------------------------------------------+
| page | bitmap_to_string(group_bitmap_xor(`user_id`)) |
+------+-----------------------------------------------+
| m    | 1,3,6,8,15                                    |
+------+-----------------------------------------------+
```

GROUP_CONCAT

Description

The GROUP_CONCAT function concatenates multiple rows of results in the result set into a string.

Syntax

```
GROUP_CONCAT([DISTINCT] <str>[, <sep>] [ORDER BY { <col_name> | <expr>} [ASC | DESC]])
```

Parameters

Parameters	Description
<str>	Required. The expression of the value to be concatenated.
<sep>	Optional. The separator between strings.
<col_name>	Optional. The column used for sorting.
<expr>	Optional. The expression used for sorting.

Return Value

Returns a value of type VARCHAR.

Example

```
-- setup
create table test(
    value varchar(10)
) distributed by hash(value) buckets 1
properties ("replication_num"="1");

insert into test values
    ("a"),
    ("b"),
    ("c"),
    ("c");
```

```
select value from test;
```

```
+-------+
| value |
+-------+
| a     |
| b     |
| c     |
```

```
| c     |
+-------+
```

```
select GROUP_CONCAT(value) from test;
```

```
+----------------------+
| GROUP_CONCAT(`value`) |
+----------------------+
| a, b, c, c           |
+----------------------+
```

```
select GROUP_CONCAT(DISTINCT value) from test;
```

```
+----------------------+
| GROUP_CONCAT(`value`) |
+----------------------+
| a, b, c              |
+----------------------+
```

```
select GROUP_CONCAT(value ORDER BY value DESC) from test;
```

```
+----------------------+
| GROUP_CONCAT(`value`) |
+----------------------+
| c, c, b, a           |
+----------------------+
```

> **Combining `DISTINCT` with `ORDER BY` inside `GROUP_CONCAT` is supported since 2.1.6.**

```
select GROUP_CONCAT(DISTINCT value ORDER BY value DESC) from test;
```

```
+----------------------+
| GROUP_CONCAT(`value`) |
+----------------------+
| c, b, a              |
+----------------------+
```

```
select GROUP_CONCAT(value, " ") from test;
```

```
+---------------------------+
| GROUP_CONCAT(`value`, ' ') |
+---------------------------+
| a b c c                   |
+---------------------------+
```

```
select GROUP_CONCAT(value, NULL) from test;
```

```
+---------------------------+
| GROUP_CONCAT(`value`, NULL)|
+---------------------------+
| NULL                      |
+---------------------------+
```

GROUPING

Description

Indicates whether a specified column expression in a GROUP BY list is aggregated or not. GROUPING returns 1 if the column is aggregated (i.e., coming from a summary row generated by ROLLUP, CUBE, or GROUPING SETS) and 0 if it is not aggregated. This function is useful for distinguishing the NULL values produced by these operations from actual NULL values in the data.

Syntax

```
GROUPING( <column_expression> )
```

Parameters

Parameter	Description
<column_expression>	A column or an expression that appears in the GROUP BY clause.

Return Value

Returns a BIGINT value. The function returns 1 if the specified column expression is aggregated and 0 if it is not.

Examples

The following example groups the `camp` column and aggregates the count of `occupation`. The GROUPING function is applied to the `camp` column to differentiate between the summary row added by the ROLLUP operation and actual NULL values in the data.

```
CREATE TABLE `roles` (
  role_id       INT,
  occupation    VARCHAR(32),
  camp          VARCHAR(32),
  register_time DATE
)
UNIQUE KEY(role_id)
DISTRIBUTED BY HASH(role_id) BUCKETS 1
PROPERTIES (
  "replication_allocation" = "tag.location.default: 1"
);

INSERT INTO `roles` VALUES
(0, 'who am I', NULL, NULL),
(1, 'mage', 'alliance', '2018-12-03 16:11:28'),
(2, 'paladin', 'alliance', '2018-11-30 16:11:28'),
(3, 'rogue', 'horde', '2018-12-01 16:11:28'),
(4, 'priest', 'alliance', '2018-12-02 16:11:28'),
(5, 'shaman', 'horde', NULL),
```

```
(6, 'warrior', 'alliance', NULL),
(7, 'warlock', 'horde', '2018-12-04 16:11:28'),
(8, 'hunter', 'horde', NULL);

SELECT
  camp,
  COUNT(occupation) AS occ_cnt,
  GROUPING(camp) AS grouping
FROM
  `roles`
GROUP BY
  ROLLUP(camp);
```

The result set shows two NULL values under the `camp` column. The first NULL, with a grouping value of 1, is the summary row added by the ROLLUP operation representing the aggregate of all `camp` groups. The second NULL, with a grouping value of 0, represents an actual NULL value from the data.

```
+----------+---------+----------+
| camp     | occ_cnt | grouping |
+----------+---------+----------+
| NULL     |       9 |        1 |
| NULL     |       1 |        0 |
| alliance |       4 |        0 |
| horde    |       4 |        0 |
+----------+---------+----------+
```

GROUPING_ID

Description

Computes the level of grouping for rows in a GROUP BY query. The GROUPING_ID function returns an integer bitmap indicating which columns in the GROUP BY list are not aggregated for a given output row. It can be used in the SELECT list, HAVING, or ORDER BY clauses when a GROUP BY is specified.

Syntax

```
GROUPING_ID(<column_expression> [, ...])
```

Parameters

Parameters	Description
<column_expression>	A column expression from the GROUP BY clause.

Return Value

Returns a BIGINT value representing the grouping bitmap for the given columns.

Examples

Example A: Identify grouping levels

```
SELECT
  department,
  CASE
    WHEN GROUPING_ID(department, level) = 0 THEN level
    WHEN GROUPING_ID(department, level) = 1 THEN CONCAT('Total: ', department)
    WHEN GROUPING_ID(department, level) = 3 THEN 'Total: Company'
    ELSE 'Unknown'
  END AS `Job Title`,
  COUNT(uid) AS `Employee Count`
FROM employee
GROUP BY ROLLUP(department, level)
ORDER BY GROUPING_ID(department, level) ASC;
```

Expected Output:

```
+-------------------+--------------------------+----------------+
| department        | Job Title                | Employee Count |
+-------------------+--------------------------+----------------+
| Board of Directors | Senior                   |              2 |
| Technology        | Senior                   |              3 |
| Sales             | Senior                   |              1 |
| Sales             | Assistant                |              2 |
```

```
| Sales              | Trainee                  |              1 |
| Marketing          | Senior                   |              1 |
| Marketing          | Trainee                  |              2 |
| Marketing          | Assistant                |              1 |
| Board of Directors | Total: Board of Directors |              2 |
| Technology         | Total: Technology        |              3 |
| Sales              | Total: Sales             |              4 |
| Marketing          | Total: Marketing         |              4 |
| NULL               | Total: Company           |             13 |
+--------------------+--------------------------+----------------+
```

Example B: Filter result set using GROUPING_ID

```
SELECT
  department,
  CASE
    WHEN GROUPING_ID(department, level) = 0 THEN level
    WHEN GROUPING_ID(department, level) = 1 THEN CONCAT('Total: ', department)
    WHEN GROUPING_ID(department, level) = 3 THEN 'Total: Company'
    ELSE 'Unknown'
  END AS `Job Title`,
  COUNT(uid) AS `Count`
FROM employee
GROUP BY ROLLUP(department, level)
HAVING `Job Title` = 'Senior';
```

Expected Output:

```
+--------------------+-----------+-------+
| department         | Job Title | Count |
+--------------------+-----------+-------+
| Board of Directors | Senior    |     2 |
| Technology         | Senior    |     3 |
| Sales              | Senior    |     1 |
| Marketing          | Senior    |     1 |
+--------------------+-----------+-------+
```

HISTOGRAM

Description

The histogram function is used to describe the distribution of the data. It uses an “equal height” bucking strategy, and divides the data into buckets according to the value of the data. It describes each bucket with some simple data, such as the number of values that fall in the bucket.

Alias

HIST

Syntax

```
HISTOGRAM(<expr>[, <num_buckets>])
```

Parameters

Parameters	Description
expr	The expression that needs to be obtained.
num_buckets	Optional. Limit the number of histogram buckets. The default value is 128.

Return Value

Returns a value of JSON type after histogram estimation. Special cases: - When the parameter is NULL, it returns NULL.

Example

```
SELECT histogram(c_float) FROM histogram_test;
```

```
+-----------------------------------------------------------------------------------------------------------------------------------------
--+
| histogram(`c_float`)
|
+-----------------------------------------------------------------------------------------------------------------------------------------
--+
| {"num_buckets":3,"buckets":[{"lower":"0.1","upper":"0.1","count":1,"pre_sum":0,"ndv":1},...]} |
+-----------------------------------------------------------------------------------------------------------------------------------------
--+
```

```
SELECT histogram(c_string, 2) FROM histogram_test;
```

```
+-----------------------------------------------------------------------------------------------------------------------------------
--+
| histogram(`c_string`)
|
+-----------------------------------------------------------------------------------------------------------------------------------
--+
| {"num_buckets":2,"buckets":[{"lower":"str1","upper":"str7","count":4,"pre_sum":0,"ndv":3},...]} |
+-----------------------------------------------------------------------------------------------------------------------------------
--+
```

Query result description:

```
{
    "num_buckets": 3,
    "buckets": [
        {
            "lower": "0.1",
            "upper": "0.2",
            "count": 2,
            "pre_sum": 0,
            "ndv": 2
        },
        {
            "lower": "0.8",
            "upper": "0.9",
            "count": 2,
            "pre_sum": 2,
            "ndv": 2
        },
        {
            "lower": "1.0",
            "upper": "1.0",
            "count": 2,
            "pre_sum": 4,
            "ndv": 1
        }
    ]
}
```

```
Field description:
- num_buckets: The number of buckets
- buckets: All buckets
    - lower: Upper bound of the bucket
    - upper: Lower bound of the bucket
    - count: The number of elements contained in the bucket
    - pre_sum: The total number of elements in the front bucket
    - ndv: The number of different values in the bucket

> Total number of histogram elements = number of elements in the last bucket(count) + total number of elements in the previous
bucket(pre_sum).
```

HLL_RAW_AGG

Description

The HLL_RAW_AGG function is an aggregate function, which is mainly used to merge multiple HyperLogLog data structures.

Alias

- HLL_UNION

Syntax

```
HLL_RAW_AGG(<hll>)
```

Parameters

Parameters	Description
<hll>	The HyperLogLog type expression to be calculated

Return Value

Returns the aggregated value of type HyperLogLog.

Example

```
select HLL_CARDINALITY(HLL_RAW_AGG(uv_set)) from test_uv;
```

```
+----------------------------------------+
|   HLL_CARDINALITY(HLL_RAW_AGG(`uv_set`)) |
+----------------------------------------+
|                                  17721 |
+----------------------------------------+
```

HLL_UNION_AGG

Description

The HLL_UNION_AGG function is an aggregate function, which is mainly used to merge multiple HyperLogLog data structures and estimate the approximate value of the combined cardinality.

Syntax

```
hll_union_agg(<hll>)
```

Parameters

Parameters	Description
<hll>	The HyperLogLog type expression to be calculated

Return Value

Returns the cardinality value of type BIGINT.

Example

```
select HLL_UNION_AGG(uv_set) from test_uv;
```

```
+------------------------+
| HLL_UNION_AGG(`uv_set`) |
+------------------------+
| 17721                  |
+------------------------+
```

INTERSECT_COUNT

Description

The INTERSECT_COUNT function is used to calculate the number of intersecting elements of the Bitmap data structure.

Syntax

```
INTERSECT_COUNT(<bitmap_column>, <column_to_filter>, <filter_values>)
```

Parameters

Parameters	Description
<bitmap_column>	The expression that needs to be obtained.
<column_to_filter>	Optional. The dimension column that needs to be filtered.
<filter_values>	Optional. Different values of the filtering dimension column.

Return Value

Returns a value of type BIGINT.

Example

```
-- setup
create table pv_bitmap(dt int, user_id bitmap bitmap_union) aggregate key(dt) distributed by hash(dt) buckets 1 properties
("replication_num"="1");
insert into pv_bitmap values (4,bitmap_from_string('1,2,3')),(3,bitmap_from_string('1,2,3,4,5'));
```

```
select dt,bitmap_to_string(user_id) from pv_bitmap where dt in (3,4);
```

```
+------+-----------------------------+
| dt   | bitmap_to_string(`user_id`) |
+------+-----------------------------+
| 4    | 1,2,3                       |
| 3    | 1,2,3,4,5                   |
+------+-----------------------------+
```

```
select intersect_count(user_id,dt,3,4) from pv_bitmap;
```

```
+---------------------------------------+
| intersect_count(`user_id`, `dt`, 3, 4) |
+---------------------------------------+
```

```
|                                    3 |
+--------------------------------------+
```

MAP_AGG

Description

The MAP_AGG function is used to form a mapping structure based on key-value pairs from multiple rows of data.

Syntax

```
MAP_AGG(<expr1>, <expr2>)
```

Parameters

Parameters	Description
<expr1>	The expression used to specify the key.
<expr2>	The expression used to specify the corresponding value.

Return Value

Returns a value of the MAP type.

Example

```
create table nation(
    n_nationkey int,
    n_name varchar(32),
    n_regionkey int
) properties('replication_num' = '1');
insert into nation values
  (0,'ALGERIA',0),(1,'ARGENTINA',1),(2,'BRAZIL',1),(3,'CANADA',1),(4,'EGYPT',4),
  (5,'ETHIOPIA',0),(6,'FRANCE',3),(7,'GERMANY',3),(8,'INDIA',2),(9,'INDONESIA',2),
  (10,'IRAN',4),(11,'IRAQ',4),(12,'JAPAN',2),(13,'JORDAN',4),(14,'KENYA',0),
  (15,'MOROCCO',0),(16,'MOZAMBIQUE',0),(17,'PERU',1),(18,'CHINA',2),(19,'ROMANIA',3),
  (20,'SAUDI ARABIA',4),(21,'VIETNAM',2),(22,'RUSSIA',3),(23,'UNITED KINGDOM',3),(24,'UNITED STATES',1);
```

```
select `n_nationkey`, `n_name`, `n_regionkey` from `nation`;
```

```
+-------------+----------------+-------------+
| n_nationkey | n_name         | n_regionkey |
+-------------+----------------+-------------+
|           0 | ALGERIA        |           0 |
|           1 | ARGENTINA      |           1 |
|           2 | BRAZIL         |           1 |
|           3 | CANADA         |           1 |
|           4 | EGYPT          |           4 |
|           5 | ETHIOPIA       |           0 |
|           6 | FRANCE         |           3 |
|           7 | GERMANY        |           3 |
```

```
|           8 | INDIA          |           2 |
|           9 | INDONESIA      |           2 |
|          10 | IRAN           |           4 |
|          11 | IRAQ           |           4 |
|          12 | JAPAN          |           2 |
|          13 | JORDAN         |           4 |
|          14 | KENYA          |           0 |
|          15 | MOROCCO        |           0 |
|          16 | MOZAMBIQUE     |           0 |
|          17 | PERU           |           1 |
|          18 | CHINA          |           2 |
|          19 | ROMANIA        |           3 |
|          20 | SAUDI ARABIA   |           4 |
|          21 | VIETNAM        |           2 |
|          22 | RUSSIA         |           3 |
|          23 | UNITED KINGDOM |           3 |
|          24 | UNITED STATES  |           1 |
+-------------+----------------+-------------+
```

```
select `n_regionkey`, map_agg(`n_nationkey`, `n_name`) from `nation` group by `n_regionkey`;
```

```
+-------------+-------------------------------------------------------------------------------+
| n_regionkey | map_agg(`n_nationkey`, `n_name`)                                              |
+-------------+-------------------------------------------------------------------------------+
|           1 | {1:"ARGENTINA", 2:"BRAZIL", 3:"CANADA", 17:"PERU", 24:"UNITED STATES"}        |
|           0 | {0:"ALGERIA", 5:"ETHIOPIA", 14:"KENYA", 15:"MOROCCO", 16:"MOZAMBIQUE"}        |
|           3 | {6:"FRANCE", 7:"GERMANY", 19:"ROMANIA", 22:"RUSSIA", 23:"UNITED KINGDOM"}     |
|           4 | {4:"EGYPT", 10:"IRAN", 11:"IRAQ", 13:"JORDAN", 20:"SAUDI ARABIA"}             |
|           2 | {8:"INDIA", 9:"INDONESIA", 12:"JAPAN", 18:"CHINA", 21:"VIETNAM"}              |
+-------------+-------------------------------------------------------------------------------+
```

```
select n_regionkey, map_agg(`n_name`, `n_nationkey` % 5) from `nation` group by `n_regionkey`;
```

```
+-------------+-------------------------------------------------------------------------------+
| n_regionkey | map_agg(`n_name`, (`n_nationkey` % 5))                                        |
+-------------+-------------------------------------------------------------------------------+
|           2 | {"INDIA":3, "INDONESIA":4, "JAPAN":2, "CHINA":3, "VIETNAM":1}                 |
|           0 | {"ALGERIA":0, "ETHIOPIA":0, "KENYA":4, "MOROCCO":0, "MOZAMBIQUE":1}           |
|           3 | {"FRANCE":1, "GERMANY":2, "ROMANIA":4, "RUSSIA":2, "UNITED KINGDOM":3}        |
|           1 | {"ARGENTINA":1, "BRAZIL":2, "CANADA":3, "PERU":2, "UNITED STATES":4}          |
|           4 | {"EGYPT":4, "IRAN":0, "IRAQ":1, "JORDAN":3, "SAUDI ARABIA":0}                 |
+-------------+-------------------------------------------------------------------------------+
```

MAX

Description

The MAX function returns the maximum value of the expression.

Syntax

```
MAX(<expr>)
```

Parameters

Parameters	Description
expr	The expression needs to be obtained

Return Value

Returns the same data type as the input expression.

Example

```
select max(scan_rows) from log_statis group by datetime;
```

```
+------------------+
| max(`scan_rows`) |
+------------------+
|          4671587 |
+------------------+
```

MAX_BY

Description

The MAX_BY function is used to return the corresponding associated value based on the maximum value of the specified column.

Syntax

```
MAX_BY(<expr1>, <expr2>)
```

Parameters

Parameters	Description
<expr1>	The expression used to specify the corresponding association.
<expr2>	The expression used to specify the maximum value for statistics.

Return Value

Returns the same data type as the input expression .

Example

```
-- setup
create table tbl(k1 int, k2 int, k3 int, k4 int) distributed by hash(k1) buckets 1 properties ("replication_num"="1");
insert into tbl values (0,3,2,100),(1,2,3,4),(4,3,2,1),(3,4,2,1);
```

```
select * from tbl;
```

```
+------+------+------+------+
| k1   | k2   | k3   | k4   |
+------+------+------+------+
|    0 | 3    | 2    |  100 |
|    1 | 2    | 3    |    4 |
|    4 | 3    | 2    |    1 |
|    3 | 4    | 2    |    1 |
+------+------+------+------+
```

```
select max_by(k1, k4) from tbl;
```

```
+--------------------+
| max_by(`k1`, `k4`) |
+--------------------+
```

```
|                  0 |
+--------------------+
```

MIN

Description

The MIN function returns the minimum value of the expression.

Syntax

```
MIN(expr)
```

Parameters

Parameters	Description
<expr>	The expression needs to be obtained

Return Value

Returns the same data type as the input expression.

Example

```
select MIN(scan_rows) from log_statis group by datetime;
```

```
+------------------+
| MIN(`scan_rows`) |
+------------------+
|                0 |
+------------------+
```

MIN_BY

Description

The MIN_BY function is used to return the corresponding associated value based on the minimum value of the specified column.

Syntax

```
MIN_BY(<expr1>, <expr2>)
```

Parameters

Parameters	Description
<expr1>	The expression used to specify the corresponding association.
<expr2>	The expression used to specify the minimum value for statistics.

Return Value

Returns the same data type as the input expression .

Example

```
select * from tbl;
```

```
+------+------+------+------+
| k1   | k2   | k3   | k4   |
+------+------+------+------+
|    0 | 3    | 2    |  100 |
|    1 | 2    | 3    |    4 |
|    4 | 3    | 2    |    1 |
|    3 | 4    | 2    |    1 |
+------+------+------+------+
```

```
select min_by(k1, k4) from tbl;
```

```
+--------------------+
| min_by(`k1`, `k4`) |
+--------------------+
|                  4 |
+--------------------+
```

PERCENTILE

Description

Calculates the exact percentile, suitable for small datasets. First sorts the specified column in descending order, then takes the exact p-th percentile. The value of `p` is between `0` and `1`. If `p` does not point to an exact position, it returns the linear interpolation of the adjacent values at position `p`. Note that this is not the average of the two numbers. Special cases: - Returns `NULL` when the input column is `NULL`

Syntax

```
PERCENTILE(<col>, <p>)
```

Parameters

Parameter	Description
`<col>`	The column to be calculated as the exact percentile, which must be an integer column.
`<p>`	The exact percentile to be calculated, a constant value, with a value range of `[0.0, 1.0]`.

Return Value

Return the exact percentile of the specified column, with a return type of `DOUBLE`.

Examples

```
-- Create sample table
CREATE TABLE sales_data
(
    product_id INT,
    sale_price DECIMAL(10, 2)
) DUPLICATE KEY(`product_id`)
DISTRIBUTED BY HASH(`product_id`) BUCKETS AUTO
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);

-- Insert sample data
INSERT INTO sales_data VALUES
(1, 10.00),
(1, 15.00),
(1, 20.00),
(1, 25.00),
(1, 30.00),
(1, 35.00),
(1, 40.00),
(1, 45.00),
(1, 50.00),
```

```
(1, 100.00);

-- Calculate different percentiles of sales prices
SELECT
    percentile(sale_price, 0.5)  as median_price,     -- Median
    percentile(sale_price, 0.75) as p75_price,        -- 75th percentile
    percentile(sale_price, 0.90) as p90_price,        -- 90th percentile
    percentile(sale_price, 0.95) as p95_price,        -- 95th percentile
    percentile(null, 0.99)       as p99_null          -- Null value at 99th percentile
FROM sales_data;
```

```
+--------------+-----------+-------------------+-------------------+----------+
| median_price | p75_price | p90_price         | p95_price         | p99_null |
+--------------+-----------+-------------------+-------------------+----------+
|         32.5 |     43.75 | 54.99999999999998 | 77.49999999999994 |     NULL |
+--------------+-----------+-------------------+-------------------+----------+
```

PERCENTILE_APPROX

Description

The PERCENTILE_APPROX function is used to calculate approximate percentiles, primarily for large datasets. Compared to the PERCENTILE function, it has the following features:

1. Memory Efficiency: Uses fixed-size memory, maintaining low memory consumption even when processing low-cardinality columns (large data volumes but the number of different elements is small)
2. Performance Advantage: Suitable for processing low-cardinality large-scale datasets with faster computation
3. Adjustable Precision: Balance between precision and performance through the compression parameter

Syntax

```
PERCENTILE_APPROX(<col>, <p> [, <compression>])
```

Parameters

Parameter	Description
<col>	The column to calculate percentiles for
<p>	Percentile value, range [0.0, 1.0], e.g., 0.99 represents the 99th percentile
<compression>	Optional parameter, compression level, range [2048, 10000], higher values increase precision but consume more memory. If not specified or out of range, uses 10000

Return Value

Returns a DOUBLE value representing the calculated approximate percentile.

Examples

```
-- Create sample table
CREATE TABLE response_times (
    request_id INT,
    response_time DECIMAL(10, 2)
) DUPLICATE KEY(`request_id`)
DISTRIBUTED BY HASH(`request_id`) BUCKETS AUTO
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);

-- Insert sample data
INSERT INTO response_times VALUES
(1, 10.5),
```

```
(2, 15.2),
(3, 20.1),
(4, 25.8),
(5, 30.3),
(6, 35.7),
(7, 40.2),
(8, 45.9),
(9, 50.4),
(10, 100.6);

-- Calculate 99th percentile using different compression levels
SELECT
    percentile_approx(response_time, 0.99) as p99_default,          -- Default compression
    percentile_approx(response_time, 0.99, 2048) as p99_fast,       -- Lower compression, faster
    percentile_approx(response_time, 0.99, 10000) as p99_accurate   -- Higher compression, more accurate
FROM response_times;
```

```
+------------------+------------------+------------------+
| p99_default      | p99_fast         | p99_accurate     |
+------------------+------------------+------------------+
| 100.5999984741211 | 100.5999984741211 | 100.5999984741211 |
+------------------+------------------+------------------+
```

PERCENTILE_ARRAY

Description

The PERCENTILE_ARRAY function calculates exact percentile arrays, allowing multiple percentile values to be computed at once. This function is primarily suitable for small datasets.

Key features: 1. Exact Calculation: Provides exact percentile results rather than approximations 2. Batch Processing: Can calculate multiple percentiles in a single operation 3. Scope: Best suited for handling small-scale datasets

Syntax

```
PERCENTILE_ARRAY(<col>, <array_p>)
```

Parameters

Parameter	Description
<col>	The column to calculate the percentile for
<array_p>	Percentile array, each element must be in the range [0.0, 1.0], e.g., [0.5, 0.95, 0.99]

Return Value

Return a DOUBLE type array, containing the calculated percentile values.

Examples

```
-- Create sample table
CREATE TABLE sales_data (
    id INT,
    amount DECIMAL(10, 2)
) DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS AUTO
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);

-- Insert sample data
INSERT INTO sales_data VALUES
(1, 10.5),
(2, 15.2),
(3, 20.1),
(4, 25.8),
(5, 30.3),
(6, 35.7),
(7, 40.2),
(8, 45.9),
```

```
(9, 50.4),
(10, 100.6);

-- Calculate multiple percentiles
SELECT percentile_array(amount, [0.25, 0.5, 0.75, 0.9]) as percentiles
FROM sales_data;
```

```
+----------------------------------------+
| percentiles                            |
+----------------------------------------+
| [21.25, 32.5, 43.75, 54.99999999999998] |
+----------------------------------------+
```

QUANTILE_UNION

Description

The QUANTILE_UNION function is used to merge intermediate results from multiple quantile calculations. This function typically works in conjunction with QUANTILE_STATE and is particularly useful in scenarios requiring multi-stage quantile calculations.

Syntax

```
QUANTILE_UNION(<query_state>)
```

Parameters

Parameter	Description
<query_state>	The intermediate state generated by the TO_QUANTILE_STATE function.

Return Value

Returns an aggregation state that can be used for further quantile calculations. The result of this function remains a QUANTILE_STATE.

Example

```
-- Create sample table
CREATE TABLE response_times (
    request_id INT,
    response_time DOUBLE,
    region STRING
) DUPLICATE KEY(request_id)
DISTRIBUTED BY HASH(request_id) BUCKETS AUTO
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);

-- Insert sample data
INSERT INTO response_times VALUES
(1, 10.5, 'east'),
(2, 15.2, 'east'),
(3, 20.1, 'west'),
(4, 25.8, 'east'),
(5, 30.3, 'west'),
(6, 35.7, 'east'),
(7, 40.2, 'west'),
(8, 45.9, 'east'),
(9, 50.4, 'west'),
(10, 100.6, 'east');
```

```
-- Calculate 50th percentile of response times by region
SELECT
    region,
    QUANTILE_PERCENT(
        QUANTILE_UNION(
            TO_QUANTILE_STATE(response_time, 2048)
        ),
        0.5
    ) AS median_response_time
FROM response_times
GROUP BY region;
```

```
+--------+----------------------+
| region | median_response_time |
+--------+----------------------+
| west   |                35.25 |
| east   |                30.75 |
+--------+----------------------+
```

RETENTION

Description

The `retention` function takes as arguments a set of conditions from 1 to 32 arguments of type `UInt8` that indicate whether a certain condition was met for the event. Any condition can be specified as an argument.

The conditions, except the first, apply in pairs: the result of the second will be true if the first and second are true, of the third if the first and third are true, etc.

To put it simply, the first digit of the return value array indicates whether `event_1` is true or false, the second digit represents the truth and falseness of `event_1` and `event_2`, and the third digit represents whether `event_1` is true or false and `event_3` is true False and, and so on. If `event_1` is false, return an array full of zeros.

Syntax

```
RETENTION(<event_1> [, <event_2>, ... , <event_n>]);
```

Parameters

Parameter	Description
`<event_n>`	The nth event condition, of type `UInt8` and value 1 or 0.

Returned value

An array of 1 and 0 with a maximum length of 32, where the final output array length matches the input parameter length.

- 1: Condition is met.
- 0: Condition is not met.

Examples

```
-- Create sample table
CREATE TABLE retention_test(
    `uid` int COMMENT 'user id',
    `date` datetime COMMENT 'date time'
) DUPLICATE KEY(uid)
DISTRIBUTED BY HASH(uid) BUCKETS AUTO
PROPERTIES (
    "replication_allocation" = "tag.location.default: 1"
);

-- Insert sample data
INSERT into retention_test values
(0, '2022-10-12'),
(0, '2022-10-13'),
(0, '2022-10-14'),
```

```
(1, '2022-10-12'),
(1, '2022-10-13'),
(2, '2022-10-12');

-- Calculate user retention
SELECT
    uid,
    RETENTION(date = '2022-10-12') AS r,
    RETENTION(date = '2022-10-12', date = '2022-10-13') AS r2,
    RETENTION(date = '2022-10-12', date = '2022-10-13', date = '2022-10-14') AS r3
FROM retention_test
GROUP BY uid
ORDER BY uid ASC;
```

```
+------+------+--------+-----------+
| uid  | r    | r2     | r3        |
+------+------+--------+-----------+
|    0 | [1]  | [1, 1] | [1, 1, 1] |
|    1 | [1]  | [1, 1] | [1, 1, 0] |
|    2 | [1]  | [1, 0] | [1, 0, 0] |
+------+------+--------+-----------+
```

SEQUENCE_COUNT

Description

Counts the number of event chains that matched the pattern. The function searches event chains that do not overlap. It starts to search for the next chain after the current chain is matched.

WARNING!

Events that occur at the same second may lay in the sequence in an undefined order affecting the result.

Syntax

```
SEQUENCE_COUNT(<pattern>, <timestamp>, <cond_1> [, <cond_2>, ..., <cond_n>]);
```

Parameters

Parameter	Description
`<pattern>`	Pattern string, see **Pattern syntax** below.
`<timestamp>`	Column considered to contain time data. Typical data types are `Date` and `DateTime`. You can also use any of the supported UInt data types.
`<cond_n>`	Conditions that describe the chain of events. Data type: `UInt8`. You can pass up to 32 condition arguments. The function takes only the events described in these conditions into account. If the sequence contains data that isn't described in a condition, the function skips them.

Pattern syntax

- `(?N)` — Matches the condition argument at position N. Conditions are numbered in the `[1, 32]` range. For example, `(?1)` matches the argument passed to the `cond_1` parameter.
- `.*` — Matches any number of events. You do not need conditional arguments to count this element of the pattern.
- `(?t operator value)` — Sets the time in seconds that should separate two events.
- We define `t` as the difference in seconds between two times, For example, pattern `(?1)(?t>1800)(?2)` matches events that occur more than 1800 seconds from each other. pattern `(?1)(?t>10000)(?2)` matches events that occur more than 10000 seconds from each other. An arbitrary number of any events can lay between these events. You can use the >=, >, <, <=, == operators.

Return Value

Number of non-overlapping event chains that are matched.

Examples

Matching examples

```
-- Create sample table
CREATE TABLE sequence_count_test1(
    `uid` int COMMENT 'user id',
    `date` datetime COMMENT 'date time',
    `number` int NULL COMMENT 'number'
) DUPLICATE KEY(uid)
DISTRIBUTED BY HASH(uid) BUCKETS AUTO
PROPERTIES (
    "replication_num" = "1"
);

-- Insert sample data
INSERT INTO sequence_count_test1(uid, date, number) values
(1, '2022-11-02 10:41:00', 1),
(2, '2022-11-02 13:28:02', 2),
(3, '2022-11-02 16:15:01', 1),
(4, '2022-11-02 19:05:04', 2),
(5, '2022-11-02 20:08:44', 3);

-- Query example
SELECT
    SEQUENCE_COUNT('(?1)(?2)', date, number = 1, number = 3) as c1,
    SEQUENCE_COUNT('(?1)(?2)', date, number = 1, number = 2) as c2,
    SEQUENCE_COUNT('(?1)(?t>=3600)(?2)', date, number = 1, number = 2) as c3
FROM sequence_count_test1;
```

```
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    1 |    2 |    2 |
+------+------+------+
```

Non-matching examples

```
-- Create sample table
CREATE TABLE sequence_count_test2(
    `uid` int COMMENT 'user id',
    `date` datetime COMMENT 'date time',
    `number` int NULL COMMENT 'number'
) DUPLICATE KEY(uid)
DISTRIBUTED BY HASH(uid) BUCKETS AUTO
PROPERTIES (
    "replication_num" = "1"
);

-- Insert sample data
INSERT INTO sequence_count_test2(uid, date, number) values
(1, '2022-11-02 10:41:00', 1),
(2, '2022-11-02 11:41:00', 7),
(3, '2022-11-02 16:15:01', 3),
(4, '2022-11-02 19:05:04', 4),
(5, '2022-11-02 21:24:12', 5);

-- Query example
SELECT
    SEQUENCE_COUNT('(?1)(?2)', date, number = 1, number = 2) as c1,
```

```
    SEQUENCE_COUNT('(?1)(?2).*', date, number = 1, number = 2) as c2,
    SEQUENCE_COUNT('(?1)(?t>3600)(?2)', date, number = 1, number = 7) as c3
FROM sequence_count_test2;
```

```
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    0 |    0 |    0 |
+------+------+------+
```

Special examples

```
-- Create sample table
CREATE TABLE sequence_count_test3(
    `uid` int COMMENT 'user id',
    `date` datetime COMMENT 'date time',
    `number` int NULL COMMENT 'number'
) DUPLICATE KEY(uid)
DISTRIBUTED BY HASH(uid) BUCKETS AUTO
PROPERTIES (
    "replication_num" = "1"
);

-- Insert sample data
INSERT INTO sequence_count_test3(uid, date, number) values
(1, '2022-11-02 10:41:00', 1),
(2, '2022-11-02 11:41:00', 7),
(3, '2022-11-02 16:15:01', 3),
(4, '2022-11-02 19:05:04', 4),
(5, '2022-11-02 21:24:12', 5);

-- Query example
SELECT SEQUENCE_COUNT('(?1)(?2)', date, number = 1, number = 5) FROM sequence_count_test3;
```

```
+-------------------------------------------------------------------+
| sequence_count('(?1)(?2)', `date`, `number` = 1, `number` = 5) |
+-------------------------------------------------------------------+
|                                                                 1 |
+-------------------------------------------------------------------+
```

This is a very simple example. The function found the event chain where number 5 follows number 1. It skipped number 7,3,4 between them, because the number is not described as an event. If we want to take this number into account when searching for the event chain given in the example, we should make a condition for it.

Now, perform this query:

```
SELECT SEQUENCE_COUNT('(?1)(?2)', date, number = 1, number = 5, number = 4) FROM sequence_count_test3;
```

```
+-----------------------------------------------------------------------------+
| sequence_count('(?1)(?2)', `date`, `number` = 1, `number` = 5, `number` = 4) |
+-----------------------------------------------------------------------------+
|                                                                           0 |
+-----------------------------------------------------------------------------+
```

The result is kind of confusing. In this case, the function couldn’t find the event chain matching the pattern, because the event for number 4 occurred between 1 and 5. If in the same case we checked the condition for number 6, the sequence would count the pattern.

```
SELECT SEQUENCE_COUNT('(?1)(?2)', date, number = 1, number = 5, number = 6) FROM sequence_count_test3;
```

```
+---------------------------------------------------------------------------+
| sequence_count('(?1)(?2)', `date`, `number` = 1, `number` = 5, `number` = 6) |
+---------------------------------------------------------------------------+
|                                                                         1 |
+---------------------------------------------------------------------------+
```

SEQUENCE_MATCH

Description

Checks whether the sequence contains an event chain that matches the pattern.

WARNING!

Events that occur at the same second may lay in the sequence in an undefined order affecting the result.

Syntax

```
SEQUENCE_MATCH(<pattern>, <timestamp>, <cond_1> [, <cond_2>, ..., <cond_n>])
```

Parameters

Parameter	Description
`<pattern>`	Pattern string. See **Pattern syntax** below.
`<timestamp>`	Column considered to contain time data. Typical data types are `Date` and `DateTime`. You can also use any of the supported UInt data types.
`<cond_n>`	Conditions that describe the chain of events. Data type: `UInt8`. You can pass up to 32 condition arguments. The function takes only the events described in these conditions into account. If the sequence contains data that isn't described in a condition, the function skips them.

Pattern syntax

- `(?N)` — Matches the condition argument at position N. Conditions are numbered in the `[1, 32]` range. For example, `(?1)` matches the argument passed to the `cond1` parameter.
- `.*` — Matches any number of events. You do not need conditional arguments to match this element of the pattern.
- `(?t operator value)` — Sets the time in seconds that should separate two events.
- We define `t` as the difference in seconds between two times, For example, pattern `(?1)(?t>1800)(?2)` matches events that occur more than 1800 seconds from each other. pattern `(?1)(?t>10000)(?2)` matches events that occur more than 10000 seconds from each other. An arbitrary number of any events can lay between these events. You can use the >=, >, <, <=, == operators.

Return value

1: if the pattern is matched.

0: if the pattern isn't matched.

Examples

Match examples

```
CREATE TABLE sequence_match_test1(
    `uid` int COMMENT 'user id',
    `date` datetime COMMENT 'date time',
    `number` int NULL COMMENT 'number'
) DUPLICATE KEY(uid)
DISTRIBUTED BY HASH(uid) BUCKETS AUTO
PROPERTIES (
    "replication_num" = "1"
);

INSERT INTO sequence_match_test1(uid, date, number) values
(1, '2022-11-02 10:41:00', 1),
(2, '2022-11-02 13:28:02', 2),
(3, '2022-11-02 16:15:01', 1),
(4, '2022-11-02 19:05:04', 2),
(5, '2022-11-02 20:08:44', 3);


SELECT
sequence_match('(?1)(?2)', date, number = 1, number = 3) as c1,
sequence_match('(?1)(?2)', date, number = 1, number = 2) as c2,
sequence_match('(?1)(?t>=3600)(?2)', date, number = 1, number = 2) as c3
FROM sequence_match_test1;
```

```
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
```

Not match examples

```
CREATE TABLE sequence_match_test2(
    `uid` int COMMENT 'user id',
    `date` datetime COMMENT 'date time',
    `number` int NULL COMMENT 'number'
) DUPLICATE KEY(uid)
DISTRIBUTED BY HASH(uid) BUCKETS AUTO
PROPERTIES (
    "replication_num" = "1"
);

INSERT INTO sequence_match_test2(uid, date, number) values
(1, '2022-11-02 10:41:00', 1),
(2, '2022-11-02 11:41:00', 7),
(3, '2022-11-02 16:15:01', 3),
(4, '2022-11-02 19:05:04', 4),
(5, '2022-11-02 21:24:12', 5);

SELECT
sequence_match('(?1)(?2)', date, number = 1, number = 2) as c1,
sequence_match('(?1)(?2).*', date, number = 1, number = 2) as c2,
```

```
sequence_match('(?1)(?t>3600)(?2)', date, number = 1, number = 7) as c3
FROM sequence_match_test2;
```

```
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    0 |    0 |    0 |
+------+------+------+
```

Special examples

```
CREATE TABLE sequence_match_test3(
    `uid` int COMMENT 'user id',
    `date` datetime COMMENT 'date time',
    `number` int NULL COMMENT 'number'
) DUPLICATE KEY(uid)
DISTRIBUTED BY HASH(uid) BUCKETS AUTO
PROPERTIES (
    "replication_num" = "1"
);

INSERT INTO sequence_match_test3(uid, date, number) values
(1, '2022-11-02 10:41:00', 1),
(2, '2022-11-02 11:41:00', 7),
(3, '2022-11-02 16:15:01', 3),
(4, '2022-11-02 19:05:04', 4),
(5, '2022-11-02 21:24:12', 5);

SELECT sequence_match('(?1)(?2)', date, number = 1, number = 5)
FROM sequence_match_test3;
```

```
+-------------------------------------------------------------------+
| sequence_match('(?1)(?2)', `date`, `number` = 1, `number` = 5) |
+-------------------------------------------------------------------+
|                                                                 1 |
+-------------------------------------------------------------------+
```

This is a very simple example. The function found the event chain where number 5 follows number 1. It skipped number 7,3,4 between them, because the number is not described as an event. If we want to take this number into account when searching for the event chain given in the example, we should make a condition for it.

Now, perform this query:

```
SELECT sequence_match('(?1)(?2)', date, number = 1, number = 5, number = 4)
FROM sequence_match_test3;
```

```
+-----------------------------------------------------------------------------+
| sequence_match('(?1)(?2)', `date`, `number` = 1, `number` = 5, `number` = 4) |
+-----------------------------------------------------------------------------+
|                                                                           0 |
+-----------------------------------------------------------------------------+
```

The result is kind of confusing. In this case, the function couldn’t find the event chain matching the pattern, because the event for number 4 occurred between 1 and 5. If in the same case we checked the condition for number 6, the sequence would match the pattern.

```
SELECT sequence_match('(?1)(?2)', date, number = 1, number = 5, number = 6)
FROM sequence_match_test3;
```

```
+-----------------------------------------------------------------------------+
| sequence_match('(?1)(?2)', `date`, `number` = 1, `number` = 5, `number` = 6) |
+-----------------------------------------------------------------------------+
|                                                                           1 |
+-----------------------------------------------------------------------------+
```

STDDEV,STDDEV_POP

Description

Returns the standard deviation of the expr expression

Alias

- STDDEV_POP

Syntax

```
STDDEV(<expr>)
```

Parameters

Parameter	Description
<expr>	The value to be calculated standard deviation

Return Value

Return the standard deviation of the expr expression

Examples

```
-- Create sample tables
CREATE TABLE score_table (
    student_id INT,
    score DOUBLE
) DISTRIBUTED BY HASH(student_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO score_table VALUES
(1, 85),
(2, 90),
(3, 82),
(4, 88),
(5, 95);

-- Calculate the standard deviation of all students' scores
SELECT STDDEV(score) as score_stddev
FROM score_table;
```

```
+-------------------+
| score_stddev      |
```

```
+-------------------+
| 4.427188724235729 |
+-------------------+
```

STDDEV_SAMP

Description

Returns the sample standard deviation of the expr expression

Syntax

```
STDDEV_SAMP(<expr>)
```

Parameters

Parameter	Description
<expr>	The value to be calculated standard deviation

Return Value

Return the sample standard deviation of the expr expression

Examples

```
-- Create sample tables
CREATE TABLE score_table (
    student_id INT,
    score DOUBLE
) DISTRIBUTED BY HASH(student_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO score_table VALUES
(1, 85),
(2, 90),
(3, 82),
(4, 88),
(5, 95);

-- Calculate the sample standard deviation of all students' scores
SELECT STDDEV_SAMP(score) as score_stddev
FROM score_table;
```

```
+-------------------+
| score_stddev      |
+-------------------+
| 4.949747468305831 |
+-------------------+
```

SUM

Description

Used to return the sum of all values of the selected field

Syntax

```
SUM(<expr>)
```

Parameters

Parameter	Description
<expr>	The field to calculate the sum of

Return Value

Return the sum of all values of the selected field.

Examples

```
-- Create sample tables
CREATE TABLE sales_table (
    product_id INT,
    price DECIMAL(10,2),
    quantity INT
) DISTRIBUTED BY HASH(product_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO sales_table VALUES
(1, 99.99, 2),
(2, 159.99, 1),
(3, 49.99, 5),
(4, 299.99, 1),
(5, 79.99, 3);

-- Calculate the total sales amount
SELECT SUM(price * quantity) as total_sales
FROM sales_table;
```

```
+-------------+
| total_sales |
+-------------+
|     1149.88 |
+-------------+
```

SUM0

Description

Used to return the sum of all values of the selected field. Unlike the SUM function, when all input values are NULL, SUM0 returns 0 instead of NULL.

Syntax

```
SUM0(<expr>)
```

Parameters

Parameter	Description
<expr>	The field to calculate the sum of

Return Value

Returns the sum of all values of the selected field. If all values are NULL, returns 0.

Examples

```
-- Create example table
CREATE TABLE sales_table (
    product_id INT,
    price DECIMAL(10,2),
    quantity INT,
    discount DECIMAL(10,2)
) DISTRIBUTED BY HASH(product_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO sales_table VALUES
(1, 99.99, 2, NULL),
(2, 159.99, 1, NULL),
(3, 49.99, 5, NULL),
(4, 299.99, 1, NULL),
(5, 79.99, 3, NULL);

-- Compare SUM and SUM0
SELECT
    SUM(discount) as sum_discount,    -- Returns NULL
    SUM0(discount) as sum0_discount   -- Returns 0
FROM sales_table;
```

```
+--------------+---------------+
| sum_discount | sum0_discount |
```

```
+--------------+---------------+
|         NULL |          0.00 |
+--------------+---------------+
```

TOPN

Description

The TOPN function returns the N most frequent values in the specified column. It is an approximate calculation function that returns results ordered by count in descending order.

Syntax

```
TOPN(<expr>, <top_num> [, <space_expand_rate>])
```

Parameters

Parameter	Description
<expr>	The column or expression to be counted.
<top_num>	The number of the most frequent values to return. It must be a positive integer.
<space_expand_rate>	Optional parameter, which is used to set the number of counters used in the Space-Saving algorithm. `counter_numbers = top_num * space_expand_rate` , the larger the value of space_expand_rate, the more accurate the result, and the default value is 50.

Return Value

Returns a JSON string containing values and their corresponding occurrence counts.

Examples

```
-- Create sample table
CREATE TABLE page_visits (
    page_id INT,
    user_id INT,
    visit_date DATE
) DISTRIBUTED BY HASH(page_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO page_visits VALUES
(1, 101, '2024-01-01'),
(2, 102, '2024-01-01'),
(1, 103, '2024-01-01'),
(3, 101, '2024-01-01'),
(1, 104, '2024-01-01'),
(2, 105, '2024-01-01'),
(1, 106, '2024-01-01'),
(4, 107, '2024-01-01');
```

```
-- Find top 3 most visited pages
SELECT TOPN(page_id, 3) as top_pages
FROM page_visits;
```

```
+--------------------+
| top_pages          |
+--------------------+
| {"1":4,"2":2,"4":1} |
+--------------------+
```

TOPN_ARRAY

Description

TOPN_ARRAY returns an array of the N most frequent values in the specified column. It is an approximate calculation function that returns results ordered by count in descending order.

Syntax

```
TOPN_ARRAY(<expr>, <top_num> [, <space_expand_rate>])
```

Parameters

Parameter	Description
<expr>	The column or expression to be counted.
<top_num>	The number of the most frequent values to return. It must be a positive integer.
<space_expand_rate>	Optional parameter, which is used to set the number of counters used in the Space-Saving algorithm. `counter_numbers = top_num * space_expand_rate` , the larger the value of space_expand_rate, the more accurate the result, and the default value is 50.

Return Value

Return an array containing the N most frequent values.

Examples

```
-- Create sample table
CREATE TABLE page_visits (
    page_id INT,
    user_id INT,
    visit_date DATE
) DISTRIBUTED BY HASH(page_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO page_visits VALUES
(1, 101, '2024-01-01'),
(2, 102, '2024-01-01'),
(1, 103, '2024-01-01'),
(3, 101, '2024-01-01'),
(1, 104, '2024-01-01'),
(2, 105, '2024-01-01'),
(1, 106, '2024-01-01'),
(4, 107, '2024-01-01');
```

```
-- Find top 3 most visited pages
SELECT TOPN_ARRAY(page_id, 3) as top_pages
FROM page_visits;
```

```
+-----------+
| top_pages |
+-----------+
| [1, 2, 4] |
+-----------+
```

TOPN_WEIGHTED

Description

The TOPN_WEIGHTED function returns the N most frequent values in the specified column with weighted counting. Unlike the regular TOPN function, TOPN_WEIGHTED allows adjusting the importance of values through weights.

Syntax

```
TOPN_WEIGHTED(<expr>, <weight>, <top_num> [, <space_expand_rate>])
```

Parameters

Parameter	Description
<expr>	The column or expression to be counted
<weight>	The column or expression to adjust the weight
<top_num>	The number of the most frequent values to return. It must be a positive integer.
<space_expand_rate>	Optional, the value to set the counter_numbers used in the Space-Saving algorithm. `counter_numbers = top_num * space_expand_rate`. The value of space_expand_rate should be greater than 1, and the default value is 50.

Return Value

Return an array containing values and weighted counts.

Examples

```
-- create example table
CREATE TABLE product_sales (
    product_id INT,
    sale_amount DECIMAL(10,2),
    sale_date DATE
) DISTRIBUTED BY HASH(product_id)
PROPERTIES (
    "replication_num" = "1"
);

-- insert test data
INSERT INTO product_sales VALUES
(1, 100.00, '2024-01-01'),
(2, 50.00, '2024-01-01'),
(1, 150.00, '2024-01-01'),
(3, 75.00, '2024-01-01'),
(1, 200.00, '2024-01-01'),
```

```
(2, 80.00, '2024-01-01'),
(1, 120.00, '2024-01-01'),
(4, 90.00, '2024-01-01');

-- find the top 3 products with highest sales amount
SELECT TOPN_WEIGHTED(product_id, sale_amount, 3) as top_products
FROM product_sales;
```

```
+--------------+
| top_products |
+--------------+
| [1, 2, 4]    |
+--------------+
```

VAR_SAMP,VARIANCE_SAMP

Description

The VAR_SAMP function calculates the sample variance of a specified expression. Unlike VARIANCE (population variance), VAR_SAMP uses n-1 as the divisor, which is considered an unbiased estimate of the population variance in statistics.

Alias

- VARIANCE_SAMP

Syntax

```
VAR_SAMP(<expr>)
```

Parameters

Parameter	Description
<expr>	The column or expression to calculate sample variance for. Must be numeric type

Return Value

Returns a DOUBLE value representing the calculated sample variance.

Examples

```
-- Create sample table
CREATE TABLE student_scores (
    student_id INT,
    score DECIMAL(4,1)
) DISTRIBUTED BY HASH(student_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO student_scores VALUES
(1, 85.5),
(2, 92.0),
(3, 78.5),
(4, 88.0),
(5, 95.5),
(6, 82.0),
(7, 90.0),
(8, 87.5);
```

```
-- Calculate sample variance of student scores
SELECT
    VAR_SAMP(score) as sample_variance,
    VARIANCE(score) as population_variance
FROM student_scores;
```

```
+------------------+---------------------+
| sample_variance  | population_variance |
+------------------+---------------------+
| 29.4107142857143 |   25.73437500000001 |
+------------------+---------------------+
```

VARIANCE,VAR_POP,VARIANCE_POP

Description

The VARIANCE function calculates the statistical variance of the specified expression. It measures how far a set of numbers are spread out from their arithmetic mean.

Alias

- VAR_POP
- VARIANCE_POP

Syntax

```
VARIANCE(<expr>)
```

Parameters

Parameter	Description
<expr>	The column or expression to calculate variance for. Must be numeric type

Return Value

Returns a DOUBLE value representing the calculated variance.

Examples

```
-- Create sample table
CREATE TABLE student_scores (
    student_id INT,
    score DECIMAL(4,1)
) DISTRIBUTED BY HASH(student_id)
PROPERTIES (
    "replication_num" = "1"
);

-- Insert test data
INSERT INTO student_scores VALUES
(1, 85.5),
(2, 92.0),
(3, 78.5),
(4, 88.0),
(5, 95.5),
(6, 82.0),
(7, 90.0),
(8, 87.5);
-- Calculate variance of student scores
```

```
SELECT VARIANCE(score) as score_variance
FROM student_scores;
```

```
+-------------------+
| score_variance    |
+-------------------+
| 25.73437499999998 |
+-------------------+
```

WINDOW_FUNNEL

Description

The WINDOW_FUNNEL function analyzes user behavior sequences by searching for event chains within a specified time window and calculating the maximum number of completed steps in the event chain. This function is particularly useful for conversion funnel analysis, such as analyzing user conversion from website visits to final purchases.

The function works according to the algorithm:

- The function searches for data that triggers the first condition in the chain and sets the event counter to 1. This is the moment when the sliding window starts.
- If events from the chain occur sequentially within the window, the counter is incremented. If the sequence of events is disrupted, the counter is not incremented.
- If the data has multiple event chains at varying points of completion, the function will only output the size of the longest chain.

Syntax

```
WINDOW_FUNNEL(<window>, <mode>, <timestamp>, <event_1>[, event_2, ... , event_n])
```

Parameters

Parameter	Description
<window>	window is the length of time window in seconds
<mode>	There are four modes in total, `default`, `deduplication`, `fixed`, and `increase`. For details, please refer to the **Mode** below.
<timestamp>	timestamp specifies column of DATETIME type, sliding time window works on it
<event_n>	evnet_n is boolean expression like eventID = 1004

Mode

```
- `default`: Defualt mode.

- `deduplication`: If the same event holds for the sequence of events, then such repeating event interrupts further processing. E.g. the array parameter is [event1='A', event2='B', event3='C', event4='D'], and the original event chain is "A-B-C-B-D". Since event B repeats, the filtered event chain can only be "A-B-C" and the max event level is 3.

- `fixed`: Don't allow interventions of other events. E.g. the array parameter is [event1='A', event2='B', event3='C', event4='D'], and the original event chain is A->B->D->C, it stops finding A->B->C at the D and the max event level is 2.

- `increase`: Apply conditions only to events with strictly increasing timestamps.
```

Return Value

Returns an integer representing the maximum number of consecutive steps completed within the specified time window.

Examples

example1: default mode

Using the `default` mode, find out the maximum number of consecutive events corresponding to different `user_id`, with a time window of `5` minutes:

```
CREATE TABLE events(
    user_id BIGINT,
    event_name VARCHAR(64),
    event_timestamp datetime,
    phone_brand varchar(64),
    tab_num int
) distributed by hash(user_id) buckets 3 properties("replication_num" = "1");

INSERT INTO
    events
VALUES
    (100123, 'login', '2022-05-14 10:01:00', 'HONOR', 1),
    (100123, 'visit', '2022-05-14 10:02:00', 'HONOR', 2),
    (100123, 'order', '2022-05-14 10:04:00', 'HONOR', 3),
    (100123, 'payment', '2022-05-14 10:10:00', 'HONOR', 4),
    (100125, 'login', '2022-05-15 11:00:00', 'XIAOMI', 1),
    (100125, 'visit', '2022-05-15 11:01:00', 'XIAOMI', 2),
    (100125, 'order', '2022-05-15 11:02:00', 'XIAOMI', 6),
    (100126, 'login', '2022-05-15 12:00:00', 'IPHONE', 1),
    (100126, 'visit', '2022-05-15 12:01:00', 'HONOR', 2),
    (100127, 'login', '2022-05-15 11:30:00', 'VIVO', 1),
    (100127, 'visit', '2022-05-15 11:31:00', 'VIVO', 5);

SELECT
    user_id,
    window_funnel(
        300,
        "default",
        event_timestamp,
        event_name = 'login',
        event_name = 'visit',
        event_name = 'order',
        event_name = 'payment'
    ) AS level
FROM
    events
GROUP BY
    user_id
order BY
    user_id;
```

```
+---------+-------+
| user_id | level |
+---------+-------+
|  100123 |     3 |
|  100125 |     3 |
|  100126 |     2 |
```

```
|  100127 |     2 |
+---------+-------+
```

For `uesr_id=100123`, because the time when the `payment` event occurred exceeds the time window, the matched event chain is `login-visit-order`.

example2: deduplication mode

Use the `deduplication` mode to find out the maximum number of consecutive events corresponding to different user_ids, with a time window of 1 hour:

```
CREATE TABLE events(
    user_id BIGINT,
    event_name VARCHAR(64),
    event_timestamp datetime,
    phone_brand varchar(64),
    tab_num int
) distributed by hash(user_id) buckets 3 properties("replication_num" = "1");

INSERT INTO
    events
VALUES
    (100123, 'login', '2022-05-14 10:01:00', 'HONOR', 1),
    (100123, 'visit', '2022-05-14 10:02:00', 'HONOR', 2),
    (100123, 'login', '2022-05-14 10:03:00', 'HONOR', 3),
    (100123, 'order', '2022-05-14 10:04:00', "HONOR", 4),
    (100123, 'payment', '2022-05-14 10:10:00', 'HONOR', 4),
    (100125, 'login', '2022-05-15 11:00:00', 'XIAOMI', 1),
    (100125, 'visit', '2022-05-15 11:01:00', 'XIAOMI', 2),
    (100125, 'order', '2022-05-15 11:02:00', 'XIAOMI', 6),
    (100126, 'login', '2022-05-15 12:00:00', 'IPHONE', 1),
    (100126, 'visit', '2022-05-15 12:01:00', 'HONOR', 2),
    (100127, 'login', '2022-05-15 11:30:00', 'VIVO', 1),
    (100127, 'visit', '2022-05-15 11:31:00', 'VIVO', 5);

SELECT
    user_id,
    window_funnel(
        3600,
        "deduplication",
        event_timestamp,
        event_name = 'login',
        event_name = 'visit',
        event_name = 'order',
        event_name = 'payment'
    ) AS level
FROM
    events
GROUP BY
    user_id
order BY
    user_id;
```

```
+---------+-------+
| user_id | level |
+---------+-------+
|  100123 |     2 |
|  100125 |     3 |
|  100126 |     2 |
```

```
|  100127 |     2 |
+---------+-------+
```

For `uesr_id=100123`, after matching the `visit` event, the `login` event appears repeatedly, so the matched event chain is `login-visit`.

example3: fixed mode

Use the `fixed` mode to find out the maximum number of consecutive events corresponding to different `user_id`, with a time window of `1` hour:

```
CREATE TABLE events(
    user_id BIGINT,
    event_name VARCHAR(64),
    event_timestamp datetime,
    phone_brand varchar(64),
    tab_num int
) distributed by hash(user_id) buckets 3 properties("replication_num" = "1");

INSERT INTO
    events
VALUES
    (100123, 'login', '2022-05-14 10:01:00', 'HONOR', 1),
    (100123, 'visit', '2022-05-14 10:02:00', 'HONOR', 2),
    (100123, 'order', '2022-05-14 10:03:00', "HONOR", 4),
    (100123, 'login2', '2022-05-14 10:04:00', 'HONOR', 3),
    (100123, 'payment', '2022-05-14 10:10:00', 'HONOR', 4),
    (100125, 'login', '2022-05-15 11:00:00', 'XIAOMI', 1),
    (100125, 'visit', '2022-05-15 11:01:00', 'XIAOMI', 2),
    (100125, 'order', '2022-05-15 11:02:00', 'XIAOMI', 6),
    (100126, 'login', '2022-05-15 12:00:00', 'IPHONE', 1),
    (100126, 'visit', '2022-05-15 12:01:00', 'HONOR', 2),
    (100127, 'login', '2022-05-15 11:30:00', 'VIVO', 1),
    (100127, 'visit', '2022-05-15 11:31:00', 'VIVO', 5);

SELECT
    user_id,
    window_funnel(
        3600,
        "fixed",
        event_timestamp,
        event_name = 'login',
        event_name = 'visit',
        event_name = 'order',
        event_name = 'payment'
    ) AS level
FROM
    events
GROUP BY
    user_id
order BY
    user_id;
```

```
+---------+-------+
| user_id | level |
+---------+-------+
|  100123 |     3 |
|  100125 |     3 |
|  100126 |     2 |
```

```
|  100127 |     2 |
+---------+-------+
```

For `uesr_id=100123`, after matching the `order` event, the event chain is interrupted by the `login2` event, so the matched event chain is `login-visit-order`.

example4: increase mode

Use the `increase` mode to find out the maximum number of consecutive events corresponding to different `user_id`, with a time window of 1 hour:

```
CREATE TABLE events(
    user_id BIGINT,
    event_name VARCHAR(64),
    event_timestamp datetime,
    phone_brand varchar(64),
    tab_num int
) distributed by hash(user_id) buckets 3 properties("replication_num" = "1");

INSERT INTO
    events
VALUES
    (100123, 'login', '2022-05-14 10:01:00', 'HONOR', 1),
    (100123, 'visit', '2022-05-14 10:02:00', 'HONOR', 2),
    (100123, 'order', '2022-05-14 10:04:00', "HONOR", 4),
    (100123, 'payment', '2022-05-14 10:04:00', 'HONOR', 4),
    (100125, 'login', '2022-05-15 11:00:00', 'XIAOMI', 1),
    (100125, 'visit', '2022-05-15 11:01:00', 'XIAOMI', 2),
    (100125, 'order', '2022-05-15 11:02:00', 'XIAOMI', 6),
    (100126, 'login', '2022-05-15 12:00:00', 'IPHONE', 1),
    (100126, 'visit', '2022-05-15 12:01:00', 'HONOR', 2),
    (100127, 'login', '2022-05-15 11:30:00', 'VIVO', 1),
    (100127, 'visit', '2022-05-15 11:31:00', 'VIVO', 5);

SELECT
    user_id,
    window_funnel(
        3600,
        "increase",
        event_timestamp,
        event_name = 'login',
        event_name = 'visit',
        event_name = 'order',
        event_name = 'payment'
    ) AS level
FROM
    events
GROUP BY
    user_id
order BY
    user_id;
```

```
+---------+-------+
| user_id | level |
+---------+-------+
|  100123 |     3 |
|  100125 |     3 |
|  100126 |     2 |
```

```
|  100127 |     2 |
+---------+-------+
```

For `uesr_id=100123`, the timestamp of the `payment` event and the timestamp of the `order` event occur in the same second and are not incremented, so the matched event chain is `login-visit-order`.

FOREACH

description

Converts an aggregate function for tables into an aggregate function for arrays that aggregates the corresponding array items and returns an array of results. For example, sum_foreach for the arrays [1, 2], [3, 4, 5]and[6, 7]returns the result [10, 13, 5] after adding together the corresponding array items.

Syntax

```
AGGREGATE_FUNCTION_FOREACH(arg...)
```

example

Prepare the sample data:

```
CREATE TABLE db (
    id INT,
    a  ARRAY<INT>,
    s  ARRAY<STRING>
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES ("replication_num" = "1");

INSERT INTO db VALUES
    (1, [1, 2, 3], ["ab", "123"]),
    (2, [20],      ["cd"]),
    (3, [100],     ["efg"]),
    (4, NULL,      NULL),
    (5, [null, 2], [null, "c"]);
```

Then run the FOREACH combinators:

```
mysql [test]>select a , s from db;
+-----------+---------------+
| a         | s             |
+-----------+---------------+
| [1, 2, 3] | ["ab", "123"] |
| [20]      | ["cd"]        |
| [100]     | ["efg"]       |
| NULL      | NULL          |
| [null, 2] | [null, "c"]   |
+-----------+---------------+

mysql [test]>select sum_foreach(a) from db;
+----------------+
| sum_foreach(a) |
+----------------+
| [121, 4, 3]    |
+----------------+

mysql [test]>select count_foreach(s) from db;
+------------------+
```

```
| count_foreach(s) |
+------------------+
| [3, 2]           |
+------------------+

mysql [test]>select array_agg_foreach(a) from db;
+-----------------------------------+
| array_agg_foreach(a)              |
+-----------------------------------+
| [[1, 20, 100, null], [2, 2], [3]] |
+-----------------------------------+

mysql [test]>select map_agg_foreach(a,a) from db;
+---------------------------------------+
| map_agg_foreach(a, a)                 |
+---------------------------------------+
| [{1:1, 20:20, 100:100}, {2:2}, {3:3}] |
+---------------------------------------+
```

Notes

- Each argument must be an ARRAY whose element type matches the corresponding argument of the nested aggregate function. The result is an ARRAY of the nested function's return type, computed position by position.
- When a single call passes multiple array arguments, all arrays in the same row must have the same length; otherwise the query fails with `Arrays passed to <function> aggregate function have different sizes`. (Array lengths may differ across rows.)
- The `percentile`, `percentile_array`, `percentile_approx`, and `percentile_approx_weighted` functions are not supported with the `foreach` combinator.

keywords

FOREACH

MERGE

Description

The aggregated intermediate results are aggregated and calculated to obtain the actual result. The type of the result is consistent with `AGGREGATE_FUNCTION`.

Syntax

```
AGGREGATE_FUNCTION_MERGE(agg_state)
```

Example

```
mysql [test]>select avg_merge(avg_state(1)) from d_table;
+-------------------------+
| avg_merge(avg_state(1)) |
+-------------------------+
|                       1 |
+-------------------------+
```

Keywords

AGG_STATE, MERGE

STATE

description

Returns the intermediate result of the aggregation function, which can be used for subsequent aggregation or to obtain the actual calculation result through the merge combiner, or can be directly written into the agg_state type table and saved. The type of the result is agg_state, and the function signature in agg_state is `AGGREGATE_FUNCTION(arg...)`.

Syntax

```
AGGREGATE_FUNCTION_STATE(arg...)
```

example

```
mysql [test]>select avg_merge(t) from (select avg_union(avg_state(1)) as t from d_table group by k1)p;
+----------------+
| avg_merge(`t`) |
+----------------+
|              1 |
+----------------+
```

keywords

AGG_STATE,STATE

UNION

description

Aggregate multiple aggregation intermediate results into one. The type of the result is agg_state, and the function signature is consistent with the input parameter.

Syntax

```
AGGREGATE_FUNCTION_UNION(agg_state)
```

example

```
mysql [test]>select avg_merge(t) from (select avg_union(avg_state(1)) as t from d_table group by k1)p;
+----------------+
| avg_merge(`t`) |
+----------------+
|              1 |
+----------------+
```

keywords

AGG_STATE, UNION

CUME_DIST

Description

CUME_DIST (Cumulative Distribution) is a window function that calculates the relative ranking of the current row value in the sorted result set. It returns the cumulative distribution value of the current row in the result set, ranging from 0 to 1. For a given row, its cumulative distribution value equals: (number of rows less than or equal to the current row value) / (total number of rows in the window partition).

Syntax

```
CUME_DIST()
```

Return Value

Returns a DOUBLE value ranging from 0 to 1.

Examples

Suppose we have a table called sales containing sales data, including salesperson name (sales_person), sales amount (sales_amount), and sales date (sales_date). We want to calculate the cumulative percentage of each salesperson's sales amount for each sales date.

```
SELECT
    sales_person,
    sales_date,
    sales_amount,
    CUME_DIST() OVER (PARTITION BY sales_date ORDER BY sales_amount ASC) AS cumulative_sales_percentage
FROM
    sales;
```

Assume the sales table contains the following data:

```
+------+--------------+------------+--------------+
| id   | sales_person | sales_date | sales_amount |
+------+--------------+------------+--------------+
|    1 | Alice        | 2024-02-01 |         2000 |
|    2 | Bob          | 2024-02-01 |         1500 |
|    3 | Alice        | 2024-02-02 |         1800 |
|    4 | Bob          | 2024-02-02 |         1200 |
|    5 | Alice        | 2024-02-03 |         2200 |
|    6 | Bob          | 2024-02-03 |         1900 |
|    7 | Tom          | 2024-02-03 |         2000 |
|    8 | Jerry        | 2024-02-03 |         2000 |
+------+--------------+------------+--------------+
```

After executing the above SQL query, the result will show each salesperson's sales amount and their cumulative percentage ranking for each sales date:

```
+--------------+------------+--------------+----------------------------+
| sales_person | sales_date | sales_amount | cumulative_sales_percentage |
+--------------+------------+--------------+----------------------------+
| Bob          | 2024-02-01 |         1500 |                        0.5 |
| Alice        | 2024-02-01 |         2000 |                          1 |
| Bob          | 2024-02-02 |         1200 |                        0.5 |
| Alice        | 2024-02-02 |         1800 |                          1 |
| Bob          | 2024-02-03 |         1900 |                       0.25 |
| Tom          | 2024-02-03 |         2000 |                       0.75 |
| Jerry        | 2024-02-03 |         2000 |                       0.75 |
| Alice        | 2024-02-03 |         2200 |                          1 |
+--------------+------------+--------------+----------------------------+
```

In this example, the CUME_DIST() function sorts sales amounts within each sales date and then calculates the cumulative percentage of each salesperson's sales amount for that date. Since we used PARTITION BY sales_date, the calculations are performed separately for each sales date, with salesperson sales amounts being calculated independently for different dates.

DENSE_RANK

Description

DENSE_RANK() is a window function used to calculate rankings within a group. Unlike RANK(), DENSE_RANK() returns consecutive rankings without gaps. The ranking values start from 1 and increment sequentially. When there are identical values, they will receive the same rank.

Syntax

```
DENSE_RANK()
```

Return Value

Returns a BIGINT type ranking value, starting from 1.

Examples

Setup — create the `int_t` table and load the rows the example uses:

```
CREATE TABLE int_t (x INT, y INT)
DISTRIBUTED BY HASH(x) BUCKETS 1
PROPERTIES ("replication_num" = "1");

INSERT INTO int_t VALUES
    (1, 1), (1, 2), (1, 2),
    (2, 1), (2, 2), (2, 3),
    (3, 1), (3, 1), (3, 2);
```

```
select x, y, dense_rank() over(partition by x order by y) as rank from int_t;
```

Identical values within a partition receive the same rank, and the ranks are consecutive (no gaps — that is how `DENSE_RANK` differs from `RANK`):

```
+------+------+------+
| x    | y    | rank |
+------+------+------+
|    1 |    1 |    1 |
|    1 |    2 |    2 |
|    1 |    2 |    2 |
|    2 |    1 |    1 |
|    2 |    2 |    2 |
|    2 |    3 |    3 |
|    3 |    1 |    1 |
|    3 |    1 |    1 |
|    3 |    2 |    2 |
+------+------+------+
```

FIRST_VALUE

Description

FIRST_VALUE() is a window function that returns the first value in an ordered set of values within a window partition. The handling of null values can be controlled using the IGNORE NULLS options.

Syntax

```
FIRST_VALUE(<expr>, <ignore_null>)
```

Parameters

Parameter	Description
expr	The expression from which to get the first value
ignore_null	Optional. When set, null values are ignored, returning the first non-null value

Return Value

Returns the same data type as the input expression.

Examples

```
WITH example_data AS (
    SELECT 1 as column1, NULL as column2, 'A' as group_name
    UNION ALL
    SELECT 1, 10, 'A'
    UNION ALL
    SELECT 1, NULL, 'A'
    UNION ALL
    SELECT 1, 20, 'A'
    UNION ALL
    SELECT 2, NULL, 'B'
    UNION ALL
    SELECT 2, 30, 'B'
    UNION ALL
    SELECT 2, 40, 'B'
)
SELECT
    group_name,
    column1,
    column2,
    FIRST_VALUE(column2) OVER (
        PARTITION BY column1
        ORDER BY column2 NULLS LAST
    ) AS first_value_default,
    FIRST_VALUE(column2, true) OVER (
        PARTITION BY column1
        ORDER BY column2
```

```
    ) AS first_value_ignore_null
FROM example_data
ORDER BY column1, column2;
```

```
+------------+---------+---------+---------------------+-------------------------+
| group_name | column1 | column2 | first_value_default | first_value_ignore_null |
+------------+---------+---------+---------------------+-------------------------+
| A          |       1 |    NULL |                  10 |                    NULL |
| A          |       1 |    NULL |                  10 |                    NULL |
| A          |       1 |      10 |                  10 |                      10 |
| A          |       1 |      20 |                  10 |                      10 |
| B          |       2 |    NULL |                  30 |                    NULL |
| B          |       2 |      30 |                  30 |                      30 |
| B          |       2 |      40 |                  30 |                      30 |
+------------+---------+---------+---------------------+-------------------------+
```

LAG

Description

LAG() is a window function that accesses data from previous rows without performing a self-join. It retrieves values from a row that is N rows before the current row within a partition.

Syntax

```
LAG ( <expr>, <offset>, <default> )
```

Parameters

Parameter	Description
expr	The expression whose value is to be retrieved
offset	Number of rows to look ahead.
default	Value to return when the offset goes beyond window bounds.

Return Value

Returns the same data type as the input expression.

Examples

Calculate the difference between each salesperson's current sales amount and the previous day's sales amount:

```
select stock_symbol, closing_date, closing_price,
lag(closing_price,1, 0) over (partition by stock_symbol order by closing_date) as "yesterday closing"
from stock_ticker
order by closing_date;
```

```
+--------------+---------------------+--------------+-------------------+
| stock_symbol | closing_date        | closing_price | yesterday closing |
| ------------ | ------------------- | ------------- | ----------------- |
| JDR          | 2014-09-13 00:00:00 | 12.86         | 0                 |
| JDR          | 2014-09-14 00:00:00 | 12.89         | 12.86             |
| JDR          | 2014-09-15 00:00:00 | 12.94         | 12.89             |
| JDR          | 2014-09-16 00:00:00 | 12.55         | 12.94             |
| JDR          | 2014-09-17 00:00:00 | 14.03         | 12.55             |
| JDR          | 2014-09-18 00:00:00 | 14.75         | 14.03             |
| JDR          | 2014-09-19 00:00:00 | 13.98         | 14.75             |
+--------------+---------------------+--------------+-------------------+
```

LAST_VALUE

Description

LAST_VALUE() is a window function that returns the last value within the window frame. The handling of null values can be controlled using the IGNORE NULLS options.

Syntax

```
LAST_VALUE(<expr>[, <ignore_null>])
```

Parameters

Parameter	Description
expr	The expression from which to get the last value
ignore_null	Optional. When set, null values are ignored, returning the last non-null value

Return Value

Returns the same data type as the input expression.

Examples

```
WITH example_data AS (
    SELECT 1 as id, 21 as myday, '04-21-11' as time_col, NULL as state
    UNION ALL
    SELECT 2, 21, '04-21-12', 2
    UNION ALL
    SELECT 3, 21, '04-21-13', 3
    UNION ALL
    SELECT 4, 22, '04-22-10-21', NULL
    UNION ALL
    SELECT 5, 22, '04-22-10-22', NULL
    UNION ALL
    SELECT 6, 22, '04-22-10-23', 5
    UNION ALL
    SELECT 7, 22, '04-22-10-24', NULL
    UNION ALL
    SELECT 8, 22, '04-22-10-25', 9
    UNION ALL
    SELECT 9, 23, '04-23-11', NULL
    UNION ALL
    SELECT 10, 23, '04-23-12', 10
    UNION ALL
    SELECT 11, 23, '04-23-13', NULL
    UNION ALL
    SELECT 12, 24, '02-24-10-21', NULL
)
SELECT
```

```
    *,
    last_value(`state`, 1) OVER(
        PARTITION BY `myday`
        ORDER BY `time_col` DESC
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
    ) as ignore_null,
    last_value(`state`, 0) OVER(
        PARTITION BY `myday`
        ORDER BY `time_col` DESC
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
    ) as not_ignore_null,
    last_value(`state`) OVER(
        PARTITION BY `myday`
        ORDER BY `time_col` DESC
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
    ) as ignore_null_default
FROM example_data
ORDER BY `id`, `myday`, `time_col`;
```

```
+------+-------+-------------+-------+-------------+-----------------+---------------------+
| id   | myday | time_col    | state | ignore_null | not_ignore_null | ignore_null_default |
+------+-------+-------------+-------+-------------+-----------------+---------------------+
|    1 |    21 | 04-21-11    |  NULL |           2 |            NULL |                NULL |
|    2 |    21 | 04-21-12    |     2 |           2 |            NULL |                NULL |
|    3 |    21 | 04-21-13    |     3 |           2 |               2 |                   2 |
|    4 |    22 | 04-22-10-21 |  NULL |        NULL |            NULL |                NULL |
|    5 |    22 | 04-22-10-22 |  NULL |           5 |            NULL |                NULL |
|    6 |    22 | 04-22-10-23 |     5 |           5 |            NULL |                NULL |
|    7 |    22 | 04-22-10-24 |  NULL |           5 |               5 |                   5 |
|    8 |    22 | 04-22-10-25 |     9 |           9 |            NULL |                NULL |
|    9 |    23 | 04-23-11    |  NULL |          10 |            NULL |                NULL |
|   10 |    23 | 04-23-12    |    10 |          10 |            NULL |                NULL |
|   11 |    23 | 04-23-13    |  NULL |          10 |              10 |                  10 |
|   12 |    24 | 02-24-10-21 |  NULL |        NULL |            NULL |                NULL |
+------+-------+-------------+-------+-------------+-----------------+---------------------+
```

LEAD

Description

LEAD() is a window function used to access data from subsequent rows without performing a self-join. It retrieves the value from the Nth row after the current row within a partition.

Syntax

```
LEAD ( <expr> [ , <offset> [ , <default> ] ] )
```

Parameters

Parameter	Description
expr	The expression whose value needs to be retrieved
offset	Number of rows to look backward.
default	Default value to return when the offset goes beyond the window range.

Return Value

Returns the same data type as the input expression.

Examples

Calculate the difference between each salesperson’s current sales and next day’s sales:

```
select stock_symbol, closing_date, closing_price,
case
(lead(closing_price,1, 0)
over (partition by stock_symbol order by closing_date)-closing_price) > 0
when true then "higher"
when false then "flat or lower"
end as "trending"
from stock_ticker
order by closing_date;
```

```
+--------------+---------------------+---------------+---------------+
| stock_symbol | closing_date        | closing_price | trending      |
| ------------ | ------------------- | ------------- | ------------- |
| JDR          | 2014-09-13 00:00:00 | 12.86         | higher        |
| JDR          | 2014-09-14 00:00:00 | 12.89         | higher        |
| JDR          | 2014-09-15 00:00:00 | 12.94         | flat or lower |
| JDR          | 2014-09-16 00:00:00 | 12.55         | higher        |
| JDR          | 2014-09-17 00:00:00 | 14.03         | higher        |
| JDR          | 2014-09-18 00:00:00 | 14.75         | flat or lower |
```

```
| JDR          | 2014-09-19 00:00:00 | 13.98         | flat or lower |
+--------------+---------------------+---------------+---------------+
```

NTILE

Description

NTILE() is a window function that divides an ordered dataset into a specified number of approximately equal buckets. The buckets are numbered sequentially starting from 1 up to the specified number of buckets. When data cannot be divided equally, extra records are allocated to lower-numbered buckets, ensuring that the number of rows in each bucket differs by at most 1.

Syntax

```
NTILE( <constant_value> )
```

Parameters

Parameter	Description
constant_value	Required. Specifies the number of buckets to create, must be a positive integer

Return Value

Returns a BIGINT bucket number, ranging from 1 to the specified number of buckets.

Usage Notes

If a statement contains both an ORDER BY clause in the NTILE function and an ORDER BY clause for the output results, these two sorts are independent: - The ORDER BY in the NTILE function determines which bucket each row is assigned to - The output ORDER BY determines the display order of the results

Examples

```
SELECT
    name,
    score,
    NTILE(4) OVER (ORDER BY score DESC) as quarter
FROM student_scores;
```

```
+----------+-------+---------+
| name     | score | quarter |
+----------+-------+---------+
| Alice    | 98    | 1       |  -- Top 25% scores
| Bob      | 95    | 1       |
| Charlie  | 90    | 2       |  -- 25-50% scores
| David    | 85    | 2       |
| Eve      | 82    | 3       |  -- 50-75% scores
| Frank    | 78    | 3       |
```

```
| Grace    | 75    | 4       |  -- Bottom 25% scores
| Henry    | 70    | 4       |
+----------+-------+---------+
```

OVERVIEW

Description

Window functions (also known as analytic functions) are special built-in functions that perform calculations while preserving the original rows. Unlike aggregate functions, window functions:

- Process data within a specific window range rather than by GROUP BY grouping
- Calculate a value for each row in the result set
- Can add additional columns in the SELECT list
- Execute last in query processing (after JOIN, WHERE, GROUP BY)

Window functions are commonly used in financial and scientific computing for trend analysis, outlier calculation, and data bucketing.

Syntax

```
function(<args>) OVER(
    [PARTITION BY <expr> [, <expr> ...]]
    [ORDER BY <expr> [ASC | DESC] [, <expr> [ASC | DESC] ...]]
    [<window_clause>]
)
```

Parameters

Parameter	Description
<args>	Input parameters for the window function, specific to the function being used
<function>	Supported functions include: AVG(), COUNT(), DENSE_RANK(), FIRST_VALUE(), LAG(), LAST_VALUE(), LEAD(), MAX(), MIN(), RANK(), ROW_NUMBER(), SUM() and all aggregate functions
<partition_by>	Similar to GROUP BY, groups data by specified columns
<order_by>	Defines the ordering of data within the window
<window_clause>	Defines the window range, syntax: ROWS BETWEEN [{ m | UNBOUNDED } PRECEDING | CURRENT ROW] [AND [CURRENT ROW | { UNBOUNDED | n } FOLLOWING]]

Return Value

Returns the same data type as the input expression.

Examples

1. Assume we have the following stock data, with stock symbol JDR and daily closing prices:

```
create table stock_ticker (stock_symbol string, closing_price decimal(8,2), closing_date datetime);
...load some data...
select * from stock_ticker order by stock_symbol, closing_date
```

```
| stock_symbol | closing_price | closing_date        |
|--------------|---------------|---------------------|
| JDR          | 12.86         | 2014-10-02 00:00:00 |
| JDR          | 12.89         | 2014-10-03 00:00:00 |
| JDR          | 12.94         | 2014-10-04 00:00:00 |
| JDR          | 12.55         | 2014-10-05 00:00:00 |
| JDR          | 14.03         | 2014-10-06 00:00:00 |
| JDR          | 14.75         | 2014-10-07 00:00:00 |
| JDR          | 13.98         | 2014-10-08 00:00:00 |
```

2. This query uses an analytic function to generate a moving_average column, which calculates the 3-day average stock price (previous day, current day, and next day). The first day has no previous day value, and the last day has no next day value, so these rows only calculate a two-day average. The Partition By clause has no effect here since all data is for JDR, but if there were other stock information, Partition By would ensure the analytic function only operates within its own partition.

```
select stock_symbol, closing_date, closing_price,
avg(closing_price) over (partition by stock_symbol order by closing_date
rows between 1 preceding and 1 following) as moving_average
from stock_ticker;
```

```
| stock_symbol | closing_date        | closing_price | moving_average |
|--------------|---------------------|---------------|----------------|
| JDR          | 2014-10-02 00:00:00 | 12.86         | 12.87          |
| JDR          | 2014-10-03 00:00:00 | 12.89         | 12.89          |
| JDR          | 2014-10-04 00:00:00 | 12.94         | 12.79          |
| JDR          | 2014-10-05 00:00:00 | 12.55         | 13.17          |
| JDR          | 2014-10-06 00:00:00 | 14.03         | 13.77          |
| JDR          | 2014-10-07 00:00:00 | 14.75         | 14.25          |
| JDR          | 2014-10-08 00:00:00 | 13.98         | 14.36          |
```

PERCENT_RANK

Description

PERCENT_RANK() is a window function that calculates the relative rank of a row within a partition or result set, returning values from 0.0 to 1.0. For a given row, it is calculated as: (rank - 1) / (total_rows - 1), where rank is the current row's rank and total_rows is the total number of rows in the partition.

Syntax

```
PERCENT_RANK()
```

Return Value

Returns a DOUBLE value ranging from 0.0 to 1.0: - Always returns 0 for the first row in the partition - Always returns 1 for the last row in the partition - Returns the same percentage rank for identical values

Examples

```
CREATE TABLE test_percent_rank (
    productLine VARCHAR,
    orderYear INT,
    orderValue DOUBLE,
    percentile_rank DOUBLE
) ENGINE=OLAP
DISTRIBUTED BY HASH(`orderYear`) BUCKETS 4
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);
```

```
INSERT INTO test_percent_rank (productLine, orderYear, orderValue, percentile_rank) VALUES
('Motorcycles', 2003, 2440.50, 0.00),
('Trains', 2003, 2770.95, 0.17),
('Trucks and Buses', 2003, 3284.28, 0.33),
('Vintage Cars', 2003, 4080.00, 0.50),
('Planes', 2003, 4825.44, 0.67),
('Ships', 2003, 5072.71, 0.83),
('Classic Cars', 2003, 5571.80, 1.00),
('Motorcycles', 2004, 2598.77, 0.00),
('Vintage Cars', 2004, 2819.28, 0.17),
('Planes', 2004, 2857.35, 0.33),
('Ships', 2004, 4301.15, 0.50),
('Trucks and Buses', 2004, 4615.64, 0.67),
('Trains', 2004, 4646.88, 0.83),
('Classic Cars', 2004, 8124.98, 1.00),
('Ships', 2005, 1603.20, 0.00),
('Motorcycles', 2005, 3774.00, 0.17),
('Planes', 2005, 4018.00, 0.50),
('Vintage Cars', 2005, 5346.50, 0.67),
```

```
('Classic Cars', 2005, 5971.35, 0.83),
('Trucks and Buses', 2005, 6295.03, 1.00);
```

```
SELECT
    productLine,
    orderYear,
    orderValue,
    ROUND(
    PERCENT_RANK()
    OVER (
        PARTITION BY orderYear
        ORDER BY orderValue
    ),2) percentile_rank
FROM
    test_percent_rank
ORDER BY
    orderYear;
```

```
+------------------+-----------+------------+-----------------+
| productLine      | orderYear | orderValue | percentile_rank |
+------------------+-----------+------------+-----------------+
| Motorcycles      |      2003 |     2440.5 |               0 |
| Trains           |      2003 |    2770.95 |            0.17 |
| Trucks and Buses |      2003 |    3284.28 |            0.33 |
| Vintage Cars     |      2003 |       4080 |             0.5 |
| Planes           |      2003 |    4825.44 |            0.67 |
| Ships            |      2003 |    5072.71 |            0.83 |
| Classic Cars     |      2003 |     5571.8 |               1 |
| Motorcycles      |      2004 |    2598.77 |               0 |
| Vintage Cars     |      2004 |    2819.28 |            0.17 |
| Planes           |      2004 |    2857.35 |            0.33 |
| Ships            |      2004 |    4301.15 |             0.5 |
| Trucks and Buses |      2004 |    4615.64 |            0.67 |
| Trains           |      2004 |    4646.88 |            0.83 |
| Classic Cars     |      2004 |    8124.98 |               1 |
| Ships            |      2005 |     1603.2 |               0 |
| Motorcycles      |      2005 |       3774 |             0.2 |
| Planes           |      2005 |       4018 |             0.4 |
| Vintage Cars     |      2005 |     5346.5 |             0.6 |
| Classic Cars     |      2005 |    5971.35 |             0.8 |
| Trucks and Buses |      2005 |    6295.03 |               1 |
+------------------+-----------+------------+-----------------+
```

RANK

Description

RANK() is a window function that returns the rank of values in an ordered dataset. Rankings start at 1 and increment sequentially. When identical values occur, they receive the same rank, but this creates gaps in the ranking sequence. For example, if the first two rows are tied for rank 1, the next different value will be ranked 3 (not 2).

Syntax

```
RANK()
```

Return Value

Returns a BIGINT rank value. Returns the same rank for identical values, but creates gaps in the sequence.

Examples

```
SELECT
    department,
    employee_name,
    salary,
    RANK() OVER (
        PARTITION BY department
        ORDER BY salary DESC
    ) as salary_rank
FROM employees;
```

```
+------------+---------------+--------+-------------+
| department | employee_name | salary | salary_rank |
+------------+---------------+--------+-------------+
| Sales      | Alice         | 10000  | 1           |
| Sales      | Bob           | 10000  | 1           |
| Sales      | Charlie       | 8000   | 3           |  -- Note this is 3, not 2
| IT         | David         | 12000  | 1           |
| IT         | Eve           | 11000  | 2           |
| IT         | Frank         | 11000  | 2           |
| IT         | Grace         | 9000   | 4           |  -- Note this is 4, not 3
+------------+---------------+--------+-------------+
```

In this example, the data is partitioned by department and ranked by salary within each department. When identical salaries occur (like Alice and Bob, Eve and Frank), they receive the same rank, but this creates gaps in subsequent rankings.

ROW_NUMBER

Description

ROW_NUMBER() is a window function that assigns a unique sequential number to each row within a partition. Numbers start at 1 and increment continuously. Unlike RANK() and DENSE_RANK(), ROW_NUMBER() assigns different numbers even for identical values, ensuring each row has a unique number.

Syntax

```
ROW_NUMBER()
```

Return Value

Returns a BIGINT sequence number, starting from 1 and incrementing continuously. Numbers are unique within each partition.

Examples

```
select x, y, row_number() over(partition by x order by y) as rank from int_t;
```

```
+-----+-----+------+
| x   | y   | rank |
| --- | --- | ---- |
| 1   | 1   | 1    |
| 1   | 2   | 2    |
| 1   | 2   | 3    |
| 2   | 1   | 1    |
| 2   | 2   | 2    |
| 2   | 3   | 3    |
| 3   | 1   | 1    |
| 3   | 1   | 2    |
| 3   | 2   | 3    |
+-----+-----+------+
```

EXPLODE

Description

The `explode` function takes an array as input and maps each element of the array to a separate row. It is typically used in conjunction with LATERAL VIEW to flatten nested data structures into a standard tabular format. The main difference between explode and `explode_outer` lies in handling empty values.

Syntax

```
EXPLODE(<array>)
EXPLODE_OUTER(<array>)
```

Required Parameters

Parameter	Description
`<arr>`	Array type

Return Value

When the array is not empty or NULL, the return values of `explode` and `explode_outer` are the same.

When the data is empty or NULL:

`explode` will not produce any rows and will filter out these records.

`explode_outer` if the array is empty, will generate a single row, but the expanded column value will be NULL. If the array is NULL, it will also retain a row and return NULL.

Examples

```
select e1 from (select 1 k1) as t lateral view explode([1,2,3]) tmp1 as e1;
```

```
+------+
| e1   |
+------+
|    1 |
|    2 |
|    3 |
+------+
```

```
select e1 from (select 1 k1) as t lateral view explode_outer(null) tmp1 as e1;
```

```
+------+
| e1   |
```

```
+------+
| NULL |
+------+
```

```
select e1 from (select 1 k1) as t lateral view explode([]) tmp1 as e1;
Empty set (0.010 sec)
```

```
select e1 from (select 1 k1) as t lateral view explode([null,1,null]) tmp1 as e1;
```

```
+------+
| e1   |
+------+
| NULL |
|    1 |
| NULL |
+------+
```

```
select e1 from (select 1 k1) as t lateral view explode_outer([null,1,null]) tmp1 as e1;
```

```
+------+
| e1   |
+------+
| NULL |
|    1 |
| NULL |
+------+
```

EXPLODE_BITMAP

Description

The `explode_bitmap` table function accepts a bitmap type data and maps each bit (bit) of the bitmap to a separate row. It is commonly used for processing bitmap data, expanding each element of the bitmap into separate records. It should be used in conjunction with LATERAL VIEW.

`explode_bitmap_outer` works similarly to `explode_bitmap`, but its behavior differs when handling NULL or empty values. It allows records with empty or NULL bitmaps to exist, and in the result, it expands an empty or NULL bitmap into NULL rows.

Syntax

```
EXPLODE_BITMAP(<bitmap>)
EXPLODE_BITMAP_OUTER(<bitmap>)
```

Parameters

Parameter	Description
`<bitmap>`	bitmap type

Return Value

Returns a row for each bit in the bitmap, with each row containing a single bit value.

Examples

```
CREATE TABLE example1 (
    k1 INT
)DUPLICATE KEY(k1)
DISTRIBUTED BY HASH(k1) BUCKETS AUTO
PROPERTIES (
"replication_allocation" = "tag.location.default: 1");
```

```
insert into example1 values(1),(2),(3),(4),(5),(6);
```

```
select k1 from example1 order by k1;
```

```
+------+
| k1   |
+------+
|    1 |
|    2 |
|    3 |
```

```
|    4 |
|    5 |
|    6 |
+------+
```

```
select k1, e1 from example1 lateral view explode_bitmap(bitmap_empty()) tmp1 as e1 order by k1, e1;
Empty set
```

```
select k1, e1 from example1 lateral view explode_bitmap(bitmap_from_string("1")) tmp1 as e1 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    1 |    1 |
|    2 |    1 |
|    3 |    1 |
|    4 |    1 |
|    5 |    1 |
|    6 |    1 |
+------+------+
```

```
select k1, e1 from example1 lateral view explode_bitmap(bitmap_from_string("1,2")) tmp1 as e1 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    1 |    1 |
|    1 |    2 |
|    2 |    1 |
|    2 |    2 |
|    3 |    1 |
|    3 |    2 |
|    4 |    1 |
|    4 |    2 |
|    5 |    1 |
|    5 |    2 |
|    6 |    1 |
|    6 |    2 |
+------+------+
```

```
select k1, e1 from example1 lateral view explode_bitmap(bitmap_from_string("1,1000")) tmp1 as e1 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    1 |    1 |
|    1 | 1000 |
|    2 |    1 |
|    2 | 1000 |
|    3 |    1 |
|    3 | 1000 |
|    4 |    1 |
|    4 | 1000 |
|    5 |    1 |
|    5 | 1000 |
|    6 |    1 |
```

```
|    6 | 1000 |
+------+------+
```

```
select k1, e1, e2 from example1
lateral view explode_bitmap(bitmap_from_string("1,1000")) tmp1 as e1
lateral view explode_split("a,b", ",") tmp2 as e2 order by k1, e1, e2;
```

```
+------+------+------+
| k1   | e1   | e2   |
+------+------+------+
|    1 |    1 | a    |
|    1 |    1 | b    |
|    1 | 1000 | a    |
|    1 | 1000 | b    |
|    2 |    1 | a    |
|    2 |    1 | b    |
|    2 | 1000 | a    |
|    2 | 1000 | b    |
|    3 |    1 | a    |
|    3 |    1 | b    |
|    3 | 1000 | a    |
|    3 | 1000 | b    |
|    4 |    1 | a    |
|    4 |    1 | b    |
|    4 | 1000 | a    |
|    4 | 1000 | b    |
|    5 |    1 | a    |
|    5 |    1 | b    |
|    5 | 1000 | a    |
|    5 | 1000 | b    |
|    6 |    1 | a    |
|    6 |    1 | b    |
|    6 | 1000 | a    |
|    6 | 1000 | b    |
+------+------+------+
```

EXPLODE_JSON_ARRAY_DOUBLE

Description

The `explode_json_array_double` table function accepts a JSON array, where each element is of double-precision floating-point type, and expands each floating-point number in the array into multiple rows, with each row containing one floating-point number. It is used in conjunction with LATERAL VIEW.

`explode_json_array_double_outer` is similar to `explode_json_array_double`, but the handling of NULL values is different.

If the JSON string itself is NULL, the `OUTER` version will return one row, with the value as NULL. The normal version will completely ignore such records.

If the JSON array is empty, the `OUTER` version will return one row, with the value as NULL. The normal version will return no results.

Syntax

```
EXPLODE_JSON_ARRAY_DOUBLE(<json>)
EXPLODE_JSON_ARRAY_DOUBLE_OUTER(<json>)
```

Return Value

Parameter	Description
<json>	json type

Parameters

Expands the JSON array, creating a row for each element, returning a double-precision floating-point column.

Examples

```
CREATE TABLE json_array_example (
    id INT,
    json_array STRING
)DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS AUTO
PROPERTIES (
"replication_allocation" = "tag.location.default: 1");
```

```
INSERT INTO json_array_example (id, json_array) VALUES
(1, '[1, 2, 3, 4, 5]'),
(2, '[1.1, 2.2, 3.3, 4.4]'),
(3, '["apple", "banana", "cherry"]'),
(4, '[{"a": 1}, {"b": 2}, {"c": 3}]'),
```

```
(5, '[]'),
(6, 'NULL');
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_DOUBLE(json_array) tmp1 AS e1
WHERE id = 2;
```

```
+------+------+
| id   | e1   |
+------+------+
|    2 |  1.1 |
|    2 |  2.2 |
|    2 |  3.3 |
|    2 |  4.4 |
+------+------+
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_DOUBLE(json_array) tmp1 AS e1
WHERE id = 6;
Empty set (0.01 sec)
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_DOUBLE_OUTER(json_array) tmp1 AS e1
WHERE id = 6;
```

```
+------+------+
| id   | e1   |
+------+------+
|    6 | NULL |
+------+------+
```

EXPLODE_JSON_ARRAY_INT

Description

The `explode_json_array_int` table function accepts a JSON array, where each element is of integer type, and expands each integer in the array into multiple rows, with each row containing one integer. It is used in conjunction with LATERAL VIEW.

`explode_json_array_int_outer` is similar to `explode_json_array_int`, but the handling of NULL values is different.

If the JSON string itself is NULL, the `OUTER` version will return one row, with the value as NULL. The normal version will completely ignore such records.

If the JSON array is empty, the `OUTER` version will return one row, with the value as NULL. The normal version will return no results.

Syntax

```
EXPLODE_JSON_ARRAY_INT(<json>)
EXPLODE_JSON_ARRAY_INT_OUTER(<json>)
```

Return Value

Parameter	Description
<json>	json type

Parameters

Expands the JSON array, creating a row for each element, returning an integer column.

Examples

```
CREATE TABLE json_array_example (
    id INT,
    json_array STRING
)DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS AUTO
PROPERTIES (
"replication_allocation" = "tag.location.default: 1");
```

```
INSERT INTO json_array_example (id, json_array) VALUES
(1, '[1, 2, 3, 4, 5]'),
(2, '[1.1, 2.2, 3.3, 4.4]'),
(3, '["apple", "banana", "cherry"]'),
(4, '[{"a": 1}, {"b": 2}, {"c": 3}]'),
```

```
(5, '[]'),
(6, 'NULL');
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_INT(json_array) tmp1 AS e1
WHERE id = 1;
```

```
+------+------+
| id   | e1   |
+------+------+
|    1 |    1 |
|    1 |    2 |
|    1 |    3 |
|    1 |    4 |
|    1 |    5 |
+------+------+
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_INT(json_array) tmp1 AS e1
WHERE id = 5;
Empty set (0.01 sec)
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_INT_OUTER(json_array) tmp1 AS e1
WHERE id = 5;
```

```
+------+------+
| id   | e1   |
+------+------+
|    5 | NULL |
+------+------+
```

EXPLODE_JSON_ARRAY_JSON

Description

The `explode_json_array_json` table function accepts a JSON array, where each element is of JSON object type, and expands each JSON object in the array into multiple rows, with each row containing one JSON object. It is used in conjunction with LATERAL VIEW.

Syntax

```
EXPLODE_JSON_ARRAY_JSON(<json>)
EXPLODE_JSON_ARRAY_JSON_OUTER(<json>)
```

Return Value

Parameter	Description
<json>	json type

Parameters

Expands the JSON array, creating a row for each element, returning a JSON object column.

Examples

```
CREATE TABLE json_array_example (
    id INT,
    json_array STRING
)DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS AUTO
PROPERTIES (
"replication_allocation" = "tag.location.default: 1");
```

```
INSERT INTO json_array_example (id, json_array) VALUES
(1, '[1, 2, 3, 4, 5]'),
(2, '[1.1, 2.2, 3.3, 4.4]'),
(3, '["apple", "banana", "cherry"]'),
(4, '[{"a": 1}, {"b": 2}, {"c": 3}]'),
(5, '[]'),
(6, 'NULL');
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_JSON(json_array) tmp1 AS e1
WHERE id = 4;
```

```
+------+---------+
| id   | e1      |
+------+---------+
|    4 | {"a":1} |
|    4 | {"b":2} |
|    4 | {"c":3} |
+------+---------+
```

EXPLODE_JSON_ARRAY_STRING

Description

The `explode_json_array_string` table function accepts a JSON array, where each element is of string type, and expands each string in the array into multiple rows, with each row containing one string. It is used in conjunction with LATERAL VIEW.

`explode_json_array_string_outer` is similar to `explode_json_array_string`, but the handling of NULL values is different.

If the JSON string itself is NULL, the `OUTER` version will return one row, with the value as NULL. The normal version will completely ignore such records.

If the JSON array is empty, the `OUTER` version will return one row, with the value as NULL. The normal version will return no results.

Syntax

```
EXPLODE_JSON_ARRAY_STRING(<json>)
EXPLODE_JSON_ARRAY_STRING_OUTER(<json>)
```

Return Value

Parameter	Description
`<json>`	json type

Parameters

Expands the JSON array, creating a row for each element, returning a string column.

Examples

```
CREATE TABLE json_array_example (
    id INT,
    json_array STRING
)DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS AUTO
PROPERTIES (
"replication_allocation" = "tag.location.default: 1");
```

```
INSERT INTO json_array_example (id, json_array) VALUES
(1, '[1, 2, 3, 4, 5]'),
(2, '[1.1, 2.2, 3.3, 4.4]'),
(3, '["apple", "banana", "cherry"]'),
(4, '[{"a": 1}, {"b": 2}, {"c": 3}]'),
```

```
(5, '[]'),
(6, 'NULL');
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_STRING(json_array) tmp1 AS e1
WHERE id = 3;
```

```
+------+--------+
| id   | e1     |
+------+--------+
|    3 | apple  |
|    3 | banana |
|    3 | cherry |
+------+--------+
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_STRING(json_array) tmp1 AS e1
WHERE id = 6;
Empty set (0.02 sec)
```

```
SELECT id, e1
FROM json_array_example
LATERAL VIEW EXPLODE_JSON_ARRAY_STRING_OUTER(json_array) tmp1 AS e1
WHERE id = 6;
```

```
+------+------+
| id   | e1   |
+------+------+
|    6 | NULL |
+------+------+
```

EXPLODE_JSON_OBJECT

Description

`explode_json_object` expands a JSON object into multiple rows, with each row containing a key-value pair. It is typically used to process JSON data and expand the JSON object into a more queryable format. This function only supports non-empty JSON objects.

`explode_json_object_outer` is similar to `explode_json_object`, but with different behavior when handling empty and NULL values. It can retain empty or NULL JSON objects and return corresponding records.

Syntax

```
EXPLODE_JSON_OBJECT(<json>)
EXPLODE_JSON_OBJECT_OUTER(<json>)
```

Parameters

Parameter	Description
`<json>`	json type

Return Value

When the JSON object is neither empty nor NULL, the return values of `explode_json_object` and `explode_json_object_outer` are the same. Each key-value pair generates one row, with the key as one column and the value as another column.

When the JSON object is empty or NULL:

`explode_json_object` will not return any rows. `explode_json_object_outer` will return one row, with the expanded columns being NULL.

Examples

```
CREATE TABLE example (
    id INT,
    value_json json
) DUPLICATE KEY(id)
DISTRIBUTED BY HASH(`id`) BUCKETS AUTO
PROPERTIES (
"replication_allocation" = "tag.location.default: 1");
```

```
INSERT INTO example VALUES
(1, '{"key1": "value1", "key2": "value2"}'),
(2, '{}'),
(3, NULL);
```

```
select * from example;
```

```
+------+-----------------------------------+
| id   | value_json                        |
+------+-----------------------------------+
|    2 | {}                                |
|    1 | {"key1":"value1","key2":"value2"} |
|    3 | NULL                              |
+------+-----------------------------------+
```

```
SELECT id, k, v
FROM example
LATERAL VIEW explode_json_object(value_json) exploded_table AS k , v;
```

```
+------+------+----------+
| id   | k    | v        |
+------+------+----------+
|    1 | key1 | "value1" |
|    1 | key2 | "value2" |
+------+------+----------+
```

```
SELECT id, k, v
FROM example
LATERAL VIEW explode_json_object_outer(value_json) exploded_table AS k, v;
```

```
+------+------+----------+
| id   | k    | v        |
+------+------+----------+
|    3 | NULL | NULL     |
|    1 | key1 | "value1" |
|    1 | key2 | "value2" |
|    2 | NULL | NULL     |
+------+------+----------+
```

EXPLODE_MAP

Description

The `explode_map` function takes a map (mapping type) and expands it into multiple rows, with each row containing a key-value pair. It is typically used in conjunction with LATERAL VIEW and can support multiple lateral views. It is supported only by the new optimizer.

The main difference between `explode_map` and `explode_map_outer` lies in the handling of null values.

Syntax

```
EXPLODE_MAP(map<k,v>)
EXPLODE_MAP_OUTER(map<k,v>)
```

Parameters

Parameter	Description
`map<k,v>`	map type

Return Value

When the map is not empty or NULL, the return values of `explode_map` and `explode_map_outer` are the same.

When the data is empty or NULL:

`explode_map` Only processes non-empty map types. If the map is empty or NULL, `explode_map` will not return any rows. `explode_map_outer` If the map is empty or NULL, explode_map_outer will retain the record with the empty or NULL map and return a row with NULL values.

Examples

```
SET enable_nereids_planner=true
```

```
SET enable_fallback_to_original_planner=false
```

```
CREATE TABLE IF NOT EXISTS `sdu`(
                    `id` INT NULL,
                    `name` TEXT NULL,
                    `score` MAP<TEXT,INT> NULL
                ) ENGINE=OLAP
                DUPLICATE KEY(`id`)
                COMMENT 'OLAP'
                DISTRIBUTED BY HASH(`id`) BUCKETS 1
```

```
                PROPERTIES ("replication_allocation" = "tag.location.default: 1");
Query OK, 0 rows affected (0.15 sec)
```

```
insert into sdu values (0, "zhangsan", {"Chinese":"80","Math":"60","English":"90"}), (1, "lisi", {"null":null}), (2, "wangwu",
{"Chinese":"88","Math":"90","English":"96"}), (3, "lisi2", {null:null}), (4, "amory", NULL);
Query OK, 5 rows affected (0.23 sec)
{'label':'label_9b35d9d9d59147f5_bffb974881ed2133', 'status':'VISIBLE', 'txnId':'4005'}
```

```
select * from sdu order by id;
```

```
+------+----------+---------------------------------------+
| id   | name     | score                                 |
+------+----------+---------------------------------------+
|    0 | zhangsan | {"Chinese":80, "Math":60, "English":90} |
|    1 | lisi     | {"null":null}                         |
|    2 | wangwu   | {"Chinese":88, "Math":90, "English":96} |
|    3 | lisi2    | {null:null}                           |
|    4 | amory    | NULL                                  |
+------+----------+---------------------------------------+
```

```
select name, k,v from sdu lateral view explode_map(score) tmp as k,v;
```

```
+----------+---------+------+
| name     | k       | v    |
+----------+---------+------+
| zhangsan | Chinese |   80 |
| zhangsan | Math    |   60 |
| zhangsan | English |   90 |
| lisi     | null    | NULL |
| wangwu   | Chinese |   88 |
| wangwu   | Math    |   90 |
| wangwu   | English |   96 |
| lisi2    | NULL    | NULL |
+----------+---------+------+
```

```
select name, k,v from sdu lateral view explode_map_outer(score) tmp as k,v;
```

```
+----------+---------+------+
| name     | k       | v    |
+----------+---------+------+
| zhangsan | Chinese |   80 |
| zhangsan | Math    |   60 |
| zhangsan | English |   90 |
| lisi     | null    | NULL |
| wangwu   | Chinese |   88 |
| wangwu   | Math    |   90 |
| wangwu   | English |   96 |
| lisi2    | NULL    | NULL |
| amory    | NULL    | NULL |
+----------+---------+------+
```

```
select name, k,v,k1,v1 from sdu lateral view explode_map_outer(score) tmp as k,v lateral view explode_map(score) tmp2 as k1,v1;
```

```
+----------+---------+------+---------+------+
| name     | k       | v    | k1      | v1   |
+----------+---------+------+---------+------+
| zhangsan | Chinese |   80 | Chinese |   80 |
| zhangsan | Chinese |   80 | Math    |   60 |
| zhangsan | Chinese |   80 | English |   90 |
| zhangsan | Math    |   60 | Chinese |   80 |
| zhangsan | Math    |   60 | Math    |   60 |
| zhangsan | Math    |   60 | English |   90 |
| zhangsan | English |   90 | Chinese |   80 |
| zhangsan | English |   90 | Math    |   60 |
| zhangsan | English |   90 | English |   90 |
| lisi     | null    | NULL | null    | NULL |
| wangwu   | Chinese |   88 | Chinese |   88 |
| wangwu   | Chinese |   88 | Math    |   90 |
| wangwu   | Chinese |   88 | English |   96 |
| wangwu   | Math    |   90 | Chinese |   88 |
| wangwu   | Math    |   90 | Math    |   90 |
| wangwu   | Math    |   90 | English |   96 |
| wangwu   | English |   96 | Chinese |   88 |
| wangwu   | English |   96 | Math    |   90 |
| wangwu   | English |   96 | English |   96 |
| lisi2    | NULL    | NULL | NULL    | NULL |
+----------+---------+------+---------+------+
```

EXPLODE_NUMBERS

Description

The `explode_numbers` table function takes an integer n and expands all numbers within the range into multiple rows, each containing a single number. It is commonly used to generate a sequence of consecutive numbers and is often paired with LATERAL VIEW.

`explode_numbers_outer`, unlike `explode_numbers`, adds a NULL row when the table function generates zero rows.

Syntax

```
EXPLODE_NUMBERS(<n>)
EXPLODE_NUMBERS_OUTER(<n>)
```

Parameters

Parameter	Description
<n>	Integer type input

Return Value

Returns a sequence of [0, n).

- Does not return any rows when n is 0 or NULL.

Examples

```
select e1 from (select 1 k1) as t lateral view explode_numbers(5) tmp1 as e1;
```

```
+------+
| e1   |
+------+
|    0 |
|    1 |
|    2 |
|    3 |
|    4 |
+------+
```

```
select e1 from (select 1 k1) as t lateral view explode_numbers(0) tmp1 as e1;
Empty set
```

```
select e1 from (select 1 k1) as t lateral view explode_numbers_outer(0) tmp1 as e1;
```

```
+------+
| e1   |
+------+
| NULL |
+------+
```

EXPLODE_SPLIT

Description

The `explode_split` table function is used to split a string into multiple substrings based on a specified delimiter and expand each substring into a separate row. Each substring is returned as an individual row, and it is typically used with LATERAL VIEW to break down long strings into individual parts for more granular queries.

`explode_split_outer` is similar to `explode_split`, but it differs in the way it handles empty or NULL strings.

Syntax

```
EXPLODE_SPLIT(<str>, <delimiter>)
EXPLODE_SPLIT_OUTER(<str>, <delimiter>)
```

Parameters

Parameter	Description
`<str>`	String type
`<delimiter>`	Delimiter

Return Value

Returns a sequence of the split substrings. If the string is empty or NULL, no rows are returned.

Examples

```
-- setup
create table example1(k1 int, k2 varchar(50)) distributed by hash(k1) buckets 1 properties ("replication_num"="1");
insert into example1 values (1,''),(2,null),(3,','),(4,'1'),(5,'1,2,3'),(6,'a, b, c');
```

```
select * from example1 order by k1;
```

```
+------+---------+
| k1   | k2      |
+------+---------+
|    1 |         |
|    2 | NULL    |
|    3 | ,       |
|    4 | 1       |
|    5 | 1,2,3   |
|    6 | a, b, c |
+------+---------+
```

```
select k1, e1 from example1 lateral view explode_split(k2, ',') tmp1 as e1 where k1 = 1 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    1 |      |
+------+------+
```

```
select k1, e1 from example1 lateral view explode_split(k2, ',') tmp1 as e1 where k1 = 2 order by k1, e1;
Empty set
```

```
select k1, e1 from example1 lateral view explode_split(k2, ',') tmp1 as e1 where k1 = 3 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    3 |      |
+------+------+
```

```
select k1, e1 from example1 lateral view explode_split(k2, ',') tmp1 as e1 where k1 = 4 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    4 | 1    |
+------+------+
```

```
select k1, e1 from example1 lateral view explode_split(k2, ',') tmp1 as e1 where k1 = 5 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    5 | 1    |
|    5 | 2    |
|    5 | 3    |
+------+------+
```

```
select k1, e1 from example1 lateral view explode_split(k2, ',') tmp1 as e1 where k1 = 6 order by k1, e1;
```

```
+------+------+
| k1   | e1   |
+------+------+
|    6 |  b   |
|    6 |  c   |
|    6 | a    |
+------+------+
```

```
CREATE TABLE example2 (
    id INT,
    str string null
```

```
)DUPLICATE KEY(id)
DISTRIBUTED BY HASH(`id`) BUCKETS AUTO
PROPERTIES (
"replication_allocation" = "tag.location.default: 1");
```

```
insert into example2 values (1,''),(2,NUll),(3,"1"),(4,"1,2,3"),(5,"a,b,c");
```

```
select id, e1 from example2 lateral view explode_split(str, ',') tmp1 as e1 where id = 2 order by id, e1;
Empty set (0.02 sec)
```

```
select id, e1 from example2 lateral view explode_split_outer(str, ',') tmp1 as e1 where id = 2 order by id, e1;
```

```
+------+------+
| id   | e1   |
+------+------+
|    2 | NULL |
+------+------+
```

BACKENDS

Description

The table function generates the backends temporary table, and you can view the BE node information in the current doris cluster.

Syntax

```
BACKENDS()
```

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV	global	

Return Value

Field	Description
BackendId	The unique identifier for each backend node.
Host	The IP address or hostname of the backend node.
HeartbeatPort	The port used for health checks (heartbeat).
BePort	The port used for communication between the backend node and the cluster for query execution.
HttpPort	The HTTP port of the backend node.
BrpcPort	The port used for BRPC communication.
ArrowFlightSqlPort	The Arrow Flight SQL port (used for integration with Apache Arrow for high-performance data transport).
LastStartTime	The timestamp for the last start time of the backend node.
LastHeartbeat	The timestamp for the last heartbeat received from the backend node.
Alive	Whether the backend node is alive (true/false).
SystemDecommissioned	Whether the backend node has been decommissioned.
TabletNum	The number of tablets managed by the backend node.
DataUsedCapacity	The amount of disk space used by the backend node (in MB).
TrashUsedCapacity	The amount of disk space used by trash (in MB).
AvailCapacity	The available disk space on the backend node.
TotalCapacity	The total disk capacity of the backend node.
UsedPct	The percentage of disk capacity used by the backend node.

Field	Description
MaxDiskUsedPct	The maximum disk usage percentage across all tablets.
RemoteUsedCapacity	The disk space used by remote storage (if applicable).
Tag	Tags associated with the backend node, typically used for categorization (e.g., location).
ErrMsg	Error messages reported by the backend node.
Version	The version of the backend node.
Status	The current status of the backend node, including success/failure reports for tablets, load times, and query statuses.
HeartbeatFailureCounter	The count of heartbeat failures, if any.
NodeRole	The role of the backend node, such as `mix`, which means the node handles both storage and query processing.
CpuCores	The number of CPU cores on the backend node.
Memory	The amount of memory on the backend node.

Examples

show backends cluster information

```
select * from backends();
```

```
+-----------+-----------+---------------+--------+----------+----------+-------------------+---------------------+-----------------
----+-------+----------------------+-----------+-----------------+------------------+--------------+--------------+---------+---
-------------+--------------------+-------------------------+--------+-------------------------+---------------------------------
---------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------+-------------------------+----------+----------+-----------+
| BackendId | Host      | HeartbeatPort | BePort | HttpPort | BrpcPort | ArrowFlightSqlPort | LastStartTime       | LastHeartbeat
| Alive | SystemDecommissioned | TabletNum | DataUsedCapacity | TrashUsedCapacity | AvailCapacity | TotalCapacity | UsedPct |
MaxDiskUsedPct | RemoteUsedCapacity | Tag                     | ErrMsg | Version                 | Status
| HeartbeatFailureCounter | NodeRole | CpuCores | Memory    |
+-----------+-----------+---------------+--------+----------+----------+-------------------+---------------------+-----------------
----+-------+----------------------+-----------+-----------------+------------------+--------------+--------------+---------+---
-------------+--------------------+-------------------------+--------+-------------------------+---------------------------------
---------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------+-------------------------+----------+----------+-----------+
| 10020     | 10.xx.xx.90 | 9050          | 9060   | 8040     | 8060     | -1                | 2025-01-13 14:11:31 | 2025-01-16
13:24:55 | true  | false               | 359       | 295.328 MB       | 0.000             | 231.236 GB    | 3.437 TB      | 93.43 %
| 93.43 %        | 0.000             | {"location" : "default"} |        | doris-0.0.0--83f899b32b |
{"lastSuccessReportTabletsTime":"2025-01-16
13:24:07","lastStreamLoadTime":1737004982210,"isQueryDisabled":false,"isLoadDisabled":false,"isActive":true,"currentFragmentNum":0,"
lastFragmentUpdateTimeTime":1737004982195} | 0                       | mix      | 96       | 375.81 GB |
+-----------+-----------+---------------+--------+----------+----------+-------------------+---------------------+-----------------
----+-------+----------------------+-----------+-----------------+------------------+--------------+--------------+---------+---
-------------+--------------------+-------------------------+--------+-------------------------+---------------------------------
---------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------+-------------------------+----------+----------+-----------+
```

CATALOGS

Description

The `CATALOGS()` function generates a temporary `catalogs` table, allowing you to view information about all the catalogs created in the current Doris instance. The result combines the information from `show catalogs` and `show catalog xxx`.

This function is used in the `FROM` clause, making it easier to query and analyze catalog data in Doris.

Syntax

```
CATALOGS()
```

Return Value

View the returned fields of the catalogs() function

```
desc function catalogs();
```

```
+-------------+--------+------+-------+---------+-------+
| Field       | Type   | Null | Key   | Default | Extra |
+-------------+--------+------+-------+---------+-------+
| CatalogId   | BIGINT | No   | false | NULL    | NONE  |
| CatalogName | TEXT   | No   | false | NULL    | NONE  |
| CatalogType | TEXT   | No   | false | NULL    | NONE  |
| Property    | TEXT   | No   | false | NULL    | NONE  |
| Value       | TEXT   | No   | false | NULL    | NONE  |
+-------------+--------+------+-------+---------+-------+
```

The field meanings are as follows:

Field	Type	Description
`CatalogId`	BIGINT	A unique identifier for each catalog. It is used to distinguish different catalogs.
`CatalogName`	TEXT	The name of the catalog. This is the identifier for the catalog within Doris.
`CatalogType`	TEXT	The type of the catalog (e.g., database, data source). It indicates the kind of catalog.
`Property`	TEXT	The name of a property related to the catalog (e.g., a configuration setting).
`Value`	TEXT	The value of the corresponding property for the catalog. It provides specific details about the catalog's configuration.

Examples

View all catalog information of the doris cluster

```
select * from catalogs()
```

```
+-----------+-------------+-------------+-----------------------------------------+-----------------------------------------
----------------------------+
| CatalogId | CatalogName | CatalogType | Property                                | Value
|
+-----------+-------------+-------------+-----------------------------------------+-----------------------------------------
----------------------------+
|     16725 | hive        | hms         | dfs.client.failover.proxy.provider.HANN    |
org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider |
|     16725 | hive        | hms         | dfs.ha.namenodes.HANN                   | nn1,nn2
|
|     16725 | hive        | hms         | create_time                             | 2023-07-13 16:24:38.968
|
|     16725 | hive        | hms         | ipc.client.fallback-to-simple-auth-allowed | true
|
|     16725 | hive        | hms         | dfs.namenode.rpc-address.HANN.nn1       | nn1_host:rpc_port
|
|     16725 | hive        | hms         | hive.metastore.uris                     | thrift://127.0.0.1:7004
|
|     16725 | hive        | hms         | dfs.namenode.rpc-address.HANN.nn2       | nn2_host:rpc_port
|
|     16725 | hive        | hms         | type                                    | hms
|
|     16725 | hive        | hms         | dfs.nameservices                        | HANN
|
|         0 | internal    | internal    | NULL                                    | NULL
|
|     16726 | es          | es          | create_time                             | 2023-07-13 16:24:44.922
|
|     16726 | es          | es          | type                                    | es
|
|     16726 | es          | es          | hosts                                   | http://127.0.0.1:9200
|
+-----------+-------------+-------------+-----------------------------------------+-----------------------------------------
----------------------------+
```

FRONTENDS

Description

The table function generates the frontends temporary table, which allows you to view the FE node information in the current doris cluster.

Syntax

```
FRONTENDS()
```

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV	global	

Return Value

Field	Description
Name	The unique name of the frontend node.
Host	The IP address or hostname of the frontend node.
EditLogPort	The port used for edit log communication.
HttpPort	The HTTP port for the frontend node.
QueryPort	The port used for query execution on the frontend node.
RpcPort	The port used for RPC communication.
ArrowFlightSqlPort	The Arrow Flight SQL port (used for integration with Apache Arrow for high-performance data transport).
Role	The role of the frontend node (e.g., `FOLLOWER`).
IsMaster	Indicates whether the node is a master node (true/false).
ClusterId	The identifier for the cluster to which the frontend node belongs.
Join	Indicates if the frontend node is part of the cluster (true/false).
Alive	Indicates if the frontend node is alive (true/false).
ReplayedJournalId	The last journal ID that was replayed by the frontend node.
LastStartTime	The timestamp of the last start time of the frontend node.
LastHeartbeat	The timestamp for the last heartbeat received from the frontend node.
IsHelper	Indicates whether the frontend node is a helper node (true/false).
ErrMsg	Any error messages reported by the frontend node.

Field	Description
Version	The version of the frontend node.
CurrentConnected	Indicates whether the frontend node is currently connected to the cluster (Yes/No).

Examples

show frontends cluster information

```
select * from frontends();
```

```
+----------------------------------------+------------+-------------+----------+-----------+---------+--------------------+--------
--+----------+-----------+------+-------+------------------+---------------------+---------------------+----------+--------+-------
-----------------+-----------------+
| Name                                   | Host       | EditLogPort | HttpPort | QueryPort | RpcPort | ArrowFlightSqlPort | Role
| IsMaster | ClusterId | Join | Alive | ReplayedJournalId | LastStartTime       | LastHeartbeat       | IsHelper | ErrMsg | Version
| CurrentConnected |
+----------------------------------------+------------+-------------+----------+-----------+---------+--------------------+--------
--+----------+-----------+------+-------+------------------+---------------------+---------------------+----------+--------+-------
-----------------+-----------------+
| fe_f4642d47_62a2_44a2_b79d_3259050ab9de | 10.xx.xx.90 | 9010        | 8030     | 9030      | 9020    | -1                 | FOLLOWER
| true     | 917153130 | true | true  | 555248            | 2025-01-13 14:11:31 | 2025-01-16 14:27:56 | true     |        | doris-
0.0.0--83f899b32b | Yes              |
+----------------------------------------+------------+-------------+----------+-----------+---------+--------------------+--------
--+----------+-----------+------+-------+------------------+---------------------+---------------------+----------+--------+-------
-----------------+-----------------+
```

FRONTENDS_DISKS

Description

The `frontends_disks` table function generates a temporary table that allows you to view the disk information of FE nodes in the current Doris cluster. Its result is essentially the same as the information displayed by the `show frontends disks` statement, but the field types in `frontends_disks()` are more explicit, and you can use the table generated by the TVF for filtering, joining, and other operations.

This function can be used in the `FROM` clause.

Syntax

```
FRONTENDS_DISKS()
```

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV	global	

Return Value

View the returned fields of the frontends_disks() function

```
desc function frontends_disks();
```

```
+------------+------+------+-------+---------+-------+
| Field      | Type | Null | Key   | Default | Extra |
+------------+------+------+-------+---------+-------+
| Name       | text | No   | false | NULL    | NONE  |
| Host       | text | No   | false | NULL    | NONE  |
| DirType    | text | No   | false | NULL    | NONE  |
| Dir        | text | No   | false | NULL    | NONE  |
| Filesystem | text | No   | false | NULL    | NONE  |
| Capacity   | text | No   | false | NULL    | NONE  |
| Used       | text | No   | false | NULL    | NONE  |
| Available  | text | No   | false | NULL    | NONE  |
| UseRate    | text | No   | false | NULL    | NONE  |
| MountOn    | text | No   | false | NULL    | NONE  |
+------------+------+------+-------+---------+-------+
```

The field meanings are as follows:

Field	Type	Explanation
`Name`	TEXT	The unique identifier for the frontend node (FE).
`Host`	TEXT	The IP address or hostname of the frontend node.

Field	Type	Explanation
DirType	TEXT	The type of directory (e.g., meta, log, audit-log, temp, deploy).
Dir	TEXT	The path to the directory on the disk for the specified directory type.
Filesystem	TEXT	The filesystem type.
Capacity	TEXT	The total storage capacity of the disk.
Used	TEXT	The amount of disk space that has been used.
Available	TEXT	The amount of disk space that is available for use.
UseRate	TEXT	The percentage of disk space that is used.
MountOn	TEXT	The mount point of the disk in the file system.

example

```
select * from frontends_disks();
```

```
+-----------------------------------------+------------+-----------+-----------------------------------------------------------+----
----------+----------+------+-----------+---------+------------+
| Name                                    | Host       | DirType   | Dir                                                       |
Filesystem   | Capacity | Used | Available | UseRate | MountOn    |
+-----------------------------------------+------------+-----------+-----------------------------------------------------------+----
----------+----------+------+-----------+---------+------------+
| fe_f4642d47_62a2_44a2_b79d_3259050ab9de | 10.x.x.6 | meta      | /mnt/disk2/doris/fe/doris-meta | /dev/nvme1n1 | 3T       | 3T   |
223G      | 94%      | /mnt/disk2                             |
| fe_f4642d47_62a2_44a2_b79d_3259050ab9de | 10.x.x.6 | log       | /mnt/disk2/doris/fe/log        | /dev/nvme1n1 | 3T       | 3T   |
223G      | 94%      | /mnt/disk2                             |
| fe_f4642d47_62a2_44a2_b79d_3259050ab9de | 10.x.x.6 | audit-log | /mnt/disk2/doris/fe/log        | /dev/nvme1n1 | 3T       | 3T   |
223G      | 94%      | /mnt/disk2                             |
| fe_f4642d47_62a2_44a2_b79d_3259050ab9de | 10.x.x.6 | temp      | /mnt/disk2/doris/fe/temp_dir   | /dev/nvme1n1 | 3T       | 3T   |
223G      | 94%      | /mnt/disk2                             |
| fe_f4642d47_62a2_44a2_b79d_3259050ab9de | 10.x.x.6 | deploy    | /mnt/disk2/doris/fe            | /dev/nvme1n1 | 3T       | 3T   |
223G      | 94%      | /mnt/disk2                             |
+-----------------------------------------+------------+-----------+-----------------------------------------------------------+----
----------+----------+------+-----------+---------+------------+
```

HDFS | Table Valued Functions

HDFS

Description

HDFS table-valued-function(tvf), allows users to read and access file contents on S3-compatible object storage, just like accessing relational table. Currently supports `csv/csv_with_names/csv_with_names_and_types/json/parquet/orc` file format.

Syntax

```
HDFS(
    "uri" = "<uri>",
    "fs.defaultFS" = "<fs_defaultFS>",
    "hadoop.username" = "<hadoop_username>",
    "format" = "<format>",
    [, "<optional_property_key>" = "<optional_property_value>" [, ...] ]
  );
```

Required Parameters

Parameter	Description
`uri`	The URI for accessing HDFS. If the URI path does not exist or the file is empty, the HDFS TVF will return an empty set.
`fs.defaultFS`	The default file system URI for HDFS
`hadoop.username`	Required, can be any string but cannot be empty.
`format`	File format, required. Currently supports `csv/csv_with_names/csv_with_names_and_types/json/parquet/orc/avro`.

Optional Parameters

`optional_property_key` in the above syntax can select the corresponding parameter from the following list as needed, and `optional_property_value` is the value of the parameter

Parameter	Description	Remarks
`hadoop.security.authentication`	HDFS security authentication type	
`hadoop.username`	Alternative HDFS username	
`hadoop.kerberos.principal`	Kerberos principal	
`hadoop.kerberos.keytab`	Kerberos keytab	

Parameter	Description	Remarks
dfs.client.read.shortcircuit	Enable short-circuit read	
dfs.domain.socket.path	Domain socket path	
dfs.nameservices	The nameservice for HA mode	
dfs.ha.namenodes.your-nameservices	Configuration for namenode in HA mode	
dfs.namenode.rpc-address.your-nameservices.your-namenode	Specify the RPC address for the namenode	
dfs.client.failover.proxy.provider.your-nameservices	Specify the proxy provider for failover	
column_separator	Column separator, default is \t	
line_delimiter	Line separator, default is \n	
compress_type	Supported types: UNKNOWN/PLAIN/GZ/LZO/BZ2/LZ4FRAME/DEFLATE/SNAPPYBLOCK. Default is UNKNOWN, and the type will be automatically inferred based on the URI suffix.	
read_json_by_line	For JSON format imports, default is true	Reference: JSON Load
strip_outer_array	For JSON format imports, default is false	Reference: JSON Load
json_root	For JSON format imports, default is empty	Reference: JSON Load
json_paths	For JSON format imports, default is empty	Reference: JSON Load
num_as_string	For JSON format imports, default is false	Reference: JSON Load
fuzzy_parse	For JSON format imports, default is false	Reference: JSON Load
trim_double_quotes	For CSV format imports, boolean type. Default is false. If true, removes the outermost double quotes from each field.	
skip_lines	For CSV format imports, integer type. Default is 0. Skips the first few lines of the CSV file. This parameter is ignored if csv_with_names or csv_with_names_and_types is set.	
path_partition_keys	Specify the partition column names carried in the file path, for example /path/to/city=beijing/date="2023-07-09", then fill in path_partition_keys="city,date", which will automatically read the corresponding column names and values from the path for import.	
resource	Specify the resource name. HDFS TVF can directly access HDFS using an existing HDFS resource. Refer to CREATE-RESOURCE for creating HDFS resources.	Supported from version 2.1.4 and above.

Access Control Requirements

Privilege	Object	Notes
USAGE_PRIV	table	
SELECT_PRIV	table	

Examples

- Read and access csv format files on hdfs storage.

```
select * from hdfs(
             "uri" = "hdfs://127.0.0.1:842/user/doris/csv_format_test/student.csv",
             "fs.defaultFS" = "hdfs://127.0.0.1:8424",
             "hadoop.username" = "doris",
             "format" = "csv");
```

```
  +------+---------+------+
  | c1   | c2      | c3   |
  +------+---------+------+
  | 1    | alice   | 18   |
  | 2    | bob     | 20   |
  | 3    | jack    | 24   |
  | 4    | jackson | 19   |
  | 5    | liming  | 18   |
  +------+---------+------+
```

- Read and access csv format files on hdfs storage in HA mode.

```
select * from hdfs(
            "uri" = "hdfs://127.0.0.1:842/user/doris/csv_format_test/student.csv",
            "fs.defaultFS" = "hdfs://127.0.0.1:8424",
            "hadoop.username" = "doris",
            "format" = "csv",
            "dfs.nameservices" = "my_hdfs",
            "dfs.ha.namenodes.my_hdfs" = "nn1,nn2",
            "dfs.namenode.rpc-address.my_hdfs.nn1" = "nanmenode01:8020",
            "dfs.namenode.rpc-address.my_hdfs.nn2" = "nanmenode02:8020",
            "dfs.client.failover.proxy.provider.my_hdfs" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider");
```

```
  +------+---------+------+
  | c1   | c2      | c3   |
  +------+---------+------+
  | 1    | alice   | 18   |
  | 2    | bob     | 20   |
  | 3    | jack    | 24   |
  | 4    | jackson | 19   |
  | 5    | liming  | 18   |
  +------+---------+------+
```

- Can be used with `desc function` :

```
desc function hdfs(
            "uri" = "hdfs://127.0.0.1:8424/user/doris/csv_format_test/student_with_names.csv",
            "fs.defaultFS" = "hdfs://127.0.0.1:8424",
            "hadoop.username" = "doris",
            "format" = "csv_with_names");
```

ICEBERG_META

Description

iceberg_meta table-valued-function(tvf), Use for read iceberg metadata, operation history, snapshots of table, file metadata etc.

Syntax

```
ICEBERG_META(
    "table" = "<table>",
    "query_type" = "<query_type>"
  );
```

Required Parameters

Each parameter in the `iceberg_meta` table function (tvf) is a `"key"="value"` pair.

Field	Description
`<table>`	The full table name, which must be specified in the format of `database_name.table_name` for the Iceberg table that you want to view.
`<query_type>`	The type of metadata you want to view. Currently, only `snapshots` is supported.

Examples

- Read and access the iceberg tabular metadata for snapshots.

  ```
  select * from iceberg_meta("table" = "ctl.db.tbl", "query_type" = "snapshots");
  ```

- Can be used with `desc function` :

  ```
  desc function iceberg_meta("table" = "ctl.db.tbl", "query_type" = "snapshots");
  ```

- Inspect the iceberg table snapshots : `sql select * from iceberg_meta("table" = "iceberg_ctl.test_db.test_tbl", "query_type" = "snapshots"); text +-----------------------+---------------------+---------------+-----------+-------------------+-----------------------------+ | committed_at | snapshot_id | parent_id | operation | manifest_list | summary | +-----------------------+---------------------+---------------+-----------+-------------------+-----------------------------+ | 2022-09-20 11:14:29 | 64123452344 | -1 | append | hdfs:/path/to/m1 | {"flink.job-id":"xxm1", ...} | | 2022-09-21 10:36:35 | 98865735822 | 64123452344 | overwrite | hdfs:/path/to/m2 | {"flink.job-id":"xxm2", ...} | | 2022-09-21 21:44:11 | 51232845315 | 98865735822 | overwrite | hdfs:/path/to/m3 | {"flink.job-id":"xxm3", ...} | +-----------------------+---------------------+---------------+-----------+-------------------+-----------------------------+`

- Filtered by snapshot_id :

```
select * from iceberg_meta("table" = "iceberg_ctl.test_db.test_tbl", "query_type" = "snapshots") where snapshot_id =
98865735822;
```

```
+-----------------------+----------------+---------------+-----------+------------------+-----------------------------+
|      committed_at     |  snapshot_id   |   parent_id   | operation |   manifest_list  |            summary          |
+-----------------------+----------------+---------------+-----------+------------------+-----------------------------+
|  2022-09-21 10:36:35  |  98865735822   |  64123452344  | overwrite | hdfs:/path/to/m2 | {"flink.job-id":"xxm2", ...} |
+-----------------------+----------------+---------------+-----------+------------------+-----------------------------+
```

JOBS

Description

Table function, generating a temporary task table, which can view job information in a certain task type.

Syntax

```
JOBS(
    "type"="<type>"
)
```

Parameters

Parameter	Description
`<type>`	The type of the job: `insert`: Insert into type job. `mv`: Materialized view job.

Return Value

This table function returns zero or more rows. The function itself does not return NULL. Individual fields can be empty or `NULL` when the corresponding metadata is unavailable.

- `jobs("type"="insert")` Job return value of type insert

Parameter	Description
Id	Job ID
Name	Job name
Definer	Job definer
ExecuteType	Execution type
RecurringStrategy	Recurring strategy
Status	Job status
ExecuteSql	Execution SQL
CreateTime	Job creation time
SucceedTaskCount	Number of successful tasks
FailedTaskCount	Number of failed tasks
CanceledTaskCount	Number of canceled tasks
Comment	Job comment

- `jobs("type"="mv")` MV type job return value

Parameter	Description
Id	Job ID. For MV jobs, it corresponds to `tasks("type"="mv").JobId`.
Name	Job name. For MV jobs, it corresponds to `mv_infos("database"="...").JobName` and `tasks("type"="mv").JobName`.
MvId	ID of the materialized view maintained by this job.
MvName	Name of the materialized view maintained by this job.
MvDatabaseId	ID of the database that contains the materialized view.
MvDatabaseName	Name of the database that contains the materialized view.
ExecuteType	Job execution type. For MV jobs, possible values are `MANUAL` and `RECURRING`. `MANUAL` means refresh tasks are triggered manually, by commit, or by internal events. `RECURRING` means refresh tasks are scheduled periodically.
RecurringStrategy	Scheduling description derived from `ExecuteType`. `MANUAL TRIGGER` means the job does not run on a timer. `EVERY ... STARTS ... [ENDS ...]` means the job runs periodically.
Status	Job status. Possible values: `PENDING` means the job is waiting for scheduling; `RUNNING` means the job can produce tasks; `PAUSED` means the job is paused and can be resumed; `STOPPED` means the job has been stopped and cannot be resumed; `FINISHED` means the job has finished.
CreateTime	Time when the job was created.

MV job enum fields

The following enum fields are commonly used when checking materialized view refresh jobs:

- `ExecuteType`: how the job creates refresh tasks.
  - `MANUAL`: the job does not run on its own timer. A task is created only when a refresh is triggered manually, by commit, or by the system.
  - `RECURRING`: the job runs on a schedule and periodically creates refresh tasks.
- `RecurringStrategy`: human-readable scheduling rule generated from `ExecuteType`.
  - `MANUAL TRIGGER`: the job is not scheduled periodically. This is the usual value when `ExecuteType` is `MANUAL`.
  - `EVERY <interval> <unit> STARTS <time> [ENDS <time>]`: the job is scheduled periodically. For example, `EVERY 10 MINUTE STARTS 2025-01-17 14:42:53`.
- `Status`: current state of the job itself.
  - `PENDING`: the job is waiting to be scheduled.
  - `RUNNING`: the job is active and can create refresh tasks.
  - `PAUSED`: the job is paused. It will not create new refresh tasks until it is resumed.
  - `STOPPED`: the job has been stopped and cannot be resumed.
  - `FINISHED`: the job has finished. This is uncommon for long-lived MV refresh jobs, but it can appear in the generic job framework.

For materialized views, one materialized view has one refresh job, and one refresh job can create many refresh tasks. Use `jobs("type"="mv").Name` to connect the job to `mv_infos("database"="...").JobName` and `tasks("type"="mv").JobName`.

Examples

View the refresh job of a materialized view.

```
select *
from jobs("type"="mv")
where MvDatabaseName = "test" and MvName = "mv1"\G
```

```
*************************** 1. row ***************************
               Id: 19508
             Name: inner_mtmv_19494
             MvId: 19494
           MvName: mv1
     MvDatabaseId: 16016
   MvDatabaseName: test
      ExecuteType: MANUAL
RecurringStrategy: MANUAL TRIGGER
           Status: RUNNING
       CreateTime: 2025-01-07 22:13:31
```

In this result:

- `Id` is the MV refresh job ID. It is the same value as `JobId` in `tasks("type"="mv")`.
- `Name` is the MV refresh job name. It is the same value as `JobName` in `mv_infos("database"="test")` and `tasks("type"="mv")`.
- `MvId` and `MvName` identify the materialized view maintained by this job.
- `MvDatabaseId` and `MvDatabaseName` identify the database that contains the materialized view.
- `ExecuteType` is `MANUAL`, so this job does not run by its own timer. A refresh task is created when the materialized view is manually refreshed, refreshed on commit, or triggered by the system.
- `RecurringStrategy` is `MANUAL TRIGGER`, which matches `ExecuteType = MANUAL`. For a scheduled materialized view, this field is displayed as `EVERY ... STARTS ...`.
- `Status` is `RUNNING`, so the job can generate refresh tasks. If it is `PAUSED`, resume the materialized view job before expecting new tasks.
- `CreateTime` is the time when this MV refresh job was created.

To view the tasks generated by this job, use the job name:

```
select TaskId, JobName, MvName, Status, CreateTime, FinishTime
from tasks("type"="mv")
where JobName = "inner_mtmv_19494"
order by CreateTime desc;
```

```
+-----------------+------------------+--------+---------+---------------------+---------------------+
| TaskId          | JobName          | MvName | Status  | CreateTime          | FinishTime          |
+-----------------+------------------+--------+---------+---------------------+---------------------+
| 437156301250803 | inner_mtmv_19494 | mv1    | SUCCESS | 2025-01-07 22:13:48 | 2025-01-07 22:17:45 |
+-----------------+------------------+--------+---------+---------------------+---------------------+
```

View all insert jobs

```
select * from jobs("type"="insert");
```

```
+----------------+----------------+---------+-------------+-----------------------------------------+---------+-----------------
----------------------------------------+--------------------+-----------------+-----------------+------------------+--------
-+
| Id             | Name           | Definer | ExecuteType | RecurringStrategy                       | Status  | ExecuteSql
| CreateTime          | SucceedTaskCount | FailedTaskCount | CanceledTaskCount | Comment |
+----------------+----------------+---------+-------------+-----------------------------------------+---------+-----------------
----------------------------------------+--------------------+-----------------+-----------------+------------------+--------
-+
| 78533940810334 | insert_tab_job | root    | RECURRING   | EVERY 10 MINUTE STARTS 2025-01-17 14:42:53 | RUNNING | INSERT INTO
test.insert_tab SELECT * FROM test.example_table | 2025-01-17 14:32:53 | 0                | 0               | 0                |
|
+----------------+----------------+---------+-------------+-----------------------------------------+---------+-----------------
----------------------------------------+--------------------+-----------------+-----------------+------------------+--------
-+
```

LOCAL

Description

Local table-valued-function(tvf), allows users to read and access local file contents on be node, just like accessing relational table. Currently supports `csv/csv_with_names/csv_with_names_and_types/json/parquet/orc` file format.

syntax

```
LOCAL(
  "file_path" = "<file_path>",
  "backend_id" = "<backend_id>",
  "format" = "<format>"
  [, "<optional_property_key>" = "<optional_property_value>" [, ...] ]
  );
```

Required Parameters

Parameter	Description	Remarks
`file_path`	The path of the file to be read, which is relative to the `user_files_secure_path` directory. The `user_files_secure_path` parameter is a BE configuration item. The path cannot include `..`, and glob syntax can be used for pattern matching, such as `logs/*.log`.	
`backend_id`	The ID of the BE node where the file is located. It can be obtained via the `show backends` command.	Before version 2.1.1, Doris only supports specifying a BE node to read local data files on that node.
`format`	The file format, which is required. Supported formats are `csv/csv_with_names/csv_with_names_and_types/json/parquet/orc`.	

Optional Parameters

Parameter	Description	Remarks
`shared_storage`	Defaults to false. If true, the specified file is located on shared storage (e.g., NAS). The shared storage must support POSIX file interfaces and be mounted on all BE nodes. When `shared_storage` is true, `backend_id` can be omitted. Doris may utilize all BE nodes to access the data. If `backend_id` is set, the data will be accessed only on the specified BE node.	Supported starting from version 2.1.2
`column_separator`	The column separator, optional, defaults to `\t`.	
`line_delimiter`	The line delimiter, optional, defaults to `\n`.	

Parameter	Description	Remarks
`compress_type`	The compression type, optional. Supported types are `UNKNOWN/PLAIN/GZ/LZO/BZ2/LZ4FRAME/DEFLATE/SNAPPYBLOCK`. Defaults to `UNKNOWN`, and the type will be automatically inferred from the `uri` suffix.	
`read_json_by_line`	For JSON format imports, optional, defaults to `true`.	Refer to: Json Load
`strip_outer_array`	For JSON format imports, optional, defaults to `false`.	Refer to: Json Load
`json_root`	For JSON format imports, optional, defaults to empty.	Refer to: Json Load
`json_paths`	For JSON format imports, optional, defaults to empty.	Refer to: Json Load
`num_as_string`	For JSON format imports, optional, defaults to `false`.	Refer to: Json Load
`fuzzy_parse`	For JSON format imports, optional, defaults to `false`.	Refer to: Json Load
`trim_double_quotes`	For CSV format imports, optional, defaults to `false`. If true, it will trim the outermost double quotes around each field in the CSV file.	For CSV format
`skip_lines`	For CSV format imports, optional, defaults to `0`, which means skipping the first few lines of the CSV file. When the format is `csv_with_names` or `csv_with_names_and_types`, this parameter is ignored.	For CSV format
`path_partition_keys`	Optional, specifies the partition column names carried in the file path, e.g., `/path/to/city=beijing/date="2023-07-09"`, then fill in `path_partition_keys="city,date"`. This will automatically read the corresponding column names and values from the path for import.	

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV	global	

Usage Notes

- For more detailed usage of local tvf, please refer to S3 tvf, The only difference between them is the way of accessing the storage system.
- Access data on NAS through local tvf

  NAS shared storage allows to be mounted to multiple nodes at the same time. Each node can access files in the shared storage just like local files. Therefore, the NAS can be thought of as a local file system, accessed through local tvf.

When setting `"shared_storage" = "true"`, Doris will think that the specified file can be accessed from any BE node. When a set of files is specified using wildcards, Doris will distribute requests to access files to multiple BE nodes, so that multiple nodes can be used to perform distributed file scanning and improve query performance.

Examples

Analyze the log file on specified BE:

```
select * from local(
        "file_path" = "log/be.out",
        "backend_id" = "10006",
        "format" = "csv")
       where c1 like "%start_time%" limit 10;
```

```
+-----------------------------------------------------+
| c1                                                  |
+-----------------------------------------------------+
| start time: 2023 年 08 月 07 日 星期一 23:20:32 CST       |
| start time: 2023 年 08 月 07 日 星期一 23:32:10 CST       |
| start time: 2023 年 08 月 08 日 星期二 00:20:50 CST       |
| start time: 2023 年 08 月 08 日 星期二 00:29:15 CST       |
+-----------------------------------------------------+
```

Read and access csv format files located at path `${DORIS_HOME}/student.csv`:

```
select * from local(
      "file_path" = "student.csv",
      "backend_id" = "10003",
      "format" = "csv");
```

```
+------+---------+--------+
| c1   | c2      | c3     |
+------+---------+--------+
| 1    | alice   | 18     |
| 2    | bob     | 20     |
| 3    | jack    | 24     |
| 4    | jackson | 19     |
| 5    | liming  | d18    |
+------+---------+--------+
```--+---------+--------+
```

Query files on NAS:

```
select * from local(
        "file_path" = "/mnt/doris/prefix_*.txt",
        "format" = "csv",
        "column_separator" =",",
        "shared_storage" = "true");
```

```
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
| 1    | 2    | 3    |
```
```

```
| 1    | 2    | 3    |
| 1    | 2    | 3    |
| 1    | 2    | 3    |
| 1    | 2    | 3    |
+------+------+------+
```

Can be used with `desc function` :

```
desc function local(
      "file_path" = "student.csv",
      "backend_id" = "10003",
      "format" = "csv");
```

```
+-------+------+------+-------+---------+-------+
| Field | Type | Null | Key   | Default | Extra |
+-------+------+------+-------+---------+-------+
| c1    | TEXT | Yes  | false | NULL    | NONE  |
| c2    | TEXT | Yes  | false | NULL    | NONE  |
| c3    | TEXT | Yes  | false | NULL    | NONE  |
+-------+------+------+-------+---------+-------+
```

MV_INFOS

Description

Table function, generating temporary tables for asynchronous materialized views, which can view information about asynchronous materialized views created in a certain database.

Syntax

```
MV_INFOS("database"="<database>")
```

Parameters

Parameter	Description
<database>	Required. String type. Name of the database whose asynchronous materialized view metadata you want to query.

Return Value

This table function returns zero or more rows describing asynchronous materialized views. The function itself does not return NULL. Individual fields can be empty or `NULL` when the corresponding metadata is unavailable.

Field	Type	Description
Id	BIGINT	Materialized view ID
Name	TEXT	Materialized view name
JobName	TEXT	Name of the refresh job corresponding to the materialized view. It can be used to query `jobs("type"="mv")` and `tasks("type"="mv")`.
State	TEXT	Metadata state of the materialized view. Possible values: `INIT`, `NORMAL`, and `SCHEMA_CHANGE`.
SchemaChangeDetail	TEXT	Reason why `State` becomes `SCHEMA_CHANGE`. Empty when the materialized view is not in schema change state.
RefreshState	TEXT	State of the latest refresh. Possible values: `INIT`, `SUCCESS`, and `FAIL`.
RefreshInfo	TEXT	Refresh strategy defined for the materialized view, including build mode, refresh method, and trigger mode.
QuerySql	TEXT	SQL query defined for the materialized view
EnvInfo	TEXT	Environment information when the materialized view was created
MvProperties	TEXT	Materialized view properties
MvPartitionInfo	TEXT	Partition information of the materialized view
SyncWithBaseTables	BOOLEAN	Whether the data of the materialized view is synchronized with its base tables. To check which partition is not synchronized, use SHOW PARTITIONS.

`RefreshInfo` is displayed as `BUILD <BuildMode> REFRESH <RefreshMethod> ON <RefreshTrigger> [schedule]`. The meaning of each part is described in the enum section below.

MV info enum fields

The following enum fields are commonly used when checking materialized view definitions and health:

- `State`: metadata state of the materialized view.
  - `INIT`: the materialized view has been created but has not reached normal usable metadata state yet.
  - `NORMAL`: the materialized view metadata is normal. This is the expected state in most cases.
  - `SCHEMA_CHANGE`: a schema change on a base table or related object affected this materialized view. Check `SchemaChangeDetail` for the reason. The materialized view can usually be queried directly, but it may not be available for transparent rewrite until it is refreshed successfully.
- `RefreshState`: result state of the latest refresh.
  - `INIT`: no refresh result has been recorded yet.
  - `SUCCESS`: the latest refresh finished successfully.
  - `FAIL`: the latest refresh failed. Use `JobName` to query `tasks("type"="mv")` and check the failed task's `ErrorMsg`.
- `RefreshInfo.BuildMode`: when Doris builds the materialized view data.
  - `IMMEDIATE`: build the materialized view immediately after creation.
  - `DEFERRED`: do not build it at creation time. The materialized view needs a later refresh before it has fresh data.
- `RefreshInfo.RefreshMethod`: how Doris chooses data to refresh.
  - `COMPLETE`: always refresh the materialized view completely.
  - `AUTO`: Doris decides whether to refresh all partitions or only changed partitions.
- `RefreshInfo.RefreshTrigger`: what triggers refresh tasks.
  - `MANUAL`: refresh is triggered manually.
  - `COMMIT`: refresh is triggered by data changes on related base tables.
  - `SCHEDULE`: refresh is triggered by a schedule. The schedule details appear after `ON SCHEDULE` in `RefreshInfo`.
- `MvPartitionInfo.partitionType`: how the materialized view is partitioned.
  - `FOLLOW_BASE_TABLE`: materialized view partitions follow a base table partition column.
  - `SELF_MANAGE`: the materialized view manages its own partitions.

> **Version**
>
> Doris 2.1.x supports `FOLLOW_BASE_TABLE` and `SELF_MANAGE` for `MvPartitionInfo.partitionType`. The `EXPR` partition type is supported since Doris 3.0.0.

- `SyncWithBaseTables`: whether materialized view data is synchronized with base tables.
  - `1` or `true`: synchronized.
  - `0` or `false`: not fully synchronized. For partitioned materialized views, use `SHOW PARTITIONS FROM <mv_name>` to check partition-level synchronization.

Examples

View a materialized view under database `test`.

```
select *
from mv_infos("database"="test")
where Name = "mv1"\G
```

```
*************************** 1. row ***************************
                Id: 19494
              Name: mv1
           JobName: inner_mtmv_19494
             State: NORMAL
SchemaChangeDetail:
      RefreshState: SUCCESS
       RefreshInfo: BUILD DEFERRED REFRESH AUTO ON MANUAL
          QuerySql: SELECT `internal`.`test`.`user`.`k2`, `internal`.`test`.`user`.`k3` FROM `internal`.`test`.`user`
           EnvInfo: EnvInfo{ctlId='0', dbId='16813'}
      MvProperties: {partition_sync_limit=100, partition_sync_time_unit=YEAR}
   MvPartitionInfo: MTMVPartitionInfo{partitionType=FOLLOW_BASE_TABLE, relatedTable=user, relatedCol='k2', partitionCol='k2'}
SyncWithBaseTables: 1
```

In this result:

- `Id` and `Name` identify the materialized view.
- `JobName` is the refresh job name for this materialized view. Use it to query the job or refresh tasks, for example `select * from jobs("type"="mv") where Name = "inner_mtmv_19494";`.
- `State` is `NORMAL`, which means the materialized view metadata is normal. `INIT` means the materialized view has just been created or initialized. `SCHEMA_CHANGE` means a schema change on a base table affected this materialized view; in this state, check `SchemaChangeDetail`.
- `SchemaChangeDetail` is empty because `State` is not `SCHEMA_CHANGE`.
- `RefreshState` is `SUCCESS`, which means the latest refresh succeeded. `INIT` means no successful refresh state has been recorded yet. `FAIL` means the latest refresh failed; query `tasks("type"="mv")` with `JobName` to find the failed task and its `ErrorMsg`.
- `RefreshInfo` is `BUILD DEFERRED REFRESH AUTO ON MANUAL`. `BUILD DEFERRED` means the materialized view was not built immediately at creation time. `REFRESH AUTO` means Doris decides whether to refresh all partitions or only changed partitions. `ON MANUAL` means refresh is triggered manually rather than by a schedule.
- `QuerySql` is the query definition of the materialized view.
- `EnvInfo` records the environment information when the materialized view was created.
- `MvProperties` shows the properties of the materialized view. In this example, partition synchronization is limited by `partition_sync_limit=100` and `partition_sync_time_unit=YEAR`.
- `MvPartitionInfo` shows how the materialized view is partitioned. In Doris 2.1.x, `FOLLOW_BASE_TABLE` means the materialized view follows a base table partition column, and `SELF_MANAGE` means the materialized view manages its own partitions. The `EXPR` partition type is supported since Doris 3.0.0.
- `SyncWithBaseTables` is `1`, which means the materialized view data is synchronized with its base tables. `0` means it is not fully synchronized. For partitioned materialized views, use `SHOW PARTITIONS FROM <mv_name>` to check partition-level synchronization.

To view the latest refresh task of this materialized view:

```
select TaskId, JobName, MvName, Status, ErrorMsg, CreateTime, FinishTime
from tasks("type"="mv")
where JobName = "inner_mtmv_19494"
order by CreateTime desc
limit 1;
```

```
+-----------------+------------------+--------+---------+----------+---------------------+---------------------+
| TaskId          | JobName          | MvName | Status  | ErrorMsg | CreateTime          | FinishTime          |
+-----------------+------------------+--------+---------+----------+---------------------+---------------------+
| 437156301250803 | inner_mtmv_19494 | mv1    | SUCCESS |          | 2025-01-07 22:13:48 | 2025-01-07 22:17:45 |
+-----------------+------------------+--------+---------+----------+---------------------+---------------------+
```

NUMBERS

Description

Table function that generates a temporary table containing only one column with the column name `number` and all element values are `const_value` if `const_value` is specified, otherwise they are [0,`number`) incremented.

Syntax

```
NUMBERS(
    "number" = "<number>"
    [, "const_value" = "<const_value>" ]
  );
```

Required Parameters

Field	Description
number	The number of rows

Optional Parameters

Field	Description
const_value	Specifies the constant value generated

Return Value

Field	Type	Description
number	BIGINT	The value returned for each row

Examples

```
select * from numbers("number" = "5");
```

```
+--------+
| number |
+--------+
|      0 |
|      1 |
|      2 |
|      3 |
|      4 |
+--------+
```

```
select * from numbers("number" = "5", "const_value" = "-123");
```

```
+--------+
| number |
+--------+
|   -123 |
|   -123 |
|   -123 |
|   -123 |
|   -123 |
+--------+
```

PARTITION_VALUES

Description

Table function that generates a temporary table of partition values, allowing you to view the list of partition values for a specific TABLE.

This function is used in the FROM clause and only supports hive tables.

Syntax

```
PARTITION_VALUES(
    "catalog"="<catalog>",
    "database"="<database>",
    "table"="<table>"
)
```

Required Parameters

Field	Description
<catalog>	Specifies the catalog name of the cluster to query.
<database>	Specifies the database name of the cluster to query.
<table>	Specifies the table name of the cluster to query.

Return Value

The table to be queried has several partition fields, and the table will have several columns accordingly.

Example

The table creation statement for `text_partitioned_columns` in the hive3 CATALOG under multi_catalog is as follows:

```
CREATE TABLE `text_partitioned_columns`(
  `t_timestamp` timestamp)
PARTITIONED BY (
 `t_int` int,
 `t_float` float,
 `t_string` string)
```

Data is as follows:

```
mysql> select * from text_partitioned_columns;
+---------------------------+-------+---------+----------+
| t_timestamp               | t_int | t_float | t_string |
+---------------------------+-------+---------+----------+
```

```
| 2023-01-01 00:00:00.000000 |  NULL |     0.1 | test1    |
| 2023-01-02 00:00:00.000000 |  NULL |     0.2 | test2    |
| 2023-01-03 00:00:00.000000 |   100 |     0.3 | test3    |
+----------------------------+-------+---------+----------+
```

View the list of partition values for `text_partitioned_columns` in the hive3 CATALOG under multi_catalog:

```
select * from partition_values("catalog"="hive3", "database" = "multi_catalog","table" = "text_partitioned_columns");
```

```
+-------+---------+----------+
| t_int | t_float | t_string |
+-------+---------+----------+
|   100 |     0.3 | test3    |
|  NULL |     0.2 | test2    |
|  NULL |     0.1 | test1    |
+-------+---------+----------+
```

PARTITIONS

Description

The table function generates a temporary partition TABLE, which allows you to view the PARTITION list of a certain TABLE.

Syntax

```
PARTITIONS(
    "catalog"="<catalog>",
    "database"="<database>",
    "table"="<table>"
)
```

Required Parameters

Field	Description
<catalog>	Specify the cluster catalog name to be queried.
<database>	Specify the cluster database name to be queried.
<table>	Specify the cluster table name to be queried.

Return Value

Field	Description
PartitionId	Partition ID
PartitionName	Name of the partition
VisibleVersion	Partition version
VisibleVersionTime	Time when the partition version was committed
State	State of the partition
PartitionKey	Partition key
Range	Range of the partition
DistributionKey	Distribution key for partitioning
Buckets	Number of buckets in the partition
ReplicationNum	Number of replicas in the partition
StorageMedium	Storage medium used for the partition
CooldownTime	Cooldown time for the partition
RemoteStoragePolicy	Remote storage policy for the partition

Field	Description
LastConsistencyCheckTime	Time of the last consistency check for the partition
DataSize	Size of the data in the partition
IsInMemory	Whether the partition is stored in memory
ReplicaAllocation	Replication strategy for the partition
IsMutable	Whether the partition is mutable
SyncWithBaseTables	Whether the partition is synchronized with the base table (for async materialized views)
UnsyncTables	Which base table is unsynchronized (for async materialized view partitions)

Examples

View the partition list of example_table under test in the internal catalog

```
select * from partitions("catalog"="internal","database"="test","table"="example_table");
```

```
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------+-----------------+---------+----------------+---------------+---------------------+------
--------------+--------------------------+----------+------------+-------------------------+-----------+--------------------+------
--------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------+-----------------+---------+----------------+---------------+---------------------+------
--------------+--------------------------+----------+------------+-------------------------+-----------+--------------------+------
--------+
|       43209 | p1            |              1 | 2025-01-17 12:35:22 | NORMAL | created_at   | [types: [DATEV2]; keys: [0000-01-01];
..types: [DATEV2]; keys: [2023-01-01]; ) | id              |      10 |              1 | HDD           | 9999-12-31 23:59:59 |
| \N                      | 0.000    |          0 | tag.location.default: 1 |         1 |                 1 | \N           |
|       43210 | p2            |              1 | 2025-01-17 12:35:22 | NORMAL | created_at   | [types: [DATEV2]; keys: [2023-01-01];
..types: [DATEV2]; keys: [2024-01-01]; ) | id              |      10 |              1 | HDD           | 9999-12-31 23:59:59 |
| \N                      | 0.000    |          0 | tag.location.default: 1 |         1 |                 1 | \N           |
|       43211 | p3            |              1 | 2025-01-17 12:35:22 | NORMAL | created_at   | [types: [DATEV2]; keys: [2024-01-01];
..types: [DATEV2]; keys: [2025-01-01]; ) | id              |      10 |              1 | HDD           | 9999-12-31 23:59:59 |
| \N                      | 0.000    |          0 | tag.location.default: 1 |         1 |                 1 | \N           |
|       43212 | p4            |              1 | 2025-01-17 12:35:22 | NORMAL | created_at   | [types: [DATEV2]; keys: [2025-01-01];
..types: [DATEV2]; keys: [2026-01-01]; ) | id              |      10 |              1 | HDD           | 9999-12-31 23:59:59 |
| \N                      | 0.000    |          0 | tag.location.default: 1 |         1 |                 1 | \N           |
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------+-----------------+---------+----------------+---------------+---------------------+------
--------------+--------------------------+----------+------------+-------------------------+-----------+--------------------+------
--------+
```

View the partition information with partition name p1 under example_table

```
select * from partitions("catalog"="internal","database"="test","table"="example_table") where PartitionName = "p1";
```

```
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------+-----------------+---------+----------------+---------------+---------------------+------
--------------+--------------------------+----------+------------+-------------------------+-----------+--------------------+------
--------+
```

```
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+--------------+----------------+---------------------+--------+--------------+-------------------------------------
----------------------------------------+----------------+---------+---------------+--------------+--------------------+------
--------------+-------------------------+----------+------------+-------------------------+-----------+--------------------+------
--------+
|       43209 | p1            |              1 | 2025-01-17 12:35:22 | NORMAL | created_at   | [types: [DATEV2]; keys: [0000-01-01];
..types: [DATEV2]; keys: [2023-01-01]; ) | id              |      10 |              1 | HDD           | 9999-12-31 23:59:59 |
| \N                       | 0.000    |          0 | tag.location.default: 1 |         1 |                 1 | \N           |
+-------------+--------------+----------------+---------------------+--------+--------------+-------------------------------------
----------------------------------------+----------------+---------+---------------+--------------+--------------------+------
--------------+-------------------------+----------+------------+-------------------------+-----------+--------------------+------
--------+
```

View the partition ID with the partition name p1 under example_table

```
select PartitionId from partitions("catalog"="internal","database"="test","table"="example_table") where PartitionName = "p1";
```

```
+-------------+
| PartitionId |
+-------------+
|       43209 |
+-------------+
```

QUERY

Description

Query table function (table-valued-function, tvf) can be used to transparently transmit query statements directly to a catalog for data query

Supported by Doris version 2.1.3, currently only transparent query jdbc catalog is supported. You need to create the corresponding catalog in Doris first.

Syntax

```
QUERY(
    "catalog" = "<catalog>",
    "query" = "<query_sql>"
  );
```

Required Parameters

Each parameter in the query table function tvf is a `"key"="value"` pair.

Field	Description
`catalog`	Catalog name, which needs to be filled in according to the name of the catalog.
`query`	The query statement to be executed.

Examples

Can be used with `desc function`

```
desc function query("catalog" = "jdbc", "query" = "select * from test.student");
```

```
+-------+------+------+-------+---------+-------+
| Field | Type | Null | Key   | Default | Extra |
+-------+------+------+-------+---------+-------+
| id    | int  | Yes  | true  | NULL    |       |
| name  | text | Yes  | false | NULL    | NONE  |
+-------+------+------+-------+---------+-------+
```

Transparent query for tables in jdbc catalog data source

```
select * from query("catalog" = "jdbc", "query" = "select * from test.student");
```

```
+------+---------+
| id   | name    |
+------+---------+
```

```
| 1    | alice   |
| 2    | bob     |
| 3    | jack    |
+------+---------+
```

```
select * from query("catalog" = "jdbc", "query" = "select * from test.score");
```

```
+------+---------+
| id   | score   |
+------+---------+
| 1    | 100     |
| 2    | 90      |
| 3    | 80      |
+------+---------+
```

Transparent join query for tables in jdbc catalog data source

```
select * from query("catalog" = "jdbc", "query" = "select a.id, a.name, b.score from test.student a join test.score b on a.id =
b.id");
```

```
+------+---------+---------+
| id   | name    | score   |
+------+---------+---------+
| 1    | alice   | 100     |
| 2    | bob     | 90      |
| 3    | jack    | 80      |
+------+---------+---------+
```

S3 | Table Valued Functions

S3

Description

The S3 table-valued function (TVF) allows users to read and access files stored on S3-compatible object storage as if accessing relational table-formatted data. Currently supports `csv/csv_with_names/csv_with_names_and_types/json/parquet/orc` file formats.

Syntax

```
S3(
    "uri" = "<uri>",
    "s3.access_key" = "<s3_access_key>",
    "s3.secret_key" = "<s3_secret_key>",
    "s3.region" = "<s3_region>",
    "s3.endpoint" = "<s3_endpoint>",
    "format" = "<format>"
    [, "<optional_property_key>" = "<optional_property_value>" [, ...] ]
  )
```

Required Parameters

Parameter	Description
`uri`	URI for accessing S3. The function will use either Path Style or Virtual-hosted Style based on the `use_path_style` parameter
`s3.access_key`	Access key for S3
`s3.secret_key`	Secret key for S3
`s3.region`	S3 region
`s3.endpoint`	S3 endpoint address
`format`	File format, supports `csv/csv_with_names/csv_with_names_and_types/json/parquet/orc`

Optional Parameters

Parameter	Default	Description	Notes
s3.session_token	-	S3 session token	
use_path_style	false	S3 SDK uses Virtual-hosted Style by default. Some object storage systems may not support Virtual-hosted Style. Set this parameter to force Path Style access. For example, MinIO only allows Path Style by default, so use_path_style=true should be set when accessing MinIO	
force_parsing_by_standard_uri	false	Forces parsing of non-standard URIs as standard URIs	
column_separator	\t	Column separator	
line_delimiter	\n	Line delimiter	
compress_type	UNKNOWN	Compression type. Currently supports UNKNOWN/PLAIN/GZ/LZO/BZ2/LZ4FRAME/DEFLATE/SNAPPYBLOCK, will auto-detect based on uri suffix	
read_json_by_line	"true"	For JSON format import	JSON Load
strip_outer_array	"false"	For JSON format import	JSON Load
json_root	Empty	For JSON format import	JSON Load

Parameter	Default	Description	Notes
`jsonpaths`	Empty	For JSON format import	JSON Load
`num_as_string`	false	For JSON format import	JSON Load
`fuzzy_parse`	false	For JSON format import	
`trim_double_quotes`	false	For CSV format import, removes the outermost double quotes from each field	
`skip_lines`	0	Skips the first N lines of CSV file. This parameter is ineffective when using `csv_with_names` or `csv_with_names_and_types` format	
`path_partition_keys`	-	Specifies partition column names in file path. For example, for path `/path/to/city=beijing/date="2023-07-09"`, set `path_partition_keys="city,date"` to automatically read column names and values from path	
`resource`	-	Specifies Resource name. S3 TVF can use existing S3 Resource to access S3 directly. For creating S3 Resource, refer to CREATE-RESOURCE. Supported from version 2.1.4.	
`enable_mapping_varbinary`	false	Maps BYTE_ARRAY type to STRING when reading PARQUET/ORC. When enabled, maps to VARBINARY type instead.	Supported from version 4.0.3

Notes

1. For AWS S3, standard URI styles include:
    - AWS Client Style (Hadoop S3 Style)

      ```
      s3://my-bucket/path/to/file?versionId=abc123&partNumber=77&partNumber=88
      ```

    - Virtual Host Style

```
https://my-bucket.s3.us-west-1.amazonaws.com/resources/doc.txt?versionId=abc123&partNumber=77&partNumber=88
```

- Path Style

```
https://s3.us-west-1.amazonaws.com/my-bucket/resources/doc.txt?versionId=abc123&partNumber=77&partNumber=88
```

Besides these three standard URI styles, other URI styles are also supported (though less common):

- Virtual Host AWS Client (Hadoop S3) Mixed Style

```
s3://my-bucket.s3.us-west-1.amazonaws.com/resources/doc.txt?versionId=abc123&partNumber=77&partNumber=88
```

- Path AWS Client (Hadoop S3) Mixed Style

```
s3://s3.us-west-1.amazonaws.com/my-bucket/resources/doc.txt?versionId=abc123&partNumber=77&partNumber=88
```

For detailed usage examples, see the examples below.

2. To directly query TVF or create a View based on this TVF, USAGE permission on the Resource is required. To query a View created based on TVF, only SELECT permission on the View is needed.

Examples

- Read and access CSV format files on S3-compatible object storage

```
select * from s3("uri" = "http://127.0.0.1:9312/test2/student1.csv",
                 "s3.access_key"= "minioadmin",
                 "s3.secret_key" = "minioadmin",
                 "s3.endpoint" = "http://127.0.0.1:9312",
                 "s3.region" = "us-east-1",
                 "format" = "csv",
                 "use_path_style" = "true") order by c1;
```

- Can be used with `desc function`

```
desc function s3("uri" = "http://127.0.0.1:9312/test2/student1.csv",
                  "s3.access_key"= "minioadmin",
                  "s3.secret_key" = "minioadmin",
                  "s3.endpoint" = "http://127.0.0.1:9312",
                  "s3.region" = "us-east-1",
                  "format" = "csv",
                  "use_path_style" = "true");
```

- **Different URL Schema Usage**

  `http://`, `https://` usage examples:

```
-- Note URI Bucket notation and use_path_style parameter setting, same for HTTP.
-- Since "use_path_style"="true" is set, Path Style will be used to access S3.
select * from s3(
    "uri" = "https://endpoint/bucket/file/student.csv",
```

```
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.endpoint" = "endpoint",
    "s3.region" = "region",
    "format" = "csv",
    "use_path_style"="true");

-- Note URI Bucket notation and use_path_style parameter setting, same for HTTP.
-- Since "use_path_style"="false" is set, Virtual-hosted Style will be used to access S3.
select * from s3(
    "uri" = "https://bucket.endpoint/bucket/file/student.csv",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.endpoint" = "endpoint",
    "s3.region" = "region",
    "format" = "csv",
    "use_path_style"="false");

-- Alibaba Cloud OSS and Tencent Cloud COS use Virtual-hosted Style to access S3.
-- OSS
select * from s3(
    "uri" = "http://example-bucket.oss-cn-beijing.aliyuncs.com/your-folder/file.parquet",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.endpoint" = "oss-cn-beijing.aliyuncs.com",
    "s3.region" = "oss-cn-beijing",
    "format" = "parquet",
    "use_path_style" = "false");
-- COS
select * from s3(
    "uri" = "https://example-bucket.cos.ap-hongkong.myqcloud.com/your-folder/file.parquet",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.endpoint" = "cos.ap-hongkong.myqcloud.com",
    "s3.region" = "ap-hongkong",
    "format" = "parquet",
    "use_path_style" = "false");

-- MinIO
select * from s3(
    "uri" = "s3://bucket/file.csv",
    "s3.endpoint" = "http://172.21.0.101:9000",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.region" = "us-east-1",
    "format" = "csv"
);

-- Baidu Cloud BOS uses S3-compatible Virtual-hosted Style to access S3.
-- BOS
select * from s3(
    "uri" = "https://example-bucket.s3.bj.bcebos.com/your-folder/file.parquet",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.region" = "bj",
    "s3.endpoint" = "http://bj.bcebos.com",
    "format" = "parquet",
    "use_path_style" = "false");
```

- `s3://` usage example:

```
-- Note URI Bucket notation, no need to set use_path_style parameter.
-- Virtual-hosted Style will be used to access S3.
```

```
select * from s3(
    "uri" = "s3://bucket/file/student.csv",
    "s3.endpoint"= "endpont",
    "s3.region"= "region",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "format" = "csv");
```

- Other supported URI style examples:

```
-- Virtual Host AWS Client (Hadoop S3) Mixed Style. Use by setting use_path_style=false and
force_parsing_by_standard_uri=true.
select * from s3(
    "URI" = "s3://my-bucket.s3.us-west-1.amazonaws.com/resources/doc.txt?versionId=abc123&partNumber=77&partNumber=88",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.endpoint"= "endpont",
    "s3.region"= "region",
    "format" = "csv",
    "use_path_style"="false",
    "force_parsing_by_standard_uri"="true");

-- Path AWS Client (Hadoop S3) Mixed Style. Use by setting use_path_style=true and force_parsing_by_standard_uri=true.
select * from s3(
    "URI" = "s3://s3.us-west-1.amazonaws.com/my-bucket/resources/doc.txt?versionId=abc123&partNumber=77&partNumber=88",
    "s3.access_key"= "ak",
    "s3.secret_key" = "sk",
    "s3.endpoint"= "endpont",
    "s3.region"= "region",
    "format" = "csv",
    "use_path_style"="true",
    "force_parsing_by_standard_uri"="true");
```

- **CSV Format**

  Since S3 table-valued-function doesn't know the Table Schema in advance, it will read the file once to parse the Table Schema.

  `csv` format: S3 table-valued-function reads files on S3 and processes them as CSV files, using the first line to parse Table Schema. The number of columns `n` in the first line will be used as the number of columns in Table Schema, and column names will be automatically named as `c1, c2, ..., cn`, with all column types set to `String`. Example:

  `student1.csv` file content:

  ```
  1,ftw,12
  2,zs,18
  3,ww,20
  ```

  Using S3 TVF:

  ```
  select * from s3("uri" = "http://127.0.0.1:9312/test2/student1.csv",
                   "s3.access_key"= "minioadmin",
                   "s3.secret_key" = "minioadmin",
                   "s3.endpoint" = "http://127.0.0.1:9312",
                   "s3.region" = "us-east-1",
                   "format" = "csv",
  ```

```
                "use_path_style" = "true") order by c1;

+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
| 1    | ftw  | 12   |
| 2    | zs   | 18   |
| 3    | ww   | 20   |
+------+------+------+
```

Can use `desc function S3()` to view Table Schema:

```
Desc function s3("uri" = "http://127.0.0.1:9312/test2/student1.csv",
                 "s3.access_key"= "minioadmin",
                 "s3.secret_key" = "minioadmin",
                 "s3.endpoint" = "http://127.0.0.1:9312",
                 "s3.region" = "us-east-1",
                 "format" = "csv",
                 "use_path_style" = "true");

+-------+------+------+-------+---------+-------+
| Field | Type | Null | Key   | Default | Extra |
+-------+------+------+-------+---------+-------+
| c1    | TEXT | Yes  | false | NULL    | NONE  |
| c2    | TEXT | Yes  | false | NULL    | NONE  |
| c3    | TEXT | Yes  | false | NULL    | NONE  |
+-------+------+------+-------+---------+-------+
```

- `csv_with_names` **Format**

  `csv_with_names` format: Parses the first line of the file as the number and names of columns in Table Schema, with all column types set to `String`. Example:

  `student_with_names.csv` file content:

```
id,name,age
1,ftw,12
2,zs,18
3,ww,20
```

  Using S3 TVF:

```
select * from s3("uri" = "http://127.0.0.1:9312/test2/student_with_names.csv",
                 "s3.access_key"= "minioadmin",
                 "s3.secret_key" = "minioadmin",
                 "s3.endpoint" = "http://127.0.0.1:9312",
                 "s3.region" = "us-east-1",
                 "format" = "csv_with_names",
                 "use_path_style" = "true") order by id;

+------+------+------+
| id   | name | age  |
+------+------+------+
| 1    | ftw  | 12   |
| 2    | zs   | 18   |
| 3    | ww   | 20   |
+------+------+------+
```

- Similarly, use `desc function S3()` to view Table Schema

```
desc function s3("uri" = "http://127.0.0.1:9312/test2/student_with_names.csv",
                 "s3.access_key"= "minioadmin",
                 "s3.secret_key" = "minioadmin",
                 "s3.endpoint" = "http://127.0.0.1:9312",
                 "s3.region" = "us-east-1",
                 "format" = "csv_with_names",
                 "use_path_style" = "true");

  +-------+------+------+-------+---------+-------+
  | Field | Type | Null | Key   | Default | Extra |
  +-------+------+------+-------+---------+-------+
  | id    | TEXT | Yes  | false | NULL    | NONE  |
  | name  | TEXT | Yes  | false | NULL    | NONE  |
  | age   | TEXT | Yes  | false | NULL    | NONE  |
  +-------+------+------+-------+---------+-------+
```

- `csv_with_names_and_types` **Format**

  `csv_with_names_and_types` format: Currently does not support parsing column types from CSV files. When using this format, S3 TVF will parse the first line as the number and names of columns in Table Schema, with all column types set to String, and will ignore the second line of the file.

  `student_with_names_and_types.csv` file content:

```
id,name,age
INT,STRING,INT
1,ftw,12
2,zs,18
3,ww,20
```

  Using S3 TVF:

```
select * from s3("uri" = "http://127.0.0.1:9312/test2/student_with_names_and_types.csv",
                 "s3.access_key"= "minioadmin",
                 "s3.secret_key" = "minioadmin",
                 "s3.endpoint" = "http://127.0.0.1:9312",
                 "s3.region" = "us-east-1",
                 "format" = "csv_with_names_and_types",
                 "use_path_style" = "true") order by id;

+------+------+------+
| id   | name | age  |
+------+------+------+
| 1    | ftw  | 12   |
| 2    | zs   | 18   |
| 3    | ww   | 20   |
+------+------+------+
```

  Similarly, use `desc function S3()` to view Table Schema:

```
Desc function s3("uri" = "http://127.0.0.1:9312/test2/student_with_names_and_types.csv",
                 "s3.access_key"= "minioadmin",
                 "s3.secret_key" = "minioadmin",
                 "s3.endpoint" = "http://127.0.0.1:9312",
                 "s3.region" = "us-east-1",
```

```
                "format" = "csv_with_names_and_types",
                "use_path_style" = "true");

  +-------+------+------+-------+---------+-------+
  | Field | Type | Null | Key   | Default | Extra |
  +-------+------+------+-------+---------+-------+
  | id    | TEXT | Yes  | false | NULL    | NONE  |
  | name  | TEXT | Yes  | false | NULL    | NONE  |
  | age   | TEXT | Yes  | false | NULL    | NONE  |
  +-------+------+------+-------+---------+-------+
```

- **JSON Format**

  `json` format: JSON format involves many optional parameters. For the meaning of each parameter, refer to: Json Load. When querying JSON format files with S3 TVF, it locates a JSON object based on `json_root` and `jsonpaths` parameters, uses the `key` in the object as column names in Table Schema, with all column types set to String. Example:

  `data.json` file content:

```
[{"id":1, "name":"ftw", "age":18}]
[{"id":2, "name":"xxx", "age":17}]
[{"id":3, "name":"yyy", "age":19}]
```

  Using S3 TVF to query:

```
select * from s3(
    "uri" = "http://127.0.0.1:9312/test2/data.json",
    "s3.access_key"= "minioadmin",
    "s3.secret_key" = "minioadmin",
    "s3.endpoint" = "http://127.0.0.1:9312",
    "s3.region" = "us-east-1",
    "format" = "json",
    "strip_outer_array" = "true",
    "read_json_by_line" = "true",
    "use_path_style"="true");
+------+------+------+
| id   | name | age  |
+------+------+------+
| 1    | ftw  | 18   |
| 2    | xxx  | 17   |
| 3    | yyy  | 19   |
+------+------+------+

select * from s3(
    "uri" = "http://127.0.0.1:9312/test2/data.json",
    "s3.access_key"= "minioadmin",
    "s3.secret_key" = "minioadmin",
    "s3.endpoint" = "http://127.0.0.1:9312",
    "s3.region" = "us-east-1",
    "format" = "json",
    "strip_outer_array" = "true",
    "jsonpaths" = "[\"$.id\", \"$.age\"]",
    "use_path_style"="true");

+------+------+
| id   | age  |
+------+------+
| 1    | 18   |
| 2    | 17   |
```

```
| 3    | 19   |
+------+------+
```

- **Parquet Format**

  `parquet` format: S3 TVF supports parsing column names and types from Parquet files for Table Schema. Example:

```
select * from s3(
   "uri" = "http://127.0.0.1:9312/test2/test.snappy.parquet",
   "s3.access_key"= "minioadmin",
   "s3.secret_key" = "minioadmin",
   "s3.endpoint" = "http://127.0.0.1:9312",
   "s3.region" = "us-east-1",
   "format" = "parquet",
   "use_path_style"="true") limit 5;

+-----------+-----------------------------------------+----------------+----------+-------------------------+--------+----
---------+--------------+--------------------+
| p_partkey | p_name                                  | p_mfgr         | p_brand  | p_type                  | p_size |
p_container | p_retailprice | p_comment           |
+-----------+-----------------------------------------+----------------+----------+-------------------------+--------+----
---------+--------------+--------------------+
|         1 | goldenrod lavender spring chocolate lace | Manufacturer#1 | Brand#13 | PROMO BURNISHED COPPER  |      7 |
JUMBO PKG   |           901 | ly. slyly ironi     |
|         2 | blush thistle blue yellow saddle         | Manufacturer#1 | Brand#13 | LARGE BRUSHED BRASS     |      1 | LG
CASE     |           902 | lar accounts amo    |
|         3 | spring green yellow purple cornsilk      | Manufacturer#4 | Brand#42 | STANDARD POLISHED BRASS |     21 |
WRAP CASE   |           903 | egular deposits hag |
|         4 | cornflower chocolate smoke green pink    | Manufacturer#3 | Brand#34 | SMALL PLATED BRASS      |     14 | MED
DRUM    |           904 | p furiously r       |
|         5 | forest brown coral puff cream            | Manufacturer#3 | Brand#32 | STANDARD POLISHED TIN   |     15 | SM
PKG      |           905 |  wake carefully     |
+-----------+-----------------------------------------+----------------+----------+-------------------------+--------+----
---------+--------------+--------------------+
```

```
MySQL [(none)]> desc function s3(
    "uri" = "http://127.0.0.1:9312/test2/test.snappy.parquet",
    "s3.access_key"= "minioadmin",
    "s3.secret_key" = "minioadmin",
    "s3.endpoint" = "http://127.0.0.1:9312",
    "s3.region" = "us-east-1",
    "format" = "parquet",
    "use_path_style"="true");

+---------------+--------------+------+-------+---------+-------+
| Field         | Type         | Null | Key   | Default | Extra |
+---------------+--------------+------+-------+---------+-------+
| p_partkey     | INT          | Yes  | false | NULL    | NONE  |
| p_name        | TEXT         | Yes  | false | NULL    | NONE  |
| p_mfgr        | TEXT         | Yes  | false | NULL    | NONE  |
| p_brand       | TEXT         | Yes  | false | NULL    | NONE  |
| p_type        | TEXT         | Yes  | false | NULL    | NONE  |
| p_size        | INT          | Yes  | false | NULL    | NONE  |
| p_container   | TEXT         | Yes  | false | NULL    | NONE  |
| p_retailprice | DECIMAL(9,0) | Yes  | false | NULL    | NONE  |
| p_comment     | TEXT         | Yes  | false | NULL    | NONE  |
+---------------+--------------+------+-------+---------+-------+
```

- **ORC Format**

`orc` format: Usage is the same as `parquet` format, just set the `format` parameter to `orc`.

```
select * from s3(
    "uri" = "http://127.0.0.1:9312/test2/test.snappy.orc",
    "s3.access_key"= "minioadmin",
    "s3.secret_key" = "minioadmin",
    "s3.endpoint" = "http://127.0.0.1:9312",
    "s3.region" = "us-east-1",
    "format" = "orc",
    "use_path_style"="true") limit 5;

  +-----------+-----------------------------------------+----------------+----------+-------------------------+--------+--
-----------+---------------+--------------------+
| p_partkey | p_name                                  | p_mfgr         | p_brand  | p_type                  | p_size |
p_container | p_retailprice | p_comment          |
+-----------+-----------------------------------------+----------------+----------+-------------------------+--------+----
---------+---------------+--------------------+
|         1 | goldenrod lavender spring chocolate lace | Manufacturer#1 | Brand#13 | PROMO BURNISHED COPPER  |      7 |
JUMBO PKG   |           901 | ly. slyly ironi     |
|         2 | blush thistle blue yellow saddle         | Manufacturer#1 | Brand#13 | LARGE BRUSHED BRASS     |      1 | LG
CASE     |           902 | lar accounts amo    |
|         3 | spring green yellow purple cornsilk      | Manufacturer#4 | Brand#42 | STANDARD POLISHED BRASS |     21 |
WRAP CASE   |           903 | egular deposits hag |
|         4 | cornflower chocolate smoke green pink    | Manufacturer#3 | Brand#34 | SMALL PLATED BRASS      |     14 | MED
DRUM    |           904 | p furiously r       |
|         5 | forest brown coral puff cream            | Manufacturer#3 | Brand#32 | STANDARD POLISHED TIN   |     15 | SM
PKG      |           905 |  wake carefully     |
+-----------+-----------------------------------------+----------------+----------+-------------------------+--------+----
---------+---------------+--------------------+
```

- **URI with Wildcards**

  URI can use wildcards to read multiple files. Note: When using wildcards, ensure all files have the same format (especially `csv`, `csv_with_names`, `csv_with_names_and_types` are different formats). S3 TVF will use the first file to parse Table Schema.

  With the following two CSV files:

  ```
  // file1.csv
  1,aaa,18
  2,qqq,20
  3,qwe,19

  // file2.csv
  5,cyx,19
  6,ftw,21
  ```

  You can use wildcards in URI to query:

  ```
  select * from s3(
          "uri" = "http://127.0.0.1:9312/test2/file*.csv",
          "s3.access_key"= "minioadmin",
          "s3.secret_key" = "minioadmin",
          "s3.endpoint" = "http://127.0.0.1:9312",
          "s3.region" = "us-east-1",
          "format" = "csv",
          "use_path_style"="true");

  +------+------+------+
  ```

```
| c1   | c2   | c3   |
+------+------+------+
| 1    | aaa  | 18   |
| 2    | qqq  | 20   |
| 3    | qwe  | 19   |
| 5    | cyx  | 19   |
| 6    | ftw  | 21   |
+------+------+------+
```

- **Using S3 TVF with `insert into` and `cast`**

```
-- Create Doris internal table
CREATE TABLE IF NOT EXISTS ${testTable}
    (
        id int,
        name varchar(50),
        age int
    )
    COMMENT "my first table"
    DISTRIBUTED BY HASH(id) BUCKETS 32
    PROPERTIES("replication_num" = "1");

-- Insert data using S3
insert into ${testTable} (id,name,age)
select cast (id as INT) as id, name, cast (age as INT) as age
from s3(
    "uri" = "${uri}",
    "s3.access_key"= "${ak}",
    "s3.secret_key" = "${sk}",
    "s3.endpoint" = "http://127.0.0.1:9312",
    "s3.region" = "us-east-1",
    "format" = "${format}",
    "strip_outer_array" = "true",
    "read_json_by_line" = "true",
    "use_path_style" = "true");
```

TASKS

Description

Table function, generates a temporary table of tasks, which allows you to view the information of tasks generated by jobs in the current Doris cluster.

Syntax

```
TASKS(
    "type"="<type>"
)
```

Parameters

Parameter	Description
`<type>`	Type of the task: `insert`: insert into type task. `mv`: materialized view type task.

Return Value

This table function returns zero or more rows. The function itself does not return NULL. Individual fields can be empty, `NULL`, or `\N` when the corresponding task metadata is unavailable.

- `tasks("type"="insert")` tasks return value of type insert

Field Name	Description
TaskId	Task id
JobId	Job id
JobName	Job name
Label	Label
Status	Task status
ErrorMsg	Task failure information
CreateTime	Task creation time
FinishTime	Task completion time
TrackingUrl	Tracking URL
LoadStatistic	Load statistics
User	User

- `tasks("type"="mv")` tasks return value of type MV

Field Name	Description
TaskId	Unique ID of the refresh task. Each materialized view refresh creates a new task record.
JobId	ID of the job that generated this task. It corresponds to `jobs("type"="mv").Id`.
JobName	Name of the job that generated this task. It corresponds to `mv_infos("database"="...").JobName` and `jobs("type"="mv").Name`.
MvId	ID of the materialized view refreshed by this task.
MvName	Name of the materialized view refreshed by this task.
MvDatabaseId	ID of the database that contains the materialized view.
MvDatabaseName	Name of the database that contains the materialized view.
Status	Task status. Possible values: `PENDING` means the task is waiting to run; `RUNNING` means the task is running; `SUCCESS` means the task finished successfully; `FAILED` means the task failed; `CANCELED` means the task was canceled.
ErrorMsg	Error message when `Status` is `FAILED` or `CANCELED`. Empty when the task succeeds.
CreateTime	Time when the task record was created.
StartTime	Time when the task started running. It can be `\N` if the task has not started.
FinishTime	Time when the task finished. It can be `\N` if the task is still pending or running.
DurationMs	Runtime in milliseconds, calculated as `FinishTime - StartTime`. It can be `\N` if the task has not finished.
TaskContext	JSON string describing how the task was triggered and what the user requested. Common fields include `triggerMode`, `partitions`, and `isComplete`. `triggerMode` values are `MANUAL`, `COMMIT`, and `SYSTEM`.
RefreshMode	Actual refresh scope decided by this task. Possible values: `COMPLETE` means all materialized view partitions were refreshed; `PARTIAL` means only some partitions were refreshed; `NOT_REFRESH` means no partition needed refreshing.
NeedRefreshPartitions	JSON array of materialized view partitions that needed refreshing in this task. Empty array means no partition needed refreshing.
CompletedPartitions	JSON array of partitions that were refreshed successfully. Compare it with `NeedRefreshPartitions` to see whether all required partitions completed.
Progress	Refresh progress in the format `percentage (completed/total)`, for example `100.00% (1/1)`. It can be `\N` when there is no partition to refresh.

MV task enum fields

The following enum fields are commonly used when checking materialized view refresh tasks:

- `Status`: lifecycle state of the task.
  - `PENDING`: the task has been created but has not started running. It is waiting for scheduling or resources.
  - `RUNNING`: the task is currently running.
  - `SUCCESS`: the task finished successfully.
  - `FAILED`: the task failed. Check `ErrorMsg` first. In Doris 2.1.x, use the task time fields and materialized view name to search FE or BE logs for more details.
  - `CANCELED`: the task was canceled before it finished.
- `TaskContext.triggerMode`: why this task was created.
  - `MANUAL`: created by a manual refresh command, such as `REFRESH MATERIALIZED VIEW`.
  - `COMMIT`: created because data changes on related base tables triggered refresh.
  - `SYSTEM`: created by an internal system action, for example the initial build of a materialized view created with immediate build.
- `TaskContext.isComplete`: whether the refresh request asks for a complete refresh.
  - `true`: the request asks Doris to refresh all materialized view partitions.
  - `false`: the request does not force complete refresh. Doris can decide the actual refresh scope based on partition freshness.
- `RefreshMode`: actual refresh scope selected by the task after checking partitions.
  - `COMPLETE`: all materialized view partitions that belong to the MV were selected for refresh.
  - `PARTIAL`: only some materialized view partitions were selected for refresh.
  - `NOT_REFRESH`: no partition needed refresh. In this case, `NeedRefreshPartitions` is usually empty and `Progress` can be `\N`.

> **Version**
>
> Doris 2.1.x does not provide `LastQueryId` in `tasks("type"="mv")`. To troubleshoot refresh failures, use `ErrorMsg`, `CreateTime`, `StartTime`, `FinishTime`, `MvDatabaseName`, and `MvName` to locate related FE or BE logs.

Examples

View the latest refresh task of a materialized view.

```
select *
from tasks("type"="mv")
where MvDatabaseName = "test" and MvName = "mv1"
order by CreateTime desc
limit 1\G
```

```
*************************** 1. row ***************************
               TaskId: 437156301250803
                JobId: 19508
              JobName: inner_mtmv_19494
                MvId: 19494
              MvName: mv1
        MvDatabaseId: 16016
```

```
       MvDatabaseName: test
               Status: SUCCESS
             ErrorMsg:
           CreateTime: 2025-01-07 22:13:48
            StartTime: 2025-01-07 22:13:48
           FinishTime: 2025-01-07 22:17:45
           DurationMs: 236985
          TaskContext: {"triggerMode":"MANUAL","partitions":[],"isComplete":false}
          RefreshMode: COMPLETE
NeedRefreshPartitions: ["p_20210101_MAXVALUE","p_20200101_20210101"]
  CompletedPartitions: ["p_20210101_MAXVALUE","p_20200101_20210101"]
             Progress: 100.00% (2/2)
```

In this result:

- `TaskId` identifies this refresh execution. It is different for every refresh.
- `JobId` and `JobName` identify the MV refresh job. The job can be queried with `select * from jobs("type"="mv") where Name = "inner_mtmv_19494";`.
- `MvId`, `MvName`, `MvDatabaseId`, and `MvDatabaseName` identify the materialized view refreshed by the task.
- `Status` is `SUCCESS`, so the task finished successfully. If it is `FAILED`, check `ErrorMsg` first.
- `CreateTime`, `StartTime`, `FinishTime`, and `DurationMs` show when the task was created, when it started, when it finished, and how long it ran.
- `TaskContext` shows this task was manually triggered. `partitions: []` means the user did not specify a partition list in the refresh command.
- `RefreshMode` is `COMPLETE`, so this task refreshed all partitions that the materialized view had to refresh.
- `NeedRefreshPartitions` lists two partitions that needed refreshing, and `CompletedPartitions` lists the same two partitions, so all required partitions completed.
- `Progress` is `100.00% (2/2)`, which also means two of two required partitions completed.

> **Note**
>
> The number of stored and displayed task records is controlled by the FE configuration item `max_persistence_task_count`. The default value is 100. When the number of task records exceeds this limit, older task records are discarded. If the value is less than 1, task records are not persisted. Restart FE after changing this configuration.

View tasks for all insert tasks

```
select * from tasks("type"="insert");
```

```
+----------------+----------------+----------------+------------------------------+---------+----------+---------------------+-----
----------------+---------------------+-------------+---------------+------+
| TaskId         | JobId          | JobName        | Label                        | Status  | ErrorMsg | CreateTime          |
StartTime           | FinishTime          | TrackingUrl | LoadStatistic | User |
+----------------+----------------+----------------+------------------------------+---------+----------+---------------------+-----
----------------+---------------------+-------------+---------------+------+
| 79133848479750 | 78533940810334 | insert_tab_job | 78533940810334_79133848479750 | SUCCESS |          | 2025-01-17 14:42:54 |
2025-01-17 14:42:54 | 2025-01-17 14:42:54 |             |               | root |
+----------------+----------------+----------------+------------------------------+---------+----------+---------------------+-----
----------------+---------------------+-------------+---------------+------+
```

SELECT

Description

Mainly introduces the use of Select syntax

grammar:

```
SELECT
    [hint_statement, ...]
    [ALL | DISTINCT | DISTINCTROW | ALL EXCEPT ( col_name1 [, col_name2, col_name3, ...] )]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
      [TABLET tabletid_list]
      [TABLESAMPLE sample_value [ROWS | PERCENT]
        [REPEATABLE pos_seek]]
    [WHERE where_condition]
    [GROUP BY [GROUPING SETS | ROLLUP | CUBE] {col_name | expr | position}]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [INTO OUTFILE 'file_name']
```

1. **Syntax Description:**

    1. select_expr, ... Columns retrieved and displayed in the result, when using an alias, as is optional.
    2. select_expr, ... Retrieved target table (one or more tables (including temporary tables generated by subqueries)
    3. where_definition retrieves the condition (expression), if there is a WHERE clause, the condition filters the row data. where_condition is an expression that evaluates to true for each row to be selected. Without the WHERE clause, the statement selects all rows. In WHERE expressions, you can use any MySQL supported functions and operators except aggregate functions
    4. `ALL | DISTINCT` : to refresh the result set, all is all, distinct/distinctrow will refresh the duplicate columns, the default is all
    5. `ALL EXCEPT`: Filter on the full (all) result set, except specifies the name of one or more columns to be excluded from the full result set. All matching column names will be ignored in the output.

    This feature is supported since the Apache Doris 1.2 version

    6. `INTO OUTFILE 'file_name'` : save the result to a new file (which did not exist before), the difference lies in the save format.
    7. `Group by having`: Group the result set, and brush the result of group by when having appears. `Grouping Sets`, `Rollup`, `Cube` are extensions of group by, please refer to GROUPING SETS DESIGN for details.

8. `Order by`: Sort the final result, Order by sorts the result set by comparing the size of one or more columns.

   Order by is a time-consuming and resource-intensive operation, because all data needs to be sent to 1 node before it can be sorted, and the sorting operation requires more memory than the non-sorting operation.

   If you need to return the top N sorted results, you need to use the LIMIT clause; in order to limit memory usage, if the user does not specify the LIMIT clause, the first 65535 sorted results are returned by default.

9. `Limit n`: limit the number of lines in the output result, `limit m,n` means output n records starting from the mth line.You should use `order by` before you use `limit m,n`, otherwise the data may be inconsistent each time it is executed.

10. The `Having` clause does not filter the row data in the table, but filters the results produced by the aggregate function.

    Typically `having` is used with aggregate functions (eg :`COUNT(), SUM(), AVG(), MIN(), MAX()`) and `group by` clauses.

11. SELECT supports explicit partition selection using PARTITION containing a list of partitions or subpartitions (or both) following the name of the table in `table_reference`

12. `[TABLET tids] TABLESAMPLE n [ROWS | PERCENT] [REPEATABLE seek]`: Limit the number of rows read from the table in the FROM clause, select a number of Tablets pseudo-randomly from the table according to the specified number of rows or percentages, and specify the number of seeds in REPEATABLE to return the selected samples again. In addition, you can also manually specify the TableID, Note that this can only be used for OLAP tables.

13. `hint_statement`: hint in front of the selectlist indicates that hints can be used to influence the behavior of the optimizer in order to obtain the desired execution plan. Details refer to [joinHint using document] (https://doris.apache.org/docs/dev/query-acceleration/hints/hints-overview)

Syntax constraints:

1. SELECT can also be used to retrieve calculated rows without referencing any table.
2. All clauses must be ordered strictly according to the above format, and a HAVING clause must be placed after the GROUP BY clause and before the ORDER BY clause.
3. The alias keyword AS is optional. Aliases can be used for group by, order by and having
4. Where clause: The WHERE statement is executed to determine which rows should be included in the GROUP BY section, and HAVING is used to determine which rows in the result set should be used.
5. The HAVING clause can refer to the total function, but the WHERE clause cannot refer to, such as count, sum, max, min, avg, at the same time, the where clause can refer to other functions except the total function. Column aliases cannot be used in the Where clause to define conditions.
6. Group by followed by with rollup can count the results one or more times.

Join query:

Doris supports JOIN syntax

```
JOIN
table_references:
    table_reference [, table_reference] ...
table_reference:
    table_factor
  | join_table
table_factor:
    tbl_name [[AS] alias]
        [{USE|IGNORE|FORCE} INDEX (key_list)]
  | ( table_references )
  | { OJ table_reference LEFT OUTER JOIN table_reference
        ON conditional_expr }
join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference LEFT [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [LEFT [OUTER]] JOIN table_factor
  | table_reference RIGHT [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [RIGHT [OUTER]] JOIN table_factor
join_condition:
    ON conditional_expr
```

UNION Grammar:

```
SELECT ...
UNION [ALL| DISTINCT] SELECT ......
[UNION [ALL| DISTINCT] SELECT ...]
```

`UNION` is used to combine the results of multiple `SELECT` statements into a single result set.

The column names in the first `SELECT` statement are used as the column names in the returned results. The selected columns listed in the corresponding position of each `SELECT` statement should have the same data type. (For example, the first column selected by the first statement should be of the same type as the first column selected by other statements.)

The default behavior of `UNION` is to remove duplicate rows from the result. The optional `DISTINCT` keyword has no effect other than the default, since it also specifies duplicate row removal. With the optional `ALL` keyword, no duplicate row removal occurs, and the result includes all matching rows in all `SELECT` statements

WITH statement:

To specify common table expressions, use the `WITH` clause with one or more comma-separated clauses. Each subclause provides a subquery that generates the result set and associates the name with the subquery. The following example defines `WITH` clauses in CTEs named `cte1` and `cte2`, and refers to the `WITH` clause below their top-level `SELECT`:

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

In a statement containing the `WITH` clause, each CTE name can be referenced to access the corresponding CTE result set.

CTE names can be referenced in other CTEs, allowing CTEs to be defined based on other CTEs.

Recursive CTE is currently not supported.

Examples

1. Query the names of students whose ages are 18, 20, 25

```
select Name from student where age in (18,20,25);
```

2. ALL EXCEPT Example

```
-- Query all information except the students' age
select * except(age) from student;
```

3. GROUP BY Example

```
--Query the tb_book table, group by type, and find the average price of each type of book,
select type,avg(price) from tb_book group by type;
```

4. DISTINCT Use

```
--Query the tb_book table to remove duplicate type data
select distinct type from tb_book;
```

5. ORDER BY Example

   Sort query results in ascending (default) or descending (DESC) order. Ascending NULL is first, descending NULL is last

```
--Query all records in the tb_book table, sort them in descending order by id, and display three records
select * from tb_book order by id desc limit 3;
```

6. LIKE fuzzy query

   Can realize fuzzy query, it has two wildcards: % and _, % can match one or more characters, _ can match one character

```
--Find all books whose second character is h
select * from tb_book where name like('_h%');
```

7. LIMIT limits the number of result rows

```
--1. Display 3 records in descending order
select * from tb_book order by price desc limit 3;

--2. Display 4 records from id=1
select * from tb_book where id limit 1,4;
```

8. CONCAT join multiple columns

```
--Combine name and price into a new string output
select id,concat(name,":",price) as info,type from tb_book;
```

9. Using functions and expressions

```
--Calculate the total price of various books in the tb_book table
select sum(price) as total,type from tb_book group by type;
--20% off price
select *,(price * 0.8) as "20%" from tb_book;
```

10. UNION Example

```
SELECT a FROM t1 WHERE a = 10 AND B = 1 ORDER BY a LIMIT 10
UNION
SELECT a FROM t2 WHERE a = 11 AND B = 2 ORDER BY a LIMIT 10;
```

11. WITH clause example

```
WITH cte AS
(
  SELECT 1 AS col1, 2 AS col2
  UNION ALL
  SELECT 3, 4
)
SELECT col1, col2 FROM cte;
```

12. JOIN Example

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)
```

Equivalent to

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)
```

13. INNER JOIN

```
SELECT t1.name, t2.salary
  FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

SELECT t1.name, t2.salary
  FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
```

14. LEFT JOIN

```
SELECT left_tbl.*
  FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
  WHERE right_tbl.id IS NULL;
```

15. RIGHT JOIN

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    2 | y    |    2 | z    |
| NULL | NULL |    3 | w    |
+------+------+------+------+
```

16. TABLESAMPLE

```
--Pseudo-randomly sample 1000 rows in t1. Note that several Tablets are actually selected according to the statistics of the table, and the total number of selected Tablet rows may be greater than 1000, so if you want to explicitly return 1000 rows, you need to add Limit.
SELECT * FROM t1 TABLET(10001) TABLESAMPLE(1000 ROWS) REPEATABLE 2 limit 1000;
```

Keywords

```
SELECT
```

Best Practice

1. Some additional knowledge about the SELECT clause

    - An alias can be specified for select_expr using AS alias_name. Aliases are used as column names in expressions and can be used in GROUP BY, ORDER BY or HAVING clauses. The AS keyword is a good habit to use when specifying aliases for columns.
    - table_references after FROM indicates one or more tables participating in the query. If more than one table is listed, a JOIN operation is performed. And for each specified table, you can define an alias for it
    - The selected column after SELECT can be referenced in ORDER IN and GROUP BY by column name, column alias or integer (starting from 1) representing the column position

      ```
      SELECT college, region, seed FROM tournament
        ORDER BY region, seed;

      SELECT college, region AS r, seed AS s FROM tournament
        ORDER BY r, s;

      SELECT college, region, seed FROM tournament
        ORDER BY 2, 3;
      ```

    - If ORDER BY appears in a subquery and also applies to the outer query, the outermost ORDER BY takes precedence.
    - If GROUP BY is used, the grouped columns are automatically sorted in ascending order (as if there was an ORDER BY statement followed by the same columns). If you want to avoid the overhead of GROUP BY due to automatic sorting, adding ORDER BY NULL can solve it:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

- When sorting columns in a SELECT using ORDER BY or GROUP BY, the server sorts values using only the initial number of bytes indicated by the max_sort_length system variable.
- Having clauses are generally applied last, just before the result set is returned to the MySQL client, and is not optimized. (while LIMIT is applied after HAVING)

  The SQL standard requires: HAVING must refer to a column in the GROUP BY list or used by an aggregate function. However, MySQL extends this by allowing HAVING to refer to columns in the Select clause list, as well as columns from outer subqueries.

  A warning is generated if the column referenced by HAVING is ambiguous. In the following statement, col2 is ambiguous:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

- Remember not to use HAVING where WHERE should be used. HAVING is paired with GROUP BY.
- The HAVING clause can refer to aggregate functions, while WHERE cannot.

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

- The LIMIT clause can be used to constrain the number of rows returned by a SELECT statement. LIMIT can have one or two arguments, both of which must be non-negative integers.

```
/*Retrieve 6~15 rows in the result set*/
SELECT * FROM tbl LIMIT 5,10;
/*Then if you want to retrieve all rows after a certain offset is set, you can set a very large constant for the
second parameter. The following query fetches all data from row 96 onwards */
SELECT * FROM tbl LIMIT 95,18446744073709551615;
/*If LIMIT has only one parameter, the parameter specifies the number of rows that should be retrieved, and the
offset defaults to 0, that is, starting from the first row*/
```

- SELECT…INTO allows query results to be written to a file

2. Modifiers of the SELECT keyword

- deduplication

  The ALL and DISTINCT modifiers specify whether to deduplicate rows in the result set (should not be a column).

  ALL is the default modifier, that is, all rows that meet the requirements are to be retrieved.

  DISTINCT removes duplicate rows.

3. The main advantage of subqueries

- Subqueries allow structured queries so that each part of a statement can be isolated.

- Some operations require complex unions and associations. Subqueries provide other ways to perform these operations

4. Speed up queries

- Use Doris's partition and bucket as data filtering conditions as much as possible to reduce the scope of data scanning
- Make full use of Doris's prefix index fields as data filter conditions to speed up query speed

5. UNION

- Using only the union keyword has the same effect as using union disitnct. Since the deduplication work is more memory-intensive, the query speed using the union all operation will be faster and the memory consumption will be less. If users want to perform order by and limit operations on the returned result set, they need to put the union operation in the subquery, then select from subquery, and finally put the subquery and order by outside the subquery.

```
select * from (select age from student_01 union all select age from student_02) as t1
order by age limit 4;

+-------------+
| age |
+-------------+
| 18 |
| 19 |
| 20 |
| 21 |
+-------------+
4 rows in set (0.01 sec)
```

6. JOIN

- In the inner join condition, in addition to supporting equal-valued joins, it also supports unequal-valued joins. For performance reasons, it is recommended to use equal-valued joins.
- Other joins only support equivalent joins

EXPLAIN

Description

The EXPLAIN statement displays Doris's query execution plan for a given query. Doris's query optimizer aims to create an efficient plan using statistical data, data characteristics, and features like HASH JOIN, partitioning, and bucketing. However, due to theoretical and practical constraints, the plan may sometimes under perform.

To improve performance, it's essential to analyze the current plan. This article teaches how to use the EXPLAIN statement for optimization.

Syntax

```
plain text {EXPLAIN | DESC} [VERBOSE] <query_block>
```

Required Parameters

> This is the query statement for which you want the explain plan.

Optional Parameters

[VERBOSE]

> Whether to display detailed information is determined by the `VERBOSE` specification. With `VERBOSE`, comprehensive details are shown, including specifics on each operator, the tuple IDs they use, and detailed descriptions for each tuple. Without it, concise information is provided.

Return Results

Basic Concepts

To better understand the information displayed by `EXPLAIN`, let's introduce a few core concepts of the Doris execution plan.

Name	Explanation
PLAN	Execution plan. A query is translated into an execution plan by the execution planner, which is then executed by the execution engine.
FRAGMENT	Execution fragment. Since Doris is a distributed execution engine, a complete execution plan is divided into multiple single-node execution fragments. A FRAGMENT table represents a complete single-node execution fragment. Multiple FRAGMENTS combine to form a complete PLAN.

Name	Explanation
PLAN NODE	Operator. The smallest unit of the execution plan. A FRAGMENT consists of multiple operators. Each operator is responsible for a specific execution logic, such as aggregation, joins, etc.

Return Result Structure

The result of the Doris `EXPLAIN` statement is a complete PLAN. Within the PLAN, FRAGMENTS are ordered from back to front based on the execution sequence. Within each FRAGMENT, operators (PLAN NODES) are also ordered from back to front based on the execution sequence.

An example is provided below:

```
+------------------------------------------------+
| Explain String(Nereids Planner)                |
+------------------------------------------------+
| PLAN FRAGMENT 0                                |
|   OUTPUT EXPRS:                                |
|     cnt[#10]                                   |
|     cnt[#11]                                   |
|   PARTITION: UNPARTITIONED                     |
|                                                |
|   HAS_COLO_PLAN_NODE: false                    |
|                                                |
|   VRESULT SINK                                 |
|      MYSQL_PROTOCAL                            |
|                                                |
|   7:VEXCHANGE                                  |
|      offset: 0                                 |
|      distribute expr lists:                    |
|                                                |
| PLAN FRAGMENT 1                                |
|                                                |
|   PARTITION: RANDOM                            |
|                                                |
|   HAS_COLO_PLAN_NODE: false                    |
|                                                |
|   STREAM DATA SINK                             |
|     EXCHANGE ID: 07                            |
|     UNPARTITIONED                              |
|                                                |
|   6:VHASH JOIN(354)                            |
|   |  join op: INNER JOIN(BROADCAST)[]          |
|   |  equal join conjunct: cnt[#7] = cnt[#5]    |
|   |  cardinality=1                             |
|   |  vec output tuple id: 8                    |
|   |  vIntermediate tuple ids: 7                |
|   |  hash output slot ids: 5 7                 |
|   |  distribute expr lists:                    |
|   |  distribute expr lists:                    |
|   |                                            |
|   |----4:VEXCHANGE                             |
|   |       offset: 0                            |
|   |       distribute expr lists:               |
|   |                                            |
|   5:VEXCHANGE                                  |
|      offset: 0                                 |
|      distribute expr lists:                    |
|                                                |
| PLAN FRAGMENT 2                                |
|   ...                                          |
```

```
|                                               |
| PLAN FRAGMENT 3                               |
|   ...                                         |
+-----------------------------------------------+
```

Operators are linked to their child nodes with dashed lines. When an operator has multiple children, they are arranged vertically, representing a right-to-left order. In the example above, operator 6 (VHASH JOIN) has operator 5 (EXCHANGE) as its left child and operator 4 (EXCHANGE) as its right child.

Fragment Field Descriptions

Name	Description
PARTITION	Displays the data distribution of the current Fragment
HAS_COLO_PLAN_NODE	Indicates if the fragment contains colocate operators
Sink	The method of fragment data output, see the table below for details

Sink Methods

Name	Description
STREAM DATA SINK	Outputs data to the next Fragment. It includes two lines of information. First line: The downstream EXCHANGE NODE to which data is sent. Second line: The method of data distribution. - UNPARTITIONED means each downstream instance receives the full data set. This typically occurs in broadcast joins or when single-instance logic is required, such as global limit or order by. - RANDOM means each downstream instance receives a random subset of data without repetition. - HASH_PARTITIONED uses the listed slots as keys to hash and send data shards to the same downstream instance. This is often used upstream of partition hash joins or the second stage of two-phase aggregations.
RESULT SINK	Sends result data to the FE. The first line indicates the protocol used for data transmission, currently supporting MySQL and arrow protocols.
OLAP TABLE SINK	Writes data to an OLAP table.
MultiCastDataSinks	A multicast operator that contains multiple STREAM DATA SINKs. Each STREAM DATA SINK sends the full data set to its downstream.

Tuple Information Description

When using VERBOSE mode, Tuple information is output. Tuple information describes the SLOT details within a row of data, including SLOT type, nullable status, etc.

The output contains multiple TupleDescriptors, each containing multiple SlotDescriptors. An example is shown below:

```
Tuples:
TupleDescriptor{id=0, tbl=t1}
  SlotDescriptor{id=0, col=c1, colUniqueId=0, type=int, nullable=true, isAutoIncrement=false, subColPath=null}
  SlotDescriptor{id=2, col=c3, colUniqueId=2, type=int, nullable=true, isAutoIncrement=false, subColPath=null}
```

TupleDescriptor

Name	Description
id	The id of the tuple descriptor
tbl	The corresponding table of the tuple, or `null` if not applicable

SlotDescriptor

Name	Description
id	The id of the slot descriptor
col	The corresponding column of the slot, or left blank if not applicable
colUniqueId	The unique id of the corresponding column, or -1 if not applicable
type	The type of the slot
nullable	Indicates if the corresponding data can be null
isAutoIncrement	Indicates if the column is auto-incremented
subColPath	The sub-column path within the column, currently only applies to variant types

Operator Descriptions

Operator List

Name	Description
AGGREGATE	Aggregation operator
ANALYTIC	Window function operator
ASSERT NUMBER OF ROWS	Operator to check the number of downstream output rows
EXCHANGE	Data exchange receiver operator
MERGING-EXCHANGE	Data exchange receiver with sorting and row limit functionality
HASH JOIN	Hash join operator
NESTED LOOP JOIN	Nested loop join operator
PartitionTopN	Intra-partition data pre-filtering operator
REPEAT_NODE	Data replication operator
DataGenScanNode	Table-valued function operator
EsScanNode	ES table scan operator
HIVE_SCAN_NODE	Hive table scan operator
HUDI_SCAN_NODE	Hudi table scan operator
ICEBERG_SCAN_NODE	Iceberg table scan operator

Name	Description
PAIMON_SCAN_NODE	Paimon table scan operator
JdbcScanNode	Jdbc table scan operator
OlapScanNode	Olap table scan operator
SELECT	Filtering operator
UNION	Set union operator
EXCEPT	Set difference operator
INTERSECT	Set intersection operator
SORT	Sorting operator
TOP-N	Sort and return top N results operator
TABLE FUNCTION NODE	Table function operator (lateral view)

Common Fields

Name	Description
limit	Limits the number of output rows
offset	Number of rows to skip before outputting
conjuncts	Filters the results of the current node. Executed before projections.
projections	Projection operations after the current operator. Executed after conjuncts.
project output tuple id	The output tuple after projection. The slot arrangement within the data tuple can be seen via tuple desc.
cardinality	Estimated row count by the optimizer
distribute expr lists	The original data distribution method for the child nodes of the current node
Expression's slot id	The specific slot corresponding to the slot id can be found in the tuple list in verbose mode. This list provides information such as slot type and nullable attributes. Represented as `[#5]` after the expression.

AGGREGATE

Name	Description
(Aggregation Phase)	The aggregation phase is represented by two terms. The first term can be either update (local aggregation) or merge (global aggregation). The second term indicates whether the current data is serialized (serialize) or has completed final calculations (finalize).
STREAMING	Only local aggregation operators in multi-stage aggregation truncation have this flag. Indicates that the current aggregation node may use STREAMING mode, where input data is passed directly to the next stage of aggregation without actual computation.
output	The output of the current aggregation operator. All local pre-aggregation functions are prefixed with partial.

Name	Description
group by	The key for aggregation

ANALYTIC

Name	Description
functions	The name of the current window function
partition by	Corresponds to the partition by clause in the over clause of the window function. Windowing expression.
order by	Sorting expression and order within the window
window	Window range

ASSERT NUMBER OF ROWS

Name	Description
EQ	The downstream output must match this row count constraint

HASH JOIN

Name	Description
join op	Type of join
equal join conjunct	Equality condition in the join condition
other join predicates	Conditions in the join condition, excluding equality
mark join predicates	Conditions used in mark join
other predicates	Filtering predicates after join execution
runtime filters	Generated runtime filters
output slot ids	List of final output slots
hash output slot ids	List of output slots after hash join execution, but before other join conditions are applied
isMarkJoin	Indicates whether it is a mark join

NESTED LOOP JOIN

Name	Description
join op	Type of join operation
join conjuncts	Conditions for joining
mark join predicates	Conditions used in mark join
predicates	Filter predicates after join
runtime filters	Generated runtime filters

Name	Description
output slot ids	List of final output slots
isMarkJoin	Whether it is a mark join

PartitionTopN

Name	Description
functions	Window functions applying grouped filter optimization
has global limit	Presence of a global limit on the number of rows
partition limit	Limit on the number of rows within each partition
partition topn phase	Current phase: TWO_PHASE_GLOBAL_PTOPN for global phase after shuffling by partition key, TWO_PHASE_LOCAL_PTOPN for local phase before shuffling by partition key

REPEAT_NODE

Name	Description
repeat	Number of repetitions for each row and corresponding slot ids for aggregation columns
exprs	List of expressions for output data after repetition

DataGenScanNode

Name	Description
table value function	Table function name

EsScanNode

Name	Description
SORT COLUMN	Columns for sorting results
LOCAL_PREDICATES	Filters executed within Doris
REMOTE_PREDICATES	Filters executed within ES
ES index/type	ES index and type for querying

HIVE_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned
scanRanges	Information on scan splits

Name	Description
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan

HUDI_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned
scanRanges	Information on scan splits
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan
hudiNativeReadSplits	Number of splits read using native method

ICEBERG_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned
scanRanges	Information on scan splits
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan
icebergPredicatePushdown	Filters pushed down to iceberg API

PAIMON_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned
scanRanges	Information on scan splits
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan
paimonNativeReadSplits	Number of splits read using native method

NESTED LOOP JOIN

Name	Description
join op	Type of join operation
join conjuncts	Conditions for joining
mark join predicates	Conditions used in mark join
predicates	Filter predicates after join
runtime filters	Generated runtime filters
output slot ids	List of final output slots
isMarkJoin	Whether it is a mark join

PartitionTopN

Name	Description
functions	Window functions applying grouped filter optimization
has global limit	Presence of a global limit on the number of rows
partition limit	Limit on the number of rows within each partition
partition topn phase	Current phase: TWO_PHASE_GLOBAL_PTOPN for global phase after shuffling by partition key, TWO_PHASE_LOCAL_PTOPN for local phase before shuffling by partition key

REPEAT_NODE

Name	Description
repeat	Number of repetitions for each row and corresponding slot ids for aggregation columns
exprs	List of expressions for output data after repetition

DataGenScanNode

Name	Description
table value function	Table function name

EsScanNode

Name	Description
SORT COLUMN	Columns for sorting results
LOCAL_PREDICATES	Filters executed within Doris
REMOTE_PREDICATES	Filters executed within ES
ES index/type	ES index and type for querying

HIVE_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned
scanRanges	Information on scan splits
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan

HUDI_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned

Name	Description
scanRanges	Information on scan splits
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan
hudiNativeReadSplits	Number of splits read using native method

ICEBERG_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned
scanRanges	Information on scan splits
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan
icebergPredicatePushdown	Filters pushed down to iceberg API

PAIMON_SCAN_NODE

Name	Description
inputSplitNum	Number of scan splits
totalFileSize	Total file size being scanned
scanRanges	Information on scan splits
partition	Number of partitions being scanned
backends	Specific data info for each BE to scan
cardinality	Estimated number of rows by optimizer
avgRowSize	Estimated average row size by optimizer
numNodes	Number of BEs used by the current operator
pushdown agg	Aggregations pushed down to scan

Name	Description
paimonNativeReadSplits	Number of splits read using native method

JdbcScanNode

Name	Description
TABLE	JDBC-side table name to scan
QUERY	Query used for scanning

OlapScanNode

Name	Description
TABLE	Table being scanned. Parentheses indicate the name of the hit synchronized materialized view.
SORT INFO	Present when SCAN pre-sorting is planned. Indicates partial pre-sorting and pre-truncation of SCAN output.
SORT LIMIT	Present when SCAN pre-sorting is planned. Indicates the truncation length for pre-truncation.
TOPN OPT	Present when TOP-N Runtime Filter is planned.
PREAGGREGATION	Indicates whether pre-aggregation is enabled. Relevant for MOR aggregation and primary key models. ON means data at the storage layer satisfies upper-layer needs without extra aggregation. OFF means extra aggregation is performed.
partitions	Number of partitions currently scanned, total partitions, and list of scanned partition names.
tablets	Number of tablets scanned and total tablets in the table.
tabletList	List of tablets scanned.
avgRowSize	Estimated row size by the optimizer.
numNodes	Number of BEs assigned to the current scan.
pushAggOp	Results are returned by reading zonemap metadata. Supports MIN, MAX, COUNT aggregation information.

UNION

Name	Description
constant exprs	List of constant expressions to be included in the output.
child exprs	Children’s outputs projected through this expression list as input to the set operator.

EXCEPT

Name	Description
child exprs	Children’s outputs projected through this expression list as input to the set operator.

INTERSECT

Name	Description
child exprs	Children's outputs projected through this expression list as input to the set operator.

SORT

Name	Description
order by	Sorting key and specific sort order.

TABLE FUNCTION NODE

Name	Description
table function	Name of the table function used.
lateral view tuple id	Tuple ID corresponding to newly generated columns.
output slot id	List of slot IDs for columns output after column pruning.

TOP-N

Name	Description
order by	Sorting key and specific sort order.
TOPN OPT	Present when TOP-N runtime filter optimization is hit.
OPT TWO PHASE	Present when TOP-N deferred materialization is hit.

INSERT

Description

The change statement is to complete the data insertion operation.

```
INSERT INTO table_name
    [ PARTITION (p1, ...) ]
    [ WITH LABEL label]
    [ (column [, ...]) ]
    [ [ hint [, ...] ] ]
    { VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

Parameters

> tablet_name: The destination table for importing data. Can be of the form `db_name.table_name`
>
> partitions: Specify the partitions to be imported, which must be partitions that exist in `table_name`. Multiple partition names are separated by commas
>
> label: specify a label for the Insert task
>
> column_name: The specified destination column, must be a column that exists in `table_name`
>
> expression: the corresponding expression that needs to be assigned to a column
>
> DEFAULT: let the corresponding column use the default value
>
> query: a common query, the result of the query will be written to the target
>
> hint: some indicator used to indicate the execution behavior of `INSERT`. You can choose one of this values: `/*+ STREAMING */`, `/*+ SHUFFLE */` or `/+ *NOSHUFFLE* /`. 1. STREAMING: At present, it has no practical effect and is only reserved for compatibility with previous versions. (In the previous version, adding this hint would return a label, but now it defaults to returning a label) 2. SHUFFLE: When the target table is a partition table, enabling this hint will do repartiiton. 3. NOSHUFFLE: Even if the target table is a partition table, repartiiton will not be performed, but some other operations will be performed to ensure that the data is correctly dropped into each partition.

For a Unique table with merge-on-write enabled, you can also perform partial columns updates using the insert statement. To perform partial column updates with the insert statement, you need to set the session variable enable_unique_key_partial_update to true (the default value for this variable is false, meaning partial columns updates with the insert statement are not allowed by default). When performing partial columns updates, the columns being inserted must contain at least all the Key columns and specify the columns you want to update. If the Key column values for the inserted row already exist in the original table, the data in the row with the same key column values will be updated. If the Key column values for the inserted row do not exist in the original table, a new row will be inserted into the table. In this case, columns not specified in the insert statement must either have default values or be nullable. These missing columns will first attempt to be

populated with default values, and if a column has no default value, it will be filled with null. If a column cannot be null, the insert operation will fail.

Please note that the default value of the session variable `enable_insert_strict`, which controls whether the insert statement operates in strict mode, is true. In other words, the insert statement is in strict mode by default, and in this mode, updating non-existing keys in partial column updates is not allowed. Therefore, when using the insert statement for partial columns update and wishing to insert non-existing keys, you need to set `enable_unique_key_partial_update` to true and simultaneously set `enable_insert_strict` to false.

Notice:

When executing the `INSERT` statement, the default behavior is to filter the data that does not conform to the target table format, such as the string is too long. However, for business scenarios that require data not to be filtered, you can set the session variable `enable_insert_strict` to `true` to ensure that `INSERT` will not be executed successfully when data is filtered out.

Examples

The `test` table contains two columns `c1`, `c2`.

1. Import a row of data into the `test` table

```
INSERT INTO test VALUES (1, 2);
INSERT INTO test (c1, c2) VALUES (1, 2);
INSERT INTO test (c1, c2) VALUES (1, DEFAULT);
INSERT INTO test (c1) VALUES (1);
```

The first and second statements have the same effect. When no target column is specified, the column order in the table is used as the default target column. The third and fourth statements express the same meaning, use the default value of the `c2` column to complete the data import.

2. Import multiple rows of data into the `test` table at one time

```
INSERT INTO test VALUES (1, 2), (3, 2 + 2);
INSERT INTO test (c1, c2) VALUES (1, 2), (3, 2 * 2);
INSERT INTO test (c1) VALUES (1), (3);
INSERT INTO test (c1, c2) VALUES (1, DEFAULT), (3, DEFAULT);
```

The first and second statements have the same effect, import two pieces of data into the `test` table at one time The effect of the third and fourth statements is known, and the default value of the `c2` column is used to import two pieces of data into the `test` table

3. Import a query result into the `test` table

```
INSERT INTO test SELECT * FROM test2;
INSERT INTO test (c1, c2) SELECT * from test2;
```

4. Import a query result into the `test` table, specifying the partition and label

```
INSERT INTO test PARTITION(p1, p2) WITH LABEL `label1` SELECT * FROM test2;
```

```
INSERT INTO test WITH LABEL `label1` (c1, c2) SELECT * from test2;
```

Keywords

```
INSERT
```

Best Practice

1. View the returned results

   The INSERT operation is a synchronous operation, and the return of the result indicates the end of the operation. Users need to perform corresponding processing according to the different returned results.

   1. The execution is successful, the result set is empty

      If the result set of the insert corresponding to the select statement is empty, it will return as follows:

      ```
      mysql> insert into tbl1 select * from empty_tbl;
      Query OK, 0 rows affected (0.02 sec)
      ```

      `Query OK` indicates successful execution. `0 rows affected` means that no data was imported.

   2. The execution is successful, the result set is not empty

      In the case where the result set is not empty. The returned results are divided into the following situations:

      1. Insert executes successfully and is visible:

         ```
         mysql> insert into tbl1 select * from tbl2;
         Query OK, 4 rows affected (0.38 sec)
         {'label':'insert_8510c568-9eda-4173-9e36-6adc7d35291c', 'status':'visible', 'txnId':'4005'}

         mysql> insert into tbl1 with label my_label1 select * from tbl2;
         Query OK, 4 rows affected (0.38 sec)
         {'label':'my_label1', 'status':'visible', 'txnId':'4005'}

         mysql> insert into tbl1 select * from tbl2;
         Query OK, 2 rows affected, 2 warnings (0.31 sec)
         {'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'visible', 'txnId':'4005'}

         mysql> insert into tbl1 select * from tbl2;
         Query OK, 2 rows affected, 2 warnings (0.31 sec)
         {'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
         ```

         `Query OK` indicates successful execution. `4 rows affected` means that a total of 4 rows of data were imported. `2 warnings` indicates the number of lines to be filtered.

         Also returns a json string:

         ```
         {'label':'my_label1', 'status':'visible', 'txnId':'4005'}
         {'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
         ```

```
{'label':'my_label1', 'status':'visible', 'txnId':'4005', 'err':'some other error'}
```

`label` is a user-specified label or an automatically generated label. Label is the ID of this Insert Into import job. Each import job has a unique Label within a single database.

`status` indicates whether the imported data is visible. Show `visible` if visible, `committed` if not visible.

`txnId` is the id of the import transaction corresponding to this insert.

The `err` field shows some other unexpected errors.

When you need to view the filtered rows, the user can pass the following statement

```
show load where label="xxx";
```

The URL in the returned result can be used to query the wrong data. For details, see the summary of **Viewing Error Lines** later.

Invisibility of data is a temporary state, this batch of data will eventually be visible

You can view the visible status of this batch of data with the following statement:

```
show transaction where id=4005;
```

If the `TransactionStatus` column in the returned result is `visible`, the representation data is visible.

3. Execution failed

   Execution failure indicates that no data was successfully imported, and the following is returned:

   ```
   mysql> insert into tbl1 select * from tbl2 where k1 = "a";
   ERROR 1064 (HY000): all partitions have no load data. url: http://10.74.167.16:8042/api/_load_error_log?
   file=__shard_2/error_log_insert_stmt_ba8bb9e158e4879-ae8de8507c0bf8a2_ba8bb9e158e4879_ae8de8507c0
   ```

   Where `ERROR 1064 (HY000): all partitions have no load data` shows the reason for the failure. The following url can be used to query the wrong data:

   ```
   show load warnings on "url";
   ```

   You can view the specific error line.

2. Timeout time

   The timeout for an INSERT operation is governed by max(insert_timeout, query_timeout). Both are environment variables, with insert_timeout defaulting to 4 hours and query_timeout defaulting to 5 minutes. If the operation exceeds the timeout, the job will be canceled. The introduction of insert_timeout is to ensure that INSERT statements have a longer default timeout, allowing import tasks to be unaffected by the shorter default timeout typically applied to regular queries.

3. Label and atomicity

   The INSERT operation also guarantees the atomicity of imports, see the Import Transactions and Atomicity documentation.

   When using `CTE(Common Table Expressions)` as the query part in an insert operation, the `WITH LABEL` and `column` parts must be specified.

4. Filter Threshold

   Unlike other import methods, INSERT operations cannot specify a filter threshold (`max_filter_ratio`). The default filter threshold is 1, which means that rows with errors can be ignored.

   For business scenarios that require data not to be filtered, you can set session variable `enable_insert_strict` to `true` to ensure that when there is data When filtered out, `INSERT` will not be executed successfully.

5. Performance issues

   There is no single row insertion using the `VALUES` method. If you must use it this way, combine multiple rows of data into one INSERT statement for bulk commit.

INSERT OVERWRITE

Description

The function of this statement is to overwrite a table or some partitions of a table

```
INSERT OVERWRITE table table_name
    [ PARTITION (p1, ... | *) ]
    [ WITH LABEL label]
    [ (column [, ...]) ]
    [ [ hint [, ...] ] ]
    { VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

Parameters

table_name: the destination table to overwrite. This table must exist. It can be of the form `db_name.table_name`

partitions: the table partitions that needs to be overwritten. The following two formats are supported

1. partition names. must be one of the existing partitions in `table_name` separated by a comma
2. asterisk(*)。Enable auto-detect-partition. The write operation will automatically detect the partitions involved in the data and overwrite those partitions. This format is supported since Apache Doris 2.1.3 version.

label: specify a label for the Insert task

column_name: the specified destination column must be one of the existing columns in `table_name`

expression: the corresponding expression that needs to be assigned to a column

DEFAULT: let the column use the default value

query: a common query, the result of the query will overwrite the target.

hint: some indicator used to indicate the execution behavior of `INSERT`. You can choose one of this values: `/*+ STREAMING */`, `/*+ SHUFFLE */` or `/*+ NOSHUFFLE */`.

1. STREAMING: At present, it has no practical effect and is only reserved for compatibility with previous versions. (In the previous version, adding this hint would return a label, but now it defaults to returning a label)
2. SHUFFLE: When the target table is a partition table, enabling this hint will do repartiiton.
3. NOSHUFFLE: Even if the target table is a partition table, repartiiton will not be performed, but some other operations will be performed to ensure that the data is correctly dropped into each partition.

Notice:

1. In the current version, the session variable `enable_insert_strict` is set to `true` by default. If some data that does not conform to the format of the target table is filtered out during the execution of the `INSERT OVERWRITE` statement, such as when overwriting a partition and not all partition conditions are satisfied, overwriting the target table will fail.
2. The `INSERT OVERWRITE` statement first creates a new table, inserts the data to be overwritten into the new table, and then atomically replaces the old table with the new table and modifies its name. Therefore, during the process of overwriting the table, the data in the old table can still be accessed normally until the overwriting is completed.

For Auto Partition Table

If the target table of the INSERT OVERWRITE is an autopartitioned table, the behaviour is controlled by the Session Variable `enable_auto_create_when_overwrite` controls the behaviour as follows:

1. If PARTITION is not specified (overwrite the whole table), when `enable_auto_create_when_overwrite` is `true`, the table is overwritten and partitions are created according to the table's auto-partitioning rules for data that does not have a corresponding partition, and those datas is admit. If `enable_auto_create_when_overwrite` is `false`, data for which no partition is found will accumulate error rows until it fails.
2. If an overwrite PARTITION is specified, the AUTO PARTITION table behaves as a normal partitioned table during this process, and data that does not satisfy the conditions of an existing partition is filtered instead of creating a new partition.
3. If you specify PARTITION as `partition(*)` (auto detect partition and overwrite), when `enable_auto_create_when_overwrite` is `true`, for the data that have corresponding partitions in the table, overwrite their corresponding partitions, and leave the other existing partitions unchanged. At the same time, for data without corresponding partitions, create partitions according to the table's auto-partitioning rules, and accommodate the data without corresponding partitions. If `enable_auto_create_when_overwrite` is `false`, data for which no partition is found will accumulate error rows until it fails.

`enable_auto_create_when_overwrite` was introduced since 2.1.8. In versions without `enable_auto_create_when_overwrite`, the behaviour is as if the variable had a value of `false`.

The quick check conclusion is as follows:

1. For auto-partition tables that have `enable_auto_create_when_overwrite` enabled:

	Partitions to Overwrite	Clear Other Partitions	Auto Create for Unpartitioned Data
Unlabeled (Full Table)	All	√	√
Designated Partition	Explicit partition	×	×
`partition(*)`	The partition that the data belongs to	×	√

2. For trivial tables, auto-partition tables disabled `enable_auto_create_when_overwrite`:

	Partitions to Overwrite	Clear Other Partitions	Auto Create for Unpartitioned Data
Unlabeled (Full Table)	All	√	×
Designated Partition	Explicit partition	×	×
`partition(*)`	The partition that the data belongs to	×	×

Examples are shown below:

```
mysql> create table auto_list(
    ->              k0 varchar null
    ->          )
    ->          auto partition by list (k0)
    ->          (
    ->              PARTITION p1 values in (("Beijing"), ("BEIJING")),
    ->              PARTITION p2 values in (("Shanghai"), ("SHANGHAI")),
    ->              PARTITION p3 values in (("xxx"), ("XXX")),
    ->              PARTITION p4 values in (("list"), ("LIST")),
    ->              PARTITION p5 values in (("1234567"), ("7654321"))
    ->          )
    ->          DISTRIBUTED BY HASH(`k0`) BUCKETS 1
    ->          properties("replication_num" = "1");
Query OK, 0 rows affected (0.14 sec)

mysql> insert into auto_list values ("Beijing"),("Shanghai"),("xxx"),("list"),("1234567");
Query OK, 5 rows affected (0.22 sec)

mysql> insert overwrite table auto_list partition(*) values ("BEIJING"), ("new1");
Query OK, 2 rows affected (0.28 sec)

mysql> select * from auto_list;
+----------+ --- p1 is overwritten, new1 gets the new partition, and the other partitions remain unchanged.
| k0       |
+----------+
| 1234567  |
| BEIJING  |
| list     |
| xxx      |
| new1     |
| Shanghai |
+----------+
6 rows in set (0.48 sec)

mysql> insert overwrite table auto_list values ("SHANGHAI"), ("new2");
Query OK, 2 rows affected (0.17 sec)

mysql> select * from auto_list;
+----------+ --- The whole table is overwritten, and new2 gets the new partition.
| k0       |
+----------+
| new2     |
| SHANGHAI |
+----------+
2 rows in set (0.15 sec)
```

Examples

Assuming there is a table named `test`. The table contains two columns `c1` and `c2`, and two partitions `p1` and `p2`

```
CREATE TABLE IF NOT EXISTS test (
  `c1` int NOT NULL DEFAULT "1",
  `c2` int NOT NULL DEFAULT "4"
) ENGINE=OLAP
UNIQUE KEY(`c1`)
PARTITION BY LIST (`c1`)
(
PARTITION p1 VALUES IN ("1","2","3"),
PARTITION p2 VALUES IN ("4","5","6")
)
DISTRIBUTED BY HASH(`c1`) BUCKETS 3
PROPERTIES (
  "replication_allocation" = "tag.location.default: 1",
  "in_memory" = "false",
  "storage_format" = "V2"
);
```

Overwrite Table

1. Overwrite the `test` table using the form of `VALUES`.

   ```
   // Single-row overwrite.
   INSERT OVERWRITE table test VALUES (1, 2);
   INSERT OVERWRITE table test (c1, c2) VALUES (1, 2);
   INSERT OVERWRITE table test (c1, c2) VALUES (1, DEFAULT);
   INSERT OVERWRITE table test (c1) VALUES (1);
   // Multi-row overwrite.
   INSERT OVERWRITE table test VALUES (1, 2), (3, 2 + 2);
   INSERT OVERWRITE table test (c1, c2) VALUES (1, 2), (3, 2 * 2);
   INSERT OVERWRITE table test (c1, c2) VALUES (1, DEFAULT), (3, DEFAULT);
   INSERT OVERWRITE table test (c1) VALUES (1), (3);
   ```

   - The first and second statements have the same effect. If the target column is not specified during overwriting, the column order in the table will be used as the default target column. After the overwrite is successful, there is only one row of data in the `test` table.
   - The third and fourth statements have the same effect. The unspecified column `c2` will be overwritten with the default value 4. After the overwrite is successful, there is only one row of data in the `test` table.
   - The fifth and sixth statements have the same effect. Expressions (such as `2+2`, `2*2`) can be used in the statement. The result of the expression will be computed during the execution of the statement and then overwritten into the `test` table. After the overwrite is successful, there are two rows of data in the `test` table.
   - The seventh and eighth statements have the same effect. The unspecified column `c2` will be overwritten with the default value 4. After the overwrite is successful, there are two rows of data in the `test` table.

2. Overwrite the `test` table in the form of a query statement. The data format of the `test2` table and the `test` table must be consistent. If they are not consistent, implicit data type conversion will be triggered.

   ```
   INSERT OVERWRITE table test SELECT * FROM test2;
   INSERT OVERWRITE table test (c1, c2) SELECT * from test2;
   ```

   - The first and second statements have the same effect. The purpose of these statements is to take data from the `test2` table and overwrite the `test` table with the taken data. After the overwrite is successful, the data in the `test` table will be consistent with the data in the `test2` table.

3. Overwrite the `test` table and specify a label.

```
INSERT OVERWRITE table test WITH LABEL `label1` SELECT * FROM test2;
INSERT OVERWRITE table test WITH LABEL `label2` (c1, c2) SELECT * from test2;
```

- Users can use the `SHOW LOAD;` command to check the status of the job imported by this `label`. It should be noted that the label is unique.

Overwrite Table Partition

When using INSERT OVERWRITE to rewrite partitions, we actually encapsulate the following three steps into a single transaction and execute it. If it fails halfway through, the operations that have been performed will be rolled back: 1. Assuming that partition `p1` is specified to be rewritten, first create an empty temporary partition `pTMP` with the same structure as the target partition to be rewritten. 2. Write data to `pTMP`. 3. replace `p1` with the `pTMP` atom

The following is examples:

1. Overwrite partitions `P1` and `P2` of the `test` table using the form of `VALUES`.

```
// Single-row overwrite.
INSERT OVERWRITE table test PARTITION(p1,p2) VALUES (1, 2);
INSERT OVERWRITE table test PARTITION(p1,p2) (c1, c2) VALUES (1, 2);
INSERT OVERWRITE table test PARTITION(p1,p2) (c1, c2) VALUES (1, DEFAULT);
INSERT OVERWRITE table test PARTITION(p1,p2) (c1) VALUES (1);
// Multi-row overwrite.
INSERT OVERWRITE table test PARTITION(p1,p2) VALUES (1, 2), (4, 2 + 2);
INSERT OVERWRITE table test PARTITION(p1,p2) (c1, c2) VALUES (1, 2), (4, 2 * 2);
INSERT OVERWRITE table test PARTITION(p1,p2) (c1, c2) VALUES (1, DEFAULT), (4, DEFAULT);
INSERT OVERWRITE table test PARTITION(p1,p2) (c1) VALUES (1), (4);
```

   Unlike overwriting an entire table, the above statements are overwriting partitions in the table. Partitions can be overwritten one at a time or multiple partitions can be overwritten at once. It should be noted that only data that satisfies the corresponding partition filtering condition can be overwritten successfully. If there is data in the overwritten data that does not satisfy any of the partitions, the overwrite will fail. An example of a failure is shown below.

```
INSERT OVERWRITE table test PARTITION(p1,p2) VALUES (7, 2);
```

   The data overwritten by the above statements (`c1=7`) does not satisfy the conditions of partitions `P1` and `P2`, so the overwrite will fail.

2. Overwrite partitions `P1` and `P2` of the `test` table in the form of a query statement. The data format of the `test2` table and the `test` table must be consistent. If they are not consistent, implicit data type conversion will be triggered.

```
INSERT OVERWRITE table test PARTITION(p1,p2) SELECT * FROM test2;
INSERT OVERWRITE table test PARTITION(p1,p2) (c1, c2) SELECT * from test2;
```

3. Overwrite partitions `P1` and `P2` of the `test` table and specify a label.

```
INSERT OVERWRITE table test PARTITION(p1,p2) WITH LABEL `label3` SELECT * FROM test2;
INSERT OVERWRITE table test PARTITION(p1,p2) WITH LABEL `label4` (c1, c2) SELECT * from test2;
```

Overwrite Auto Detect Partition

> This feature is available since version 2.1.3.

When the PARTITION clause specified by the INSERT OVERWRITE command is `PARTITION(*)`, this overwrite will automatically detect the partition where the data is located. Example:

```
mysql> create table test(
    -> k0 int null
    -> )
    -> partition by range (k0)
    -> (
    -> PARTITION p10 values less than (10),
    -> PARTITION p100 values less than (100),
    -> PARTITION pMAX values less than (maxvalue)
    -> )
    -> DISTRIBUTED BY HASH(`k0`) BUCKETS 1
    -> properties("replication_num" = "1");
Query OK, 0 rows affected (0.11 sec)

mysql> insert into test values (1), (2), (15), (100), (200);
Query OK, 5 rows affected (0.29 sec)

mysql> select * from test order by k0;
+------+
| k0   |
+------+
|    1 |
|    2 |
|   15 |
|  100 |
|  200 |
+------+
5 rows in set (0.23 sec)

mysql> insert overwrite table test partition(*) values (3), (1234);
Query OK, 2 rows affected (0.24 sec)

mysql> select * from test order by k0;
+------+
| k0   |
+------+
|    3 |
|   15 |
| 1234 |
+------+
3 rows in set (0.20 sec)
```

As you can see, all data in partitions `p10` and `pMAX`, where data 3 and 1234 are located, are overwritten, while partition `p100` remains unchanged. This operation can be interpreted as syntactic sugar for specifying a specific partition to be overwritten by the PARTITION clause during an INSERT OVERWRITE operation, which is

implemented in the same way as specify a partition to overwrite. The `PARTITION(*)` syntax eliminates the need to manually fill in all the partition names when overwriting a large number of partitions.

Keywords

INSERT OVERWRITE, OVERWRITE, AUTO DETECT

SHOW LAST INSERT

Description

This syntax is used to view the result of the latest insert operation in the current session connection

Syntax

```
SHOW LAST INSERT
```

Example of returned result:

```
     TransactionId: 64067
             Label: insert_ba8f33aea9544866-8ed77e2844d0cc9b
          Database: default_cluster:db1
             Table: t1
TransactionStatus: VISIBLE
        LoadedRows: 2
      FilteredRows: 0
```

Return Value

- TransactionId: transaction id
- Label: the label corresponding to the insert task
- Database: the database corresponding to insert
- Table: the table corresponding to insert
- TransactionStatus: transaction status
    - PREPARE: preparation stage
    - PRECOMMITTED: Pre-commit stage
    - COMMITTED: The transaction succeeded, but the data was not visible
    - VISIBLE: The transaction succeeded and the data is visible
    - ABORTED: Transaction failed
- LoadedRows: Number of imported rows
- FilteredRows: The number of rows being filtered

Examples

Keywords

```
SHOW, LAST, INSERT
```

Best Practice

UPDATE

Description

This statement is used to update the data. The UPDATE statement currently only supports the UNIQUE KEY model.

The UPDATE operation currently only supports updating the Value column. The update of the Key column can refer to Using FlinkCDC to update Key column.

Syntax

```
[cte]
UPDATE target_table [table_alias]
    SET assignment_list
    [ FROM additional_tables]
    WHERE condition
```

Required Parameters

- target_table: The target table of the data to be updated. Can be of the form 'db_name.table_name'
- assignment_list: The target column to be updated, in the format 'col_name = value, col_name = value'
- where condition: the condition that is expected to be updated, an expression that returns true or false can be

Optional Parameters

- cte: Common Table Expression, eg 'WITH a AS SELECT * FROM tbl'
- table_alias: alias of table
- FROM additional_tables: Specifies one or more tables to use for selecting rows to update or for setting new values. Note that if you want use target table here, you should give it a alias explicitly.

Note

The current UPDATE statement only supports row updates on the UNIQUE KEY model.

Examples

The `test` table is a unique model table, which contains four columns: k1, k2, v1, v2. Where k1, k2 are keys, v1, v2 are values, and the aggregation method is Replace.

1. Update the v1 column in the 'test' table that satisfies the conditions k1 =1 , k2 =2 to 1

```
UPDATE test SET v1 = 1 WHERE k1=1 and k2=2;
```

2. Increment the v1 column of the k1=1 column in the ‘test’ table by 1

```
UPDATE test SET v1 = v1+1 WHERE k1=1;
```

3. use the result of `t2` join `t3` to update `t1`

```
-- create t1, t2, t3 tables
CREATE TABLE t1
  (id INT, c1 BIGINT, c2 STRING, c3 DOUBLE, c4 DATE)
UNIQUE KEY (id)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1', "function_column.sequence_col" = "c4");

CREATE TABLE t2
  (id INT, c1 BIGINT, c2 STRING, c3 DOUBLE, c4 DATE)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1');

CREATE TABLE t3
  (id INT)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1');

-- insert data
INSERT INTO t1 VALUES
  (1, 1, '1', 1.0, '2000-01-01'),
  (2, 2, '2', 2.0, '2000-01-02'),
  (3, 3, '3', 3.0, '2000-01-03');

INSERT INTO t2 VALUES
  (1, 10, '10', 10.0, '2000-01-10'),
  (2, 20, '20', 20.0, '2000-01-20'),
  (3, 30, '30', 30.0, '2000-01-30'),
  (4, 4, '4', 4.0, '2000-01-04'),
  (5, 5, '5', 5.0, '2000-01-05');

INSERT INTO t3 VALUES
  (1),
  (4),
  (5);

-- update t1
UPDATE t1
  SET t1.c1 = t2.c1, t1.c3 = t2.c3 * 100
  FROM t2 INNER JOIN t3 ON t2.id = t3.id
  WHERE t1.id = t2.id;
```

the expect result is only update the row where id = 1 in table t1

```
+----+----+----+--------+------------+
| id | c1 | c2 | c3     | c4         |
+----+----+----+--------+------------+
| 1  | 10 | 1  | 1000.0 | 2000-01-01 |
| 2  | 2  | 2  |    2.0 | 2000-01-02 |
| 3  | 3  | 3  |    3.0 | 2000-01-03 |
+----+----+----+--------+------------+
```

4. using cte

```
create table orders(
    o_orderkey bigint,
    o_totalprice decimal(15, 2)
) unique key(o_orderkey)
distributed by hash(o_orderkey) buckets 1
properties (
    "replication_num" = "1"
);

insert into orders values
(1, 34.1),
(2, 432.8);

create table lineitem(
    l_linenumber int,
    o_orderkey bigint,
    l_discount  decimal(15, 2)
) unique key(l_linenumber)
distributed by hash(l_linenumber) buckets 1
properties (
    "replication_num" = "1"
);

insert into lineitem values
(1, 1, 1.23),
(2, 1, 3.21),
(3, 2, 18.08),
(4, 2, 23.48);

with discount_orders as (
    select * from orders
    where o_totalprice > 100
)
update lineitem  set l_discount = l_discount*0.9
from discount_orders
where lineitem.o_orderkey = discount_orders.o_orderkey;
```

Keywords

```
UPDATE
```

Best Practice

DELETE

Description

This statement is used to conditionally delete data in the specified table (base index) partition.

This operation will also delete the data of the rollup index related to this base index.

Syntax

Syntax 1: This syntax can only specify filter predicates

```
DELETE FROM table_name [PARTITION partition_name | PARTITIONS (partition_name [, partition_name])]
WHERE
column_name op { value | value_list } [ AND column_name op { value | value_list } ...];
```

Syntax 2: This syntax can only used on UNIQUE KEY model

```
[cte]
DELETE FROM table_name
    [PARTITION partition_name | PARTITIONS (partition_name [, partition_name])]
    [USING additional_tables]
    WHERE condition
```

Required Parameters

- table_name: Specifies the table from which rows are removed.
- column_name: column belong to table_name
- op: Logical comparison operator, The optional types of op include: =, >, <, >=, <=, !=, in, not in
- value | value_list: value or value list used for logial comparison
- WHERE condition: Specifies a condition to use to select rows for removal

Optional Parameters

- cte: Common Table Expression, e.g. 'WITH a AS SELECT * FROM tbl'.
- PARTITION partition_name | PARTITIONS (partition_name [, partition_name]): Specifies the partition or partitions to select rows for removal
- table_alias: alias of table
- USING additional_tables: If you need to refer to additional tables in the WHERE clause to help identify the rows to be removed, then specify those table names in the USING clause. You can also use the USING clause to specify subqueries that identify the rows to be removed.

Note

1. Only conditions on the key column can be specified when using AGGREGATE (UNIQUE) model.

2. When the selected key column does not exist in a rollup, delete cannot be performed.
3. Wheny you use syntax 1, conditions can only have an “and” relationship. If you want to achieve an “or” relationship, you need to write the conditions in two DELETE statements.
4. In syntax 1, if it is a partitioned table, you can specify a partition. If not specified, Doris will infer partition from the given conditions. In two cases, Doris cannot infer the partition from conditions: 1) the conditions do not contain partition columns; 2) The operator of the partition column is not in. When the partitioned table is not a Unique table, and it does not specify the partition, or the partition cannot be inferred from the conditions, the session variable delete_without_partition needs to be true to make delete statement be applied to all partitions.

> **Tips**
>
> This feature is supported since the Apache Doris 1.2 version

5. This statement may reduce query efficiency for a period of time after execution. The degree of impact depends on the number of delete conditions specified in the statement. The more conditions you specify, the greater the impact.

Examples

1. Delete the data row whose k1 column value is 3 in my_table partition p1

```
DELETE FROM my_table PARTITION p1
    WHERE k1 = 3;
```

2. Delete the data rows where the value of column k1 is greater than or equal to 3 and the value of column k2 is “abc” in my_table partition p1

```
DELETE FROM my_table PARTITION p1
WHERE k1 >= 3 AND k2 = "abc";
```

3. Delete the data rows where the value of column k1 is greater than or equal to 3 and the value of column k2 is “abc” in my_table partition p1, p2

```
DELETE FROM my_table PARTITIONS (p1, p2)
WHERE k1 >= 3 AND k2 = "abc";
```

4. use the result of `t2` join `t3` to romve rows from `t1`,delete table only support unique key model

```
-- create t1, t2, t3 tables
CREATE TABLE t1
  (id INT, c1 BIGINT, c2 STRING, c3 DOUBLE, c4 DATE)
UNIQUE KEY (id)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1', "function_column.sequence_col" = "c4");

CREATE TABLE t2
  (id INT, c1 BIGINT, c2 STRING, c3 DOUBLE, c4 DATE)
DISTRIBUTED BY HASH (id)
```

```
PROPERTIES('replication_num'='1');

CREATE TABLE t3
  (id INT)
DISTRIBUTED BY HASH (id)
PROPERTIES('replication_num'='1');

-- insert data
INSERT INTO t1 VALUES
  (1, 1, '1', 1.0, '2000-01-01'),
  (2, 2, '2', 2.0, '2000-01-02'),
  (3, 3, '3', 3.0, '2000-01-03');

INSERT INTO t2 VALUES
  (1, 10, '10', 10.0, '2000-01-10'),
  (2, 20, '20', 20.0, '2000-01-20'),
  (3, 30, '30', 30.0, '2000-01-30'),
  (4, 4, '4', 4.0, '2000-01-04'),
  (5, 5, '5', 5.0, '2000-01-05');

INSERT INTO t3 VALUES
  (1),
  (4),
  (5);

-- remove rows from t1
DELETE FROM t1
  USING t2 INNER JOIN t3 ON t2.id = t3.id
  WHERE t1.id = t2.id;
```

the expect result is only remove the row where id = 1 in table t1

```
+----+----+----+--------+------------+
| id | c1 | c2 | c3     | c4         |
+----+----+----+--------+------------+
| 2  | 2  | 2  |    2.0 | 2000-01-02 |
| 3  | 3  | 3  |    3.0 | 2000-01-03 |
+----+----+----+--------+------------+
```

5. using cte

```
create table orders(
 o_orderkey bigint,
 o_totalprice decimal(15, 2)
) unique key(o_orderkey)
distributed by hash(o_orderkey) buckets 1
properties (
"replication_num" = "1"
);

insert into orders values
(1, 34.1),
(2, 432.8);

create table lineitem(
l_linenumber int,
o_orderkey bigint,
l_discount  decimal(15, 2)
) unique key(l_linenumber)
distributed by hash(l_linenumber) buckets 1
properties (
"replication_num" = "1"
```

```
);

insert into lineitem values
(1, 1, 1.23),
(2, 1, 3.21),
(3, 2, 18.08),
(4, 2, 23.48);

with discount_orders as (
select * from orders
where o_totalprice > 100
)
delete from lineitem
using discount_orders
where lineitem.o_orderkey = discount_orders.o_orderkey;
```

Keywords

```
DELETE
```

Best Practice

SHOW DELETE

Description

This statement is used to display the historical delete tasks that have been successfully executed

Syntax

```
SHOW DELETE [FROM <db_name>]
```

Optional Parameters

: The name of the database to be displayed

Return Value

Column	Description
TableName	The name of the table where the delete operation was executed.
PartitionName	The name of the partition affected by the delete operation.
CreateTime	The timestamp when the delete operation was issued.
DeleteCondition	The condition used for the delete operation, specifying which rows were deleted.
State	The status of the delete operation

Example

1. Go to the test database to view all historical delete tasks sql show delete; text +---------------+---------------+---------------------+-----------------+----------+ | TableName | PartitionName | CreateTime | DeleteCondition | State | +---------------+---------------+---------------------+-----------------+----------+ | iceberg_table | * | 2025-03-14 15:53:32 | id EQ "1" | FINISHED | +---------------+---------------+---------------------+-----------------+----------+

2. Display all historical delete tasks of database database

```
show delete from tpch;
```

```
+-----------+---------------+---------------------+-------------------+----------+
| TableName | PartitionName | CreateTime          | DeleteCondition   | State    |
+-----------+---------------+---------------------+-------------------+----------+
| customer  | *             | 2025-03-14 15:45:19 | c_custkey EQ "18" | FINISHED |
+-----------+---------------+---------------------+-------------------+----------+
```

BROKER LOAD

Description

Broker Load is a data import method in Doris, primarily used to import large scale data from remote storage systems such as HDFS or S3. It is initiated through the MySQL API and is an asynchronous import method. The import progress and results can be queried using the `SHOW LOAD` statement.

In earlier versions, S3 and HDFS Load relied on the Broker process. Now, data is read directly from the data source without relying on an additional Broker process. Nevertheless, due to the similar syntax, S3 Load, HDFS Load, and Broker Load are collectively referred to as Broker Load.

Syntax

```
LOAD LABEL [<db_name>.]<load_label>
(
[ { MERGE | APPEND | DELETE } ]
DATA INFILE
(
"<file_path>"[, ...]
)
[ NEGATIVE ]
INTO TABLE `<table_name>`
[ PARTITION ( <partition_name> [ , ... ] ) ]
[ COLUMNS TERMINATED BY "<column_separator>" ]
[ LINES TERMINATED BY "<line_delimiter>" ]
[ FORMAT AS "<file_type>" ]
[ COMPRESS_TYPE AS "<compress_type>" ]
[ (<column_list>) ]
[ COLUMNS FROM PATH AS (<column_name> [ , ... ] ) ]
[ SET (<column_mapping>) ]
[ PRECEDING FILTER <predicate> ]
[ WHERE <predicate> ]
[ DELETE ON <expr> ]
[ ORDER BY <source_sequence> ]
[ PROPERTIES ("<key>"="<value>" [, ...] ) ]
)
WITH BROKER "<broker_name>"
(   <broker_properties>
    [ , ... ])
[ PROPERTIES (
    <load_properties>
    [ , ... ]) ]
[COMMENT "<comment>" ];
```

Required Parameters

1. `<db_name>` > Specifies the name of the database for import.

2. `<load_label>` > Each import task needs to specify a unique Label. The job progress can be queried later using this Label.

3. `<table_name>` > Specifies the table corresponding to the import task.

4. <file_path> > Specifies the file path to be imported. Multiple paths can be specified, and wildcards can be used. The path must ultimately match a file; if it only matches a directory, the import will fail.

5. <broker_name> > Specifies the name of the Broker service to be used. For example, in public - cloud Doris, the Broker service name is bos.

6. <broker_properties> > Specifies the information required by the broker. This information is typically used to enable the Broker to access the remote storage system, such as BOS or HDFS. > >text > (> "username" = "user", > "password" = "pass", > ... >) >

Optional Parameters

1. merge | append | delete
> Data merge type. The default is append, indicating that this import is a normal append-write operation. merge and delete types are only applicable to tables with the unique key model. The merge type needs to be used in conjunction with the [delete on] statement to mark the delete flag column. The delete type indicates that all data imported this time is deletion data.

2. negative
> Indicates “negative” import. This method is only applicable to aggregate data tables with an integer sum aggregation type. It negates the integer values corresponding to the sum aggregation columns in the imported data, which is used to offset incorrect data.

3. <partition_name>
> Specifies to import only certain partitions of the table, for example: partition (p1, p2,...). Other data outside the partition range will be ignored.

4. <column_separator>
> Specifies the column separator, which is only valid in CSV format and can only specify single-byte separators.

5. <line_delimiter>
> Specifies the line separator, which is only valid in CSV format and can only specify single-byte separators.

6. <file_type>
> Specifies the file format, supporting csv (default), parquet, and orc formats.

7. <compress_type>
> Specifies the file compression type, supporting gz, bz2, and lz4frame.

8. <column_list>
> Specifies the column order in the original file.

9. columns from path as (<c1>, <c2>,...)
> Specifies the columns to be extracted from the import file path.

10. <column_mapping>
> Specifies the column conversion function.

11. preceding filter <predicate>
> Data is first spliced into the original data rows according to column list and columns from path as, then filtered

according to the preceding filter condition.

12. `where <predicate>`
> Filters the imported data according to the condition.

13. `delete on <expr>`
> Used in conjunction with the `merge` import mode and is only applicable to tables with the unique key model. It specifies the column representing the delete flag in the imported data and the calculation relationship.

14. `<source_sequence>`
> Only applicable to tables with the unique key model. It specifies the column representing the sequence column in the imported data, mainly to ensure the data order during import.

15. `properties ("<key>"="<value>",...)`
> Specifies the parameters for the import file format, applicable to formats such as CSV, JSON, etc. For example, parameters such as `json_root`, `jsonpaths`, and `fuzzy_parse` can be specified.
> `enclose`: Enclosure character; when a CSV data field contains a line separator or column separator, a single-byte character can be specified as the enclosure character to prevent accidental truncation. For example, if the column separator is “,”, and the enclosure character is ““”, and the data is”a,’b,c’”, then “b,c” will be parsed as one field.
> Note: When `enclose` is set to `"`, `trim_double_quotes` must be set to `true`.
> `escape`: Escape character, used to escape characters in the field that are the same as the enclosure character. For example, if the data is “a,‘b,’c’”, the enclosure character is ““”, and you want “b,’c” to be parsed as one field, you need to specify a single-byte escape character, such as `"`", and then modify the data to “a,‘b,’c’”.

16. `<load_properties>`
> The optional parameters are as follows and can be added based on the actual environment.

Parameter	Parameter Description
timeout	Import timeout period, with a default of 4 hours and the unit in seconds.
max_filter_ratio	The maximum tolerable ratio of filterable data (due to reasons such as data irregularities), with a default of zero tolerance and a value range from 0 to 1.
exec_mem_limit	Import memory limit, with a default of 2GB and the unit in bytes.
strict_mode	Whether to impose strict restrictions on the data, with a default of `false`.
partial_columns	A boolean type. When set to `true`, it indicates using partial - column updates, with a default value of `false`. It can only be set when the table model is Unique and uses Merge on Write.
timezone	Specifies the time zone, which affects some functions affected by the time zone, such as `strftime`, `alignment_timestamp`, `from_unixtime`, etc. For details, please refer to the Time Zone documentation. If not specified, “Asia/Shanghai” will be used.
load_parallelism	Import concurrency. The default is 1. Increasing the import concurrency will start multiple execution plans to execute the import task simultaneously, speeding up the import process.
send_batch_parallelism	Sets the parallelism for sending batch data. If the value of the parallelism exceeds `max_send_batch_parallelism_per_job` in the BE configuration, the value of `max_send_batch_parallelism_per_job` will be used.

Parameter	Parameter Description
load_to_single_tablet	A boolean type. When set to `true`, it indicates supporting importing data into a single tablet of the corresponding partition, with a default value of `false`. The number of tasks in the job depends on the overall concurrency and can only be set when importing an OLAP table with a random bucket.
priority	Sets the priority of the import task, with options of `HIGH/NORMAL/LOW` and a default of `NORMAL`. For import tasks in the `PENDING` state, tasks with a higher priority will enter the `LOADING` state first.
comment	Specifies the remarks information for the import task.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
LOAD_PRIV	Table	Import permissions for the specified database table.

Examples

1. Import a batch of data from HDFS. The imported file is `file.txt`, separated by commas, and imported into the table `my_table`.

```
LOAD LABEL example_db.label1
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/file.txt")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY ","
)
WITH BROKER hdfs
(
    "username"="hdfs_user",
    "password"="hdfs_password"
);
```

2. Import data from HDFS using wildcards to match two batches of files and import them into two tables respectively. Use wildcards to match two batches of files, `file - 10*` and `file - 20*`, and import them into the tables `my_table1` and `my_table2` respectively. For `my_table1`, specify to import into partition `p1`, and import the values of the second and third columns in the source file after adding 1.

```
LOAD LABEL example_db.label2
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/file-10*")
    INTO TABLE `my_table1`
    PARTITION (p1)
    COLUMNS TERMINATED BY ","
    (k1, tmp_k2, tmp_k3)
    SET (
        k2 = tmp_k2 + 1,
        k3 = tmp_k3 + 1
    ),
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/file-20*")
    INTO TABLE `my_table2`
    COLUMNS TERMINATED BY ","
    (k1, k2, k3)
```

```
)
WITH BROKER hdfs
(
    "username"="hdfs_user",
    "password"="hdfs_password"
);
```

3. Import a batch of data from HDFS. Specify the separator as the default Hive separator `\\x01`, and use the wildcard * to specify all files in all directories under the `data` directory. Use simple authentication and configure namenode HA at the same time.

```
LOAD LABEL example_db.label3
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/user/doris/data/*/*")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY "\\x01"
)
WITH BROKER my_hdfs_broker
(
    "username" = "",
    "password" = "",
    "fs.defaultFS" = "hdfs://my_ha",
    "dfs.nameservices" = "my_ha",
    "dfs.ha.namenodes.my_ha" = "my_namenode1, my_namenode2",
    "dfs.namenode.rpc-address.my_ha.my_namenode1" = "nn1_host:rpc_port",
    "dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
    "dfs.client.failover.proxy.provider.my_ha" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

4. Import data in Parquet format and specify the `FORMAT` as `parquet`. By default, it is determined by the file suffix.

```
LOAD LABEL example_db.label4
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/file")
    INTO TABLE `my_table`
    FORMAT AS "parquet"
    (k1, k2, k3)
)
WITH BROKER hdfs
(
    "username"="hdfs_user",
    "password"="hdfs_password"
);
```

5. Import data and extract partition fields from the file path. The columns in the `my_table` are `k1, k2, k3, city, utc_date`. The directory `hdfs://hdfs_host:hdfs_port/user/doris/data/input/dir/city = beijing` contains the following files: `text hdfs://hdfs_host:hdfs_port/input/city=beijing/utc_date=2020-10-01/0000.csv hdfs://hdfs_host:hdfs_port/input/city=beijing/utc_date=2020-10-02/0000.csv hdfs://hdfs_host:hdfs_port/input/city=tianji/utc_date=2020-10-03/0000.csv hdfs://hdfs_host:hdfs_port/input/city=tianji/utc_date=2020-10-04/0000.csv` The files only contain three columns of data, `k1, k2, k3`, and the two columns of data, `city` and `utc_date`, will be extracted from the file path.

```
LOAD LABEL example_db.label10
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/input/city=beijing/*/*")
    INTO TABLE `my_table`
    FORMAT AS "csv"
    (k1, k2, k3)
    COLUMNS FROM PATH AS (city, utc_date)
)
WITH BROKER hdfs
(
    "username"="hdfs_user",
    "password"="hdfs_password"
);
```

6. Filter the data to be imported. Only rows where `k1 = 1` in the original data and `k1 > k2` after conversion will be imported.

```
LOAD LABEL example_db.label6
(
    DATA INFILE("hdfs://host:port/input/file")
    INTO TABLE `my_table`
    (k1, k2, k3)
    SET (
        k2 = k2 + 1
    )
    PRECEDING FILTER k1 = 1
    WHERE k1 > k2
)
WITH BROKER hdfs
(
    "username"="user",
    "password"="pass"
);
```

7. Import data, extract the time partition field from the file path, and the time contains `%3A` (in the HDFS path, ：is not allowed, so all ：will be replaced by `%3A`).

```
LOAD LABEL example_db.label7
(
    DATA INFILE("hdfs://host:port/user/data/*/test.txt")
    INTO TABLE `tbl12`
    COLUMNS TERMINATED BY ","
    (k2,k3)
    COLUMNS FROM PATH AS (data_time)
    SET (
        data_time=str_to_date(data_time, '%Y-%m-%d %H%%3A%i%%3A%s')
    )
)
WITH BROKER hdfs
(
    "username"="user",
    "password"="pass"
);
```

The directory contains the following files:

```
/user/data/data_time=2020-02-17 00%3A00%3A00/test.txt
/user/data/data_time=2020-02-18 00%3A00%3A00/test.txt
```

The table structure is:

```
data_time DATETIME,
k2        INT,
k3        INT
```

8. Import a batch of data from HDFS, specifying the timeout period and the filtering ratio. Use the broker `my_hdfs_broker` with plain - text authentication. Delete the columns in the original data that match the columns where `v2 > 100` in the imported data, and import other columns normally.

```
LOAD LABEL example_db.label8
(
    MERGE DATA INFILE("HDFS://test:802/input/file")
    INTO TABLE `my_table`
    (k1, k2, k3, v2, v1)
    DELETE ON v2 > 100
)
WITH HDFS
(
    "hadoop.username"="user",
    "password"="pass"
)
PROPERTIES
(
    "timeout" = "3600",
    "max_filter_ratio" = "0.1"
);
```

Use the `MERGE` method for import. `my_table` must be a table with the Unique Key model. When the value of the `v2` column in the imported data is greater than 100, the row will be considered a deletion row.

The timeout period for the import task is 3600 seconds, and an error rate of up to 10% is allowed.

9. Specify the `source_sequence` column during import to ensure the replacement order in the `UNIQUE_KEYS` table:

```
LOAD LABEL example_db.label9
(
    DATA INFILE("HDFS://test:802/input/file")
    INTO TABLE `my_table`
    COLUMNS TERMINATED BY ","
    (k1,k2,source_sequence,v1,v2)
    ORDER BY source_sequence
)
WITH HDFS
(
    "hadoop.username"="user",
    "password"="pass"
)
```

`my_table` must be a table with the Unique Key model and a `Sequence Col` must be specified. The data will be ordered according to the values in the `source_sequence` column of the source data.

10. Import a batch of data from HDFS, specifying the file format as `json` and setting `json_root` and `jsonpaths`:

```
LOAD LABEL example_db.label10
(
    DATA INFILE("HDFS://test:port/input/file.json")
    INTO TABLE `my_table`
    FORMAT AS "json"
    PROPERTIES(
      "json_root" = "$.item",
      "jsonpaths" = "[$.id, $.city, $.code]"
    )
)
WITH BROKER HDFS (
    "hadoop.username" = "user",
    "password" = ""
)
PROPERTIES
(
    "timeout"="1200",
    "max_filter_ratio"="0.1"
);
```

`jsonpaths` can be used in conjunction with `column list` and `SET (column_mapping)`:

```
LOAD LABEL example_db.label10
(
    DATA INFILE("HDFS://test:port/input/file.json")
    INTO TABLE `my_table`
    FORMAT AS "json"
    (id, code, city)
    SET (id = id * 10)
    PROPERTIES(
      "json_root" = "$.item",
      "jsonpaths" = "[$.id, $.code, $.city]"
    )
)
WITH BROKER HDFS (
    "hadoop.username" = "user",
    "password" = ""
)
PROPERTIES
(
    "timeout"="1200",
    "max_filter_ratio"="0.1"
);
```

11. Import data in CSV format from Tencent Cloud COS.

```
LOAD LABEL example_db.label10
(
    DATA INFILE("cosn://my_bucket/input/file.csv")
    INTO TABLE `my_table`
    (k1, k2, k3)
)
WITH BROKER "broker_name"
(
    "fs.cosn.userinfo.secretId" = "xxx",
    "fs.cosn.userinfo.secretKey" = "xxxx",
    "fs.cosn.bucket.endpoint_suffix" = "cos.xxxxxxxxx.myqcloud.com"
)
```

12. Remove double quotes and skip the first 5 rows when importing CSV data.

```
LOAD LABEL example_db.label12
(
    DATA INFILE("cosn://my_bucket/input/file.csv")
    INTO TABLE `my_table`
    (k1, k2, k3)
    PROPERTIES("trim_double_quotes" = "true", "skip_lines" = "5")
)
WITH BROKER "broker_name"
(
    "fs.cosn.userinfo.secretId" = "xxx",
    "fs.cosn.userinfo.secretKey" = "xxxx",
    "fs.cosn.bucket.endpoint_suffix" = "cos.xxxxxxxxx.myqcloud.com"
)
```

MYSQL LOAD

Description

Use the MySQL client to import local data files into Doris. MySQL Load is a synchronous import method, which returns the import result immediately after execution. You can determine whether the import is successful based on the return result of the `LOAD DATA` statement. MySQL Load can ensure the atomicity of a batch of import tasks, meaning that either all imports succeed or all fail.

Syntax

```
LOAD DATA
[ LOCAL ]
INFILE "<file_name>"
INTO TABLE "<tbl_name>"
[ PARTITION (<partition_name> [, ... ]) ]
[ COLUMNS TERMINATED BY "<column_separator>" ]
[ LINES TERMINATED BY "<line_delimiter>" ]
[ IGNORE <number> {LINES | ROWS} ]
[ (<col_name_or_user_var> [, ... ] ) ]
[ SET (col_name={<expr> | DEFAULT} [, col_name={<expr> | DEFAULT}] ...) ]
[ PROPERTIES ("<key>" = "<value>" [ , ... ]) ]
```

Required Parameters

1. `<file_name>`

> Specify the path of the local file, which can be either a relative or an absolute path. Currently, only a single file is supported, and multiple files are not supported.

2. `<tbl_name>`

> The table name can include the database name, as shown in the examples. If the database name is omitted, the current user's database will be used.

Optional Parameters

1. `LOCAL`

> Specifying `LOCAL` indicates reading files from the client. Omitting it means reading files from the local storage of the FE server. The function of importing files from the FE server is disabled by default. You need to set `mysql_load_server_secure_path` on the FE node to specify a secure path to enable this function.

2. `<partition_name>`

> Multiple partitions can be specified for import, separated by commas.

3. `<column_separator>`

> Specify the column separator.

4. `<line_delimiter>`

> Specify the line delimiter.

5. `IGNORE <number> { LINES | ROWS }`

> Users can skip the header of the CSV file or any number of lines. This syntax can also be replaced with `IGNORE num ROWS`.

6. `<col_name_or_user_var>`

> Column mapping syntax. For specific parameters, refer to the column mapping section of Data Transformation during Import.

7. `properties ("<key>"="<value>",...)`

Parameter	Parameter Description
max_filter_ratio	The maximum tolerable ratio of filterable data (due to reasons such as data irregularities), with a default of zero tolerance.
timeout	Specify the import timeout period in seconds. The default is 600 seconds, and the valid range is from 1 second to 259,200 seconds.
strict_mode	Users can specify whether to enable strict mode for this import. The default is disabled.
timezone	Specify the time zone for this import. The default is the East Eight Time Zone. This parameter will affect the results of all time - zone - related functions involved in the import.
exec_mem_limit	The import memory limit, with a default of 2GB in bytes.
trim_double_quotes	A boolean type with a default value of `false`. When set to `true`, it means trimming the outermost double quotes of each field in the imported file.

Parameter	Parameter Description
enclose	Enclosure character. When a CSV data field contains a line separator or column separator, to prevent accidental truncation, a single - byte character can be specified as the enclosure character for protection. For example, if the column separator is “,”, the enclosure character is ““”, and the data is”a,’b,c’”, then “b,c” will be parsed as one field. Note: When `enclose` is set to `""`, `trim_double_quotes` must be set to `true`.
escape	Escape character. Used to escape characters in the CSV field that are the same as the enclosure character. For example, if the data is “a,‘b,’c’”, the enclosure character is ““”, and you want”b,’c” to be parsed as one field, you need to specify a single - byte escape character, such as ““, and then modify the data to”a,’b,’c’”.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
LOAD_PRIV	Table	Import permissions for the specified database table.

Usage Notes

- The MySQL Load statement starts with the syntax `LOAD DATA` and does not require specifying a LABEL.

Examples

1. Import data from the client’s local file `testData` into the table `testTbl` in the database `testDb`. Specify a timeout of 100 seconds.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("timeout"="100")
```

2. Import data from the server’s local file `/root/testData` (you need to set the FE configuration `mysql_load_server_secure_path` to `/root`) into the table `testTbl` in the database `testDb`. Specify a timeout of 100 seconds.

```
LOAD DATA
INFILE '/root/testData'
INTO TABLE testDb.testTbl
PROPERTIES ("timeout"="100")
```

3. Import data from the client’s local file `testData` into the table `testTbl` in the database `testDb`, allowing an error rate of 20%.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("max_filter_ratio"="0.2")
```

4. Import data from the client's local file `testData` into the table `testTbl` in the database `testDb`, allowing an error rate of 20%, and specify the column names of the file.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
(k2, k1, v1)
PROPERTIES ("max_filter_ratio"="0.2")
```

5. Import data from the local file `testData` into partitions `p1` and `p2` of the table `testTbl` in the database `testDb`, allowing an error rate of 20%.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PARTITION (p1, p2)
PROPERTIES ("max_filter_ratio"="0.2")
```

6. Import data from the local CSV file `testData` with a line separator of `0102` and a column separator of `0304` into the table `testTbl` in the database `testDb`.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
COLUMNS TERMINATED BY '0304'
LINES TERMINATED BY '0102'
```

7. Import data from the local file `testData` into partitions `p1` and `p2` of the table `testTbl` in the database `testDb` and skip the first 3 lines.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PARTITION (p1, p2)
IGNORE 3 LINES
```

8. Import data with strict mode filtering and set the time zone to `Africa/Abidjan`.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("strict_mode"="true", "timezone"="Africa/Abidjan")
```

9. Limit the import memory to 10GB and set a timeout of 10 minutes for the data import.

```
LOAD DATA LOCAL
INFILE 'testData'
INTO TABLE testDb.testTbl
PROPERTIES ("exec_mem_limit"="10737418240", "timeout"="600")
```

SHOW LOAD

Description

This statement is used to display the execution status of the specified import task.

Syntax

```
SHOW LOAD
[FROM <db_name>]
[
   WHERE
   [LABEL  = [ "<your_label>" | LIKE "<label_matcher>"]]
   [ STATE = { " PENDING " | " ETL " | " LOADING " | " FINISHED " | " CANCELLED " } ]
]
[ORDER BY { <col_name> | <expr> | <position> }]
[LIMIT <limit>[OFFSET <offset>]];
```

Optional Parameters

1. `<db_name>`

> If `db_name` is not specified, the current default database will be used.

2. `<label_matcher>`

> When using `LABEL LIKE = "<label_matcher>"`, it will match import tasks whose labels contain `label_matcher`.

3. `<your_label>`

> When using `LABEL = "<your_label>"`, it will precisely match the specified label.

4. STATE = { ” PENDING ” | ” ETL ” | ” LOADING ” | ” FINISHED ” | ” CANCELLED ” }

> Specifying `PENDING` means matching jobs with the `STATE = "PENDING"` status. The same applies to other status keywords.

5. `<col_name>`

> Specify the column name in the result set for sorting.

6. `<expr>`

> Use an expression for sorting.

7. `<position>`

> Sort by the position of the column in the `SELECT` list (starting from 1).

8. `<limit>`

> If `LIMIT` is specified, it will display `limit` matching records. Otherwise, all records will be displayed.

9. `<offset>`

> Specify to start displaying query results from the offset `offset`. By default, the offset is 0.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
LOAD_PRIV	Database	Import permissions for the database tables are required.

Return Value

Returns the detailed status of the specified import task.

Examples

1. Display all import tasks in the default database.

   ```
   SHOW LOAD;
   ```

2. Display import tasks in the specified database where the label contains the string “2014_01_02”, and show the oldest 10 tasks.

```
SHOW LOAD FROM example_db WHERE LABEL LIKE "2014_01_02" LIMIT 10;
```

3. Display import tasks in the specified database with the specified label “load_example_db_20140102” and sort them in descending order by `LoadStartTime`.

```
SHOW LOAD FROM example_db WHERE LABEL = "load_example_db_20140102" ORDER BY LoadStartTime DESC;
```

4. Display import tasks in the specified database with the specified label “load_example_db_20140102”, the state “loading”, and sort them in descending order by `LoadStartTime`.

```
SHOW LOAD FROM example_db WHERE LABEL = "load_example_db_20140102" AND STATE = "loading" ORDER BY LoadStartTime DESC;
```

5. Display import tasks in the specified database, sort them in descending order by `LoadStartTime`, and start displaying 10 query results from offset 5.

```
SHOW LOAD FROM example_db ORDER BY LoadStartTime DESC limit 5,10;
SHOW LOAD FROM example_db ORDER BY LoadStartTime DESC limit 10 offset 5;
```

6. Command to check the import status during small - batch imports.

```
curl --location-trusted -u {user}:{passwd} http://{hostname}:{port}/api/{database}/_load_info?label={labelname}
```

SHOW STREAM LOAD

Description

This statement is used to display the execution of the specified Stream Load task

grammar:

```
SHOW STREAM LOAD
[FROM db_name]
[
  WHERE
  [LABEL [ = "your_label" | LIKE "label_matcher"]]
  [STATUS = ["SUCCESS"|"FAIL"]]
]
[ORDER BY...]
[LIMIT limit][OFFSET offset];
```

illustrate:

1. By default, BE does not record Stream Load records. If you want to view records that need to be enabled on BE, the configuration parameter is: `enable_stream_load_record=true`. For details, please refer to BE Configuration Items
2. If db_name is not specified, the current default db is used
3. If LABEL LIKE is used, it will match the tasks whose label of the Stream Load task contains label_matcher
4. If LABEL = is used, it will match the specified label exactly
5. If STATUS is specified, matches STREAM LOAD status
6. You can use ORDER BY to sort on any combination of columns
7. If LIMIT is specified, limit matching records are displayed. Otherwise show all
8. If OFFSET is specified, the query results are displayed starting at offset offset. By default the offset is 0.

Example

1. Show all Stream Load tasks of the default db

   ```
   SHOW STREAM LOAD;
   ```

2. Display the Stream Load task of the specified db, the label contains the string “2014_01_02”, and display the oldest 10

   ```
   SHOW STREAM LOAD FROM example_db WHERE LABEL LIKE "2014_01_02" LIMIT 10;
   ```

3. Display the Stream Load task of the specified db and specify the label as “load_example_db_20140102”

   ```
   SHOW STREAM LOAD FROM example_db WHERE LABEL = "load_example_db_20140102";
   ```

4. Display the Stream Load task of the specified db, specify the status as “success”, and sort by StartTime in descending order

```
SHOW STREAM LOAD FROM example_db WHERE STATUS = "success" ORDER BY StartTime DESC;
```

5. Display the import tasks of the specified db and sort them in descending order of StartTime, and display 10 query results starting from offset 5

```
SHOW STREAM LOAD FROM example_db ORDER BY StartTime DESC limit 5,10;
SHOW STREAM LOAD FROM example_db ORDER BY StartTime DESC limit 10 offset 5;
```

Keywords

SHOW, STREAM, LOAD

SHOW CREATE LOAD

Description

This statement is used to display the creation statement of an import job.

Syntax

```
SHOW CREATE LOAD FOR <load_name>;
```

Required Parameters

`<load_name>`

> The name of the routine import job.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN/NODE_PRIV	Database	Cluster administrator privileges are required.

Return Value

Returns the creation statement of the specified import job.

Examples

- Display the creation statement of the specified import job in the default database.

```
SHOW CREATE LOAD for test_load
```

CANCEL LOAD

Description

This statement is used to cancel an import job with a specified `label`, or to cancel import jobs in batches through fuzzy matching.

Syntax

```
CANCEL LOAD
[FROM <db_name>]
WHERE [LABEL = "<load_label>" | LABEL like "<label_pattern>" | STATE = { "PENDING" | "ETL" | "LOADING" } ]
```

Required Parameters

1. `<db_name>`

> The name of the database where the import job to be cancelled resides.

Optional Parameters

1. `<load_label>`

> If `LABEL = "<load_label>"` is used, it precisely matches the specified label.

2. `<label_pattern>`

> If `LABEL LIKE "<label_pattern>"` is used, it matches import tasks whose labels contain the `label_pattern`.

3. STATE = { ” PENDING ” | ” ETL ” | ” LOADING ” }

> Specifying `PENDING` means cancelling jobs with the `STATE = "PENDING"` status. The same applies to other statuses.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
LOAD_PRIV	Database	Import permissions for the database tables are required.

Usage Notes

- Cancelling jobs based on the `State` is supported starting from version 1.2.0.
- Only incomplete import jobs in the `PENDING`, `ETL`, or `LOADING` states can be cancelled.
- When performing batch cancellation, Doris does not guarantee that all corresponding import jobs will be cancelled atomically. That is, only some import jobs may be cancelled successfully. Users can check the job status using the `SHOW LOAD` statement and try to execute the `CANCEL LOAD` statement again.

Examples

1. Cancel the import job with the label `example_db_test_load_label` in the database `example_db`.

```
CANCEL LOAD
FROM example_db
WHERE LABEL = "example_db_test_load_label";
```

2. Cancel all import jobs containing `example_` in the database `example_db`.

```
CANCEL LOAD
FROM example_db
WHERE LABEL like "%example_%";
```

3. Cancel import jobs in the `LOADING` state.

```
CANCEL LOAD
FROM example_db
WHERE STATE = "loading";
```

SHOW LOAD WARNINGS

Description

If an import task fails and the error message is `ETL_QUALITY_UNSATISFIED`, it indicates that there are import quality issues. This statement is used to view these import tasks with quality issues.

Syntax

```
SHOW LOAD WARNINGS
[FROM <db_name>]
[
   WHERE
   [LABEL  = [ "<your_label>" ]]
   [LOAD_JOB_ID = ["<job_id>"]]
]
```

Optional Parameters

1. `<db_name>`

> If `db_name` is not specified, the current default database will be used.

2. `<your_label>`

> If `LABEL = <your_label>` is used, it will precisely match the specified label.

3. `<job_id>`

> If `LOAD_JOB_ID = <job_id>` is specified, it will precisely match the specified JOB_ID.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
LOAD_PRIV	Database	Import permissions for the database tables are required.

Return Value

Returns the data with quality issues in the import tasks of the specified database.

Examples

- Display the data with quality issues in the import tasks of the specified database, where the label is specified as “load_demo_20210112”.

```
SHOW LOAD WARNINGS FROM demo WHERE LABEL = "load_demo_20210112"
```

CREATE ROUTINE LOAD

Description

The Routine Load feature allows users to submit a resident import task that continuously reads data from a specified data source and imports it into Doris.

Currently, it only supports importing CSV or Json format data from Kafka through unauthenticated or SSL authentication methods. Example of importing Json format data

Syntax

```
CREATE ROUTINE LOAD [<db>.]<job_name> [ON <tbl_name>]
[<merge_type>]
[<load_properties>]
[<job_properties>]
FROM <data_source> [<data_source_properties>]
[COMMENT "<comment>"]
```

Required Parameters

1. `[<db>.]<job_name>`

> The name of the import job. Within the same database, only one job with the same name can be running.

2. `FROM <data_source>`

> The type of data source. Currently supports: KAFKA

3. `<data_source_properties>`

> 1. `<kafka_broker_list>`
>
>   Kafka broker connection information. Format is ip:host. Multiple brokers are separated by commas.
>
>    ```
>    "kafka_broker_list" = "broker1:9092,broker2:9092"
>    ```
>
> 2. `<kafka_topic>`
>
>   Specifies the Kafka topic to subscribe to.

```
"kafka_topic" = "my_topic"
```

Optional Parameters

1. `<tbl_name>`

Specifies the name of the table to import into. This is an optional parameter. If not specified, the dynamic table method is used, which requires the data in Kafka to contain table name information.

Currently, only supports getting table names from Kafka's Value, and it needs to follow this format: for json example: `table_name|{"col1": "val1", "col2": "val2"}`, where `tbl_name` is the table name, with `|` as the separator between table name and table data.

For csv format data, it's similar: `table_name|val1,val2,val3`. Note that `table_name` here must match the table name in Doris, otherwise the import will fail.

Tips: Dynamic tables do not support the `columns_mapping` parameter. If your table structure matches the table structure in Doris and there is a large amount of table information to import, this method will be the best choice.

2. `<merge_type>`

Data merge type. Default is APPEND, which means the imported data are ordinary append write operations. MERGE and DELETE types are only available for Unique Key model tables. The MERGE type needs to be used with the [DELETE ON] statement to mark the Delete Flag column. The DELETE type means that all imported data are deleted data.

Tips: When using dynamic multiple tables, please note that this parameter should be consistent with each dynamic table's type, otherwise it will result in import failure.

3. `<load_properties>`

Used to describe imported data. The composition is as follows:

```
[column_separator],
[columns_mapping],
[preceding_filter],
[where_predicates],
[partitions],
[DELETE ON],
[ORDER BY]
```

1. `<column_separator>`

   Specifies the column separator, defaults to `\t`

```
COLUMNS TERMINATED BY ","
```

2. <columns_mapping>

   Used to specify the mapping relationship between file columns and table columns, as well as various column transformations. For a detailed introduction to this part, you can refer to the [Column Mapping, Transformation and Filtering] document.

   ```
   (k1, k2, tmpk1, k3 = tmpk1 + 1)
   ```

   Tips: Dynamic tables do not support this parameter.

3. <preceding_filter>

   Filter raw data. For detailed information about this part, please refer to the [Column Mapping, Transformation and Filtering] document.

   ```
   WHERE k1 > 100 and k2 = 1000
   ```

   Tips: Dynamic tables do not support this parameter.

4. <where_predicates>

   Filter imported data based on conditions. For detailed information about this part, please refer to the [Column Mapping, Transformation and Filtering] document.

   ```
   WHERE k1 > 100 and k2 = 1000
   ```

   Tips: When using dynamic multiple tables, please note that this parameter should match the columns of each dynamic table, otherwise the import will fail. When using dynamic multiple tables, we only recommend using this parameter for common public columns.

5. <partitions>

   Specify which partitions of the destination table to import into. If not specified, data will be automatically imported into the corresponding partitions.

   ```
   PARTITION(p1, p2, p3)
   ```

   Tips: When using dynamic multiple tables, please note that this parameter should match each dynamic table, otherwise the import will fail.

6. <DELETE ON>

   Must be used with MERGE import mode, only applicable to Unique Key model tables. Used to specify the Delete Flag column and calculation relationship in the imported data.

   ```
   DELETE ON v3 >100
   ```

   Tips: When using dynamic multiple tables, please note that this parameter should match each dynamic table, otherwise the import will fail.

7. <ORDER BY>

Only applicable to Unique Key model tables. Used to specify the Sequence Col column in the imported data. Mainly used to ensure data order during import.

Tips: When using dynamic multiple tables, please note that this parameter should match each dynamic table, otherwise the import will fail.

4. `<job_properties>`

Used to specify general parameters for routine import jobs.

```
PROPERTIES (
    "key1" = "val1",
    "key2" = "val2"
)
```

Currently, we support the following parameters:

1. `<desired_concurrent_number>`

   The desired concurrency. A routine import job will be divided into multiple subtasks for execution. This parameter specifies how many tasks can run simultaneously for a job. Must be greater than 0. Default is 5.

   This concurrency is not the actual concurrency. The actual concurrency will be determined by considering the number of cluster nodes, load conditions, and data source conditions.

   `"desired_concurrent_number" = "3"`

2. `<max_batch_interval>/<max_batch_rows>/<max_batch_size>`

   These three parameters represent:

   1. Maximum execution time for each subtask, in seconds. Must be greater than or equal to 1. Default is 10.
   2. Maximum number of rows to read for each subtask. Must be greater than or equal to 200000. Default is 20000000.
   3. Maximum number of bytes to read for each subtask. Unit is bytes, range is 100MB to 10GB. Default is 1G.

   These three parameters are used to control the execution time and processing volume of a subtask. When any one reaches the threshold, the task ends.

   ```
   "max_batch_interval" = "20",
   "max_batch_rows" = "300000",
   "max_batch_size" = "209715200"
   ```

3. `<max_error_number>`

   Maximum number of error rows allowed within the sampling window. Must be greater than or equal to 0. Default is 0, meaning no error rows are allowed.

The sampling window is `max_batch_rows * 10`. If the number of error rows within the sampling window exceeds `max_error_number`, the routine job will be suspended and require manual intervention to check data quality issues.

Rows filtered by where conditions are not counted as error rows.

4. `<strict_mode>`

   Whether to enable strict mode, default is off. If enabled, when non-null original data's column type conversion results in NULL, it will be filtered. Specified as:

   `"strict_mode" = "true"`

   Strict mode means: strictly filter column type conversions during the import process. The strict filtering strategy is as follows:

   1. For column type conversion, if strict mode is true, erroneous data will be filtered. Here, erroneous data refers to: original data that is not null but results in null value after column type conversion.
   2. For columns generated by function transformation during import, strict mode has no effect.
   3. For columns with range restrictions, if the original data can pass type conversion but cannot pass range restrictions, strict mode has no effect. For example: if the type is decimal(1,0) and the original data is 10, it can pass type conversion but is outside the column's declared range. Strict mode has no effect on such data.

   Relationship between strict mode and source data import

   Here's an example using TinyInt column type

   Note: When columns in the table allow null values

source data	source data example	string to int	strict_mode	result
null	`\N`	N/A	true or false	NULL
not null	aaa or 2000	NULL	true	invalid data(filtered)
not null	aaa	NULL	false	NULL
not null	1	1	true or false	correct data

   Here's an example using Decimal(1,0) column type

   Note: When columns in the table allow null values

source data	source data example	string to int	strict_mode	result
null	`\N`	N/A	true or false	NULL
not null	aaa	NULL	true	invalid data(filtered)
not null	aaa	NULL	false	NULL
not null	1 or 10	1	true or false	correct data

Note: Although 10 is a value exceeding the range, because its type meets decimal requirements, strict mode has no effect on it. 10 will eventually be filtered in other ETL processing flows, but won't be filtered by strict mode.

5. <timezone>

   Specifies the timezone used for the import job. Defaults to the Session's timezone parameter. This parameter affects all timezone-related function results involved in the import.

   "timezone" = "Asia/Shanghai"

6. <format>

   Specifies the import data format, default is csv, json format is supported.

   "format" = "json"

7. <jsonpaths>

   When importing json format data, jsonpaths can be used to specify fields to extract from Json data.

   -H "jsonpaths: [\"$.k2\", \"$.k1\"]"

8. <strip_outer_array>

   When importing json format data, strip_outer_array set to true indicates that Json data is presented as an array, where each element in the data will be treated as a row. Default value is false.

   -H "strip_outer_array: true"

9. <json_root>

   When importing json format data, json_root can be used to specify the root node of Json data. Doris will parse elements extracted from the root node through json_root. Default is empty.

   -H "json_root: $.RECORDS"

10. <send_batch_parallelism>

    Integer type, used to set the parallelism of sending batch data. If the parallelism value exceeds `max_send_batch_parallelism_per_job` in BE configuration, the BE serving as the coordination point will use the value of `max_send_batch_parallelism_per_job`.

    "send_batch_parallelism" = "10"

11. <load_to_single_tablet>

    Boolean type, true indicates support for a task to import data to only one tablet of the corresponding partition, default value is false. This parameter is only allowed to be set when importing data to olap tables with random bucketing.

    "load_to_single_tablet" = "true"

12. <partial_columns>

Boolean type, true indicates using partial column updates, default value is false. This parameter is only allowed to be set when the table model is Unique and uses Merge on Write. Dynamic multiple tables do not support this parameter.

```
"partial_columns" = "true"
```

13. `<max_filter_ratio>`

    Maximum filter ratio allowed within the sampling window. Must be between greater than or equal to 0 and less than or equal to 1. Default value is 0.

    The sampling window is `max_batch_rows * 10`. If within the sampling window, error rows/total rows exceeds `max_filter_ratio`, the routine job will be suspended and require manual intervention to check data quality issues.

    Rows filtered by where conditions are not counted as error rows.

14. `<enclose>`

    Enclosure character. When csv data fields contain row or column separators, to prevent accidental truncation, a single-byte character can be specified as an enclosure for protection. For example, if the column separator is “,” and the enclosure is ““, for data”a,’b,c’”, “b,c” will be parsed as one field.

    Note: When enclose is set to ", trim_double_quotes must be set to true.

15. `<escape>`

    Escape character. Used to escape characters in csv fields that are the same as the enclosure character. For example, if the data is “a,‘b,’c’”, enclosure is “’”, and you want “b,’c” to be parsed as one field, you need to specify a single-byte escape character, such as `\`, and modify the data to `a,'b,\'c'`.

5. Optional properties in `data_source_properties`

1. `<kafka_partitions>/<kafka_offsets>`

   Specifies the kafka partitions to subscribe to and the starting offset for each partition. If a time is specified, consumption will start from the nearest offset greater than or equal to that time.

   offset can be specified as a specific offset greater than or equal to 0, or:

   - `OFFSET_BEGINNING`: Start subscribing from where data exists.
   - `OFFSET_END`: Start subscribing from the end.
   - Time format, such as: “2021-05-22 11:00:00”

   If not specified, defaults to subscribing to all partitions under the topic from `OFFSET_END`.

   ```
   "kafka_partitions" = "0,1,2,3",
   "kafka_offsets" = "101,0,OFFSET_BEGINNING,OFFSET_END"
   ```

```
"kafka_partitions" = "0,1,2,3",
"kafka_offsets" = "2021-05-22 11:00:00,2021-05-22 11:00:00,2021-05-22 11:00:00"
```

Note: Time format cannot be mixed with OFFSET format.

2. <property>

   Specifies custom kafka parameters. Functions the same as the “–property” parameter in kafka shell.

   When the value of a parameter is a file, the keyword “FILE:” needs to be added before the value.

   For information about how to create files, please refer to the CREATE FILE command documentation.

   For more supported custom parameters, please refer to the client configuration items in the official CONFIGURATION documentation of librdkafka. For example:

   ```
   "property.client.id" = "12345",
   "property.ssl.ca.location" = "FILE:ca.pem"
   ```

   2.1 When using SSL to connect to Kafka, the following parameters need to be specified:

   ```
   ```text
   "property.security.protocol" = "ssl",
   "property.ssl.ca.location" = "FILE:ca.pem",
   "property.ssl.certificate.location" = "FILE:client.pem",
   "property.ssl.key.location" = "FILE:client.key",
   "property.ssl.key.password" = "abcdefg"
   ```

   Among them:

   `property.security.protocol` and `property.ssl.ca.location` are required, used to specify the connection method as SSL and the location of the CA certificate.

   If client authentication is enabled on the Kafka server side, the following also need to be set:

   ```text
   "property.ssl.certificate.location"
   "property.ssl.key.location"
   "property.ssl.key.password"
   ```

   Used to specify the client's public key, private key, and private key password respectively.
   ```

   2.2 Specify default starting offset for kafka partitions

   If <kafka_partitions>/<kafka_offsets> is not specified, all partitions will be consumed by default.

   In this case, <kafka_default_offsets> can be specified to set the starting offset. Default is OFFSET_END, meaning subscription starts from the end.

   Example:

```
"property.kafka_default_offsets" = "OFFSET_BEGINNING"
```

6. `COMMENT`

```
Comment information for the routine load task.
```

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
LOAD_PRIV	Table	CREATE ROUTINE LOAD belongs to table LOAD operation

Usage Notes

- Dynamic tables do not support the `columns_mapping` parameter
- When using dynamic multiple tables, parameters like merge_type, where_predicates, etc., need to conform to each dynamic table's requirements
- Time format cannot be mixed with OFFSET format
- `kafka_partitions` and `kafka_offsets` must correspond one-to-one
- When `enclose` is set to ", `trim_double_quotes` must be set to true.

Examples

- Create a Kafka routine load task named test1 for example_tbl in example_db. Specify column separator, group.id and client.id, and automatically consume all partitions by default, starting subscription from where data exists (OFFSET_BEGINNING)

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS TERMINATED BY ",",
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100)
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "property.group.id" = "xxx",
    "property.client.id" = "xxx",
```

```
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

- Create a Kafka routine dynamic multi-table load task named test1 for example_db. Specify column separator, group.id and client.id, and automatically consume all partitions by default, starting subscription from where data exists (OFFSET_BEGINNING)

  Assuming we need to import data from Kafka into test1 and test2 tables in example_db, we create a routine load task named test1, and write data from test1 and test2 to a Kafka topic named `my_topic`. This way, we can import data from Kafka into two tables through one routine load task.

```
CREATE ROUTINE LOAD example_db.test1
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "property.group.id" = "xxx",
    "property.client.id" = "xxx",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

- Create a Kafka routine load task named test1 for example_tbl in example_db. The import task is in strict mode.

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100),
PRECEDING FILTER k1 = 1,
WHERE k1 > 100 and k2 like "%doris%"
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "true"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2,3",
    "kafka_offsets" = "101,0,0,200"
);
```

- Import data from Kafka cluster using SSL authentication. Also set client.id parameter. Import task is in non-strict mode, timezone is Africa/Abidjan

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(k1, k2, k3, v1, v2, v3 = k1 * 100),
```

```
WHERE k1 > 100 and k2 like "%doris%"
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false",
    "timezone" = "Africa/Abidjan"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "property.security.protocol" = "ssl",
    "property.ssl.ca.location" = "FILE:ca.pem",
    "property.ssl.certificate.location" = "FILE:client.pem",
    "property.ssl.key.location" = "FILE:client.key",
    "property.ssl.key.password" = "abcdefg",
    "property.client.id" = "my_client_id"
);
```

- Import Json format data. Use field names in Json as column name mapping by default. Specify importing partitions 0,1,2, all starting offsets are 0

```
CREATE ROUTINE LOAD example_db.test_json_label_1 ON table1
COLUMNS(category,price,author)
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false",
    "format" = "json"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2",
    "kafka_offsets" = "0,0,0"
);
```

- Import Json data, extract fields through Jsonpaths, and specify Json document root node

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
COLUMNS(category, author, price, timestamp, dt=from_unixtime(timestamp, '%Y%m%d'))
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false",
    "format" = "json",
    "jsonpaths" = "[\"$.category\",\"$.author\",\"$.price\",\"$.timestamp\"]",
    "json_root" = "$.RECORDS"
    "strip_outer_array" = "true"
)
FROM KAFKA
```

```
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2",
    "kafka_offsets" = "0,0,0"
);
```

- Create a Kafka routine load task named test1 for example_tbl in example_db with condition filtering.

```
CREATE ROUTINE LOAD example_db.test1 ON example_tbl
WITH MERGE
COLUMNS(k1, k2, k3, v1, v2, v3),
WHERE k1 > 100 and k2 like "%doris%",
DELETE ON v3 >100
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "20",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200",
    "strict_mode" = "false"
)
FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2,3",
    "kafka_offsets" = "101,0,0,200"
);
```

- Import data into a Unique Key model table containing sequence columns

```
CREATE ROUTINE LOAD example_db.test_job ON example_tbl
COLUMNS TERMINATED BY ",",
COLUMNS(k1,k2,source_sequence,v1,v2),
ORDER BY source_sequence
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "30",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200"
) FROM KAFKA
(
    "kafka_broker_list" = "broker1:9092,broker2:9092,broker3:9092",
    "kafka_topic" = "my_topic",
    "kafka_partitions" = "0,1,2,3",
    "kafka_offsets" = "101,0,0,200"
);
```

- Start consuming from a specified time point

```
CREATE ROUTINE LOAD example_db.test_job ON example_tbl
PROPERTIES
(
    "desired_concurrent_number"="3",
    "max_batch_interval" = "30",
    "max_batch_rows" = "300000",
    "max_batch_size" = "209715200"
) FROM KAFKA
```

```
(
    "kafka_broker_list" = "broker1:9092,broker2:9092",
    "kafka_topic" = "my_topic",
    "property.kafka_default_offsets" = "2021-05-21 10:00:00"
);
```

ALTER ROUTINE LOAD

Description

This syntax is used to modify an existing routine load job. Only jobs in PAUSED state can be modified.

Syntax

```
ALTER ROUTINE LOAD FOR [<db>.]<job_name>
[<job_properties>]
FROM [<data_source>]
[<data_source_properties>]
```

Required Parameters

1. `[<db>.]<job_name>`

> Specifies the name of the job to be modified. The identifier must begin with a letter character and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks.
>
> The identifier cannot use reserved keywords. For more details, please refer to identifier requirements and reserved keywords.

Optional Parameters

1. `<job_properties>`

> Specifies the job parameters to be modified. Currently supported parameters include:
>
> - desired_concurrent_number
> - max_error_number
> - max_batch_interval
> - max_batch_rows
> - max_batch_size
> - jsonpaths
> - json_root
> - strip_outer_array
> - strict_mode
> - timezone
> - num_as_string
> - fuzzy_parse
> - partial_columns

- max_filter_ratio

2. `<data_source_properties>`

Properties related to the data source. Currently supports:

- `<kafka_partitions>`
- `<kafka_offsets>`
- `<kafka_broker_list>`
- `<kafka_topic>`
- Custom properties, such as `<property.group.id>`

3. `<data_source>`

The type of data source. Currently supports:

- KAFKA

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
LOAD_PRIV	Table	ALTER ROUTINE LOAD requires LOAD privilege on the table

Notes

- `kafka_partitions` and `kafka_offsets` are used to modify the offset of kafka partitions to be consumed, and can only modify currently consumed partitions. New partitions cannot be added.

Examples

- Modify `desired_concurrent_number` to 1

```
ALTER ROUTINE LOAD FOR db1.label1
PROPERTIES
(
    "desired_concurrent_number" = "1"
);
```

- Modify `desired_concurrent_number` to 10, modify partition offsets, and modify group id

```
ALTER ROUTINE LOAD FOR db1.label1
PROPERTIES
(
    "desired_concurrent_number" = "10"
)
FROM kafka
(
    "kafka_partitions" = "0, 1, 2",
    "kafka_offsets" = "100, 200, 100",
    "property.group.id" = "new_group"
);
```

PAUSE ROUTINE LOAD

Description

This syntax is used to pause one or all Routine Load jobs. Paused jobs can be restarted using the RESUME command.

Syntax

```
PAUSE [ALL] ROUTINE LOAD FOR <job_name>
```

Required Parameters

1. `<job_name>`

> Specifies the name of the job to pause. If ALL is specified, job_name is not required.

Optional Parameters

1. `[ALL]`

> Optional parameter. If ALL is specified, it indicates pausing all routine load jobs.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
LOAD_PRIV	Table	PAUSE ROUTINE LOAD requires LOAD privilege on the table

Notes

- After a job is paused, it can be restarted using the RESUME command
- The pause operation will not affect tasks that have already been dispatched to BE, these tasks will continue to complete

Examples

- Pause a routine load job named test1.

```
PAUSE ROUTINE LOAD FOR test1;
```

- Pause all routine load jobs.

```
PAUSE ALL ROUTINE LOAD;
```

RESUME ROUTINE LOAD

Description

This syntax is used to restart one or all paused Routine Load jobs. The restarted job will continue consuming from the previously consumed offset.

Syntax

```
RESUME [ALL] ROUTINE LOAD FOR <job_name>
```

Required Parameters

1. `<job_name>`

> Specifies the name of the job to restart. If ALL is specified, job_name is not required.

Optional Parameters

1. `[ALL]`

> Optional parameter. If ALL is specified, it indicates restarting all paused routine load jobs.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
LOAD_PRIV	Table	RESUME ROUTINE LOAD requires LOAD privilege on the table

Notes

- Only jobs in PAUSED state can be restarted
- Restarted jobs will continue consuming data from the last consumed position
- If a job has been paused for too long, the restart may fail due to expired Kafka data

Examples

- Restart a routine load job named test1.

```
RESUME ROUTINE LOAD FOR test1;
```

- Restart all routine load jobs.

```
RESUME ALL ROUTINE LOAD;
```

STOP ROUTINE LOAD

Description

This syntax is used to stop a Routine Load job. Unlike the PAUSE command, stopped jobs cannot be restarted. If you need to import data again, you'll need to create a new import job.

Syntax

```
STOP ROUTINE LOAD FOR <job_name>;
```

Required Parameters

1. `<job_name>`

> Specifies the name of the job to stop. It can be in the following formats:
>
> - `<job_name>`: Stop a job with the specified name in the current database
> - `<db_name>.<job_name>`: Stop a job with the specified name in the specified database

Access Control Requirements

Users executing this SQL command must have at least the following permission:

Privilege	Object	Notes
LOAD_PRIV	Table	STOP ROUTINE LOAD requires LOAD permission on the table

Notes

- The stop operation is irreversible; stopped jobs cannot be restarted using the RESUME command
- The stop operation takes effect immediately, and running tasks will be interrupted
- It's recommended to check the job status using the SHOW ROUTINE LOAD command before stopping a job
- If you only want to temporarily pause a job, use the PAUSE command instead

Examples

- Stop a routine load job named test1

  ```
  STOP ROUTINE LOAD FOR test1;
  ```

- Stop a routine load job in a specified database

```
STOP ROUTINE LOAD FOR example_db.test1;
```

SHOW ROUTINE LOAD

Description

This statement is used to display the running status of Routine Load jobs. You can view the status information of either a specific job or all jobs.

Syntax

```
SHOW [ALL] ROUTINE LOAD [FOR <jobName>];
```

Optional Parameters

1. `[ALL]`

> Optional parameter. If specified, all jobs (including stopped or cancelled jobs) will be displayed. Otherwise, only currently running jobs will be shown.

2. `[FOR <jobName>]`

> Optional parameter. Specifies the job name to view. If not specified, all jobs under the current database will be displayed.
>
> Supports the following formats:
>
> - `<job_name>`: Shows the job with the specified name in the current database
> - `<db_name>.<job_name>`: Shows the job with the specified name in the specified database

Return Results

Field Name	Description
Id	Job ID
Name	Job name
CreateTime	Job creation time
PauseTime	Most recent job pause time
EndTime	Job end time
DbName	Corresponding database name
TableName	Corresponding table name (shows ‘multi-table’ for multiple tables)

Field Name	Description
IsMultiTbl	Whether it's a multi-table job
State	Job running status
DataSourceType	Data source type: KAFKA
CurrentTaskNum	Current number of subtasks
JobProperties	Job configuration details
DataSourceProperties	Data source configuration details
CustomProperties	Custom configurations
Statistic	Job running statistics
Progress	Job running progress
Lag	Job delay status
ReasonOfStateChanged	Reason for job state change
ErrorLogUrls	URLs to view filtered data that failed quality checks
OtherMsg	Other error messages

Access Control Requirements

Users executing this SQL command must have at least the following permission:

Privilege	Object	Notes
LOAD_PRIV	Table	SHOW ROUTINE LOAD requires LOAD permission on the table

Notes

- State descriptions:
  - NEED_SCHEDULE: Job is waiting to be scheduled
  - RUNNING: Job is running
  - PAUSED: Job is paused
  - STOPPED: Job has ended
  - CANCELLED: Job has been cancelled
- Progress description:
  - For Kafka data source, shows the consumed offset for each partition
  - For example, {"0":"2"} means the consumption progress of Kafka partition 0 is 2
- Lag description:
  - For Kafka data source, shows the consumption delay for each partition
  - For example, {"0":10} means the consumption lag of Kafka partition 0 is 10

Examples

- Show all routine load jobs (including stopped or cancelled ones) named test1

```
SHOW ALL ROUTINE LOAD FOR test1;
```

- Show currently running routine load jobs named test1

```
SHOW ROUTINE LOAD FOR test1;
```

- Show all routine load jobs (including stopped or cancelled ones) in example_db. Results can be one or multiple rows.

```
use example_db;
SHOW ALL ROUTINE LOAD;
```

- Show all currently running routine load jobs in example_db

```
use example_db;
SHOW ROUTINE LOAD;
```

- Show currently running routine load job named test1 in example_db

```
SHOW ROUTINE LOAD FOR example_db.test1;
```

- Show all routine load jobs (including stopped or cancelled ones) named test1 in example_db. Results can be one or multiple rows.

```
SHOW ALL ROUTINE LOAD FOR example_db.test1;
```

SHOW ROUTINE LOAD TASK

Description

This syntax is used to view the currently running subtasks of a specified Routine Load job.

Syntax

```
SHOW ROUTINE LOAD TASK WHERE JobName = <job_name>;
```

Required Parameters

1. `<job_name>`

> The name of the routine load job to view.

Return Results

The return results include the following fields:

Field Name	Description
TaskId	Unique ID of the subtask
TxnId	Import transaction ID corresponding to the subtask
TxnStatus	Import transaction status of the subtask. Null indicates the subtask has not yet been scheduled
JobId	Job ID corresponding to the subtask
CreateTime	Creation time of the subtask
ExecuteStartTime	Time when the subtask was scheduled for execution, typically later than creation time
Timeout	Subtask timeout, typically twice the `max_batch_interval` set in the job
BeId	BE node ID executing this subtask
DataSourceProperties	Starting offset of Kafka Partition that the subtask is preparing to consume. It's a Json format string. Key is Partition Id, Value is the starting offset for consumption

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
LOAD_PRIV	Table	SHOW ROUTINE LOAD TASK requires LOAD privilege on the table

Notes

- A null TxnStatus doesn't indicate task error, it may mean the task hasn't been scheduled yet
- The offset information in DataSourceProperties can be used to track data consumption progress
- When Timeout is reached, the task will automatically end regardless of whether data consumption is complete

Examples

- Show subtask information for a routine load task named test1.

```
SHOW ROUTINE LOAD TASK WHERE JobName = "test1";
```

SHOW CREATE ROUTINE LOAD

Description

This statement is used to display the creation statement of a routine load job.

The result shows the current consuming Kafka partitions and their corresponding offsets to be consumed. The result may not be a real-time consumption point, and should be based on the result of show routine load.

Syntax

```
SHOW [ALL] CREATE ROUTINE LOAD for <load_name>;
```

Required Parameters

1. `<load_name>`

> The name of the routine load job

Optional Parameters

1. `[ALL]`

> Optional parameter that represents retrieving all jobs, including historical jobs

Access Control Requirements

Users executing this SQL command must have at least the following permission:

Privilege	Object	Notes
LOAD_PRIV	Table	SHOW CREATE ROUTINE LOAD requires LOAD permission on the table

Examples

- Show the creation statement of a specified routine load job in the default database

  ```
  SHOW CREATE ROUTINE LOAD for test_load
  ```

CREATE SYNC JOB

Description

The data synchronization (Sync Job) function allows users to submit a persistent data synchronization job. It incrementally synchronizes the CDC (Change Data Capture) of data update operations from a MySQL database by reading the Binlog from a specified remote source. Currently, the synchronization job supports connecting to Canal, obtaining parsed Binlog data from the Canal server, and importing it into Doris.

Users can view the status of synchronization jobs via SHOW SYNC JOB.

Syntax

```
CREATE SYNC [<db>.]<job_name>
(<channel_desc> [, ... ])
<binlog_desc>
```

where:

```
channel_desc
  : FROM <mysql_db>.<src_tbl> INTO <des_tbl> [ <columns_mapping> ]
```

```
binlog_desc
  : FROM BINLOG ("<key>" = "<value>" [, ... ])
```

Required Parameters

1. `<job_name>`

> Specifies the unique name of the synchronization job within the current database. Only one job with the same `<job_name>` can be running at a time.

2. `<channel_desc>`

> Describes the mapping relationship between the MySQL source table and the Doris target table.
>
> - `<mysql_db.src_tbl>`: Specifies the source table in MySQL (including the database name).
> - `<des_tbl>`: Specifies the target table in Doris. The target table must be unique, and its batch delete function must be enabled.
> - `<columns_mapping>` (Optional): Defines the mapping between columns of the source and target tables. If omitted, columns are mapped one-to-one in order. Note that the form `col_name = expr` is not supported.

3. `<binlog_desc>`

> Describes the remote data source for the Binlog.
>
> The properties for the Canal data source (keys prefixed with `canal.`) include:
>
> - `canal.server.ip`: Address of the Canal server.
> - `canal.server.port`: Port of the Canal server.
> - `canal.destination`: Identifier of the Canal instance.
> - `canal.batchSize`: Maximum batch size to fetch (default is 8192).
> - `canal.username`: Username for the Canal instance.
> - `canal.password`: Password for the Canal instance.
> - `canal.debug` (Optional): If set to true, prints detailed batch and row information.

Usage Notes

- Currently, the synchronization job only supports connecting to a Canal server.
- Only one synchronization job with the same `<job_name>` can run concurrently within a database.
- The target table specified in `<channel_desc>` must have its batch delete function enabled.

Access Control Requirements

Users executing this SQL command must have at least the following privileges: | Privilege | Object | Notes | | : ——————- | :———- | :——————- | | LOAD_PRIV | Table | This operation can only be performed by users or roles who have the LOAD_PRIV privilege for the imported table. |

Examples

1. **Create a simple synchronization job**

   Create a synchronization job named `job1` in the `test_db` database that maps the MySQL source table `mysql_db1.tbl1` to the Doris target table `test_tbl`, connecting to a local Canal server.

   ```
   CREATE SYNC `test_db`.`job1`
   (
     FROM `mysql_db1`.`tbl1` INTO `test_tbl`
   )
   FROM BINLOG
   (
     "type" = "canal",
     "canal.server.ip" = "127.0.0.1",
     "canal.server.port" = "11111",
     "canal.destination" = "example",
     "canal.username" = "",
     "canal.password" = ""
   );
   ```

2. **Create a synchronization job with multiple channels and explicit column mapping**

Create a synchronization job named `job1` in the `test_db` database for multiple MySQL source tables with one-to-one mapping and explicitly specified column orders.

```
CREATE SYNC `test_db`.`job1`
(
  FROM `mysql_db`.`t1` INTO `test1` (k1, k2, v1),
  FROM `mysql_db`.`t2` INTO `test2` (k3, k4, v2)
)
FROM BINLOG
(
  "type" = "canal",
  "canal.server.ip" = "xx.xxx.xxx.xx",
  "canal.server.port" = "12111",
  "canal.destination" = "example",
  "canal.username" = "username",
  "canal.password" = "password"
);
```

PAUSE SYNC JOB

Description

Pause a running resident data synchronization job in a database identified by `job_name`. The suspended job will stop synchronizing data while retaining its latest consumption position until it is resumed by the user.

Syntax

```
PAUSE SYNC JOB [<db>.]<job_name>
```

Required Parameters

1. `<job_name>`

> Specifies the name of the synchronization job to be paused.

Optional Parameters

1. `<db>` > If a database is specified using the `[<db>.]` prefix, the job is located in that database; otherwise, the current database is used.

Access Control Requirements

Any user or role can perform this operation.

Example

1. Pause the data synchronization job named `job_name`.

```
PAUSE SYNC JOB `job_name`;
```

RESUME SYNC JOB

Description

Resume a resident data synchronization job that has been suspended in a database by its `job_name`. Once resumed, the job continues to synchronize data starting from the latest position before the suspension.

Syntax

```
RESUME SYNC JOB [<db>.]<job_name>
```

Required Parameters

1. `<job_name>`

> Specifies the name of the data synchronization job to be resumed.

Optional Parameters

1. `<db>` > If a database is specified using the `[<db>.]` prefix, the job is located in that database; otherwise, the current database is used.

Access Control Requirements

Any user or role can perform this operation.

Examples

1. Resume the data synchronization job named `job_name`.

```
RESUME SYNC JOB `job_name`;
```

STOP SYNC JOB

Description

Stop a running resident data synchronization job in a database by specifying its `job_name`. Once stopped, the job will cease synchronizing data and release its occupied resources.

Syntax

```
STOP SYNC JOB [<db>.]<job_name>
```

Required Parameters

1. `<job_name>`

> Specifies the name of the data synchronization job to be stopped.

Optional Parameters

1. `<db>` > If a database is specified using the `[<db>.]` prefix, the job is located in that database; otherwise, the current database is used.

Access Control Requirements

Any user or role can perform this operation.

Example

1. Stop the data synchronization job named `job_name`.

```
STOP SYNC JOB `job_name`;
```

SHOW SYNC JOB

Description

This statement displays the status of resident data synchronization jobs in all databases.

Syntax

```
SHOW SYNC JOB [FROM <db_name>]
```

Optional Parameters

1. <db_name> > <db_name>represents the database name, which is used to specify the database from which the information of the synchronization jobs is to be shown.

Access Control Requirements

Users executing this SQL command must have at least one of the following privileges:

Privilege	Object	Notes
ADMIN_PRIV, SELECT_PRIV, LOAD_PRIV, ALTER_PRIV, CREATE_PRIV, DROP_PRIV, SHOW_VIEW_PRIV	Database db_name	This operation requires at least one of the listed privileges on the target database.

Examples

1. Display the status of all data synchronization jobs in the current database.

   ```
   SHOW SYNC JOB;
   ```

2. Display the status of all data synchronization jobs in the test_db database.

   ```
   SHOW SYNC JOB FROM `test_db`;
   ```

SYNC

Description

This statement is used to synchronize metadata for non-master Frontend (FE) nodes. In Apache Doris, only the master FE node can write metadata, while other FE nodes forward metadata write operations to the master. After the master completes the metadata writing operation, non-master nodes may experience a short delay in replaying the metadata. You can use this statement to force synchronization of metadata.

Syntax

```
SYNC;
```

Access Control Requirements

Any user or role can perform this operation.

Examples

Synchronize metadata:

````
```sql
SYNC;
```
````

EXPORT | Load And Export

EXPORT

Description

The `EXPORT` command is used to export data from a specified table to files at a specified location. Currently supports exporting to remote storage such as HDFS, S3, BOS, COS (Tencent Cloud) through Broker processes, S3 protocol, or HDFS protocol.

`EXPORT` is an asynchronous operation. This command submits an `EXPORT JOB` to Doris and returns immediately upon successful submission. You can use the SHOW EXPORT command to check the progress after execution.

Syntax:

```
EXPORT TABLE <table_name>
[ PARTITION ( <partation_name> [ , ... ] ) ]
[ <where_clause> ]
TO <export_path>
[ <properties> ]
WITH <target_storage>
[ <broker_properties> ];
```

Required Parameters

1. `<table_name>`

The name of the table to be exported. Supports exporting data from Doris local tables, views, and catalog external tables.

2. `<export_path>`

The export file path. Can be a directory or a file directory with file prefix, such as `hdfs://path/to/my_file_`

Optional Parameters

1. `<where_clause>`

You can specify filter conditions for the exported data.

2. `<partation_name>`

You can export only certain specified partitions of the specified table. Only valid for Doris local tables.

3. `<properties>`

Used to specify some export parameters.

```
[ PROPERTIES ("<key>"="<value>" [, ... ]) ]
```

The following parameters can be specified:
- `label`: Optional parameter to specify the Label for this Export task. When not specified, the system will randomly generate a Label.

- `column_separator`: Specifies the column separator for export, default is `\t`, supports multi-byte. This parameter is only used for CSV file format.
- `line_delimiter`: Specifies the line delimiter for export, default is `\n`, supports multi-byte. This parameter is only used for CSV file format.
- `columns`: Specifies certain columns of the export table.
- `format`: Specifies the file format for the export job, supports: parquet, orc, csv, csv_with_names, csv_with_names_and_types. Default is CSV format.
- `max_file_size`: Single file size limit for export job. If the result exceeds this value, it will be split into multiple files. `max_file_size` value range is [5MB, 2GB], default is 1GB. (When specifying export to orc file format, the actual split file size will be a multiple of 64MB, e.g.: specifying max_file_size = 5MB will actually split at 64MB; specifying max_file_size = 65MB will actually split at 128MB)
- `parallelism`: Concurrency of the export job, default is `1`. The export job will start `parallelism` number of threads to execute `select into outfile` statements. (If the number of Parallelism is greater than the number of Tablets in the table, the system will automatically set Parallelism to the size of the number of Tablets, i.e., each `select into outfile` statement is responsible for one Tablet)
- `delete_existing_files`: Default is `false`. If specified as `true`, all files in the directory specified by `export_path` will be deleted first, then data will be exported to that directory. For example: “export_path” = “/user/tmp”, will delete all files and directories under “/user/”; “file_path” = “/user/tmp/”, will delete all files and directories under “/user/tmp/”.
- `with_bom`: Default is `false`. If specified as `true`, the exported file encoding will be UTF8 encoding with BOM (only effective for csv-related file formats).
- `data_consistency`: Can be set to `none` / `partition`, default is `partition`. Indicates at what granularity to split the export table, `none` represents Tablets level, `partition` represents Partition level.
- `timeout`: Timeout for export job, default is 2 hours, unit is seconds.
- `compress_type`: (Supported since 2.1.5) When specifying the export file format as Parquet / ORC files, you can specify the compression method used by Parquet / ORC files. Parquet file format can specify compression methods as SNAPPY, GZIP, BROTLI, ZSTD, LZ4, and PLAIN, with default value SNAPPY. ORC file format can specify compression methods as PLAIN, SNAPPY, ZLIB, and ZSTD, with default value ZLIB. This parameter is supported starting from version 2.1.5. (PLAIN means no compression)

:::caution Note
To use the delete_existing_files parameter, you also need to add the configuration `enable_delete_existing_files = true` in fe.conf and restart fe, then delete_existing_files will take effect. delete_existing_files = true is a dangerous operation, it’s recommended to use only in test environments.
:::

4. `<target_storage>`
Storage medium, optional BROKER, S3, HDFS.

5. `<broker_properties>`
Different properties need to be specified according to different storage media of `<target_storage>`.

- **BROKER**
  Data can be written to remote storage through Broker processes. Here you need to define relevant connection information for Broker to use.

  ```
  WITH BROKER "broker_name"
  ("<key>"="<value>" [,...])
  ```

  Broker related properties:

  - `username`: Username
  - `password`: Password
  - `hadoop.security.authentication`: Specify authentication method as kerberos
  - `kerberos_principal`: Specify kerberos principal
  - `kerberos_keytab`: Specify the path to kerberos keytab file. This file must be an absolute path to a file on the server where the Broker process is located and accessible by the Broker process

- **HDFS**

  Data can be written directly to remote HDFS.

  ```
  WITH HDFS ("<key>"="<value>" [,...])
  ```

  HDFS related properties:

  - `fs.defaultFS`: namenode address and port
  - `hadoop.username`: HDFS username
  - `dfs.nameservices`: name service name, consistent with hdfs-site.xml
  - `dfs.ha.namenodes.[nameservice ID]`: list of namenode ids, consistent with hdfs-site.xml
  - `dfs.namenode.rpc-address.[nameservice ID].[name node ID]`: rpc address of Name node, same number as namenode count, consistent with hdfs-site.xml

  For Hadoop clusters with kerberos authentication enabled, the following additional PROPERTIES attributes need to be set:

  - `dfs.namenode.kerberos.principal`: Principal name of HDFS namenode service
  - `hadoop.security.authentication`: Set authentication method to kerberos
  - `hadoop.kerberos.principal`: Set the Kerberos principal used when Doris connects to HDFS
  - `hadoop.kerberos.keytab`: Set the local file path of keytab

- **S3**

  Data can be written directly to remote S3 object storage.

  ```
  WITH S3 ("<key>"="<value>" [,...])
  ```

S3 related properties:

- `s3.endpoint`
- `s3.region`
- `s3.secret_key`
- `s3.access_key`
- `use_path_style`: (Optional) Default is `false`. S3 SDK uses Virtual-hosted Style by default. However, some object storage systems may not have enabled or support Virtual-hosted Style access. In this case, you can add the `use_path_style` parameter to force the use of Path Style access.

Return Values

Column Name	Type	Description
jobId	long	Unique identifier of the export job.
label	string	Label of the export job.
dbId	long	Identifier of the database.
tableId	long	Identifier of the table.
state	string	Current state of the job.
path	string	Path of the export files.
partitions	string	List of exported partition names, multiple partition names separated by commas.
progress	int	Current progress of the export job (percentage).
createTimeMs	string	Millisecond value of job creation time, formatted as date time.
exportStartTimeMs	string	Millisecond value of export job start time, formatted as date time.
exportFinishTimeMs	string	Millisecond value of export job end time, formatted as date time.
failMsg	string	Error message when export job fails.

Access Control

Users executing this SQL command must have at least the following permissions:

Permission	Object	Description
SELECT_PRIV	Database	Requires read permission on database and table.

Notes

Concurrent Execution

An Export job can set the `parallelism` parameter to export data concurrently. The `parallelism` parameter actually specifies the number of threads executing the EXPORT job. When `"data_consistency" = "none"` is set, each thread will be responsible for exporting part of the table's Tablets.

The underlying execution logic of an Export job is actually `SELECT INTO OUTFILE` statements. Each thread set by the `parallelism` parameter will execute independent `SELECT INTO OUTFILE` statements.

The specific logic for splitting Export jobs into multiple `SELECT INTO OUTFILE` is: evenly distribute all tablets of the table to all parallel threads, for example: - num(tablets) = 40, parallelism = 3, then these 3 threads are responsible for 14, 13, 13 tablets respectively. - num(tablets) = 2, parallelism = 3, then Doris will automatically set parallelism to 2, with each thread responsible for one tablet.

When the tablets a thread is responsible for exceeds the `maximum_tablets_of_outfile_in_export` value (default is 10, can be modified by adding `maximum_tablets_of_outfile_in_export` parameter in fe.conf), that thread will be split into multiple `SELECT INTO OUTFILE` statements, for example: - A thread is responsible for 14 tablets, `maximum_tablets_of_outfile_in_export = 10`, then this thread is responsible for two `SELECT INTO OUTFILE` statements. The first `SELECT INTO OUTFILE` statement exports 10 tablets, the second `SELECT INTO OUTFILE` statement exports 4 tablets. The two `SELECT INTO OUTFILE` statements are executed serially by this thread.

When the amount of data to be exported is very large, you can consider appropriately increasing the `parallelism` parameter to increase concurrent export. If machine cores are tight and cannot increase `parallelism` while the export table has many Tablets, you can consider increasing `maximum_tablets_of_outfile_in_export` to increase the number of tablets a `SELECT INTO OUTFILE` statement is responsible for, which can also speed up export.

If you want to export Table at Partition granularity, you can set Export property `"data_consistency" = "partition"`. In this case, concurrent threads of Export tasks will be divided into multiple Outfile statements at Partition granularity. Different Outfile statements export different Partitions, while data exported by the same Outfile statement must belong to the same Partition. For example: after setting `"data_consistency" = "partition"`

- num(partition) = 40, parallelism = 3, then these 3 threads are responsible for 14, 13, 13 Partitions respectively.
- num(partition) = 2, parallelism = 3, then Doris will automatically set Parallelism to 2, with each thread responsible for one Partition.

Memory Limits

Usually an Export job's query plan only has `scan-export` two parts, not involving computation logic that requires too much memory. So usually the default memory limit of 2GB can meet the requirements.

But in some scenarios, such as when a query plan needs to scan too many Tablets on the same BE, or when Tablet data versions are too many, it may cause insufficient memory. You can adjust the Session variable `exec_mem_limit` to increase the memory usage limit.

Other Matters

- It's not recommended to export large amounts of data at once. The recommended maximum export data volume for one Export job is tens of GB. Too large exports will cause more garbage files and higher retry costs. If table data volume is too large, it's recommended to export by partition.
- If Export job fails, already generated files will not be deleted and need to be manually deleted by users.
- Export jobs will scan data and occupy IO resources, which may affect system query latency.

- Currently during Export, only simple checks are performed on whether Tablets versions are consistent. It's recommended not to perform data import operations on the table during Export execution.
- An Export Job allows exporting a maximum of 2000 partitions. You can add parameter `maximum_number_of_export_partitions` in `fe.conf` and restart FE to modify this setting.

Examples

Export Data to Local

> To export data to local file system, you need to add `enable_outfile_to_local=true` in `fe.conf` and restart FE.

- Export all data in Test table to local storage, default export CSV format files

```
EXPORT TABLE test TO "file:///home/user/tmp/";
```

- Export k1,k2 columns in Test table to local storage, default export CSV file format, and set Label

```
EXPORT TABLE test TO "file:///home/user/tmp/"
PROPERTIES (
  "label" = "label1",
  "columns" = "k1,k2"
);
```

- Export rows where `k1 < 50` in Test table to local storage, default export CSV format files, and use `,` as column separator

```
EXPORT TABLE test WHERE k1 < 50 TO "file:///home/user/tmp/"
PROPERTIES (
  "columns" = "k1,k2",
  "column_separator"=","
);
```

- Export partitions p1,p2 in Test table to local storage, default export csv format files

```
EXPORT TABLE test PARTITION (p1,p2) TO "file:///home/user/tmp/"
PROPERTIES ("columns" = "k1,k2");
```

- Export all data in Test table to local storage, export other format files

```
-- parquet
EXPORT TABLE test TO "file:///home/user/tmp/"
PROPERTIES (
  "columns" = "k1,k2",
  "format" = "parquet"
);

-- orc
EXPORT TABLE test TO "file:///home/user/tmp/"
PROPERTIES (
```

```
  "columns" = "k1,k2",
  "format" = "orc"
);

-- csv(csv_with_names) , Use 'AA' as the column separator and 'zz' as the row separator
EXPORT TABLE test TO "file:///home/user/tmp/"
PROPERTIES (
  "format" = "csv_with_names",
  "column_separator"="AA",
  "line_delimiter" = "zz"
);

-- csv(csv_with_names_and_types)
EXPORT TABLE test TO "file:///home/user/tmp/"
PROPERTIES (
  "format" = "csv_with_names_and_types"
);
```

- Set `max_file_sizes` property
  When exported file is larger than 5MB, data will be split into multiple files, each file maximum 5MB.

```
-- When the exported file is larger than 5MB, the data will be split into multiple files, with each file having a maximum size of
5MB.
EXPORT TABLE test TO "file:///home/user/tmp/"
PROPERTIES (
  "format" = "parquet",
  "max_file_size" = "5MB"
);
```

- Set `parallelism` property

```
EXPORT TABLE test TO "file:///home/user/tmp/"
PROPERTIES (
  "format" = "parquet",
  "max_file_size" = "5MB",
  "parallelism" = "5"
);
```

- Set `delete_existing_files` property
  When Export exports data, it will first delete all files and directories under `/home/user/` directory, then export data to this directory.

```
-- When exporting data, all files and directories under the `/home/user/` directory will be deleted first, and then the data will be
exported to this directory.
EXPORT TABLE test TO "file:///home/user/tmp"
PROPERTIES (
  "format" = "parquet",
  "max_file_size" = "5MB",
  "delete_existing_files" = "true"
);
```

Export to S3

- Export all data in s3_test table to S3, using invisible character `\x07` as column or row separator. If you need to export data to minio, you also need to specify `use_path_style=true`.

```
EXPORT TABLE s3_test TO "s3://bucket/a/b/c"
PROPERTIES (
  "column_separator"="\\x07",
  "line_delimiter" = "\\x07"
) WITH S3 (
  "s3.endpoint" = "xxxxx",
  "s3.region" = "xxxxx",
  "s3.secret_key"="xxxx",
  "s3.access_key" = "xxxxx"
)
```

Export to HDFS

- Export all data in Test table to HDFS, export file format is Parquet, export job single file size limit is 512MB, keep all files in the specified directory.

```
EXPORT TABLE test TO "hdfs://hdfs_host:port/a/b/c/"
PROPERTIES(
    "format" = "parquet",
    "max_file_size" = "512MB",
    "delete_existing_files" = "false"
)
with HDFS (
"fs.defaultFS"="hdfs://hdfs_host:port",
"hadoop.username" = "hadoop"
);
```

Export through Broker Node

Need to start Broker process first and add this Broker in FE. - Export all data in Test table to HDFS

```
EXPORT TABLE test TO "hdfs://hdfs_host:port/a/b/c"
WITH BROKER "broker_name"
(
  "username"="xxx",
  "password"="yyy"
);
```

- Export partitions p1,p2 in testTbl table to HDFS, using “,” as column separator, and specify Label

```
EXPORT TABLE testTbl PARTITION (p1,p2) TO "hdfs://hdfs_host:port/a/b/c"
PROPERTIES (
  "label" = "mylabel",
  "column_separator"=","
)
WITH BROKER "broker_name"
(
  "username"="xxx",
  "password"="yyy"
);
```

- Export all data in testTbl table to HDFS, using invisible character `\x07` as column or row separator.

```
EXPORT TABLE testTbl TO "hdfs://hdfs_host:port/a/b/c"
PROPERTIES (
```

```
  "column_separator"="\\x07",
  "line_delimiter" = "\\x07"
)
WITH BROKER "broker_name"
(
  "username"="xxx",
  "password"="yyy"
)
```

CANCEL EXPORT

Description

This statement is used to undo an export job for the specified label. Or batch undo export jobs via fuzzy matching.

Syntax

```
CANCEL EXPORT
[ FROM <db_name> ]
WHERE [ LABEL = "<export_label>" | LABEL like "<label_pattern>" | STATE = "<state>" ]
```

Optional Parameters

1. `<db_name>`

The name of the database to which the exported data task belongs. If omitted, the default is the current database.

2. `<export_label>`

Each import needs to be assigned a unique Label. Stopping this task requires specifying the label.

3. `<label_pattern>`

A label expression for fuzzy matching. If you want to undo multiple EXPORT jobs, you can use `LIKE` for fuzzy matching.

4. `<state>`

state options: `PENDING`,`IN_QUEUE`,`EXPORTING`.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Database	Requires modification access to the database.

Usage Notes

1. Only pending export jobs in PENDING, IN_QUEUE,EXPORTING state can be canceled.
2. When performing batch undo, Doris does not guarantee the atomic undo of all corresponding export jobs. That is, it is possible that only some of the export jobs were successfully undone. The user can

view the job status through the SHOW EXPORT statement and try to execute the CANCEL EXPORT statement repeatedly.

3. When the job of the `EXPORTING` state is revoked, part of the data may have been exported to the storage system, and the user needs to process (delete) this section to export data.

Examples

- Cancel the export job whose label is `example_db_test_export_label` on the database example_db.

```
CANCEL EXPORT
FROM example_db
WHERE LABEL = "example_db_test_export_label" and STATE = "EXPORTING";
```

- Cancel all export jobs containing example* on the database example*db.

```
CANCEL EXPORT
FROM example_db
WHERE LABEL like "%example%";
```

- Cancel all export jobs which state are “PENDING”

```
CANCEL EXPORT
FROM example_db
WHERE STATE = "PENDING";
```

SHOW EXPORT

Description

This statement is used to display the execution status of a specified export job.

Syntax

```
SHOW EXPORT
[ FROM <db_name> ]
  [
    WHERE
      [ ID = <job_id> ]
      [ STATE = { "PENDING" | "EXPORTING" | "FINISHED" | "CANCELLED" } ]
      [ LABEL = <label> ]
   ]
[ ORDER BY <column_name> [ ASC | DESC ] [, column_name [ ASC | DESC ] ... ] ]
[ LIMIT <limit> ];
```

Optional Parameters

1. `<db_name>`: Optional parameter. If not specified, the current default database will be used.

2. `<job_id>`: Optional parameter. Used to specify the export job ID to display.

3. `<label>`: Optional parameter. Used to specify the label of the export job to display.

4. `<column_name>`: Optional parameter. Used to specify the column name for sorting.

5. `<limit>`: Optional parameter. If specified, only the specified number of matching records will be displayed; if not specified, all records will be displayed.

Return Value

Column Name	Column Type	Note
JobLabel	String	Unique ID of the job

Column Name	Data Type	Note
Label	String	The label of the export job. If not specified, the system will generate one by default.
State	String	Job status: - `PENDING`: Job waiting for scheduling - `EXPORTING`: Data exporting - `FINISHED`: Job successful - `CANCELLED`: Job failed
Progress	String	Job progress. This progress is measured in query plan units. For example, if there are 10 threads and 3 are completed, the progress is 30%.
JobInfo	String	Job information displayed in JSON format: - db: Database name - tbl: Table name - partitions: Specified partitions for export, `empty` list means all partitions - column_separator: Column delimiter for the exported file - line_delimiter: Line delimiter for the exported file - tablet num: Total number of involved tablets - broker: Name of the broker used - coord num: Number of query plans - max_file_size: Maximum size of an exported file - delete_existing_files: Whether to delete existing files and directories in the export directory - columns: Columns to export, empty value means export all columns - format: File format of the export
Path	String	Export path on remote storage

Column Name	Data Type	Note
CreationTime	Job creation time	
StartTime	Job start time	
FinishTime	Job finish time	
TimeOut	Job timeout (in seconds). The time is calculated from CreateTime.	
ErrorMsg	If the job encounters an error, the error reason will be displayed here.	

Column Name	Data Type	Note
Output info	String	If the export job is successful, the specific `SELECT INTO OUTFILE` result information will be displayed here.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
SELECT_PRIV	Database (Database)	Requires read access to the database and table.

Examples

- Display all export jobs for the default db

```
SHOW EXPORT;
```

- Display export jobs for a specified db, ordered by StartTime in descending order

```
 SHOW EXPORT FROM example_db ORDER BY StartTime DESC;
```

- Display export jobs for a specified db where the state is “exporting”, ordered by StartTime in descending order

```
SHOW EXPORT FROM example_db WHERE STATE = "exporting" ORDER BY StartTime DESC;
```

- Display export job for a specified db and job_id

```
  SHOW EXPORT FROM example_db WHERE ID = job_id;
```

- Display export job for a specified db and label

```
SHOW EXPORT FROM example_db WHERE LABEL = "mylabel";
```

OUTFILE

Description

This statement is used to export query results to a file using the `SELECT INTO OUTFILE` command. Currently, it supports exporting to remote storage, such as HDFS, S3, BOS, COS (Tencent Cloud), through the Broker process, S3 protocol, or HDFS protocol.

Syntax

```
<query_stmt>
INTO OUTFILE "<file_path>"
[ FORMAT AS <format_as> ]
[ <properties> ]
```

Required Parameters

1. `<query_stmt>`

The query statement must be a valid SQL statement. Please refer to the query statement documentation.

2. `<file_path>`

file_path points to the path where the file is stored and the file prefix. Such as `hdfs://path/to/my_file_`.

The final filename will consist of `my_file_`, the file number and the file format suffix. The file serial number starts from 0, and the number is the number of files to be divided. Such as:
- my_file_abcdefg_0.csv - my_file_abcdefg_1.csv - my_file_abcdegf_2.csv

You can also omit the file prefix and specify only the file directory, such as: `hdfs://path/to/`

Optional Parameters

1. `<format_as>`

Specifies the export format. Supported formats include :
- `CSV` (Default) - `PARQUET` - `CSV_WITH_NAMES` - `CSV_WITH_NAMES_AND_TYPES` - `ORC`

> Note: PARQUET, CSV_WITH_NAMES, CSV_WITH_NAMES_AND_TYPES, and ORC are supported starting in version 1.2 .

2. `<properties>`

```
[ PROPERTIES ("<key>"="<value>" [, ... ]) ]
```

Specify related properties. Currently exporting via the Broker process, S3 protocol, or HDFS protocol is supported.

File properties - `column_separator`: column separator,is only for CSV format. mulit-bytes is supported starting in version 1.2, such as: “\x01”, “abc”. - `line_delimiter`: line delimiter,is only for CSV format. mulit-bytes supported starting in version 1.2, such as: “\x01”, “abc”. - `max_file_size`: the size limit of a single file, if the result exceeds this value, it will be cut into multiple files, the value range of max_file_size is [5MB, 2GB] and the default is 1GB. (When specified that the file format is ORC, the size of the actual division file will be a multiples of 64MB, such as: specify max_file_size = 5MB, and actually use 64MB as the division; specify max_file_size = 65MB, and will actually use 128MB as cut division points.) - `delete_existing_files`: default `false`. If it is specified as true, you will first delete all files specified in the directory specified by the file_path, and then export the data to the directory.For example: “file_path” = “/user/tmp”, then delete all files and directory under “/user/”; “file_path” = “/user/tmp/”, then delete all files and directory under “/user/tmp/” - `file_suffix`: Specify the suffix of the export file. If this parameter is not specified, the default suffix for the file format will be used.

Broker properties *(need to be prefixed with* `broker`*)* - `broker.name: broker`: broker name - `broker.hadoop.security.authentication`: specify the authentication method as kerberos - `broker.kerberos_principal`: specifies the principal of kerberos - `broker.kerberos_keytab`: specifies the path to the keytab file of kerberos. The file must be the absolute path to the file on the server where the broker process is located. and can be accessed by the Broker process

HDFS properties - `fs.defaultFS`: namenode address and port - `hadoop.username`: hdfs username - `dfs.nameservices`: if hadoop enable HA, please set fs nameservice. See hdfs-site.xml - `dfs.ha.namenodes.[nameservice ID]`: unique identifiers for each NameNode in the nameservice. See hdfs-site.xml - `dfs.namenode.rpc-address.[nameservice ID].[name node ID]`: the fully-qualified RPC address for each NameNode to listen on. See hdfs-site.xml - `dfs.client.failover.proxy.provider.[nameservice ID]`: the Java class that HDFS clients use to contact the Active NameNode, usually it is org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider

For a kerberos-authentication enabled Hadoop cluster, additional properties need to be set: - `dfs.namenode.kerberos.principal`: HDFS namenode service principal - `hadoop.security.authentication`: kerberos - `hadoop.kerberos.principal`: the Kerberos pincipal that Doris will use when connectiong to HDFS. - `hadoop.kerberos.keytab`: HDFS client keytab location.

For the S3 protocol, you can directly execute the S3 protocol configuration: - `s3.endpoint` - `s3.access_key` - `s3.secret_key` - `s3.region` - `use_path_style`: (optional) default false . The S3 SDK uses the virtual-hosted style by default. However, some object storage systems may not be enabled or support virtual-hosted style access. At this time, we can add the use_path_style parameter to force the use of path style access method.

> Note that to use the `delete_existing_files` parameter, you also need to add the configuration `enable_delete_existing_files = true` to the fe.conf file and restart the FE. Only then will the `delete_existing_files` parameter take effect. Setting `delete_existing_files = true` is a dangerous operation and it is recommended to only use it in a testing environment.

Return Value

The results returned by the `Outfile` statement are explained as follows:

Column	DataType	Note
FileNumber	int	The total number of files generated.
TotalRows	int	The number of rows in the result set.
FileSize	int	The total size of the exported files, in bytes.
URL	string	The prefix of the exported file paths. Multiple files are numbered sequentially with suffixes like `_0`, `_1`.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
SELECT_PRIV	Database	Requires read access to the database and table.

Usage Notes

DataType Mapping

- All file formats support the export of basic data types, while only csv/orc/csv_with_names/csv_with_names_and_types currently support the export of complex data types (ARRAY/MAP/STRUCT). Nested complex data types are not supported.
- Parquet and ORC file formats have their own data types. The export function of Doris can automatically export the Doris data types to the corresponding data types of the Parquet/ORC file format. The following are the data type mapping relationship of the Doris data types and the Parquet/ORC file format data types:

1. The mapping relationship between the Doris data types to the ORC data types is: | Doris Type | Orc Type | |————————-|————| | boolean | boolean | | tinyint | tinyint | | smallint | smallint | | int | int | | bigint | bigint | | largeInt | string | | date | string | | datev2 | string | | datetime | string | | datetimev2 | timestamp | | float | float | | double | double | | char / varchar / string | string | | decimal | decimal | | struct | struct | | map | map | | array | array |
2. When Doris exports data to the Parquet file format, the Doris memory data will be converted to Arrow memory data format first, and then the paraquet file format is written by Arrow. The mapping relationship between the Doris data types to the ARROW data types is: | Doris Type | Arrow Type | | ————————-|————| | boolean | boolean | | tinyint | int8 | | smallint | int16 | | int | int32 | | bigint | int64 | | largeInt | utf8 | | date | utf8 | | datev2 | utf8 | | datetime | utf8 | | datetimev2 | utf8 | | float | float32 | | double | float64 | | char / varchar / string | utf8 | | decimal | decimal128 | | struct | struct | | map | map | | array | list |

Export data volume and export efficiency

This function essentially executes an SQL query command. The final result is a single-threaded output. Therefore, the time-consuming of the entire export includes the time-consuming of the query itself and the time-consuming of writing the final result set. If the query is large, you need to set the session variable `query_timeout` to appropriately extend the query timeout.

Management of export files

Doris does not manage exported files. Including the successful export, or the remaining files after the export fails, all need to be handled by the user.

Export to local file

To export to a local file, you need configure `enable_outfile_to_local=true` in fe.conf.

```
select * from tbl1 limit 10
INTO OUTFILE "file:///home/work/path/result_";
```

The ability to export to a local file is not available for public cloud users, only for private deployments. And the default user has full control over the cluster nodes. Doris will not check the validity of the export path filled in by the user. If the process user of Doris does not have write permission to the path, or the path does not exist, an error will be reported. At the same time, for security reasons, if a file with the same name already exists in this path, the export will also fail.

Doris does not manage files exported locally, nor does it check disk space, etc. These files need to be managed by the user, such as cleaning and so on.

Results Integrity Guarantee

This command is a synchronous command, so it is possible that the task connection is disconnected during the execution process, so that it is impossible to live the exported data whether it ends normally, or whether it is complete. At this point, you can use the `success_file_name` parameter to request that a successful file identifier be generated in the directory after the task is successful. Users can use this file to determine whether the export ends normally.

Concurrent Export

Setting the session variable `set enable_parallel_outfile = true;` enables concurrent export using outfile. For detailed usage, see Export Query Result.

Examples

- Use the broker method to export, and export the simple query results to the file `hdfs://path/to/result.txt`. Specifies that the export format is CSV. Use `my_broker` and set kerberos authentication information. Specify the column separator as `,` and the row separator as `\n`.

  ```
  SELECT * FROM tbl
  INTO OUTFILE "hdfs://path/to/result_"
  FORMAT AS CSV
  PROPERTIES
  (
      "broker.name" = "my_broker",
      "broker.hadoop.security.authentication" = "kerberos",
      "broker.kerberos_principal" = "doris@YOUR.COM",
      "broker.kerberos_keytab" = "/home/doris/my.keytab",
      "column_separator" = ",",
      "line_delimiter" = "\n",
  ```

```
    "max_file_size" = "100MB"
);
```

If the final generated file is not larger than 100MB, it will be: `result_0.csv`. If larger than 100MB, it may be `result_0.csv, result_1.csv, ....`

- Export the simple query results to the file `hdfs://path/to/result.parquet`. Specify the export format as PARQUET. Use `my_broker` and set kerberos authentication information.

```
SELECT c1, c2, c3 FROM tbl
INTO OUTFILE "hdfs://path/to/result_"
FORMAT AS PARQUET
PROPERTIES
(
    "broker.name" = "my_broker",
    "broker.hadoop.security.authentication" = "kerberos",
    "broker.kerberos_principal" = "doris@YOUR.COM",
    "broker.kerberos_keytab" = "/home/doris/my.keytab"
);
```

- Export the query result of the CTE statement to the file `hdfs://path/to/result.txt`. The default export format is CSV. Use `my_broker` and set hdfs high availability information. Use the default row and column separators.

```
WITH
x1 AS
(SELECT k1, k2 FROM tbl1),
x2 AS
(SELECT k3 FROM tbl2)
SELEC k1 FROM x1 UNION SELECT k3 FROM x2
INTO OUTFILE "hdfs://path/to/result_"
PROPERTIES
(
    "broker.name" = "my_broker",
    "broker.username"="user",
    "broker.password"="passwd",
    "broker.dfs.nameservices" = "my_ha",
    "broker.dfs.ha.namenodes.my_ha" = "my_namenode1, my_namenode2",
    "broker.dfs.namenode.rpc-address.my_ha.my_namenode1" = "nn1_host:rpc_port",
    "broker.dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
    "broker.dfs.client.failover.proxy.provider" =
"org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

If the final generated file is not larger than 1GB, it will be: `result_0.csv`. If larger than 1GB, it may be `result_0.csv, result_1.csv, ....`

- Export the query result of the UNION statement to the file `bos://bucket/result.txt`. Specify the export format as PARQUET. Use `my_broker` and set hdfs high availability information. The PARQUET format does not require a column delimiter to be specified. After the export is complete, an identity file is generated.

```
SELECT k1 FROM tbl1 UNION SELECT k2 FROM tbl1
INTO OUTFILE "bos://bucket/result_"
FORMAT AS PARQUET
PROPERTIES
```

```
(
    "broker.name" = "my_broker",
    "broker.bos_endpoint" = "http://bj.bcebos.com",
    "broker.bos_accesskey" = "xxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "broker.bos_secret_accesskey" = "yyyyyyyyyyyyyyyyyyyyyyyyy"
);
```

- Export the query result of the select statement to the file `s3a://${bucket_name}/path/result.txt`. Specify the export format as csv. After the export is complete, an identity file is generated.

```
select k1,k2,v1 from tbl1 limit 100000
into outfile "s3a://my_bucket/export/my_file_"
FORMAT AS CSV
PROPERTIES
(
    "broker.name" = "hdfs_broker",
    "broker.fs.s3a.access.key" = "xxx",
    "broker.fs.s3a.secret.key" = "xxxx",
    "broker.fs.s3a.endpoint" = "https://cos.xxxxxx.myqcloud.com/",
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "max_file_size" = "1024MB",
    "success_file_name" = "SUCCESS"
)
```

  If the final generated file is not larger than 1GB, it will be: `my_file_0.csv`. If larger than 1GB, it may be `my_file_0.csv, result_1.csv, ....` Verify on cos

```
    1. A path that does not exist will be automatically created
    2. Access.key/secret.key/endpoint needs to be confirmed with students of cos. Especially the value of endpoint does not
need to fill in bucket_name.
```

- Use the s3 protocol to export to bos, and enable concurrent export.

```
set enable_parallel_outfile = true;
select k1 from tb1 limit 1000
into outfile "s3://my_bucket/export/my_file_"
format as csv
properties
(
    "s3.endpoint" = "http://s3.bd.bcebos.com",
    "s3.access_key" = "xxxx",
    "s3.secret_key" = "xxx",
    "s3.region" = "bd"
)
```

  The resulting file is prefixed with `my_file_{fragment_instance_id}_`.

- Use the s3 protocol to export to bos, and enable concurrent export of session variables. Note: However, since the query statement has a top-level sorting node, even if the concurrently exported session variable is enabled for this query, it cannot be exported concurrently.

```
set enable_parallel_outfile = true;
select k1 from tb1 order by k1 limit 1000
into outfile "s3://my_bucket/export/my_file_"
format as csv
```

```
properties
(
    "s3.endpoint" = "http://s3.bd.bcebos.com",
    "s3.access_key" = "xxxx",
    "s3.secret_key" = "xxx",
    "s3.region" = "bd"
)
```

- Use hdfs export to export simple query results to the file `hdfs://${host}:${fileSystem_port}/path/to/result.txt`. Specify the export format as CSV and the user name as work. Specify the column separator as ，and the row separator as `\n`.

```
-- fileSystem_port 默认值为 9000
SELECT * FROM tbl
INTO OUTFILE "hdfs://${host}:${fileSystem_port}/path/to/result_"
FORMAT AS CSV
PROPERTIES
(
    "fs.defaultFS" = "hdfs://ip:port",
    "hadoop.username" = "work"
);
```

If the Hadoop cluster is highly available and Kerberos authentication is enabled, you can refer to the following SQL statement:

```
SELECT * FROM tbl
INTO OUTFILE "hdfs://path/to/result_"
FORMAT AS CSV
PROPERTIES
(
'fs.defaultFS'='hdfs://hacluster/',
'dfs.nameservices'='hacluster',
'dfs.ha.namenodes.hacluster'='n1,n2',
'dfs.namenode.rpc-address.hacluster.n1'='192.168.0.1:8020',
'dfs.namenode.rpc-address.hacluster.n2'='192.168.0.2:8020',
'dfs.client.failover.proxy.provider.hacluster'='org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider',
'dfs.namenode.kerberos.principal'='hadoop/_HOST@REALM.COM'
'hadoop.security.authentication'='kerberos',
'hadoop.kerberos.principal'='doris_test@REALM.COM',
'hadoop.kerberos.keytab'='/path/to/doris_test.keytab'
);
```

If the final generated file is not larger than 100MB, it will be: `result_0.csv`. If larger than 100MB, it may be `result_0.csv, result_1.csv, ....`

- Export the query result of the select statement to the file `cosn://${bucket_name}/path/result.txt` on Tencent Cloud Object Storage (COS). Specify the export format as csv. After the export is complete, an identity file is generated.

```
select k1,k2,v1 from tbl1 limit 100000
into outfile "cosn://my_bucket/export/my_file_"
FORMAT AS CSV
PROPERTIES
(
    "broker.name" = "broker_name",
    "broker.fs.cosn.userinfo.secretId" = "xxx",
    "broker.fs.cosn.userinfo.secretKey" = "xxxx",
```

```
    "broker.fs.cosn.bucket.endpoint_suffix" = "cos.xxxxxx.myqcloud.com",
    "column_separator" = ",",
    "line_delimiter" = "\n",
    "max_file_size" = "1024MB",
    "success_file_name" = "SUCCESS"
)
```

CLEAN LABEL

Description

Used to manually clean up the labels of historical import jobs. After cleaning up, the labels can be reused. Commonly used in some automatic import tasks set by programs. When repeated execution, set the label of the imported fixed string. Before each import task is initiated, execute the statement to clean up the label.

Syntax

```
CLEAN LABEL [ <label> ] FROM <db_name>;
```

Required Parameters

1. `<db_name>`
label The name of the library.

Optional Parameters

1. `<label>`
The label to be cleaned. If omitted, the default is all labels in the current database.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Database	Requires modification access to the database.

Examples

- Clean label label1 from database db1

  ```
  CLEAN LABEL label1 FROM db1;
  ```

- Clean all labels from database db1

  ```
  CLEAN LABEL FROM db1;
  ```

CREATE REPOSITORY

Description

This statement is used to create a repository. Repositories are used for backup or restore.

Syntax

```
CREATE [READ ONLY] REPOSITORY <repo_name>
    WITH [ S3 | HDFS ]
    ON LOCATION <repo_location>
    PROPERTIES (
              -- S3 or HDFS storage property
              <storage_property>
              [ , ... ]
    )
```

Required Parameters

> The unique name of the repository.

> The storage path of the repository.

> The properties of the repository. The corresponding parameters should be selected based on whether S3 or HDFS is chosen as the storage medium.

Optional parameters are as follows, and additional parameters can be added based on the actual environment.

Parameter	Description
s3.endpoint	S3 service endpoint
s3.access_key	S3 access key
s3.secret_key	S3 secret key
s3.region	S3 region
use_path_style	Whether to use path-style access for S3 (applies to MinIO)
fs.defaultFS	Hadoop default file system URI
hadoop.username	Hadoop username

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV	Entire cluster management permissions	Only the root or superuser can create repositories

Usage notes

- If it is a read-only repository, restoration can only be performed on the repository. If not, both backup and restoration operations can be performed.
- The properties (PROPERTIES) vary depending on whether it is S3 or HDFS, as shown in the example.
- For ON LOCATION, if it is S3, the following should be the S3 Bucket Name.
- When performing data migration, the same repository must be created in both the source and destination clusters so that the destination cluster can view the data snapshot from the source cluster's backup.
- Any user can view the repositories that have been created by using the SHOW REPOSITORIES command.

Examples

Create a repository named s3_repo.

```
CREATE REPOSITORY `s3_repo`
WITH S3
ON LOCATION "s3://s3-repo"
PROPERTIES
(
    "s3.endpoint" = "http://s3-REGION.amazonaws.com",
    "s3.region" = "s3-REGION",
    "s3.access_key" = "AWS_ACCESS_KEY",
    "s3.secret_key"="AWS_SECRET_KEY"
);
```

Create a repository named hdfs_repo.

```
CREATE REPOSITORY `hdfs_repo`
WITH hdfs
ON LOCATION "hdfs://hadoop-name-node:54310/path/to/repo/"
PROPERTIES
(
    "fs.defaultFS"="hdfs://hadoop-name-node:54310",
    "hadoop.username"="user"
);
```

Create a repository named minio_repo to link minio storage directly through the S3 protocol.

```
CREATE REPOSITORY `minio_repo`
WITH S3
ON LOCATION "s3://minio_repo"
PROPERTIES
(
    "s3.endpoint" = "http://minio.com",
    "s3.access_key" = "MINIO_USER",
    "s3.secret_key"="MINIO_PASSWORD",
    "s3.region" = "REGION",
    "use_path_style" = "true"
);
```

Create a repository named minio_repo via temporary security credentials.

```
CREATE REPOSITORY `minio_repo`
WITH S3
ON LOCATION "s3://minio_repo"
PROPERTIES
(
    "s3.endpoint" = "AWS_ENDPOINT",
    "s3.access_key" = "AWS_TEMP_ACCESS_KEY",
    "s3.secret_key" = "AWS_TEMP_SECRET_KEY",
    "s3.session_token" = "AWS_TEMP_TOKEN",
    "s3.region" = "AWS_REGION"
)
```

Create repository using Tencent COS

```
CREATE REPOSITORY `cos_repo`
WITH S3
ON LOCATION "s3://backet1/"
PROPERTIES
(
    "s3.access_key" = "ak",
    "s3.secret_key" = "sk",
    "s3.endpoint" = "http://cos.ap-beijing.myqcloud.com",
    "s3.region" = "ap-beijing"
);
```

DROP REPOSITORY

Description

This statement is used to delete a created repository.

Syntax

```
DROP REPOSITORY <repo_name>;
```

Required Parameters

> The unique name of the repository.

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV	Entire cluster management permissions	Only the root or superuser can create repositories

Usage notes

- Deleting the repository only removes its mapping in Doris and does not delete the actual repository data. After deletion, the repository can be mapped again by specifying the same LOCATION.

Examples

Delete the repository named example_repo:

```
DROP REPOSITORY `example_repo`;
```

SHOW CREATE REPOSITORY

Description

This statement is used to demonstrate the creation statement of a repository.

Syntax

```
SHOW CREATE REPOSITORY for <repo_name>;
```

Required Parameters

> The unique name of the repository.

Examples

Show the creation statement of the specified repository

```
SHOW CREATE REPOSITORY for test_repository;
```

SHOW REPOSITORIES

Description

This statement is used to view the currently created warehouse

Syntax

```
SHOW REPOSITORIES;
```

Return Value

Field	Description
RepoId	The unique identifier (ID) of the repository
RepoName	The name of the repository
CreateTime	The creation time of the repository
IsReadOnly	Whether the repository is read-only. `false` means not read-only, `true` means read-only
Location	The root directory used for backing up data in the repository
Broker	-
Type	The repository type, currently supporting S3 and HDFS
ErrMsg	The error message of the repository. Typically `NULL` if no error occurs

Examples

View the created repository:

```
SHOW REPOSITORIES;
```

```
+--------+--------------+---------------------+------------+----------+--------+------+--------+
| RepoId | RepoName     | CreateTime          | IsReadOnly | Location | Broker | Type | ErrMsg |
+--------+--------------+---------------------+------------+----------+--------+------+--------+
| 43411  | example_repo | 2025-01-17 18:50:47 | false      | s3://rep1  | -      | S3   | NULL   |
+--------+--------------+---------------------+------------+----------+--------+------+--------+
```

BACKUP

Description

This statement is used to back up the data under the specified database. This command is an asynchronous operation. After the submission is successful, you need to check the progress through the SHOW BACKUP command.

Syntax

```
BACKUP SNAPSHOT [<db_name>.]<snapshot_name>
TO `<repository_name>`
[ { ON | EXCLUDE } ]
    ( <table_name> [ PARTITION ( <partition_name> [, ...] ) ]
    [, ...] ) ]

[ PROPERTIES ( "<key>" = "<value>" [ , ... ] )]
```

Required Parameters

1.<db_name>

The name of the database to which the data to be backed up belongs.

2.<snapshot_name>

Specify the data snapshot name. The snapshot name cannot be repeated and is globally unique.

3.<repository_name>

Warehouse name. You can create a repository via CREATE REPOSITORY.

Optional Parameters

1.<table_name>

The name of the table to be backed up. If not specified, the entire database will be backed up.

- The ON clause identifies the tables and partitions that need to be backed up. If no partition is specified, all partitions of the table are backed up by default
- Tables and partitions that do not require backup are identified in the EXCLUDE clause. Back up all partition data for all tables in this database except the specified table or partition.

2.<partition_name>

The name of the partition to be backed up. If not specified, all partitions of the corresponding table will be backed up.

3.[PROPERTIES ("<key>" = "<value>" [, ...])]

Data snapshot attributes, in the format: `<key>` = `<value>`, currently supports the following properties:

- “type” = “full”: indicates that this is a full update (default)
- “timeout” = “3600”: The task timeout period, the default is one day. in seconds.

Access Control Requirements

The user executing this SQL command must have at least the following privileges: | Privilege | Object | Notes | |:————|:———-|:——| | LOAD_PRIV | USER or ROLE | This operation can only be performed by users or roles with LOAD_PRIV permissions |

Usage Notes

- Only backing up tables of type OLAP is supported.
- Only one backup operation can be performed under the same database.
- The backup operation will back up the underlying table and Synchronous materialized view of the specified table or partition, and only one replica will be backed up. Asynchronous materialized view is not supported.
- Efficiency of backup operations:The efficiency of backup operations depends on the amount of data, the number of Compute Nodes, and the number of files. Each Compute Node where the backup data shard is located will participate in the upload phase of the backup operation. The greater the number of nodes, the higher the upload efficiency. The amount of file data refers only to the number of shards, and the number of files in each shard. If there are many shards, or there are many small files in the shards, the backup operation time may be increased.

Example

1. Fully backup the table example_tbl under example_db to the warehouse example_repo:

```
BACKUP SNAPSHOT example_db.snapshot_label1
TO example_repo
ON (example_tbl)
PROPERTIES ("type" = "full");
```

2. Under the full backup example_db, the p1, p2 partitions of the table example_tbl, and the table example_tbl2 to the warehouse example_repo:

```
BACKUP SNAPSHOT example_db.snapshot_label2
TO example_repo
ON
(
    example_tbl PARTITION (p1,p2),
    example_tbl2
);
```

3. Full backup of all tables except table example_tbl under example_db to warehouse example_repo:

```
BACKUP SNAPSHOT example_db.snapshot_label3
TO example_repo
EXCLUDE (example_tbl);
```

4. Fully back up tables under example_db to the repository example_repo:

```
BACKUP SNAPSHOT example_db.snapshot_label3
TO example_repo;
```

CANCEL BACKUP

Description

This statement is used to cancel an ongoing BACKUP task.

Syntax

```
CANCEL BACKUP FROM <db_name>;
```

Parameters

1.<db_name>

The name of the database to which the backup task belongs.

Example

1. Cancel the BACKUP task under example_db.

```
CANCEL BACKUP FROM example_db;
```

RESTORE

Description

This statement is used to restore the data backed up by the BACKUP command to the specified database. This command is an asynchronous operation. After the submission is successful, you need to check the progress through the SHOW RESTORE command.

Syntax

```
RESTORE SNAPSHOT [<db_name>.]<snapshot_name>
FROM `<repository_name>`
[ { ON | EXCLUDE } ] (
    `<table_name>` [PARTITION (`<partition_name>`, ...)] [AS `<table_alias>`]
    [, ...] ) ]
)
[ PROPERTIES ( "<key>" = "<value>" [ , ... ] )]
```

Required Parameters

1.<db_name>

The name of the database to which the data to be restored belongs

2.<snapshot_name>

Data snapshot name

3.<repository_name>

Warehouse name. You can create a repository via CREATE REPOSITORY

4.[PROPERTIES ("<key>" = "<value>" [, ...])]

Restoration operation attributes, the format is <key> = <value>， currently supports the following properties:

- “backup_timestamp” = “2018-05-04-16-45-08”: Specifies which time version of the corresponding backup to restore, required. This information can be obtained with the SHOW SNAPSHOT ON repo; statement.
- “replication_num” = “3”: Specifies the number of replicas for the restored table or partition. Default is 3. If restoring an existing table or partition, the number of replicas must be the same as the number of replicas of the existing table or partition. At the same time, there must be enough hosts to accommodate multiple replicas.
- “reserve_replica” = “true”: Default is false. When this property is true, the replication_num property is ignored and the restored table or partition will have the same number of replication as before the backup. Supports multiple tables or multiple partitions within a table with different replication number.
- “reserve_dynamic_partition_enable” = “true”: Default is false. When this property is true, the restored table will have the same value of ‘dynamic_partition_enable’ as before the backup. if this property is not true, the restored table will set ‘dynamic_partition_enable=false’.

- “timeout” = “3600”: The task timeout period, the default is one day. in seconds.
- “meta_version” = 40: Use the specified meta_version to read the previously backed up metadata. Note that this parameter is used as a temporary solution and is only used to restore the data backed up by the old version of Doris. The latest version of the backup data already contains the meta version, no need to specify it.
- “clean_tables” : Indicates whether to clean up tables that do not belong to the restore target. For example, if the target db before the restore has tables that are not present in the snapshot, specifying `clean_tables` can drop these extra tables and move them into the recycle bin during the restore.
    - This feature is supported since the Apache Doris 2.1.6 version
- “clean_partitions”：Indicates whether to clean up partitions that do not belong to the restore target. For example, if the target table before the restore has partitions that are not present in the snapshot, specifying `clean_partitions` can drop these extra partitions and move them into the recycle bin during the restore.
    - This feature is supported since the Apache Doris 2.1.6 version
- “atomic_restore”：The data will be loaded into a temporary table first, and then the original table will be replaced atomically to ensure that the read and write of the target table are not affected during the recovery process.
- “force_replace”：Force replace when the table exists and the schema is different with the backup table.
    - Note that to enable “force_replace”, you must enable “atomic_restore” ### Optional Parameters

1.`<table_name>`

The name of the table to be restored. If not specified, the entire database will be restored.

- The tables and partitions that need to be restored are identified in the ON clause. If no partition is specified, all partitions of the table are restored by default. The specified table and partition must already exist in the warehouse backup.
- Tables and partitions that do not require recovery are identified in the EXCLUDE clause. All partitions of all other tables in the warehouse except the specified table or partition will be restored.

2.`<partition_name>`

The name of the partition to be restored. If not specified, all partitions of the corresponding table will be restored.

3.`<table_alias>`

table alias

Access Control Requirements

The user executing this SQL command must have at least the following privileges: | Privilege | Object | Notes | |:————|:———-|:——| | LOAD_PRIV | USER or ROLE | This operation can only be performed by users or roles with LOAD_PRIV permissions |

Usage Notes

- Restoring tables of type OLAP is only supported.
- There can only be one executing BACKUP or RESTORE task under the same database.

- You can restore the backed up tables in the warehouse to replace the existing tables of the same name in the database, but you must ensure that the table structures of the two tables are exactly the same. The table structure includes: table name, column, partition, Rollup, etc.
- You can specify some partitions of the recovery table, and the system will check whether the partition Range or List can match.
- The table name backed up in the warehouse can be restored to a new table through the AS statement. But the new table name cannot already exist in the database. The partition name cannot be modified.
- Efficiency of recovery operations:In the case of the same cluster size, the time-consuming of the restore operation is basically the same as the time-consuming of the backup operation. If you want to speed up the recovery operation, you can first restore only one copy by setting the `replication_num` parameter, and then adjust the number of copies by ALTER TABLE PROPERTY, complete the copy.

Example

1. Restore the table backup_tbl in backup snapshot_1 from example_repo to database example_db1, the time version is “2018-05-04-16-45-08”. Revert to 1 copy:

```
RESTORE SNAPSHOT example_db1.`snapshot_1`
FROM `example_repo`
ON ( `backup_tbl` )
PROPERTIES
(
    "backup_timestamp"="2018-05-04-16-45-08",
    "replication_num" = "1"
);
```

2. Restore partitions p1, p2 of table backup_tbl in backup snapshot_2 from example_repo, and table backup_tbl2 to database example_db1, rename it to new_tbl, and the time version is “2018-05-04-17-11-01”. The default reverts to 3 replicas:

```
RESTORE SNAPSHOT example_db1.`snapshot_2`
FROM `example_repo`
ON
(
    `backup_tbl` PARTITION (`p1`, `p2`),
    `backup_tbl2` AS `new_tbl`
)
PROPERTIES
(
    "backup_timestamp"="2018-05-04-17-11-01"
);
```

3. Restore all tables except for table backup_tbl in backup snapshot_3 from example_repo to database example_db1, the time version is “2018-05-04-18-12-18”.

```
RESTORE SNAPSHOT example_db1.`snapshot_3`
FROM `example_repo`
EXCLUDE ( `backup_tbl` )
PROPERTIES
(
    "backup_timestamp"="2018-05-04-18-12-18"
);
```

SHOW RESTORE

Description

This statement is used to view RESTORE tasks

Syntax

```
SHOW [BRIEF] RESTORE [FROM <db_name>]
```

Parameters

1.`<db_name>`

The name of the database to which the recovery task belongs.

Return Value

- brief: only show key information of RESTORE tasks, columns RestoreObjs, Progress, TaskErrMsg will not show

Column	Description
JobId	Unique job id
Label	The name of the backup to restore
Timestamp	The time version of the backup to restore
DbName	belongs to the database
State	current stage:
AllowLoad	Whether to allow import when restoring (currently not supported)
ReplicationNum	Specifies the number of replicas to restore
ReserveReplica	Whether to keep a copy
ReplicaAllocation	Whether to keep dynamic partitioning enabled
RestoreJobs	Tables and partitions to restore
CreateTime	task submission time
MetaPreparedTime	Metadata preparation completion time
SnapshotFinishedTime	Snapshot completion time
DownloadFinishedTime	Snapshot download completion time
FinishedTime	Job finish time

Column	Description
UnfinishedTasks	Displays unfinished subtask ids during SNAPSHOTING, DOWNLOADING and COMMITING stages
Progress	Task progress
TaskErrMsg	Display task error messages
Status	If the job fails, display the failure message
Timeout	Job timeout, in seconds

Example

1. View the latest RESTORE task under example_db.

```
SHOW RESTORE FROM example_db;
```

CANCEL RESTORE

Description

This statement is used to cancel an ongoing RESTORE task.

Syntax

```
CANCEL RESTORE FROM <db_name>;
```

Parameters

1.<db_name>

The name of the database to which the recovery task belongs.

Usage Notes

- When cancellation is around a COMMIT or later stage of recovery, the table being recovered may be rendered inaccessible. At this time, data recovery can only be performed by executing the recovery job again.

Example

1. Cancel the RESTORE task under example_db.

```
CANCEL RESTORE FROM example_db;
```

SHOW SNAPSHOT

Description

This statement is used to view backups that already exist in the repository.

Syntax

```
SHOW SNAPSHOT ON `<repo_name>`
[WHERE SNAPSHOT = "<snapshot_name>" [AND TIMESTAMP = "<backup_timestamp>"]];
```

Parameters

1.<repo_name>

Back up the selected repository name.

2.<snapshot_name>

Backup name.

3.<backup_timestamp>

Backup timestamp.

Return Value

Column	Description
Snapshot	The name of the backup
Timestamp	corresponds to the time version of the backup
Status	corresponds to the time version of the backup
Database	The name of the database to which the backup data originally belonged
Details	In the form of Json, the data directory and file structure of the entire backup are displayed

Example

1. View the existing backups in the repository example_repo

```
SHOW SNAPSHOT ON example_repo;
```

2. View only the backup named backup1 in the repository example_repo:

```
SHOW SNAPSHOT ON example_repo WHERE SNAPSHOT = "backup1";
```

3. View the details of the backup named backup1 in the warehouse example_repo with the time version “2018-05-05-15-34-26”:

```
SHOW SNAPSHOT ON example_repo WHERE SNAPSHOT = "backup1" AND TIMESTAMP = "2018-05-05-15-34-26";
```

CREATE USER

Description

The `CREATE USER` statement is used to create a Doris user.

Syntax

```
CREATE USER [IF EXISTS] <user_identity> [IDENTIFIED BY <password>]
[DEFAULT ROLE <role_name>]
[<password_policy>]
[<comment>]

password_policy:
    1. PASSWORD_HISTORY { <n> | DEFAULT }
    2. PASSWORD_EXPIRE { DEFAULT | NEVER | INTERVAL <n> { DAY | HOUR | SECOND }}
    3. FAILED_LOGIN_ATTEMPTS <n>
    4. PASSWORD_LOCK_TIME { UNBOUNDED | <n> { DAY | HOUR | SECOND }}
```

Required Parameters

1. `<user_identity>`

> A user_identity uniquely identifies a user.The syntax is:‘user_name’@‘host’. `user_identity` consists of two parts, user_name and host, where username is the username. Host identifies the host address where the client connects. The host part can use % for fuzzy matching. If no host is specified, it defaults to ‘%’, which means the user can connect to Doris from any host. The host part can also be specified as a domain, the syntax is: ‘user_name’@[‘domain’], even if it is surrounded by square brackets, Doris will think this is a domain and try to resolve its ip address.

Optional Parameters

1. `<password>`

> Specify the user password.

2. `<role_name>`

> Specify the user role. If a role (ROLE) is specified, the newly created user will be automatically granted the permissions of the role. If not specified, the user has no permissions by default. The specified ROLE must already exist.

3. `<password_policy>`

> `password_policy` is a clause used to specify policies related to password authentication login. Currently, the following policies are supported:
>
> `PASSWORD_HISTORY { <n> | DEFAULT }`
>
> Whether to allow the current user to use historical passwords when resetting their passwords. For example, `PASSWORD_HISTORY 10` means that it is forbidden to use the password set in the past 10 times as a new password. If set to `PASSWORD_HISTORY DEFAULT`, the value in the global variable `password_history` will be used. `0` means do not enable this feature. Default is 0.
>
> `PASSWORD_EXPIRE { DEFAULT | NEVER | INTERVAL <n> { DAY | HOUR | SECOND }}`
>
> Set the expiration time of the current user's password. For example `PASSWORD_EXPIRE INTERVAL 10 DAY` means the password will expire in 10 days. `PASSWORD_EXPIRE NEVER` means that the password does not expire. If set to `PASSWORD_EXPIRE DEFAULT`, the value in the global variable `default_password_lifetime` is used. Defaults to NEVER (or 0), which means it will not expire.
>
> `FAILED_LOGIN_ATTEMPTS <n>`
>
> When the current user logs in, if the user logs in with the wrong password for n times, the account will be locked.For example, `FAILED_LOGIN_ATTEMPTS 3` means that if you log in wrongly for 3 times, the account will be locked.
>
> `PASSWORD_LOCK_TIME { UNBOUNDED | <n> { DAY | HOUR | SECOND }}`
>
> When the account is locked, the lock time is set. For example, `PASSWORD_LOCK_TIME 1 DAY` means that the account will be locked for one day.

4. `<comment>`

> Specify the user comment.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	This operation can only be performed by users or roles with ADMIN_PRIV permissions

Example

- Create a passwordless user (if host is not specified, it is equivalent to jack@'%')

```
CREATE USER 'jack';
```

- Create a user with a password to allow login from ‘172.10.1.10’

```
CREATE USER jack@'172.10.1.10' IDENTIFIED BY '123456';
```

- In order to avoid passing plaintext, use case 2 can also be created in the following way

```
CREATE USER jack@'172.10.1.10' IDENTIFIED BY PASSWORD '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9';
The encrypted content can be obtained through PASSWORD(), for example:
SELECT PASSWORD('123456');
```

- Create a user that is allowed to log in from the ‘192.168’ subnet, and specify its role as example_role

```
CREATE USER 'jack'@'192.168.%' DEFAULT ROLE 'example_role';
```

- Create a user that is allowed to log in from the domain ‘example_domain’

```
CREATE USER 'jack'@['example_domain'] IDENTIFIED BY '12345';
```

- Create a user and assign a role

```
CREATE USER 'jack'@'%' IDENTIFIED BY '12345' DEFAULT ROLE 'my_role';
```

- Create a user, set the password to expire after 10 days, and set the account to be locked for one day if you log in failed for 3 times.

```
CREATE USER 'jack' IDENTIFIED BY '12345' PASSWORD_EXPIRE INTERVAL 10 DAY FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1 DAY;
```

- Create a user and restrict non-resetable passwords to the last 8 passwords used.

```
CREATE USER 'jack' IDENTIFIED BY '12345' PASSWORD_HISTORY 8;
```

- Create a user with comment

```
CREATE USER 'jack' COMMENT "this is my first user";
```

ALTER USER

Description

The `ALTER USER` statement is used to modify a user's account attributes, including passwords, and password policies, etc.

Syntax

```
ALTER USER [IF EXISTS] <user_identity> [IDENTIFIED BY <password>]
[<password_policy>]
[<comment>]

password_policy:
    1. PASSWORD_HISTORY { <n> | DEFAULT }
    2. PASSWORD_EXPIRE { DEFAULT | NEVER | INTERVAL <n> { DAY | HOUR | SECOND }}
    3. FAILED_LOGIN_ATTEMPTS <n>
    4. PASSWORD_LOCK_TIME { UNBOUNDED | <n> { DAY | HOUR | SECOND }}
    5. ACCOUNT_UNLOCK
```

Required Parameters

1. `<user_identity>`

> A user_identity uniquely identifies a user.The syntax is:'user_name'@'host'. `user_identity` consists of two parts, user_name and host, where username is the username. Host identifies the host address where the client connects. The host part can use % for fuzzy matching. If no host is specified, it defaults to '%', which means the user can connect to Doris from any host. The host part can also be specified as a domain, the syntax is: 'user_name'@['domain'], even if it is surrounded by square brackets, Doris will think this is a domain and try to resolve its ip address.

Optional Parameters

1. `<password>`

> Specify the user password.

2. `<password_policy>`

> `password_policy` is a clause used to specify policies related to password authentication login. Currently, the following policies are supported:
>
> `PASSWORD_HISTORY { <n> | DEFAULT}`

> Whether to allow the current user to use historical passwords when resetting their passwords. For example, `PASSWORD_HISTORY 10` means that it is forbidden to use the password set in the past 10 times as a new password. If set to `PASSWORD_HISTORY DEFAULT`, the value in the global variable `password_history` will be used. `0` means do not enable this feature. Default is 0.
>
> `PASSWORD_EXPIRE { DEFAULT | NEVER | INTERVAL <n> { DAY | HOUR | SECOND }}`
>
> Set the expiration time of the current user's password. For example `PASSWORD_EXPIRE INTERVAL 10 DAY` means the password will expire in 10 days. `PASSWORD_EXPIRE NEVER` means that the password does not expire. If set to `PASSWORD_EXPIRE DEFAULT`, the value in the global variable `default_password_lifetime` is used. Defaults to NEVER (or 0), which means it will not expire.
>
> `FAILED_LOGIN_ATTEMPTS <n>`
>
> When the current user logs in, if the user logs in with the wrong password for n times, the account will be locked.For example, `FAILED_LOGIN_ATTEMPTS 3` means that if you log in wrongly for 3 times, the account will be locked.
>
> `PASSWORD_LOCK_TIME { UNBOUNDED | <n> { DAY | HOUR | SECOND }}`
>
> When the account is locked, the lock time is set. For example, `PASSWORD_LOCK_TIME 1 DAY` means that the account will be locked for one day.
>
> `ACCOUNT_UNLOCK`
>
> `ACCOUNT_UNLOCK` is used to unlock a locked user.

3. `<comment>`

> Specify the user comment.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	This operation can only be performed by users or roles with ADMIN_PRIV permissions

Usage Notes

1. Starting from version 2.0, this command no longer supports modifying user roles. Please use GRANT and REVOKE for related operations.
2. In an ALTER USER command, only one of the following account attributes can be modified at the same time:

- Change password
- Modify `PASSWORD_HISTORY`
- Modify `PASSWORD_EXPIRE`
- Modify `FAILED_LOGIN_ATTEMPTS` and `PASSWORD_LOCK_TIME`
- Unlock users

Example

- Change the user's password

```
ALTER USER jack@'%' IDENTIFIED BY "12345";
```

- Modify the user's password policy

```
ALTER USER jack@'%' FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1 DAY;
```

- Unlock a user

```
ALTER USER jack@'%' ACCOUNT_UNLOCK
```

- Modify the user's comment

```
ALTER USER jack@'%' COMMENT "this is my first user"
```

DROP USER

Description

The `DROP USER` statement is used to delete a user.

Syntax

```
DROP USER <user_identity>
```

Required Parameters

1. `<user_identity>`

> The specified user identity.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	This operation can only be performed by users or roles with ADMIN_PRIV permissions

Example

- Delete user jack@‘192.%’

```
DROP USER 'jack'@'192.%'
```

SET PASSWORD

Description

The `SET PASSWORD` statement is used to modify a user's login password.

Syntax

```
SET PASSWORD [FOR <user_identity>] =
    [PASSWORD(<plain_password>)]|[<hashed_password>]
```

Required Parameters

1. `<plain_password>`

> The input is a plaintext password. Taking the password `123456` as an example, directly use the string `123456`.

2. `<hashed_password>`

> The input is an encrypted password. Taking the password 123456 as an example, directly use the string `*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9`, which is the return value of the function `PASSWORD('123456')`.

Optional Parameters

1. `<user_identity>`

> The user_identity here must exactly match the user_identity specified when creating a user with CREATE USER, otherwise an error will be reported that the user does not exist. If user_identity is not specified, the current user is 'username'@'ip', which may not match any user_identity. Current users can be viewed through SHOW GRANTS.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	User or Role has the ADMIN_PRIV privilege to modify all user's password, otherwise only the current user's password can be modified.

Usage Notes

- If the `FOR user_identity` field does not exist, then change the current user's password.

Example

- Modify the current user's password

```
SET PASSWORD = PASSWORD('123456')
SET PASSWORD = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'
```

- Modify the specified user password

```
SET PASSWORD FOR 'jack'@'192.%' = PASSWORD('123456')
SET PASSWORD FOR 'jack'@['domain'] = '*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9'
```

CREATE ROLE

Description

The `CREATE ROLE` statement is used to create an unprivileged role, which can be subsequently granted with the GRANT command.

Syntax

```
CREATE ROLE <role_name> [<comment>];
```

Required Parameters

1. `<role_name>`

> The name of the role.

Optional Parameters

1. `<comment>`

> The comment of the role.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	This operation can only be performed by users or roles with ADMIN_PRIV permissions

Example

- Create a role

```
CREATE ROLE role1;
```

- Create a role with comment

```
CREATE ROLE role2 COMMENT "this is my first role";
```

ALTER ROLE

Description

The `ALTER ROLE` statement is used to modify a role.

Syntax

```
ALTER ROLE <role_name> COMMENT <comment>;
```

Required Parameters

1. `<role_name>`

> The name of the role.

Optional Parameters

1. `<comment>`

> The comment of the role.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	This operation can only be performed by users or roles with ADMIN_PRIV permissions

Example

- Modify the role's comment

```
ALTER ROLE role1 COMMENT "this is my first role";
```

DROP ROLE

Description

The `DROP ROLE` statement is used to removes a role.

Syntax

```
DROP ROLE [IF EXISTS] <role_name>;
```

Required Parameters

1. `<role_name>`:

> The name of the role.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	This operation can only be performed by users or roles with ADMIN_PRIV permissions

Usage Notes

- Deleting a role does not affect the permissions of users who previously belonged to the role. It is only equivalent to decoupling the role from the user. The permissions that the user has obtained from the role will not change.

Example

- Drop a role1

```
DROP ROLE role1;
```

SHOW ROLES

Description

The `SHOW ROLES` statement is used to display all created role information, including role name, included users and permissions.

Syntax

```
SHOW ROLES
```

Return Value

Column	DataType	Note
Name	string	Role Name
Comment	string	Comment
Users	string	Included Users
GlobalPrivs	string	Global Privileges
CatalogPrivs	string	Catalog Privileges
DatabasePrivs	string	Database Privileges
TablePrivs	string	Table Privileges
ResourcePrivs	string	Resource Privileges
WorkloadGroupPrivs	string	Workload Group Privileges

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
GRANT_PRIV	USER or ROLE	This operation can only be performed by users or roles with GRANT_PRIV permissions

Usage Notes

Doris creates a default role for each user. If you want to display the default role, you can execute the command `set show_user_default_role=true;`.

Example

- View created roles

```
SHOW ROLES
```

GRANT TO

Description

The GRANT command is used to:

1. Grant specified privileges to a user or role.
2. Grant specified roles to a user.

Related Commands

- REVOKE FROM
- SHOW GRANTS
- CREATE ROLE
- CREATE WORKLOAD GROUP
- CREATE RESOURCE

Syntax

Grant specified privileges to a user or role

```
GRANT <privilege_list>
ON { <priv_level>
    | RESOURCE <resource_name>
    | WORKLOAD GROUP <workload_group_name>
   }
TO { <user_identity> | ROLE <role_name> }
```

Grant specified roles to a user

```
GRANT <role_list> TO <user_identity>
```

Required Parameters

1. `<privilege_list>`

A comma-separated list of privileges to be granted. Currently supported privileges include:

- NODE_PRIV: Cluster node operation permissions, including node online and offline operations.
- ADMIN_PRIV: All privileges except NODE_PRIV.
- GRANT_PRIV: Privilege for operation privileges, including creating and deleting users, roles, authorization and revocation, setting passwords, etc.
- SELECT_PRIV: Read permission on the specified database or table.
- LOAD_PRIV: Import privileges on the specified database or table.
- ALTER_PRIV: Schema change permission for the specified database or table.
- CREATE_PRIV: Create permission on the specified database or table.
- DROP_PRIV: Drop privilege on the specified database or table.

- USAGE_PRIV: Access to the specified resource and Workload Group permissions.
- SHOW_VIEW_PRIV: Permission to view view creation statements.

Legacy privilege conversion:

- ALL and READ_WRITE will be converted to: SELECT_PRIV, LOAD_PRIV, ALTER_PRIV, CREATE_PRIV, DROP_PRIV.
- READ_ONLY is converted to SELECT_PRIV.

2. `<priv_level>`

Supports the following four forms:

- ..*: Privileges can be applied to all catalogs and all databases and tables within them.
- catalog_name..: Privileges can be applied to all databases and tables in the specified catalog.
- catalog_name.db.*: Privileges can be applied to all tables under the specified database.
- catalog_name.db.tbl: Privileges can be applied to the specified table under the specified database.

3. `<resource_name>`

Specifies the resource name, supporting `%` and `*` to match all resources, but does not support wildcards, such as res*.

4. `<workload_group_name>`

Specifies the workload group name, supporting `%` and `*` to match all workload groups, but does not support wildcards.

5. `<user_identity>`

Specifies the user to receive the privileges. Must be a user_identity created with CREATE USER. The host in user_identity can be a domain name. If it is a domain name, the effective time of the authority may be delayed by about 1 minute.

6. `<role_name>`

Specifies the role to receive the privileges. If the specified role does not exist, it will be created automatically.

7. `<role_list>`

A comma-separated list of roles to be assigned. The specified roles must exist.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
GRANT_PRIV	User or Role	Only users or roles with the GRANT_PRIV privilege can perform the REVOKE operation.

Examples

- Grant permissions to all catalogs and databases and tables to the user:

```
GRANT SELECT_PRIV ON *.*.* TO 'jack'@'%';
```

- Grant permissions to the specified database table to the user:

```
GRANT SELECT_PRIV,ALTER_PRIV,LOAD_PRIV ON ctl1.db1.tbl1  TO 'jack'@'192.8.%';
```

- Grant permissions to the specified database table to the role:

```
GRANT LOAD_PRIV ON ctl1.db1.* TO ROLE 'my_role';
```

- Grant access to all resources to users:

```
GRANT USAGE_PRIV ON RESOURCE * TO 'jack'@'%';
```

- Grant the user permission to use the specified resource:

```
GRANT USAGE_PRIV ON RESOURCE 'spark_resource' TO 'jack'@'%';
```

- Grant access to specified resources to roles:

```
GRANT USAGE_PRIV ON RESOURCE 'spark_resource' TO ROLE 'my_role';
```

- Grant the specified role to a user:

```
GRANT 'role1','role2' TO 'jack'@'%';
```

- Grant the specified workload group ‘g1’ to user jack:

```
GRANT USAGE_PRIV ON WORKLOAD GROUP 'g1' TO 'jack'@'%';
```

- Match all workload groups granted to user jack:

```
GRANT USAGE_PRIV ON WORKLOAD GROUP '%' TO 'jack'@'%';
```

- Grant the workload group ‘g1’ to the role my_role:

```
GRANT USAGE_PRIV ON WORKLOAD GROUP 'g1' TO ROLE 'my_role';
```

- Allow jack to view the creation statement of view1 under db1:

```
GRANT SHOW_VIEW_PRIV ON db1.view1 TO 'jack'@'%';
```

REVOKE FROM

Description

The REVOKE command has the following functions:

1. Revoke the specified permission of a user or a role.
2. Revoke the specified role previously granted to a user.

Related Commands

- GRANT TO
- SHOW GRANTS
- CREATE ROLE
- CREATE WORKLOAD GROUP
- CREATE RESOURCE

Syntax

Revoke the specified permission of a user or a role

```
REVOKE <privilege_list>
ON { <priv_level>
   | RESOURCE <resource_name>
   | WORKLOAD GROUP <workload_group_name>
   }
FROM { <user_identity> | ROLE <role_name> }
```

Revoke the specified role previously granted to a user

```
REVOKE <role_list> FROM <user_identity>
```

Required Parameters

1. `<privilege_list>`

A comma-separated list of privileges to be revoked. Supported privileges include:

- NODE_PRIV: Cluster node operation permissions
- ADMIN_PRIV: Administrator privileges
- GRANT_PRIV: Authorization privileges
- SELECT_PRIV: Query privileges
- LOAD_PRIV: Data import privileges
- ALTER_PRIV: Modify privileges
- CREATE_PRIV: Create privileges
- DROP_PRIV: Delete privileges
- USAGE_PRIV: Usage privileges

- SHOW_VIEW_PRIV: View definition privileges

2. `<priv_level>`

Specifies the scope of the privileges. Supported formats include:

- ..*: All catalogs, databases, and tables
- catalog_name..: Specifies all databases and tables in the specified catalog
- catalog_name.db.*: Specifies all tables in the specified database
- catalog_name.db.tbl: Specifies a specific table in the specified database

3. `<resource_name>`

Specifies the resource name. Supports `%` (matches any string) and _ (matches any single character) wildcard characters.

4. `<workload_group_name>`

Specifies the workload group name. Supports `%` (matches any string) and _ (matches any single character) wildcard characters.

5. `<user_identity>`

Specifies the user identity. The user must be a user created with the CREATE USER statement. The host part of the user identity can be a domain name, and the permission revocation time may have a delay of 1 minute for domain names.

6. `<role_name>`

Specifies the role name. The role must exist.

7. `<role_list>`

A comma-separated list of roles to be revoked. All specified roles must exist.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
GRANT_PRIV	User or Role	Only users or roles with the GRANT_PRIV privilege can perform the REVOKE operation.

Examples

- Revoke the SELECT privilege on a specific database from a user:

```
REVOKE SELECT_PRIV ON db1.* FROM 'jack'@'192.%';
```

- Revoke the usage privilege on a specific resource from a user:

```
REVOKE USAGE_PRIV ON RESOURCE 'spark_resource' FROM 'jack'@'192.%';
```

- Revoke roles from a user:

```
REVOKE 'role1','role2' FROM 'jack'@'192.%';
```

- Revoke the usage privilege on a specific workload group from a user:

```
REVOKE USAGE_PRIV ON WORKLOAD GROUP 'g1' FROM 'jack'@'%';
```

- Revoke the usage privilege on all workload groups from a user:

```
REVOKE USAGE_PRIV ON WORKLOAD GROUP '%' FROM 'jack'@'%';
```

SHOW PRIVILEGES

Description

`SHOW PRIVILEGES` statement is used to display the list of currently available privileges in the database system. It helps users understand the types of privileges supported by the system and the details of each privilege.

Syntax

```
SHOW PRIVILEGES
```

Return Value

Column	Description
Privilege	Privilege name
Context	Applicable range
Comment	Description

Access Control Requirements

The user executing this SQL command does not need to have specific privileges.

Examples

View all privileges

```
SHOW PRIVILEGES
```

```
+-------------+-----------------------------------------------------+-----------------------------------------------+
| Privilege   | Context                                             | Comment                                       |
+-------------+-----------------------------------------------------+-----------------------------------------------+
| Node_priv   | GLOBAL                                              | Privilege for cluster node operations         |
| Admin_priv  | GLOBAL                                              | Privilege for admin user                      |
| Grant_priv  | GLOBAL,CATALOG,DATABASE,TABLE,RESOURCE,WORKLOAD GROUP | Privilege for granting privilege              |
| Select_priv | GLOBAL,CATALOG,DATABASE,TABLE                       | Privilege for select data in tables           |
| Load_priv   | GLOBAL,CATALOG,DATABASE,TABLE                       | Privilege for loading data into tables        |
| Alter_priv  | GLOBAL,CATALOG,DATABASE,TABLE                       | Privilege for alter database or table         |
| Create_priv | GLOBAL,CATALOG,DATABASE,TABLE                       | Privilege for creating database or table      |
| Drop_priv   | GLOBAL,CATALOG,DATABASE,TABLE                       | Privilege for dropping database or table      |
| Usage_priv  | RESOURCE,WORKLOAD GROUP                             | Privilege for using resource or workloadGroup |
+-------------+-----------------------------------------------------+-----------------------------------------------+
```

SHOW GRANTS

Description

This statement is used to view user permissions.

Syntax

```
SHOW [ALL] GRANTS [FOR <user_identity>];
```

Optional Parameters

1. `[ALL]`

Whether to view the permissions of all users.

2. `<user_identity>`

Specify the user whose permissions are to be viewed. The `user_identity` must be created by the `CREATE USER` command.

Return Value

Column	Description
UserIdentity	User identity
Comment	Comment
Password	Whether the password is set
Roles	Roles
GlobalPrivs	Global privileges
CatalogPrivs	Catalog privileges
DatabasePrivs	Database privileges
TablePrivs	Table privileges
ColPrivs	Column privileges
ResourcePrivs	Resource privileges
WorkloadGroupPrivs	WorkloadGroup privileges

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
GRANT_PRIV	User or Role	User or Role has the `GRANT_PRIV` privilege to view all user permissions, otherwise only the current user's permissions can be viewed

Usage Notes

- `SHOW ALL GRANTS` can view all users' permissions, but requires the `GRANT_PRIV` permission.
- If the `user_identity` is specified, the permissions of the specified user are viewed. And the `user_identity` must be created by the `CREATE USER` command.
- If the `user_identity` is not specified, the permissions of the current user are viewed.
- Doris implements permission control based on the RBAC (Role-Based Access Control) model. Therefore, the permissions displayed here are actually the combined permissions of all roles assigned to the user. If you want to check which specific role a permission comes from, you can use the SHOW ROLES command to view the details.

Examples

1. View all user permission information.

```
SHOW ALL GRANTS;
```

```
+--------------+---------+----------+----------+---------------------+--------------+-----------------------------------
---------------------------------+------------+----------+--------------+--------------------+
| UserIdentity | Comment | Password | Roles    | GlobalPrivs         | CatalogPrivs | DatabasePrivs
| TablePrivs | ColPrivs | ResourcePrivs | WorkloadGroupPrivs |
+--------------+---------+----------+----------+---------------------+--------------+-----------------------------------
---------------------------------+------------+----------+--------------+--------------------+
| 'root'@'%'   | ROOT    | No       | operator | Node_priv,Admin_priv | NULL         | internal.information_schema:
Select_priv; internal.mysql: Select_priv | NULL       | NULL     | NULL          | normal: Usage_priv |
| 'admin'@'%'  | ADMIN   | No       | admin    | Admin_priv           | NULL         | internal.information_schema:
Select_priv; internal.mysql: Select_priv | NULL       | NULL     | NULL          | normal: Usage_priv |
| 'jack'@'%'   |         | No       |          | NULL                 | NULL         | internal.information_schema:
Select_priv; internal.mysql: Select_priv | NULL       | NULL     | NULL          | normal: Usage_priv |
+--------------+---------+----------+----------+---------------------+--------------+-----------------------------------
---------------------------------+------------+----------+--------------+--------------------+
```

2. View the permissions of the specified user

```
SHOW GRANTS FOR jack@'%';
```

```
+--------------+---------+----------+-------+-------------+--------------+-----------------------------------------------
----------------------+------------+----------+--------------+--------------------+
| UserIdentity | Comment | Password | Roles | GlobalPrivs | CatalogPrivs | DatabasePrivs
| TablePrivs | ColPrivs | ResourcePrivs | WorkloadGroupPrivs |
+--------------+---------+----------+-------+-------------+--------------+-----------------------------------------------
----------------------+------------+----------+--------------+--------------------+
| 'jack'@'%'   |         | No       |       | NULL        | NULL         | internal.information_schema: Select_priv;
internal.mysql: Select_priv | NULL       | NULL     | NULL          | normal: Usage_priv |
+--------------+---------+----------+-------+-------------+--------------+-----------------------------------------------
----------------------+------------+----------+--------------+--------------------+
```

3. View the permissions of the current user

```
SHOW GRANTS;
```

```
+--------------+---------+----------+----------+----------------------+-------------+-----------------------------------
---------------------------------+------------+----------+--------------+--------------------+
| UserIdentity | Comment | Password | Roles    | GlobalPrivs          | CatalogPrivs | DatabasePrivs
| TablePrivs | ColPrivs | ResourcePrivs | WorkloadGroupPrivs |
+--------------+---------+----------+----------+----------------------+-------------+-----------------------------------
---------------------------------+------------+----------+--------------+--------------------+
| 'root'@'%'   | ROOT    | No       | operator | Node_priv,Admin_priv | NULL         | internal.information_schema:
Select_priv; internal.mysql: Select_priv | NULL       | NULL     | NULL          | normal: Usage_priv |
+--------------+---------+----------+----------+----------------------+-------------+-----------------------------------
---------------------------------+------------+----------+--------------+--------------------+
```

REFRESH LDAP

Description

This statement is used to refresh the cache information of LDAP in Doris. When modifying user information in the LDAP service or modifying the role permissions corresponding to LDAP user groups in Doris, the changes may not take effect immediately due to caching, and the cache can be refreshed through this statement.

Syntax

```
REFRESH LDAP [ALL | FOR <user_name>];
```

Optional Parameters

1. `[ALL]`

Whether to refresh the LDAP cache information of all users.

2. `<user_name>`

The user whose LDAP cache information needs to be refreshed

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	User or Role	Only users or roles with the `ADMIN_PRIV` permission can refresh the LDAP cache information of all users. Otherwise, they can only refresh the LDAP cache information of the current user

Usage Notes

- The default timeout for LDAP information cache in Doris is 12 hours, which can be viewed by `SHOW FRONTEND CONFIG LIKE 'ldap_user_cache_timeout_s';`.
- `REFRESH LDAP ALL` refreshes the LDAP cache information of all users, but requires the `ADMIN_PRIV` permission.
- If `user_name` is specified, the LDAP cache information of the specified user will be refreshed.
- If `user_name` is not specified, the LDAP cache information of the current user will be refreshed.

Examples

1. Refresh the cache information of all LDAP users.

```
REFRESH LDAP ALL;
```

2. Refresh the cache information of the current LDAP user.

```
REFRESH LDAP;
```

3. Refresh the cache information of the specified LDAP user jack.

```
REFRESH LDAP FOR jack;
```

SET LDAP_ADMIN_PASSWORD

Description

The `SET LDAP_ADMIN_PASSWORD` command is used to set the LDAP administrator password. When using LDAP authentication, doris needs to use the administrator account and password to query the LDAP service for login user information.

Syntax

```
SET LDAP_ADMIN_PASSWORD = PASSWORD('<plain_password>')
```

Required Parameters

1. `<plain_password>`

LDAP administrator password

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	USER or ROLE	This operation can only be performed by users or roles with `ADMIN_PRIV` permissions

Examples

- Set the LDAP administrator password

  ```
  SET LDAP_ADMIN_PASSWORD = PASSWORD('123456')
  ```

SET PROPERTY

Description

Set user attributes, including resources assigned to users, importing clusters, etc.

Related Commands

- CREATE USER
- SHOW PROPERTY

Syntax

```
SET PROPERTY [ FOR '<user_name>' ] '<key_1>' = '<value_1>' [, '<key_2>' = '<value_2>', ...];
```

Required Parameters

1. `<key_n>`

Super user privileges:

- `max_user_connections`: The maximum number of connections.
- `max_query_instances`: The number of instances that a user can use to execute a query at the same time.
- `sql_block_rules`: Set sql block rules. Once set, queries sent by this user will be rejected if they match the rules.
- `cpu_resource_limit`: Limit the cpu resources for queries. See the introduction to the session variable `cpu_resource_limit` for details. -1 means not set.
- `exec_mem_limit`: Limit the memory usage of the query. See the introduction to the session variable `exec_mem_limit` for details. -1 means not set.
- `resource_tags`: Specifies the user's resource tag permissions.
- `query_timeout`: Specifies the user's query timeout permissions.
- `default_workload_group`: Specifies the user's default workload group.

Note: If the attributes `cpu_resource_limit`, `exec_mem_limit` are not set, the value in the session variable will be used by default.

2. `<value_n>`

The value set for the specified key.

Optional Parameters

1. `<user_name>`

The username of the user whose properties are to be set. If omitted, the properties will be set for the current user.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	User or Role	This `SET PROPERTY` operation can only be performed by users or roles with `ADMIN_PRIV` permissions.

Usage Notes

- The user attribute set here is for user, not user_identity. That is, if two users ‘jack’@‘%’ and ‘jack’@‘192.%’ are created through the CREATE USER statement, the SET PROPERTY statement can only be used for the user jack, not ‘jack’@‘%’ or ‘jack’@‘192.%’

Examples

- Modify the maximum number of user jack connections to 1000

```
SET PROPERTY FOR 'jack' 'max_user_connections' = '1000';
```

- Modify the number of available instances for user jack’s query to 3000

```
SET PROPERTY FOR 'jack' 'max_query_instances' = '3000';
```

- Modify the sql block rule of user jack

```
SET PROPERTY FOR 'jack' 'sql_block_rules' = 'rule1, rule2';
```

- Modify the cpu usage limit of user jack

```
SET PROPERTY FOR 'jack' 'cpu_resource_limit' = '2';
```

- Modify the user’s resource tag permissions

```
SET PROPERTY FOR 'jack' 'resource_tags.location' = 'group_a, group_b';
```

- Modify the user’s query memory usage limit, in bytes

```
SET PROPERTY FOR 'jack' 'exec_mem_limit' = '2147483648';
```

- Modify the user’s query timeout limit, in second

```
SET PROPERTY FOR 'jack' 'query_timeout' = '500';
```

SHOW PROPERTY

Description

This statement is used to view the attributes of the user.

Syntax

```
SHOW {ALL PROPERTIES | PROPERTY [FOR <user_name>]} [LIKE <key>]
```

Optional Parameters

1. `[ALL PROPERTIES]`

Whether to view all user attributes.

2. `<user_name>`

View the attributes of the specified user. If not specified, check the current user's.

3. `<key>`

Fuzzy matching can be done by attribute name.

Return Value

- If the statement uses `PROPERTY`

Column	Description
Key	Attribute name
Value	Attribute value

- If the statement uses `PROPERTIES`

Column	Description
User	User name
Properties	Corresponding user each `property` `key:value`

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
GRANT_PRIV	User or Role	User or role has `GRANT_PRIV` permission to view all user properties, `SHOW PROPERTY` does not require `GRANT_PRIV` permission to view the current user's properties

Usage Notes

- `SHOW ALL PROPERTIES` can be used to view all users' properties.
- If the `user_name` is specified, view the attributes of the specified user.
- If the `user_name` is not specified, view the attributes of the current user.
- `SHOW PROPERTY` does not require `GRANT_PRIV` permission to view the current user's properties.

Examples

- View the attributes of the jack user

```
SHOW PROPERTY FOR 'jack';
```

```
+-----------------------------------+--------+
| Key                               | Value  |
+-----------------------------------+--------+
| cpu_resource_limit                | -1     |
| default_load_cluster              |        |
| default_workload_group            | normal |
| exec_mem_limit                    | -1     |
| insert_timeout                    | -1     |
| max_query_instances               | 3000   |
| max_user_connections              | 1000   |
| parallel_fragment_exec_instance_num | -1     |
| query_timeout                     | -1     |
| resource_tags                     |        |
| sql_block_rules                   |        |
+-----------------------------------+--------+
```

- View the limit-related properties of the user jack

```
SHOW PROPERTY FOR 'jack' LIKE '%limit%';
```

```
+--------------------+-------+
| Key                | Value |
+--------------------+-------+
| cpu_resource_limit | -1    |
| exec_mem_limit     | -1    |
+--------------------+-------+
```

- View all users limit related properties

```
SHOW ALL PROPERTIES LIKE '%limit%';
```

```
+-------+------------------------------------------------------------+
| User  | Properties                                                 |
```

```
+-------+-------------------------------------------------------------+
| root  | {
  "cpu_resource_limit": "-1",
  "exec_mem_limit": "-1"
} |
| admin | {
  "cpu_resource_limit": "-1",
  "exec_mem_limit": "-1"
} |
| jack  | {
  "cpu_resource_limit": "-1",
  "exec_mem_limit": "-1"
} |
+-------+-------------------------------------------------------------+
```

SWITCH CATALOG

Description

This statement is used to switch catalog.

Syntax

```
SWITCH <catalog_name>
```

Required Parameters

1. `<catalog_name>` > The name of the catalog to switch to.

Access Control Requirements

Privilege	Object	Notes
SELECT_PRIV	Catalog	SELECT_PRIV privilege is required on the catalog to be switched to.

Examples

1. Switch to the catalog `hive`

    ```
    SWITCH hive;
    ```

USE DATABASE

Description

Used to switch to the specified database or compute group.

Syntax

```
USE { [<catalog_name>.]<database_name>[@<compute_group_name>] | @<compute_group_name> }
```

Required Parameters

Switch to the specified database.

1. `<database_name>` > The name of the database to switch to. > If no catalog is specified, the current catalog is used by default.

Switch to the specified compute group only.

1. `<compute_group_name>` > The name of the compute group to switch to.

Optional Parameters

Switch to the specified database.

1. `<catalog_name>` > The name of the catalog to switch to.

2. `<compute_group_name>` > The name of the compute group to switch to.

Access Control Requirements

Privilege	Object	Notes
SELECT_PRIV	Catalog, Database	SELECT_PRIV privilege is required on the catalog or database to switch to.
USAGE_PRIV	Compute Group	USAGE_PRIV privilege is required on the compute group to switch to.

Examples

1. If the `demo` database exists, try to use it:

   ```
   use demo;
   ```

2. If the `demo` database exists under the `hms_catalog` catalog, try to switch to `hms_catalog` and use it:

```
use hms_catalog.demo;
```

3. If the `demo` database exists in the current catalog and you want to use the compute group named ‘cg1’, try to access it:

```
use demo@cg1;
```

4. If you only want to use the compute group named ‘cg1’, try to access it:

```
use @cg1;
```

SET VARIABLE

Description

This statement is mainly used to modify Doris system variables. These system variables can be modified at the global and session level, and some can also be modified dynamically. You can also view these system variables with `SHOW VARIABLE`.

Syntax

```
SET variable_assignment [, variable_assignment] [ ... ]
```

Where:

```
variable_assignment
  : <user_var_name> = <expr>
  | [ <effective_scope> ] <system_var_name> = <expr>
```

Required Parameters

1. `<user_var_name>` > Specifies the variable of user level, for example : @your_variable_name, variable name starts with `@`

2. `<system_var_name>` > Specifies the variable of system level, for example : exec_mem_limit and so on

Optional Parameters

1. `<effective_scope>`

> Effective scope is one of `GLOBAL` or `SESSION` or `LOCAL`. If there is no effective scope, default value is `SESSION`. `LOCAL` is an alias of `SESSION`.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	Session	set global variables need admin privilege

Usage Notes

- Only ADMIN users can set variables to take effect globally

- The globally effective variable affects the current session and new sessions thereafter, but does not affect other sessions that currently exist.

Example

- Set the time zone to East Eighth District

```
SET time_zone = "Asia/Shanghai";
```

- Set the global execution memory size

```
SET GLOBAL exec_mem_limit = 137438953472
```

- Set a user variable

```
SET @your_variable_name = your_variable_value;
```

UNSET VARIABLE

Description

This statement is used to restore Doris system variables. These system variables can be modified at global or session level.

Syntax

```
UNSET [<effective_scope>] VARIABLE (<variable_name>)
```

Required Parameters

1. `<variable_name>` > Specifies the variable name, or if you want to unset all variables, this parameter you can give a keyword `ALL`.

Optional Parameters

1. `<effective_scope>` > Effective scope is one of `GLOBAL` or `SESSION` or `LOCAL`. If there is no effective scope, default value is `SESSION`. `LOCAL` is an alias of `SESSION`.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	Session	unset global variables need admin privilege

Usage Notes

- Only ADMIN users can unset variables to take effect globally
- When restore a variable with `GLOBAL`, it only affects your current using session and new open sessions. It does not affect other current open sessions.

Example

- Restore value of the time zone

  ```
  UNSET VARIABLE time_zone;
  ```

- Restore the global execution memory size

  ```
  UNSET GLOBAL VARIABLE exec_mem_limit;
  ```

- Restore all variables globally

```
UNSET GLOBAL VARIABLE ALL;
```

SHOW VARIABLES

Description

This statement is used to display Doris system variables, which can be queried by conditions

Syntax

```
SHOW [<effective_scope>] VARIABLES [<like_pattern> | <where>]
```

Optional Parameters

1. `<effective_scope>` > Effective scope is one of `GLOBAL` or `SESSION` or `LOCAL`. If there is no effective scope, default value is `SESSION`. `LOCAL` is an alias of `SESSION`.

2. `<like_pattern>` > Use like statement to match and filter result

3. `<where>` > Use where statement to match and filter result

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
N/A	Session	No specific privilege is required to show variables

Return Value

Variable_name	Value	Default_Value	Changed
variable name1	value1	default value1	0/1
variable name2	value2	default value2	0/1

Usage Notes

- Show variables is mainly used to view the values of system variables.
- Executing the SHOW VARIABLES command does not require any privileges, it only requires being able to connect to the server.
- The column `Changed` from `Return Value`, 0 means no changed and 1 means changed.
- There are some restrictions when using the `SHOW` statement:
    - Can not use `or` in where clause
    - Column names are on the left
    - Only supports equivalent comparisons in where clause
    - Use the like statement to match with variable_name.

    - The % percent wildcard can be used anywhere in the matching pattern

Example

- The default here is to match the Variable_name, here is the exact match

```
show variables like 'max_connections';
```

- Matching through the percent sign (%) wildcard can match multiple items

```
show variables like '%connec%';
```

- Use the Where clause for matching queries

```
show variables where variable_name = 'version';
```

SHOW PROCESSLIST

Description

Displays the thread that the user is running.
This command can only display the connection information of the current FE.
If you need to view the connection information of the entire cluster, you need to add a session variable:

```
SET SHOW_ALL_FE_CONNECTION = TRUE;
```

Syntax

```
SHOW [FULL] PROCESSLIST
```

Optional Parameters

1. `FULL`

> Indicates whether to view the connection information of other users

Return Value

column	Instructions
CurrentConnected	Whether the connection is current
Id	The unique identification of this thread
User	The user who started this thread
Host	The IP address and port number of the client sending the request are recorded
LoginTime	Time to establish a connection
Catalog	In which data directory is the command currently executed
Db	On which database is the command being executed? If no database is specified, the value is NULL
Command	The command that the thread is executing at this moment
Time	Time when the previous command is submitted to the current status, in seconds
State	State of thread
QueryId	ID of the current query statement
Info	Generally, the statement executed by the thread is recorded, and only the first 100 characters are displayed by default

Common Command types are as follows:

column	Instructions
Query	The thread is executing a statement
Sleep	Waiting for the client to send it an execution statement
Quit	The thread is exiting
Kill	Executing the kill statement

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	DATABASE	If you want to view the connection information of other users, you need the ADMIN permission

Examples

```
SHOW PROCESSLIST
```

```
+------------------+------+------+----------------+---------------------+----------+------+---------+------+-------+---------------
--------------------+-----------------------+
| CurrentConnected | Id   | User | Host           | LoginTime           | Catalog  | Db   | Command | Time | State | QueryId
| Info                  |
+------------------+------+------+----------------+---------------------+----------+------+---------+------+-------+---------------
--------------------+-----------------------+
| Yes              |    0 | root | 127.0.0.1:34650 | 2025-01-21 12:01:02 | internal | test | Query   |    0 | OK    |
c84e397193a54fe7-bbe9bc219318b75e | select 1              |
|                  |    1 | root | 127.0.0.1:34776 | 2025-01-21 12:01:07 | internal |      | Sleep   |   29 | EOF   |
886ffe2894314f50-8dd73a6ca06699e4 | show full processlist |
+------------------+------+------+----------------+---------------------+----------+------+---------+------+-------+---------------
--------------------+-----------------------+
```

KILL QUERY

Description

Each Doris connection runs in a separate thread. Use this statement to terminate the thread.

Syntax

```
KILL [CONNECTION] <processlist_id>
```

Varaint Syntax

```
KILL QUERY [ { <processlist_id> | <query_id> } ]
```

Required Parameters

1. `<processlist_id>`

> Indicates the ID of the connection thread that needs to be killed

2. `<query_id>`

> Indicates the ID of the query to be killed

Optional Parameters

1. `CONNECTION`

> Indicates whether the thread is currently connected

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
GRANT_PRIV	DATABASE	GRANT permission is required for the KILL statement

Usage Notes

- The thread process list identifier can be queried from the Id column output by the SHOW PROCESSLIST
- The Connection ID can be queried from SELECT CONNECTION_ID()

Examples

```
select connection_id()
```

```
+-----------------+
| connection_id() |
+-----------------+
| 48              |
+-----------------+
```

```
SHOW PROCESSLIST
```

```
+------------------+------+------+--------------------+---------------------+----------+---------+---------+------+-------+---------
-------------------------+-----------------------------------------------------------------------------------------------+
| CurrentConnected | Id   | User | Host               | LoginTime           | Catalog  | Db      | Command | Time | State | QueryId
| Info                                                                                                                  |
+------------------+------+------+--------------------+---------------------+----------+---------+---------+------+-------+---------
-------------------------+-----------------------------------------------------------------------------------------------+
| Yes              |   48 | root | 10.16.xx.xx:44834  | 2025-01-21 16:49:47 | internal | test | Query   |    0 | OK    |
e6e4ce9567b04859-8eeab8d6b5513e38 | SHOW PROCESSLIST                                                                                                      |
|                  |   50 | root | 192.168.xx.xx:52837 | 2025-01-21 16:51:34 | internal |      | Sleep   | 1837 | EOF   |
deaf13c52b3b4a3b-b25e8254b50ff8cb | SELECT @@session.transaction_isolation                                                                                |
|                  |   51 | root | 192.168.xx.xx:52843 | 2025-01-21 16:51:35 | internal |      | Sleep   |  907 | EOF   |
437f219addc0404f-9befe7f6acf9a700 | /* ApplicationName=DBeaver Ultimate 23.1.3 - Metadata */ SHOW STATUS                                                   |
|                  |   55 | root | 192.168.xx.xx:55533 | 2025-01-21 17:09:32 | internal | test | Sleep   |  271 | EOF   |
f02603dc163a4da3-beebbb5d1ced760c | /* ApplicationName=DBeaver Ultimate 23.1.3 - SQLEditor <Console> */ SELECT DATABASE()                                 |
|                  |   47 | root | 10.16.xx.xx:35678  | 2025-01-21 16:21:56 | internal | test | Sleep   | 3528 | EOF   |
f4944c543dc34a99-b0d0f3986c8f1c98 | select * from test                                                                                                    |
+------------------+------+------+--------------------+---------------------+----------+---------+---------+------+-------+---------
-------------------------+-----------------------------------------------------------------------------------------------+
```

```
kill 51
```

CLEAN ALL PROFILE

Description

This command is used to manually clear all historical query or load profiles.

Syntax

```
CLEAN ALL PROFILE
```

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
GRANT_PRIV	DATABASE	GRANT permission is required for the CLEAN statement

Examples

```
CLEAN ALL PROFILE
```

SHOW QUERY STATS

Description

This statement is used to display the database table columns hit by the historical query

Syntax

```
SHOW QUERY STATS [ { [FOR <db_name>] | [FROM <table_name>] } ] [ALL] [VERBOSE]];
```

Optional Parameters

1. `<db_name>`

> If this parameter is set, the matching information of the database is displayed

2. `<table_name>`

> If this parameter is set, the matching status of a table is queried

3. `ALL`

> ALL Specifies whether to display the matching information of all indexes

4. `VERBOSE`

> VERBOSE displays detailed matching information

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
SELECT_PRIV	DATABASE	You must have the SELECT permission on the queried database

Usage Notes

- Query the historical query matching status of databases and tables. Data is reset after fe is restarted, and statistics are collected for each fe.
- You can use FOR DATABASE and FROM TABLE to specify the matching information of the database or table to be queried, followed by the database name or table name.
- The ALL and VERBOSE parameters can be used alone or together, but they must be used last and only for table queries.
- If no databases are being used, then simply executing 'SHOW QUERY STATS' will show the hits of all databases.
- There may be two columns in the hit result: QueryCount indicates the number of times the column has been queried, and FilterCount indicates the number of times the column has been queried as the where condition.

Examples

```
show query stats from baseall
```

```
+-------+------------+-------------+
| Field | QueryCount | FilterCount |
+-------+------------+-------------+
| k0    | 0          | 0           |
| k1    | 0          | 0           |
| k2    | 0          | 0           |
| k3    | 0          | 0           |
| k4    | 0          | 0           |
| k5    | 0          | 0           |
| k6    | 0          | 0           |
| k10   | 0          | 0           |
| k11   | 0          | 0           |
| k7    | 0          | 0           |
| k8    | 0          | 0           |
| k9    | 0          | 0           |
| k12   | 0          | 0           |
| k13   | 0          | 0           |
+-------+------------+-------------+
```

```
select k0, k1,k2, sum(k3) from baseall  where k9 > 1 group by k0,k1,k2
```

```
+------+------+--------+-------------+
| k0   | k1   | k2     | sum(`k3`)   |
+------+------+--------+-------------+
|    0 |    6 |  32767 |        3021 |
|    1 |   12 |  32767 | -2147483647 |
|    0 |    3 |   1989 |        1002 |
|    0 |    7 | -32767 |        1002 |
|    1 |    8 |    255 |  2147483647 |
|    1 |    9 |   1991 | -2147483647 |
|    1 |   11 |   1989 |       25699 |
|    1 |   13 | -32767 |  2147483647 |
|    1 |   14 |    255 |         103 |
```

```
|    0 |    1 |   1989 |        1001 |
|    0 |    2 |   1986 |        1001 |
|    1 |   15 |   1992 |        3021 |
+------+------+--------+-------------+
```

```
show query stats from baseall;
```

```
+-------+------------+-------------+
| Field | QueryCount | FilterCount |
+-------+------------+-------------+
| k0    | 1          | 0           |
| k1    | 1          | 0           |
| k2    | 1          | 0           |
| k3    | 1          | 0           |
| k4    | 0          | 0           |
| k5    | 0          | 0           |
| k6    | 0          | 0           |
| k10   | 0          | 0           |
| k11   | 0          | 0           |
| k7    | 0          | 0           |
| k8    | 0          | 0           |
| k9    | 1          | 1           |
| k12   | 0          | 0           |
| k13   | 0          | 0           |
+-------+------------+-------------+
```

```
show query stats from baseall all
```

```
+-----------+------------+
| IndexName | QueryCount |
+-----------+------------+
| baseall   | 1          |
+-----------+------------+
```

```
show query stats from baseall all verbose
```

```
+-----------+-------+------------+-------------+
| IndexName | Field | QueryCount | FilterCount |
+-----------+-------+------------+-------------+
| baseall   | k0    | 1          | 0           |
|           | k1    | 1          | 0           |
|           | k2    | 1          | 0           |
|           | k3    | 1          | 0           |
|           | k4    | 0          | 0           |
|           | k5    | 0          | 0           |
|           | k6    | 0          | 0           |
|           | k10   | 0          | 0           |
|           | k11   | 0          | 0           |
|           | k7    | 0          | 0           |
|           | k8    | 0          | 0           |
|           | k9    | 1          | 1           |
|           | k12   | 0          | 0           |
|           | k13   | 0          | 0           |
+-----------+-------+------------+-------------+
```

```
show query stats for test_query_db
```

```
+----------------------------+------------+
| TableName                  | QueryCount |
+----------------------------+------------+
| compaction_tbl             | 0          |
| bigtable                   | 0          |
| empty                      | 0          |
| tempbaseall                | 0          |
| test                       | 0          |
| test_data_type             | 0          |
| test_string_function_field | 0          |
| baseall                    | 1          |
| nullable                   | 0          |
+----------------------------+------------+
```

```
show query stats
```

```
+-----------------+------------+
| Database        | QueryCount |
+-----------------+------------+
| test_query_db   | 1          |
+-----------------+------------+
```

CLEAN ALL QUERY STATS

Description

This statement is used to clear query statistics

Syntax

```
CLEAN [ { ALL| DATABASE | TABLE } ] QUERY STATS [ { [ FOR <db_name>] | [ { FROM | IN } ] <table_name>]];
```

Required Parameters

1. `ALL`

> Use ALL to clear all query statistics

2. `DATABASE`

> Use DATABASE to clear database query statistics

3. `TABLE`

> Use TABLE to clear table query statistics

Optional Parameters

1. `<db_name>`

> If this parameter is set, the statistics of the corresponding database are cleared

2. `<table_name>`

> If this parameter is set, the statistics of the corresponding table are cleared

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	ALL	If ALL is specified, the ADMIN permission is required
ALTER_PRIV	DATABASE	If the database is specified, the ALTER permission for the corresponding database is required
ADMIN_PRIV	TABLE	If you specify a table, you need alter permission for that table

Examples

```
clean all query stats
```

```
clean database query stats for test_query_db
```

```
clean table query stats from test_query_db.baseall
```

KILL CONNECTION

Description

Kill a connection with a specified connection ID. This will also kill the query associated with this connection.

Syntax

```
KILL [ CONNECTION ] <connection_id>
```

Required Parameters

> The ID of the connection. It can be queried through the SHOW PROCESSLIST statement.

Access Control Requirements

The user executing this SQL command must be the user who owns the connection or have at least `ADMIN_PRIV` permission.

Examples

Query `connection_id`:

```
show processlist;
```

Result:

```
+------------------+----+------+-----------------+---------------------+----------+------+---------+------+-------+-----------------
------------------+------------------+--------------+--------------+
| CurrentConnected | Id | User | Host            | LoginTime           | Catalog  | Db   | Command | Time | State | QueryId
| Info             | FE           | CloudCluster |
+------------------+----+------+-----------------+---------------------+----------+------+---------+------+-------+-----------------
------------------+------------------+--------------+--------------+
| Yes              | 16 | root | 127.0.0.1:63746 | 2024-11-04 20:18:07 | internal | test | Query   | 0    | OK    |
e4d69a1cce81468d-91c9ae32b17540e9 | show processlist | 172.16.123.1 | NULL         |
+------------------+----+------+-----------------+---------------------+----------+------+---------+------+-------+-----------------
------------------+------------------+--------------+--------------+
```

Send KILL command:

```
KILL CONNECTION 16;
```

BEGIN

Description

Users can specify a Label. If not specified, the system will generate a Label automatically.

Syntax

```
BEGIN [ WITH LABEL <label> ]
```

Optional Parameter

`[ WITH LABEL <label> ]`

> Explicitly specify the Label associated with the transaction. If not specified, the system will generate a label automatically.

Notes

- If an explicit transaction is started without a commit or rollback, executing the BEGIN command again will not take effect.

Examples

Start an explicit transaction using a system-generated Label

```
mysql> BEGIN;
{'label':'txn_insert_624a0e16ef4c43d4-9814c7fa3e83a705', 'status':'PREPARE', 'txnId':''}
```

Start an explicit transaction with a specified Label

```
mysql> BEGIN WITH LABEL load_1;
{'label':'load_1', 'status':'PREPARE', 'txnId':''}
```

COMMIT

Description

Submit an explicit transaction, used in pairs with BEGIN.

Syntax

```
COMMIT
```

Notes

- If an explicit transaction is not enabled, executing this command will not take effect.

Example

The following example creates a table named `test`, starts a transaction, inserts two rows of data, commits the transaction, and then executes a query.

```
CREATE TABLE `test` (
  `ID` int NOT NULL,
  `NAME` varchar(100) NULL,
  `SCORE` int NULL
) ENGINE=OLAP
DUPLICATE KEY(`ID`)
DISTRIBUTED BY HASH(`ID`) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 3"
);

BEGIN;
INSERT INTO test VALUES(1, 'Alice', 100);
INSERT INTO test VALUES(2, 'Bob', 100);
COMMIT;
SELECT * FROM test;
```

ROLLBACK

Description

Rollback an explicit transaction. It is used in pairs with BEGIN.

Syntax

```
ROLLBACK
```

Usage Notes

- If an explicit transaction is not started, executing this command will not take effect.

Examples

The following example creates a table named `test`, starts a transaction, inserts two rows of data, rolls back the transaction, and then executes a query.

```
CREATE TABLE `test` (
  `ID` int NOT NULL,
  `NAME` varchar(100) NULL,
  `SCORE` int NULL
) ENGINE=OLAP
DUPLICATE KEY(`ID`)
DISTRIBUTED BY HASH(`ID`) BUCKETS 1
PROPERTIES (
    "replication_allocation" = "tag.location.default: 3"
);

BEGIN;
INSERT INTO test VALUES(1, 'Bob', 100);
INSERT INTO test VALUES(2, 'Bob', 100);
ROLLBACK;
SELECT * FROM test;
```

SHOW TRANSACTION

Description

This syntax is used to view transaction details for the specified transaction id or label.

Syntax

```
SHOW TRANSACTION
[FROM <db_name>]
WHERE
[id = <transaction_id> | label = <label_name>];
```

Required Parameters

1. `<transaction_id>`

The transaction ID whose details need to be viewed.

2. `<label_name>`

The label whose transaction details need to be viewed.

Optional Parameters

1. `<db_name>`

The database whose transaction details need to be viewed.

Return Value

Column Name	Description
TransactionId	Transaction ID
Label	Label associated with the import task
Coordinator	Node responsible for coordinating the transaction
TransactionStatus	Status of the transaction
PREPARE	Preparation phase
COMMITTED	Transaction succeeded, but data is not visible yet
VISIBLE	Transaction succeeded, and data is visible
ABORTED	Transaction failed
LoadJobSourceType	Type of the import task
PrepareTime	Start time of the transaction

Column Name	Description
CommitTime	Time when the transaction was successfully committed
FinishTime	Time when the data became visible
Reason	Error message
ErrorReplicasCount	Number of replicas with errors
ListenerId	ID of the related import job
TimeoutMs	Transaction timeout duration in milliseconds

Access Control Requirements

Privilege	Object	Notes
LOAD_PRIV	Database	

Examples

1. View the transaction with id 4005:

```
SHOW TRANSACTION WHERE ID=4005;
```

2. In the specified db, view the transaction with id 4005:

```
SHOW TRANSACTION FROM db WHERE ID=4005;
```

3. View the transaction whose label is label_name:

```
SHOW TRANSACTION WHERE LABEL = 'label_name';
```

CREATE CATALOG

Description

This statement is used to create an external catalog

Syntax

```
CREATE CATALOG [IF NOT EXISTS] <catalog_name> [ COMMENT "<comment>"]
    PROPERTIES ("<key>"="<value>" [, ... ]);
```

Required Parameters

1. `<catalog_name>` The name of the catalog to be created.

2. "`<key>`"="`<value>`" The parameters for creating the catalog.

Optional Parameters

1. `<comment>` The comment for the catalog to be created.

Access Control Requirements

Privilege	Object	Notes
CREATE_PRIV	Catalog	The CREATE_PRIV permission for the corresponding catalog is required.

Examples

1. Create catalog hive

```
CREATE CATALOG hive comment 'hive catalog' PROPERTIES (
    'type'='hms',
    'hive.metastore.uris' = 'thrift://127.0.0.1:7004',
    'dfs.nameservices'='HANN',
    'dfs.ha.namenodes.HANN'='nn1,nn2',
    'dfs.namenode.rpc-address.HANN.nn1'='nn1_host:rpc_port',
    'dfs.namenode.rpc-address.HANN.nn2'='nn2_host:rpc_port',
    'dfs.client.failover.proxy.provider.HANN'='org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider'
);
```

2. Create catalog es

```
CREATE CATALOG es PROPERTIES (
    "type"="es",
    "hosts"="http://127.0.0.1:9200"
);
```

3. Create catalog jdbc

mysql

```
CREATE CATALOG jdbc PROPERTIES (
    "type"="jdbc",
    "user"="root",
    "password"="123456",
    "jdbc_url" = "jdbc:mysql://127.0.0.1:3316/doris_test?useSSL=false",
    "driver_url" = "https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.25/mysql-connector-java-8.0.25.jar",
    "driver_class" = "com.mysql.cj.jdbc.Driver"
);
```

postgresql

```
CREATE CATALOG jdbc PROPERTIES (
    "type"="jdbc",
    "user"="postgres",
    "password"="123456",
    "jdbc_url" = "jdbc:postgresql://127.0.0.1:5432/demo",
    "driver_url" = "file:///path/to/postgresql-42.5.1.jar",
    "driver_class" = "org.postgresql.Driver"
);
```

clickhouse

```
CREATE CATALOG jdbc PROPERTIES (
    "type"="jdbc",
    "user"="default",
    "password"="123456",
    "jdbc_url" = "jdbc:clickhouse://127.0.0.1:8123/demo",
    "driver_url" = "file:///path/to/clickhouse-jdbc-0.3.2-patch11-all.jar",
    "driver_class" = "com.clickhouse.jdbc.ClickHouseDriver"
)
```

oracle `sql CREATE CATALOG jdbc PROPERTIES ( "type"="jdbc", "user"="doris", "password"="123456", "jdbc_url" = "jdbc:oracle:thin:@127.0.0.1:1521:helowin", "driver_url" = "file:///path/to/ojdbc8.jar", "driver_class" = "oracle.jdbc.driver.OracleDriver" );`

SQLServer `sql CREATE CATALOG sqlserver_catalog PROPERTIES ("type"="jdbc", "user"="SA", "password"="Doris123456", "jdbc_url" = "jdbc:sqlserver://localhost:1433;DataBaseName=doris_test", "driver_url" = "file:///path/to/mssql-jdbc-11.2.3.jre8.jar", "driver_class" = "com.microsoft.sqlserver.jdbc.SQLServerDriver");`

SAP HANA `sql CREATE CATALOG saphana_catalog PROPERTIES ( "type"="jdbc", "user"="SYSTEM", "password"="SAPHANA", "jdbc_url" = "jdbc:sap://localhost:31515/TEST", "driver_url" = "file:///path/to/ngdbc.jar", "driver_class" = "com.sap.db.jdbc.Driver" );`

Trino `sql CREATE CATALOG trino_catalog PROPERTIES ( "type"="jdbc", "user"="hadoop", "password"="", "jdbc_url" = "jdbc:trino://localhost:8080/hive", "driver_url" = "file:///path/to/trino-jdbc-389.jar", "driver_class" = "io.trino.jdbc.TrinoDriver" );`

OceanBase `sql CREATE CATALOG oceanbase_catalog PROPERTIES ( "type"="jdbc", "user"="root", "password"="", "jdbc_url" = "jdbc:oceanbase://localhost:2881/demo", "driver_url" =`

```
"file:///path/to/oceanbase-client-2.4.2.jar", "driver_class" = "com.oceanbase.jdbc.Driver" );
```

ALTER CATALOG

Description

This statement is used to set properties of the specified catalog.

Syntax

1. Rename the catalog

```
ALTER CATALOG <catalog_name> RENAME <new_catalog_name>;
```

2. Modify / Add properties for the catalog

```
ALTER CATALOG <catalog_name> SET PROPERTIES ('<key>' = '<value>' [, ... ]);
```

3. Modify comment for the catalog

```
ALTER CATALOG <catalog_name> MODIFY COMMENT "<new catalog comment>";
```

Required Parameters

1. `<catalog_name>`

The name of the catalog that should be modified

2. `<new_catalog_name>`

New catalog name after modification

3. `'<key>' = '<value>'`

The key and value of the catalog properties that need to be modified / added

4. `<new catalog comment>`

Modified catalog comment

Access Control Requirements

Privilege	Object	Notes
ALTER_PRIV	Catalog	The ALTER_PRIV of the catalog is required

Usage Notes

1. Rename the catalog

- The builtin catalog `internal` cannot be renamed
- Only the one who has at least Alter privilege can rename a catalog
- After renaming the catalog, use the REVOKE and GRANT commands to modify the appropriate user permissions

2. Modify / Add properties for the catalog

- property `type` cannot be modified.
- properties of builtin catalog `internal` cannot be modified.
- Update values of specified keys. If a key does not exist in the catalog properties, it will be added.

3. Modify comment for the catalog

- The builtin catalog `internal` cannot be modified

Example

1. rename catalog ctlg_hive to hive

```
ALTER CATALOG ctlg_hive RENAME hive;
```

2. modify property `hive.metastore.uris` of catalog hive

```
ALTER CATALOG hive SET PROPERTIES ('hive.metastore.uris'='thrift://172.21.0.1:9083');
```

3. modify comment of catalog hive `sql ALTER CATALOG hive MODIFY COMMENT "new catalog comment";`

DROP CATALOG

Description

This statement is used to delete the external catalog.

Syntax

```
DROP CATALOG [IF EXISTS] <catalog_name>;
```

Required Parameters

1. `<catalog_name>` The name of the catalog to be dropped.

Access Control Requirements

Privilege	Object	Notes
DROP_PRIV	Catalog	The DROP_PRIV permission for the corresponding catalog is required.

Example

1. Drop catalog hive

    ```
    DROP CATALOG hive;
    ```

SHOW CATALOG

Description

Shows properties of specified catalog

Syntax

```
SHOW CATALOG <catalog_name>
```

Required Parameters

1. `<catalog_name>`

The name of the catalog to be displayed.

Return Value

Column Name	Description
Key	Configuration property name.
Value	Configuration property value.

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV / SELECT_PRIV / LOAD_PRIV / ALTER_PRIV / CREATE_PRIV / SHOW_VIEW_PRIV / DROP_PRIV	Catalog	One of the above permissions is required.

Examples

1. View specific catalog configuration details

```
show catalog test_mysql;
```

```
+---------------+-----------------------------------------------------------------------------------------------------------------------------------------+
| Key           | Value                                                                                                                                   |
+---------------+-----------------------------------------------------------------------------------------------------------------------------------------+
| checksum      | fdf55dcef04b09f2eaf42b75e61ccc9a                                                                                                        |
| create_time   | 2025-02-17 17:21:13.099                                                                                                                 |
```

```
| driver_class   | com.mysql.cj.jdbc.Driver
|
| driver_url     | mysql-connector-j-8.3.0.jar
|
| jdbc_url       | jdbc:mysql://127.0.0.1:3306/db
|
| password       | *XXX
|
| type           | jdbc
|
| use_meta_cache | true
|
| user           | root
|
+----------------+-----------------------------------------------------------------------------------------------------------
-------------------------------------------+
```

SHOW CATALOGS

Description

This statement is used for view created catalogs

Syntax

```
SHOW CATALOGS [LIKE <catalog_name>]
```

illustrate:

1. LIKE: Fuzzy query can be performed according to the catalog name

Optional Parameters

1. `<catalog_name>`

The name of the catalog to be displayed.

Return Value

Column Name	Description
CatalogId	Unique ID of the data catalog
CatalogName	Name of the data catalog. The default built-in catalog is named “internal” and cannot be modified.
Type	Type of the data catalog
IsCurrent	Indicates whether it is the currently active data catalog
CreateTime	Creation time
LastUpdateTime	Last updated time
Comment	comments about the catalog

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV / SELECT_PRIV / LOAD_PRIV / ALTER_PRIV / CREATE_PRIV / SHOW_VIEW_PRIV / DROP_PRIV	Catalog	One of the above permissions is required.

Examples

1. View the data catalogs that have been created currently

```
SHOW CATALOGS;
```

```
 +-----------+-------------+----------+-----------+-------------------------+---------------------+------------------------
+
 | CatalogId | CatalogName | Type     | IsCurrent | CreateTime              | LastUpdateTime      | Comment
|
 +-----------+-------------+----------+-----------+-------------------------+---------------------+------------------------
+
 |    130100 | hive        | hms      |           | 2023-12-25 16:11:41.687 | 2023-12-25 20:43:18 | NULL
|
 |         0 | internal    | internal | yes       | UNRECORDED              | NULL                | Doris internal catalog
|
 +-----------+-------------+----------+-----------+-------------------------+---------------------+------------------------
+
```

2. Fuzzy query by catalog name

```
SHOW CATALOGS LIKE 'hi%';
```

```
+-----------+-------------+----------+-----------+-------------------------+---------------------+------------------------+
| CatalogId | CatalogName | Type     | IsCurrent | CreateTime              | LastUpdateTime      | Comment                |
+-----------+-------------+----------+-----------+-------------------------+---------------------+------------------------+
|    130100 | hive        | hms      |           | 2023-12-25 16:11:41.687 | 2023-12-25 20:43:18 | NULL                   |
+-----------+-------------+----------+-----------+-------------------------+---------------------+------------------------+
```

SHOW CREATE CATALOG

Description

This statement shows the creating statement of a doris catalog.

Syntax

```
SHOW CREATE CATALOG <catalog_name>;
```

Required Parameters

1. `<catalog_name>`

The name of the catalog whose creation statement needs to be viewed.

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV / SELECT_PRIV / LOAD_PRIV / ALTER_PRIV / CREATE_PRIV / SHOW_VIEW_PRIV / DROP_PRIV	Catalog	One of the above permissions is required.

Examples

1. View the creating statement of the hive oracle in doris

```
SHOW CREATE CATALOG oracle;
```

```
+---------+-----------------------------------------------------------------------------------------------------------------
------+
 | Catalog | CreateCatalog
|
 +---------+-----------------------------------------------------------------------------------------------------------------
-------+
 | oracle  |
 CREATE CATALOG `oracle` PROPERTIES (
 "user" = "XXX",
 "type" = "jdbc",
 "password" = "*XXX",
 "jdbc_url" = "XXX",
 "driver_url" = "XXX",
 "driver_class" = "oracle.jdbc.driver.OracleDriver",
 "checksum" = "XXX"
 ); |
 +---------+-----------------------------------------------------------------------------------------------------------------
-------+
```

REFRESH

Description

This statement refreshes the metadata of the specified Catalog/Database/Table.

Syntax

```
REFRESH CATALOG <catalog_name>;
REFRESH DATABASE [<catalog_name>.]<database_name>;
REFRESH TABLE [[<catalog_name>.]<database_name>.]<table_name>;
```

Required Parameters

1. `<catalog_name>`

The name of the catalog that needs to be refreshed.

2. `[<catalog_name>.]<database_name>`

The name of the database within the catalog that needs to be refreshed.

3. `[[<catalog_name>.]<database_name>.]<table_name>`

The name of the table within the catalog that needs to be refreshed.

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV / SELECT_PRIV / LOAD_PRIV / ALTER_PRIV / CREATE_PRIV / SHOW_VIEW_PRIV / DROP_PRIV	Catalog	One of the above permissions is required.

Usage Notes

When the Catalog is refreshed, the object-related Cache is forced to be invalidated. Including Partition Cache, Schema Cache, File Cache, etc.

Example

1. Refresh hive catalog

```
REFRESH CATALOG hive;
```

2. Refresh database1

```
REFRESH DATABASE ctl.database1;
REFRESH DATABASE database1;
```

3. Refresh table1

```
REFRESH TABLE ctl.db.table1;
REFRESH TABLE db.table1;
REFRESH TABLE table1;
```

CREATE DATABASE

Description

This statement is used to create a new database

Syntax

```
CREATE DATABASE [IF NOT EXISTS] <db_name>
    [PROPERTIES ("<key>"="<value>"[, ... ])];
```

Required parameters

** 1. <db_name>** > Database Name

Optional parameters

** 1. <PROPERTIES>** > Additional information about this database

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
CREATE_PRIV	Corresponding database	You need to have the create permission for the corresponding database

Precautions

If you want to specify the default replica distribution strategy for the table under db, you need to specify <replication_allocation> (the <replication_allocation> attribute of table has a higher priority than db):

```
PROPERTIES (
  "replication_allocation" = "tag.location.default:3"
)
```

Example

- Create a new database db_test

  ```
  CREATE DATABASE db_test;
  ```

- Create a new database and set the default replica distribution:

```
CREATE DATABASE `db_test`
PROPERTIES (
 "replication_allocation" = "tag.location.group_1:3"
);
```

ALTER DATABASE

Description

This statement is used to set the properties of a specified db, change the db name, and set various quotas for the db.

Syntax

```
ALTER DATABASE <db_name> RENAME <new_name>
ALTER DATABASE <db_name> SET { DATA | REPLICA | TRANSACTION } QUOTA <quota>
ALTER DATABASE <db_name> SET <PROPERTIES> ("<key>" = "<value>" [, ...])
```

Required parameters

** 1. <db_name>** > Specifies the identifier for the database to alter.

** 2. <new_db_name>** > Specifies the new identifier for the database

** 3. <quota>** > Database data volume quota or database replica number quota

** 4. <PROPERTIES>** > Additional information about this database

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
ALTER_PRIV	Corresponding database	You need to have the permission to change the corresponding database.

Precautions

After renaming the database, use the REVOKE and GRANT commands to modify the corresponding user permissions if necessary. The default data volume quota for a database is 1024 GB, and the default replica number quota is 1073741824.

Example

- Set the data volume quota for the specified database

  ```
  ALTER DATABASE example_db SET DATA QUOTA 10995116277760;
  ```

- Rename the database example_db to example_db2

```
ALTER DATABASE example_db RENAME example_db2;
```

- Set a quota for the number of copies of a specified database

```
ALTER DATABASE example_db SET REPLICA QUOTA 102400;
```

- Modify the default replica distribution strategy of the table under db (this operation is only effective for newly created tables and will not modify existing tables under db)

```
ALTER DATABASE example_db SET PROPERTIES("replication_allocation" = "tag.location.default:2");
```

- Cancel the default replica distribution policy of the table under db (this operation is only effective for newly created tables and will not modify existing tables under db)

```
ALTER DATABASE example_db SET PROPERTIES("replication_allocation" = "");
```

DROP DATABASE

Description

This statement is used to delete a database.

Syntax

```
DROP DATABASE [IF EXISTS] <db_name> [FORCE];
```

Required parameters

** 1. <db_name>** > Database Name

Optional parameters

** 1. FORCE** > Force deletion without going to the Recycle Bin

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
DROP_PRIV	Corresponding database	You need to have delete permission on the corresponding database

Precautions

If you execute DROP DATABASE FORCE, the system will not check whether there are any unfinished transactions in the database. The database will be deleted directly and cannot be restored. This operation is generally not recommended.

Example

- Deleting a Database db_test

```
DROP DATABASE db_test;
```

SHOW CREATE DATABASE

Description

This statement checks the creation information of the doris built-in database or catalog database.

Syntax

```
SHOW CREATE DATABASE [<catalog>.]<db_name>;
```

Required parameters

** 1. <db_name>** > Database Name

Optional parameters

** 1. <catalog>** > Indicates whether the table is internal or external

Return Value

Column	Description
Database	Database Name
Create Database	Corresponding database creation statement

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
SELECT_PRIV	Corresponding database	Requires read permission on the corresponding database

Example

- View the creation of the test database in doris

```
SHOW CREATE DATABASE test;
```

```
+----------+-----------------------+
| Database | Create Database       |
+----------+-----------------------+
| test     | CREATE DATABASE `test` |
+----------+-----------------------+
```

- View the creation information of the database hdfs_text in the hive catalog

```
SHOW CREATE DATABASE hdfs_text;
```

```
+-----------+-----------------------------------------------------------------------------------+
| Database  | Create Database                                                                   |
+-----------+-----------------------------------------------------------------------------------+
| hdfs_text | CREATE DATABASE `hdfs_text` LOCATION 'hdfs://HDFS1009138/hive/warehouse/hdfs_text' |
+-----------+-----------------------------------------------------------------------------------+
```

SHOW DATABASES

Description

This statement is used to display the currently visible database.

Syntax

```
SHOW DATABASES [FROM <catalog>] [<filter_expr>];
```

Optional parameters

** 1. `<catalog>`** > Corresponding catalog

** 2. `<filter_expr>`** > Filter by specified conditions

Return Value

Column	Description
Database	Database Name

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
SELECT_PRIV	Corresponding database	Requires read permission on the corresponding database

Examples

- Displays the names of all current databases.

  ```
  SHOW DATABASES;
  ```

  ```
  +--------------------+
  | Database           |
  +--------------------+
  | test               |
  | information_schema |
  +--------------------+
  ```

- Will display all database names in `hms_catalog`.

```
SHOW DATABASES FROM hms_catalog;
```

```
+---------------+
| Database      |
+---------------+
| default       |
| tpch          |
+---------------+
```

- Displays the names of all databases currently filtered by the expression `like 'infor%'`.

```
SHOW DATABASES like 'infor%';
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+
```

SHOW-DATABASE-ID

Description

This statement is used to find the corresponding database name based on the database id (only used by administrators)

Syntax

```
SHOW DATABASE <database_id>
```

Required parameters

** 1. <database_id>** > Database corresponding id number

Return Value

Column	Description
DbName	Database Name

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
ADMIN_PRIV	The entire cluster	Requires administrative privileges for the entire cluster

Example

- Find the corresponding database name according to the database id

```
SHOW DATABASE 10396;
```

```
+------------+
| DbName     |
+------------+
| example_db |
+------------+
```

CREATE TABLE

Description

Create a new table in the current or specified database. A table can have multiple columns, and each column definition includes the name, data type, and optionally the following attributes:

- Whether it is a key
- Whether it has aggregation semantics
- Whether it is a generated column
- Whether it requires a value (NOT NULL)
- Whether it is an auto-increment column
- Whether there is a default value upon insertion
- Whether there is a default value upon update

Furthermore, this command also supports the following variations:

- CREATE TABLE … AS SELECT (to create a table pre-populated with data; also known as CTAS)
- CREATE TABLE … LIKE (to create an empty copy of an existing table)

Syntax

```
CREATE [ EXTERNAL ] TABLE [ IF NOT EXISTS ] <table_name>
    (<columns_definition> [ <indexes_definition> ])
    [ ENGINE = <table_engine_type> ]
    [ <key_type> KEY (<key_cols>)
        [ CLUSTER BY (<cluster_cols>) ]
    ]
    [ COMMENT '<table_comment>' ]
    [ <partitions_definition> ]
    [ DISTRIBUTED BY { HASH (<distribute_cols>) | RANDOM }
        [ BUCKETS { <bucket_count> | AUTO } ]
    ]
    [ <roll_up_definition> ]
    [ PROPERTIES (
          -- Table property
          <table_property>
          -- Additional table properties
          [ , ... ])
    ]
```

Where:

```
columns_definition
  : -- Column definition
    <col_name> <col_type>
      [ KEY ]
      [ <col_aggregate_type> ]
      [ [NOT] NULL ]
      [ AUTO_INCREMENT(<col_auto_increment_start_value>) ]
      [ DEFAULT <col_default_value> ]
      [ ON UPDATE CURRENT_TIMESTAMP (<col_on_update_precision>) ]
```

```
      [ COMMENT '<col_comment>' ]
    -- Additional column definitions
    [ , <col_name> <col_type> [ ... ] ]
```

```
indexes_definition
  : -- Index definition
    INDEX [ IF NOT EXISTS ]
      <index_name> (<index_cols>)
      [ USING <index_type> ]
      [ PROPERTIES (
            -- Table property
            <index_property>
            -- Additional table properties
            [ , ... ])
      ]
      [ COMMENT '<index_comment>' ]
    -- Additional index definitions
    [ , <index_name> (<index_cols>) [ ... ] ]
```

```
partitions_definition
  : AUTO PARTITION BY RANGE(<auto_partition_function>(<auto_partition_arguments>))
    <origin_partitions_definition>
  | AUTO PARTITION BY LIST(<partition_cols>)
    <origin_partitions_definition>
  | PARTITION BY <partition_type> (<partition_cols>)
    <origin_partitions_definition>
```

- Where:

  ```
  <origin_partitions_definition>
  : (
      -- Partition definition
      <one_partition_definition>
      -- Additional partition definition
      [ , ... ]
    )

  <one_partition_definition>
  : PARTITION [ IF NOT EXISTS ] <partition_name>
      VALUES LESS THAN <partition_value_list>
  | PARTITION [ IF NOT EXISTS ] <partition_name>
      VALUES [ <partition_lower_bound>, <partition_upper_bound>)
  | FROM <partition_lower_bound> TO <partition_upper_bound>
      INTERVAL <n> [ <datetime_unit> ]
  | PARTITION [ IF NOT EXISTS ] <partition_name>
      VALUES IN {
          (<partition_value> [, <partition_value> [ ... ] ])
          | <partition_value>
      }
  ```

```
roll_up_definition
  : ROLLUP (
        -- Rollup definition
        <rollup_name> (<rollup_cols>)
        [ DUPLICATE KEY (<duplicate_cols>) ]
        -- Additional rollup definition
        [ , <rollup_name> (<rollup_cols>) [ ... ] ]
    )
```

Varaint Syntax

CREATE TABLE … AS SELECT (Also Referred to as CTAS)

Generates a table and populates it with data returned from the `query`:

```
CREATE
    [ EXTERNAL ]
    TABLE [ IF NOT EXISTS ] <table_name>
    [ ( <column_definitions> ) ]
    [ <index_definitions> ]
    [ ENGINE = <storage_engine_type> ]
    [ <partitioning_key_type> KEY ( <key_columns> )
        [ CLUSTER BY ( <clustering_columns> ) ]
    ]
    [ COMMENT '<table_description>' ]
    [ <partition_definitions> ]
    [ DISTRIBUTED BY { HASH ( <distribution_columns> ) | RANDOM }
        [ BUCKETS { <number_of_buckets> | AUTO } ]
    ]
    [ <rollup_definitions> ]
    [ PROPERTIES (
          "<table_properties>"
          [ , ... ]
    ) ]
AS <query>;
```

CREATE TABLE … LIKE

Creates a new table with the same column definitions as an existing table, without copying data from the existing table. All properties of the columns will be replicated in the new table. If a list of `rollup` names is specified, the corresponding `rollup` from the original table will also be replicated:

```
CREATE TABLE <new_table_name> LIKE <existing_table_name>
[ WITH ROLLUP ( <rollup_list> ) ];
```

Required Parameters

> Specifies the identifier (i.e., name) of the table; it must be unique within the database where the table is created.
>
> The identifier must start with a letter (any language character if Unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> Identifiers cannot use reserved keywords.
>
> For more details, see Identifier Requirements and Reserved Keywords.

> Specifies the column identifier (i.e., name). It must be unique within the created table.

> The identifier must start with a letter (any language character if Unicode name support is enabled), a digit, or the symbol `@`, and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> For more details, see Identifier Requirements and Reserved Keywords.

> Specifies the data type of the column.
>
> For details on the data types that can be specified for table columns, see the Data Types section.

> A required parameter in CTAS. Specifies the SELECT statement that populates the data.

> A required parameter in CREATE TABLE … LIKE. Specifies the original table to be copied.

Optional Parameters

Data Model Related Parameters

> The data model of the table. Optional values are DUPLICATE (detail model), UNIQUE (primary key model), AGGREGATE (aggregation model). For details on data models, see the Data Model section.

> The key columns of the table. In Doris, Key columns must be the first K columns of the table. Data in a single tablet will be kept in order by these columns. For restrictions on Keys and how to choose Key columns, see the various subsections in the Data Model section.

> Data local sort columns, can only be used when the data model is UNIQUE (primary key model). When `<cluster_cols>` is specified, data is sorted by `<cluster_cols>` instead of using `<key_cols>`.

> The aggregation method of the column. Can only be used when the table is an aggregation model. For details on aggregation methods, see the Aggregation Model section.

Bucketing Related Parameters

and

> Bucketing columns and bucket counts. Detail model bucket columns can be any columns, aggregation model and primary key model bucket columns must be consistent with key columns. Bucket count is any positive integer. For details on bucketing, see the Manual Bucketing and Automatic Bucketing sections.

Column Default Value Related Parameters

AUTO_INCREMENT()

> When importing data, Doris will assign a unique value within the table to rows of data in the auto-increment column that do not specify a value. `<col_auto_increment_start_value>` specifies the starting value for the auto-increment column. For more details on auto-increment columns, see the Auto-Increment Columns section.

DEFAULT

> The default value for the column. When writing without including this column, this default value is used. If the default value is not explicitly set, NULL is used. Available default values include:
>
> - NULL: Available for all types, using NULL as the default value.
> - Numeric literals: Can only be used for numeric types.
> - String literals: Can only be used for string types.
> - CURRENT_DATE: Can only be used for date types. Uses the current date as the default value.
> - CURRENT_TIMESTAMP [ ]: Can only be used for datetime types. Uses the current timestamp as the default value. `<defaultValuePrecision>` can specify the time precision.
> - PI: Can only be used for double types. Uses pi as the default value.
> - E: Can only be used for double types. Uses the mathematical constant e as the default value.
> - BITMAP_EMPTY: Can only be used when the column is of bitmap type. Fills an empty bitmap.

ON UPDATE CURRENT_TIMESTAMP ()

> When data is updated, if no value is specified for this column, the current timestamp is used to update the data in this column. Can only be used on tables with a UNIQUE (primary key model).

Index Related Parameters

> Specifies the index identifier (i.e., name). It must be unique within the created table. For more details on identifiers, see Identifier Requirements and Reserved Keywords.

> A list of columns to add an index to. Must be existing columns in the table.

> The type of index. Currently, only INVERTED is supported.

> The properties of the index. For detailed explanations, refer to the Inverted Index section.

Auto Partition Related Parameters

For a detailed introduction to partitioning, see the Automatic Partitioning section.

Manual Partitioning Related Parameters

For a detailed introduction to partitioning, see the “Manual Partitioning” section.

> Doris supports RANGE partitioning and LIST partitioning. For details, see the Manual Partitioning section.

> The partition identifier (i.e., name). It must be unique within the created table. For more details on identifiers, see Identifier Requirements and Reserved Keywords.

VALUES LESS THAN

> RANGE partitioning. The partition data range is from the lower bound to `<partition_value_list>`.
>
> If it represents the upper bound, `<partition_value_list>` can be simplified to `MAX_VALUE`.
>
> The format of `<partition_value_list>` is as follows: `((col_1_value, ...), (col_1_value, ...), ...)`

VALUES [,)

> RANGE partitioning. The partition data range is from `<partition_lower_bound>` to `<partition_upper_bound>`. Only one partition is created.
>
> The format of `<partition_lower_bound>` and `<partition_upper_bound>` is as follows: `(col_1_value, ...)`

FROM TO

INTERVAL [ ]

RANGE partitioning. The partition data range is from <partition_lower_bound> to <partition_value_list>. A partition is created every <n>.

The format of <partition_lower_bound> and <partition_upper_bound> is as follows: (col_1_value, ...)

VALUES IN {

([, [...]])

|

}

LIST partitioning. Rows where the partition column equals <partition_value> belong to this partition.

The format of <partition_value> is as follows: (col_1_value, ...)

Synchronized Materialized View Related

Note

The functionality of creating synchronized materialized views with rollup is limited and no longer recommended. It is advised to use separate statements to create synchronized materialized views. For details, please refer to the CREATE MATERIALIZED VIEW statement and the Synchronized Materialized View section.

The identifier (i.e., name) of the synchronized materialized view. It must be unique within the created table. For more details on identifiers, see Identifier Requirements and Reserved Keywords.

The columns included in the synchronized materialized view.

Table Property Related Parameters

Property Name	Function
replication_num	The number of replicas. The default number of replicas is 3. If the number of BE nodes is less than 3, you must specify a number of replicas less than or equal to the number of BE nodes. After version 0.15, this property will automatically convert to the `replication_allocation` property, e.g., `"replication_num" = "3"` will automatically convert to `"replication_allocation" = "tag.location.default:3"`.
replication_allocation	Sets the distribution of replicas based on Tags. This property can completely override the functionality of the `replication_num` property.
min_load_replica_num	Sets the minimum number of replicas required for a successful data import, with a default value of -1. When this property is less than or equal to 0, it indicates that the majority of replicas must still succeed for the data import.
is_being_synced	Used to identify whether this table is being replicated by CCR and is currently being synchronized by the syncer, with a default value of `false`. If set to `true`, the `colocate_with` and `storage_policy` properties will be cleared. The `dynamic partition and auto bucket` features will become ineffective. That is, they will appear enabled in `show create table` but will not actually take effect. When `is_being_synced` is set to `false`, these features will resume. This property is for use only by the CCR peripheral module and should not be manually set during the CCR synchronization process.
storage_medium	Declares the initial storage medium for table data.
storage_cooldown_time	Sets the expiration time for the initial storage medium of the table data. After this time, it will automatically downgrade to the first-level storage medium.
colocate_with	When the Colocation Join feature is needed, use this parameter to set the Colocation Group.
bloom_filter_columns	A list of column names specified by the user that require the addition of a Bloom Filter index. Each column's Bloom Filter index is independent and not a composite index. For example: `"bloom_filter_columns" = "k1, k2, k3"`
compression	The default compression method for Doris tables is LZ4. After version 1.1, support for specifying ZSTD as the compression method is available for higher compression ratios.

Property Name	Function
function_column.sequence_col	When using the Unique Key model, you can specify a Sequence column. When Key columns are the same, REPLACE will be performed according to the Sequence column (the larger value replaces the smaller value; otherwise, it cannot be replaced). `function_column.sequence_col` is used to map the sequence column to a specific column in the table, which can be of integer or date/time types (DATE, DATETIME). The type of this column cannot be changed after creation. If `function_column.sequence_col` is set, `function_column.sequence_type` will be ignored.
function_column.sequence_type	When using the Unique Key model, you can specify a Sequence column. When Key columns are the same, REPLACE will be performed according to the Sequence column (the larger value replaces the smaller value; otherwise, it cannot be replaced). Here, you only need to specify the type of the sequence column, which supports date/time types or integers. Doris will create a hidden sequence column.
enable_unique_key_merge_on_write	Whether the Unique table uses the Merge-on-Write implementation. This property was default disabled before version 2.1 and default enabled from version 2.1 onwards.
light_schema_change	Whether to use the Light Schema Change optimization. If set to `true`, addition and subtraction operations on value columns can be completed faster and synchronously. This feature is enabled by default in versions 2.0.0 and later.
disable_auto_compaction	Whether to disable automatic compaction for this table. If this property is set to `true`, the background automatic compaction process will skip all tablets of this table.
enable_single_replica_compaction	**Deprecated.** This feature has been removed since Doris 4.1.2 due to a data correctness risk in its peer selection, and enabling it is not recommended. Whether to enable single-replica compaction for this table. If this property is set to `true`, only one replica of all replicas of the table's tablets will perform the actual compaction action, and other replicas will pull the compacted rowset from that replica.
enable_duplicate_without_keys_by_default	When set to `true`, if no Unique, Aggregate, or Duplicate is specified when creating a table, a Duplicate model table without sort columns and prefix indexes will be created by default.

Property Name	Function
skip_write_index_on_load	Whether to enable not writing indexes during data import for this table. If this property is set to `true`, indexes will not be written during data import (currently only effective for inverted indexes), but will be delayed until compaction. This can avoid the CPU and IO resource consumption of writing indexes repeatedly during the first write and compaction, improving the performance of high-throughput imports.
compaction_policy	Configures the compaction merge policy for this table, supporting only time_series or size_basedtime_series: When the disk volume of rowsets accumulates to a certain size, version merging is performed. The merged rowset is directly promoted to the base compaction phase. This effectively reduces the write amplification of compact in scenarios with continuous imports. This policy will use parameters prefixed with time_series_compaction to adjust the execution of compaction.
time_series_compaction_goal_size_mbytes	When the compaction merge policy is time_series, this parameter is used to adjust the size of the input files for each compaction, with the output file size being comparable to the input.
time_series_compaction_file_count_threshold	When the compaction merge policy is time_series, this parameter is used to adjust the minimum number of input files for each compaction. If the number of files in a tablet exceeds this configuration, compaction will be triggered.
time_series_compaction_time_threshold_seconds	When the compaction merge policy is time_series, this parameter is used to adjust the longest interval between compactions, i.e., a compaction will be triggered if it has not been executed for a long time, in seconds.
time_series_compaction_level_threshold	When the compaction merge policy is time_series, this parameter defaults to 1. When set to 2, it is used to control that segments merged once are merged again to ensure that the segment size reaches time_series_compaction_goal_size_mbytes, achieving the effect of reducing the number of segments.
group_commit_interval_ms	Configures the Group Commit batch interval for this table. The unit is ms, with a default value of 10000ms, i.e., 10s. The timing of Group Commit depends on which of `group_commit_interval_ms` and `group_commit_data_bytes` reaches the set value first.
group_commit_data_bytes	Configures the Group Commit batch data size for this table. The unit is bytes, with a default value of 134217728, i.e., 128MB. The timing of Group Commit depends on which of `group_commit_interval_ms` and `group_commit_data_bytes` reaches the set value first.

Property Name	Function
enable_mow_light_delete	Whether to enable writing Delete predicate with Delete statements on Unique tables with Mow. If enabled, it will improve the performance of Delete statements, but partial column updates after Delete may result in some data errors. If disabled, it will reduce the performance of Delete statements to ensure correctness. The default value of this property is `false`. This property can only be enabled on Unique Merge-on-Write tables.
Dynamic Partitioning Related Properties	For dynamic partitioning, refer to Data Partitioning - Dynamic Partitioning

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Description
CREATE_PRIV	Database	
SELECT_PRIV	Table, View	Required when executing CTAS to have SELECT_PRIV on the queried table, view, or materialized view

Usage Notes

- The database (Database) must not contain tables (Table) or views (View) with the same name.
- Table names, column names, and rollup names must not use Reserved Keywords.
- CREATE TABLE … LIKE:
    - This command can only be used on internal Doris tables.
    - Only explicitly specified rollups will be copied.
    - All synchronized materialized views will not be replicated.
- CREATE TABLE … AS SELECT (CTAS):
    - If the alias of the column name in the SELECT list is a valid column, then column definitions are not required in the CTAS statement; if omitted, column names and data types will be inferred from the base query:

      ```
      CREATE TABLE <table_name> AS SELECT ...
      ```

    - Alternatively, you can explicitly specify names using the following syntax:

      ```
      CREATE TABLE <table_name> ( <col1_name>, <col2_name>, ... ) AS SELECT ...
      ```

- Partitioning and Bucketing
    - A table must specify bucketing columns but can opt out of specifying partitions. For detailed information on partitioning and bucketing, refer to the Data Partitioning documentation.
    - Tables in Doris can be either partitioned or non-partitioned. This attribute is determined at table creation and cannot be changed afterward. That is, for partitioned tables, partitions can be added

or removed in subsequent use, while non-partitioned tables cannot have partitions added later.
    - Partition and bucket columns cannot be altered after table creation; neither the types of partition and bucket columns can be changed nor can these columns be added or removed.
- Dynamic Partitioning
    - The dynamic partitioning feature is primarily used to help users manage partitions automatically. By setting certain rules, the Doris system periodically adds new partitions or removes old ones. For more assistance, refer to the Dynamic Partitioning documentation.
- Automatic Partitioning
    - Documentation for automatic partitioning can be found in Automatic Partitioning.
- Synchronized Materialized Views
    - Users can create multiple synchronized materialized views (ROLLUP) while creating a table. Synchronized materialized views can also be added after the table is created. Including them in the table creation statement facilitates the one-time creation of all synchronized materialized views.
    - If synchronized materialized views are created at the time of table creation, all subsequent data import operations will synchronously generate data for the materialized views. The number of materialized views may affect the efficiency of data import.
    - For an introduction to materialized views, please refer to the documentation on Synchronized Materialized Views.
- Indexes
    - Users can create multiple column indexes while creating a table. Indexes can also be added after the table is created.
    - If indexes are added during subsequent use and there is existing data in the table, all data will need to be rewritten; hence, the time to create an index depends on the current volume of data.

Examples

Basic Examples

Detail Model

```
CREATE TABLE t1
(
  c1 INT,
  c2 STRING
)
DUPLICATE KEY(c1)
DISTRIBUTED BY HASH(c1)
PROPERTIES (
  'replication_num' = '1'
);
```

Aggregation Model

```
CREATE TABLE t2
(
  c1 INT,
  c2 INT MAX
)
AGGREGATE KEY(c1)
DISTRIBUTED BY HASH(c1)
```

```
PROPERTIES (
  'replication_num' = '1'
);
```

Primary Key Model

```
CREATE TABLE t3
(
  c1 INT,
  c2 INT
)
UNIQUE KEY(c1)
DISTRIBUTED BY HASH(c1)
PROPERTIES (
  'replication_num' = '1'
);
```

Specifying Column Default Values

```
CREATE TABLE t5
(
  c1 INT,
  c2 INT DEFAULT 10
)
DUPLICATE KEY(c1)
DISTRIBUTED BY HASH(c1)
PROPERTIES (
  'replication_num' = '1'
);
```

Bucketing Method

```
CREATE TABLE t6
(
  c1 INT,
  c2 INT
)
DUPLICATE KEY(c1)
DISTRIBUTED BY RANDOM
PROPERTIES (
  'replication_num' = '1'
);
```

Automatic Partitioning

```
CREATE TABLE t7
(
  c1 INT,
  c2 DATETIME NOT NULL
)
DUPLICATE KEY(c1)
AUTO PARTITION BY RANGE(date_trunc(c2, 'day')) ()
DISTRIBUTED BY RANDOM
PROPERTIES (
  'replication_num' = '1'
);
```

Range Partitioning

```
CREATE TABLE t8
(
  c1 INT,
  c2 DATETIME NOT NULL
)
DUPLICATE KEY(c1)
PARTITION BY RANGE(c2) (
  FROM ('2020-01-01') TO ('2020-01-10') INTERVAL 1 DAY
)
DISTRIBUTED BY RANDOM
PROPERTIES (
  'replication_num' = '1'
);
```

List Partitioning

```
CREATE TABLE t9
(
  c1 INT,
  c2 DATE NOT NULL
)
DUPLICATE KEY(c1)
PARTITION BY LIST(c2) (
  PARTITION p1 VALUES IN (('2020-01-01'),('2020-01-02'))
)
DISTRIBUTED BY RANDOM
PROPERTIES (
  'replication_num' = '1'
);
```

Storage Medium and Cooldown Time

```
CREATE TABLE example_db.table_hash
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048),
    v2 SMALLINT DEFAULT "10"
)
UNIQUE KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2) BUCKETS 32
PROPERTIES(
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

Setting Table's Cold-Hot Tiered Data Migration Strategy with `storage_policy` Property

1. You need to create an s3 resource and storage policy first for the table to successfully associate with the migration strategy.

   ```
   -- Non-partitioned table
   CREATE TABLE IF NOT EXISTS create_table_use_created_policy
   (
       k1 BIGINT,
       k2 LARGEINT,
   ```

```
    v1 VARCHAR(2048)
)
UNIQUE KEY(k1)
DISTRIBUTED BY HASH (k1) BUCKETS 3
PROPERTIES(
    "storage_policy" = "test_create_table_use_policy",
    "replication_num" = "1"
);

-- Partitioned table
CREATE TABLE create_table_partion_use_created_policy
(
    k1 DATE,
    k2 INT,
    V1 VARCHAR(2048) REPLACE
) PARTITION BY RANGE (k1) (
    PARTITION p1 VALUES LESS THAN ("2022-01-01") ("storage_policy" = "test_create_table_partition_use_policy_1"
,"replication_num"="1"),
    PARTITION p2 VALUES LESS THAN ("2022-02-01") ("storage_policy" = "test_create_table_partition_use_policy_2"
,"replication_num"="1")
) DISTRIBUTED BY HASH(k2) BUCKETS 1;
```

Colocation Group

```
CREATE TABLE t1 (
    id int(11) COMMENT "",
    value varchar(8) COMMENT ""
)
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 10
PROPERTIES (
    "colocate_with" = "group1"
);

CREATE TABLE t2 (
    id int(11) COMMENT "",
    value1 varchar(8) COMMENT "",
    value2 varchar(8) COMMENT ""
)
DUPLICATE KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10
PROPERTIES (
    "colocate_with" = "group1"
);
```

Index

```
CREATE TABLE example_db.table_hash
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5",
    v1 CHAR(10) REPLACE,
    v2 INT SUM,
    INDEX k1_idx (k1) USING INVERTED COMMENT 'my first index'
)
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1) BUCKETS 32
PROPERTIES (
    "bloom_filter_columns" = "k2"
);
```

Setting Table's Replication Properties

```
CREATE TABLE example_db.table_hash
(
    k1 TINYINT,
    k2 DECIMAL(10, 2) DEFAULT "10.5"
)
DISTRIBUTED BY HASH(k1) BUCKETS 32
PROPERTIES (
    "replication_allocation"="tag.location.group_a:1, tag.location.group_b:2"
);
```

Dynamic Partitioning

This table creates partitions 3 days in advance and deletes partitions from 3 days ago. For example, if today is `2020-01-08`, it will create partitions named `p20200108`, `p20200109`, `p20200110`, `p20200111`. The partition ranges are as follows:

```
[types: [DATE]; keys: [2020-01-08]; ··types: [DATE]; keys: [2020-01-09]; )
[types: [DATE]; keys: [2020-01-09]; ··types: [DATE]; keys: [2020-01-10]; )
[types: [DATE]; keys: [2020-01-10]; ··types: [DATE]; keys: [2020-01-11]; )
[types: [DATE]; keys: [2020-01-11]; ··types: [DATE]; keys: [2020-01-12]; )
```

```
CREATE TABLE example_db.dynamic_partition
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048),
    v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
DUPLICATE KEY(k1, k2, k3)
PARTITION BY RANGE (k1) ()
DISTRIBUTED BY HASH(k2) BUCKETS 32
PROPERTIES(
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-3",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "32"
);
```

Setting Dynamic Partition's Replication Properties

```
CREATE TABLE example_db.dynamic_partition
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    v1 VARCHAR(2048),
    v2 DATETIME DEFAULT "2014-02-04 15:36:00"
)
PARTITION BY RANGE (k1) ()
DISTRIBUTED BY HASH(k2) BUCKETS 32
PROPERTIES(
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-3",
    "dynamic_partition.end" = "3",
```

```
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "32",
    "dynamic_partition.replication_allocation" = "tag.location.group_a:3"
 );
```

CTAS Example

```
CREATE TABLE t10
PROPERTIES (
  'replication_num' = '1'
)
AS SELECT * FROM t1;
```

CREATE TABLE ... LIKE Example

```
CREATE TABLE t11 LIKE t10;
```

DESC TABLE

Description

This statement is used to display the schema information of the specified table.

Syntax

```
DESC[RIBE] [db_name.]table_name [ALL];
```

Required Parameters

1.`<table_name>` > Specifies the table identifier (name), which must be unique within the database in which it is located. > > Identifiers must begin with an alphabetic character (or any character in a language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g. `My Object`). > > Identifiers cannot use reserved keywords. > > For more details, see Identifier Requirements and Reserved Keywords.

Optional Parameters

1.`<db_name>` > Specifies the identifier (i.e., name) for the database. > > Identifiers must begin with an alphabetic character (or any character in a given language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Database`). > > Identifiers cannot use reserved keywords. > > See Identifier Requirements and Reserved Keywords for more details.

2.`RIBE` > Returns description information of all columns in a table

3.`ALL` > Returns description information for all columns

Return Value

column name	description
IndexName	Table name
IndexKeysType	Table Model
Field	Column Name

column name	description
Type	Data Types
Null	Whether NULL values are allowed
Key	Is it a key column
Default	Default Value
Extra	Display some additional information
Visible	Visible
DefineExpr	Defining Expressions
WhereClause	Filter Conditions Related Definitions

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
SELECT_PRIV	Table	When executing DESC, you need to have the SELECT_PRIV privilege on the table being queried

Usage Notes

- If ALL is specified, the schema of all indexes (rollup) of the table is displayed.

Examples

1. Display the Base table schema

```
DESC test_table;
```

```
+---------+-------------+------+-------+---------+-------+
| Field   | Type        | Null | Key   | Default | Extra |
+---------+-------------+------+-------+---------+-------+
| user_id | bigint      | No   | true  | NULL    |       |
| name    | varchar(20) | Yes  | false | NULL    | NONE  |
```

```
| age     | int         | Yes  | false | NULL    | NONE  |
+---------+-------------+------+-------+---------+-------+
```

2. Display the schema of all indexes in the table

```
DESC demo.test_table ALL;
```

```
+------------+---------------+---------+-------------+--------------+------+-------+---------+-------+---------+------------+-------
------+
| IndexName  | IndexKeysType | Field   | Type        | InternalType | Null | Key   | Default | Extra | Visible | DefineExpr |
WhereClause |
+------------+---------------+---------+-------------+--------------+------+-------+---------+-------+---------+------------+-------
------+
| test_table | DUP_KEYS      | user_id | bigint      | bigint       | No   | true  | NULL    |       | true    |            |
|
|            |               | name    | varchar(20) | varchar(20)  | Yes  | false | NULL    | NONE  | true    |            |
|
|            |               | age     | int         | int          | Yes  | false | NULL    | NONE  | true    |            |
|
+------------+---------------+---------+-------------+--------------+------+-------+---------+-------+---------+------------+-------
------+
```

ALTER TABLE COLUMN

Description

This statement is used to perform a schema change operation on an existing table. The schema change is asynchronous, and the task is returned when the task is submitted successfully. After that, you can use the SHOW ALTER TABLE COLUMN command to view the progress.

Doris has the concept of materialized index after table construction. After successful table construction, it is the base table and the materialized index is the base index. rollup index can be created based on the base table. Both base index and rollup index are materialized indexes. If rollup_index_name is not specified during the schema change operation, the operation is based on the base table by default.

> **Doris 1.2.0 supports light schema change for light scale structure changes, and addition and subtraction operations for value columns can be completed more quickly and synchronously. You can manually specify "light_schema_change" = 'true' when creating a table. This parameter is enabled by default for versions 2.0.0 and later.**

Grammar:

```
ALTER TABLE [database.]table alter_clause;
```

The alter_clause of schema change supports the following modification methods:

1. Add a column to the specified position at the specified index

Grammar

```
ALTER TABLE [database.]table table_name ADD COLUMN column_name column_type [KEY | agg_type] [DEFAULT "default_value"]
[AFTER column_name|FIRST]
[TO rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Example

1. Add a key column new_col to example_db.my_table after key_1 (non-aggregated model)

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT KEY DEFAULT "0" AFTER key_1;
```

2. Add a value column new_col to example_db.my_table after value_1 (non-aggregate model)

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT DEFAULT "0" AFTER value_1;
```

3. Add a key column new_col (aggregate model) to example_db.my_table after key_1

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT KEY DEFAULT "0" AFTER key_1;
```

4. Add a value column to example_db.my_table after value_1 new_col SUM Aggregation type (aggregation model)

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT SUM DEFAULT "0" AFTER value_1;
```

5. Add new_col to the first column position of the example_db.my_table table (non-aggregated model)

```
ALTER TABLE example_db.my_table
ADD COLUMN new_col INT KEY DEFAULT "0" FIRST;
```

> **- If you add a value column to the aggregation model, you need to specify agg_type**
>
> - For non-aggregated models (such as DUPLICATE KEY), if you add a key column, you need to specify the KEY keyword
> - You cannot add columns that already exist in the base index to the rollup index (you can recreate a rollup index if necessary)

2. Add multiple columns to the specified index

Grammar

```
ALTER TABLE [database.]table table_name ADD COLUMN (column_name1 column_type [KEY | agg_type] DEFAULT "default_value", ...)
[TO rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Example

1. Add multiple columns to example_db.my_table, where new_col and new_col2 are SUM aggregate types (aggregate model)

```
ALTER TABLE example_db.my_table
ADD COLUMN (new_col1 INT SUM DEFAULT "0" ,new_col2 INT SUM DEFAULT "0");
```

2. Add multiple columns to example_db.my_table (non-aggregated model), where new_col1 is the KEY column and new_col2 is the value column

```
ALTER TABLE example_db.my_table
ADD COLUMN (new_col1 INT key DEFAULT "0" , new_col2 INT DEFAULT "0");
```

> **- If you add a value column to the aggregation model, you need to specify agg_type**

- If you add a key column to the aggregation model, you need to specify the KEY keyword
- You cannot add columns that already exist in the base index to the rollup index (you can recreate a rollup index if necessary)

3. Delete a column from the specified index

Grammar*

```
ALTER TABLE [database.]table table_name DROP COLUMN column_name
[FROM rollup_index_name]
```

Example

1. Delete column col1 from example_db.my_table

```
ALTER TABLE example_db.my_table DROP COLUMN col1;
```

- Cannot drop partition column

- The aggregate model cannot delete KEY columns
- If the column is removed from the base index, it will also be removed if it is included in the rollup index

4. Modify the column type and column position of the specified index

Grammar

```
ALTER TABLE [database.]table table_name MODIFY COLUMN column_name column_type [KEY | agg_type] [NULL | NOT NULL] [DEFAULT
"default_value"]
[AFTER column_name|FIRST]
[FROM rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Example

1. Modify the type of the key column col1 of the base index to BIGINT and move it to the back of the col2 column

```
ALTER TABLE example_db.my_table
MODIFY COLUMN col1 BIGINT KEY DEFAULT "1" AFTER col2;
```

:::tip Whether you modify the key column or the value column, you need to declare complete column information :::

2. Modify the maximum length of the val1 column of base index. The original val1 is (val1 VARCHAR(32) REPLACE DEFAULT “abc”)

```
ALTER TABLE example_db.my_table
MODIFY COLUMN val1 VARCHAR(64) REPLACE DEFAULT "abc";
```

:::tip You can only modify the column's data type; other attributes of the column must remain unchanged. :::

3. Modify the length of a field in the Key column of the Duplicate key table

```
ALTER TABLE example_db.my_table
MODIFY COLUMN k3 VARCHAR(50) KEY NULL COMMENT 'to 50';
```

- If you modify the value column in the aggregation model, you need to specify agg_type

- If you modify the key column for non-aggregate types, you need to specify the KEY keyword
- Only the type of the column can be modified, and other attributes of the column remain as they are (that is, other attributes need to be explicitly written in the statement according to the original attributes, see example 2 above)
- Partitioning and bucketing columns cannot be modified in any way
- The following types of conversions are currently supported (loss of precision is guaranteed by the user)
    - Conversion of TINYINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE types to larger numeric types
    - Convert TINTINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE/DECIMAL to VARCHAR
    - VARCHAR supports modifying the maximum length
    - VARCHAR/CHAR converted to TINTINT/SMALLINT/INT/BIGINT/LARGEINT/FLOAT/DOUBLE
    - Convert VARCHAR/CHAR to DATE (currently supports "%Y-%m-%d", "%y-%m-%d", "%Y%m%d", "%y%m%d", "%Y/%m/%d,"%y/%m/%d" six formats)
    - Convert DATETIME to DATE (only keep year-month-day information, for example: `2019-12-09 21:47:05` <–> `2019-12-09`)
    - DATE is converted to DATETIME (hours, minutes and seconds are automatically filled with zeros, for example: `2019-12-09` <–> `2019-12-09 00:00:00`)
    - Convert FLOAT to DOUBLE
    - INT is converted to DATE (if the INT type data is illegal, the conversion fails, and the original data remains unchanged)
    - All can be converted to STRING except DATE and DATETIME, but STRING cannot be converted to any other type

5. Reorder the column at the specified index

Grammar

```
ALTER TABLE [database.]table table_name ORDER BY (column_name1, column_name2, ...)
[FROM rollup_index_name]
[PROPERTIES ("key"="value", ...)]
```

Example

1. Adjust the order of the key and value columns of example_db.my_table (non-aggregate model)

```
CREATE TABLE `my_table`(
`k_1` INT NULL,
`k_2` INT NULL,
`v_1` INT NULL,
`v_2` varchar NULL,
`v_3` varchar NULL
) ENGINE=OLAP
DUPLICATE KEY(`k_1`, `k_2`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`k_1`) BUCKETS 5
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);

ALTER TABLE example_db.my_table ORDER BY (k_2,k_1,v_3,v_2,v_1);

mysql> desc my_table;
+-------+------------+------+-------+---------+-------+
| Field | Type       | Null | Key   | Default | Extra |
+-------+------------+------+-------+---------+-------+
| k_2   | INT        | Yes  | true  | NULL    |       |
| k_1   | INT        | Yes  | true  | NULL    |       |
| v_3   | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| v_2   | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| v_1   | INT        | Yes  | false | NULL    | NONE  |
+-------+------------+------+-------+---------+-------+
```

2. Do two actions simultaneously

```
CREATE TABLE `my_table` (
`k_1` INT NULL,
`k_2` INT NULL,
`v_1` INT NULL,
`v_2` varchar NULL,
`v_3` varchar NULL
) ENGINE=OLAP
DUPLICATE KEY(`k_1`, `k_2`)
COMMENT 'OLAP'
DISTRIBUTED BY HASH(`k_1`) BUCKETS 5
PROPERTIES (
"replication_allocation" = "tag.location.default: 1"
);

ALTER TABLE example_db.my_table
ADD COLUMN col INT DEFAULT "0" AFTER v_1,
ORDER BY (k_2,k_1,v_3,v_2,v_1,col);

mysql> desc my_table;
+-------+------------+------+-------+---------+-------+
| Field | Type       | Null | Key   | Default | Extra |
+-------+------------+------+-------+---------+-------+
| k_2   | INT        | Yes  | true  | NULL    |       |
| k_1   | INT        | Yes  | true  | NULL    |       |
| v_3   | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| v_2   | VARCHAR(*) | Yes  | false | NULL    | NONE  |
| v_1   | INT        | Yes  | false | NULL    | NONE  |
| col   | INT        | Yes  | false | 0       | NONE  |
+-------+------------+------+-------+---------+-------+
```

> **- All columns in index are written out**
>
> - the value column comes after the key column
> - You can adjust the key column only within the range of the key column. The same applies to the value column

Keywords

```
ALTER, TABLE, COLUMN, ALTER TABLE
```

Best Practice

ALTER TABLE PARTITION

Description

This statement is used to modify a table with a partition.

This operation is synchronous, and the return of the command indicates the completion of the execution.

grammar:

```
ALTER TABLE [database.]table alter_clause;
```

The alter_clause of partition supports the following modification methods

1. Add partition

grammar:

```
ADD PARTITION [IF NOT EXISTS] partition_name
partition_desc ["key"="value"]
[DISTRIBUTED BY HASH (k1[,k2 ...]) [BUCKETS num]]
```

Notice:

- partition_desc supports the following two ways of writing
    - VALUES LESS THAN [MAXVALUE|(“value1”, …)]
    - VALUES [(“value1”, …), (“value1”, …))
- The partition is left closed and right open. If the user only specifies the right boundary, the system will automatically determine the left boundary
- If the bucketing method is not specified, the bucketing method and bucket number used for creating the table would be automatically used
- If the bucketing method is specified, only the number of buckets can be modified, not the bucketing method or the bucketing column. If the bucketing method is specified but the number of buckets not be specified, the default value `10` will be used for bucket number instead of the number specified when the table is created. If the number of buckets modified, the bucketing method needs to be specified simultaneously.
- The [“key”=“value”] section can set some attributes of the partition, see CREATE TABLE
- If the user does not explicitly create a partition when creating a table, adding a partition by ALTER is not supported
- If the user uses list partition then they can add default partition to the table. The default partition will store all data not satisfying prior partition key’s constraints.
    - ALTER TABLE table_name ADD PARTITION partition_name

2. Delete the partition

grammar:

```
DROP PARTITION [IF EXISTS] partition_name [FORCE]
```

Notice:

- At least one partition must be reserved for tables using partitioning.
- After executing DROP PARTITION for a period of time, the deleted partition can be recovered through the RECOVER statement. For details, see SQL Manual - Database Management - RECOVER Statement
- If you execute DROP PARTITION FORCE, the system will not check whether there are unfinished transactions in the partition, the partition will be deleted directly and cannot be recovered, this operation is generally not recommended

3. Modify the partition properties

grammar:

```
MODIFY PARTITION p1|(p1[, p2, ...]) SET ("key" = "value", ...)
```

illustrate:

- Currently supports modifying the following properties of partitions:
    - storage_medium -storage_cooldown_time
    - replication_num
    - in_memory
- For single-partition tables, partition_name is the same as the table name.

Examples

1. Add partition, existing partition [MIN, 2013-01-01), add partition [2013-01-01, 2014-01-01), use default bucketing method

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2014-01-01");
```

2. Increase the partition and use the new number of buckets

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2015-01-01")
DISTRIBUTED BY HASH(k1) BUCKETS 20;
```

3. Increase the partition and use the new number of replicas

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES LESS THAN ("2015-01-01")
("replication_num"="1");
```

4. Modify the number of partition replicas

```
ALTER TABLE example_db.my_table
MODIFY PARTITION p1 SET("replication_num"="1");
```

5. Batch modify the specified partition

```
ALTER TABLE example_db.my_table
MODIFY PARTITION (p1, p2, p4) SET("replication_num"="1");
```

6. Batch modify all partitions

```
ALTER TABLE example_db.my_table
MODIFY PARTITION (*) SET("storage_medium"="HDD");
```

7. Delete partition

```
ALTER TABLE example_db.my_table
DROP PARTITION p1;
```

8. Batch delete partition

```
ALTER TABLE example_db.my_table
DROP PARTITION p1,
DROP PARTITION p2,
DROP PARTITION p3;
```

9. Add a partition specifying upper and lower bounds

```
ALTER TABLE example_db.my_table
ADD PARTITION p1 VALUES [("2014-01-01"), ("2014-02-01"));
```

Keywords

```
ALTER, TABLE, PARTITION, ALTER TABLE
```

Best Practice

ALTER TABLE ROLLUP

Description

This statement is used to perform a rollup modification operation on an existing table. The rollup is an asynchronous operation, and the task is returned when the task is submitted successfully. After that, you can use the SHOW ALTER command to view the progress.

grammar:

```
ALTER TABLE [database.]table alter_clause;
```

The alter_clause of rollup supports the following creation methods

1. Create a rollup index

grammar:

```
ADD ROLLUP rollup_name (column_name1, column_name2, ...)
[FROM from_index_name]
[PROPERTIES ("key"="value", ...)]
```

properties: Support setting timeout time, the default timeout time is 1 day.

2. Create rollup indexes in batches

grammar:

```
ADD ROLLUP [rollup_name (column_name1, column_name2, ...)
                    [FROM from_index_name]
                    [PROPERTIES ("key"="value", ...)],...]
```

Notice:

- If from_index_name is not specified, it will be created from base index by default
- Columns in rollup table must be columns already in from_index
- In properties, the storage format can be specified. For details, see CREATE TABLE

3. Delete rollup index

grammar:

```
DROP ROLLUP rollup_name [PROPERTIES ("key"="value", ...)]
```

4. Batch delete rollup index

grammar:

```
DROP ROLLUP [rollup_name [PROPERTIES ("key"="value", ...)],...]
```

Notice:

- cannot delete base index

Examples

1. Create index: example_rollup_index, based on base index (k1,k2,k3,v1,v2). Columnar storage.

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index(k1, k3, v1, v2);
```

2. Create index: example_rollup_index2, based on example_rollup_index (k1,k3,v1,v2)

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index2 (k1, v1)
FROM example_rollup_index;
```

3. Create index: example_rollup_index3, based on base index (k1,k2,k3,v1), with a custom rollup timeout of one hour.

```
ALTER TABLE example_db.my_table
ADD ROLLUP example_rollup_index(k1, k3, v1)
PROPERTIES("timeout" = "3600");
```

4. Delete index: example_rollup_index2

```
ALTER TABLE example_db.my_table
DROP ROLLUP example_rollup_index2;
```

5. Batch Delete rollup index

```
ALTER TABLE example_db.my_table
DROP ROLLUP example_rollup_index2,example_rollup_index3;
```

Keywords

```
ALTER, TABLE, ROLLUP, ALTER TABLE
```

Best Practice

ALTER TABLE RENAME

Description

This statement is used to rename certain names of existing table properties. This operation is synchronous, and the return of the command indicates the completion of the execution.

grammar:

```
ALTER TABLE [database.]table alter_clause;
```

The alter_clause of rename supports modification of the following names

1. Modify the table name

grammar:

```
RENAME new_table_name;
```

2. Modify the rollup index name

grammar:

```
RENAME ROLLUP old_rollup_name new_rollup_name;
```

3. Modify the partition name

grammar:

```
RENAME PARTITION old_partition_name new_partition_name;
```

4. Modify the column name

> **Tips**
>
> This feature is supported since the Apache Doris 1.2 version

Modify the column name

grammar:

```
RENAME COLUMN old_column_name new_column_name;
```

Notice: - When creating a table, you need to set 'light_schema_change=true' in the property.

Examples

1. Modify the table named table1 to table2

```
ALTER TABLE table1 RENAME table2;
```

2. Modify the rollup index named rollup1 in the table example_table to rollup2

```
ALTER TABLE example_table RENAME ROLLUP rollup1 rollup2;
```

3. Modify the partition named p1 in the table example_table to p2

```
ALTER TABLE example_table RENAME PARTITION p1 p2;
```

4. Modify the column named c1 in the table example_table to c2

```
ALTER TABLE example_table RENAME COLUMN c1 c2;
```

Keywords

```
ALTER, TABLE, RENAME, ALTER TABLE
```

Best Practice

ALTER TABLE REPLACE

Description

Atomic substitution of two tables. This operation applies only to OLAP tables.

```
ALTER TABLE [db.]tbl1 REPLACE WITH TABLE tbl2
[PROPERTIES('swap' = 'true')];
```

Replace table tbl1 with table tbl2.

If the `swap` parameter is `true`, the data in the table named `tbl1` will be the data in the original table named `tbl2` after the replacement. The data in the table named `tbl2` is the data in the original `tbl1` table. That is, two tables of data have been swapped.

If the `swap` parameter is `false`, the data in the `tbl1` table will be the data in the `tbl2` table after the replacement. The table named `tbl2` is deleted.

Theory

The replace table function actually turns the following set of operations into an atomic operation.

If you want to replace table A with table B and `swap` is `true`, do the following:

1. Rename table B as table A.
2. Rename table A as table B.

If `swap` is `false`, do as follows:

1. Delete table A.
2. Rename table B as table A.

Notice

1. The default `swap` parameter is `true`. That is, a table replacement operation is equivalent to an exchange of data between two tables.
2. If the `swap` parameter is set to false, the replaced table (table A) will be deleted and cannot be restored.
3. The replacement operation can only occur between two OLAP tables and does not check whether the table structure of the two tables is consistent.
4. The original permission Settings are not changed. Because the permission check is based on the table name.

Examples

1. Atomic swap `tbl1` with `tbl2` without dropping any tables(Note: if you delete it, you actually delete tbl1 and rename tbl2 to tbl1.)

```
ALTER TABLE tbl1 REPLACE WITH TABLE tbl2;
```

or

```
ALTER TABLE tbl1 REPLACE WITH TABLE tbl2 PROPERTIES('swap' = 'true');
```

2. Atomic swap `tbl1` with `tbl2` and deleting the `tbl2` table(Keep `tbl1` and the data of the original `tbl2`)

```
ALTER TABLE tbl1 REPLACE WITH TABLE tbl2 PROPERTIES('swap' = 'false');
```

Keywords

```
ALTER, TABLE, REPLACE, ALTER TABLE
```

Best Practice

1. Atomic overlay write operations

In some cases, the user wants to be able to rewrite the data of a certain table, but if the data is deleted first and then imported, the data cannot be viewed for a period of time in between. At this time, the user can first use the `CREATE TABLE LIKE` statement to create a new table with the same structure, import the new data into the new table, and use the replacement operation to atomically replace the old table to achieve the goal. Atomic overwrite write operations at the partition level, see temp partition documentation.

ALTER TABLE PROPERTY

> **Differences between Partition Attributes and Table Attributes**
>
> - Partition attributes generally focus on the number of buckets (buckets), storage medium (storage_medium), replication num (replication_num), and hot/cold data separation policies (storage_policy).
>   - For existing partitions, you can use ALTER TABLE {tableName} MODIFY PARTITION ({partitionName}) SET ({key}={value}) to modify them, but the number of buckets (buckets) cannot be changed.
>   - For not-created dynamic partitions, you can use ALTER TABLE {tableName} SET (dynamic_partition.{key} = {value}) to modify their attributes.
>   - For not-created auto partitions, you can use ALTER TABLE {tableName} SET ({key} = {value}) to modify their attributes.
>   - If users want to modify partition attributes, they need to modify the attributes of the already created partitions, as well as the attributes of not-created partitions.
> - Aside from the above attributes, all others are at the table level.
> - For the specific attributes, please refer to create table attributes

Description

This statement is used to modify the properties of an existing table. This operation is synchronous, and the return of the command indicates the completion of the execution.

Modify the properties of the table, currently supports modifying the bloom filter column, the colocate_with attribute and the dynamic_partition attribute, the replication_num and default.replication_num.

grammar:

```
ALTER TABLE [database.]table alter_clause;
```

The alter_clause of property supports the following modification methods.

Note:

Can also be merged into the above schema change operation to modify, see the example below

1. Modify the bloom filter column of the table

```
ALTER TABLE example_db.my_table SET ("bloom_filter_columns"="k1,k2,k3");
```

Can also be incorporated into the schema change operation above (note that the syntax for multiple clauses is slightly different)

```
ALTER TABLE example_db.my_table
DROP COLUMN col2
PROPERTIES ("bloom_filter_columns"="k1,k2,k3");
```

2. Modify the Colocate property of the table

```
ALTER TABLE example_db.my_table set ("colocate_with" = "t1");
```

3. Change the bucketing method of the table from Hash Distribution to Random Distribution

```
ALTER TABLE example_db.my_table set ("distribution_type" = "random");
```

4. Modify the dynamic partition attribute of the table (support adding dynamic partition attribute to the table without dynamic partition attribute)

```
ALTER TABLE example_db.my_table set ("dynamic_partition.enable" = "false");
```

If you need to add dynamic partition attributes to tables without dynamic partition attributes, you need to specify all dynamic partition attributes (Note: adding dynamic partition attributes is not supported for non-partitioned tables)

```
ALTER TABLE example_db.my_table set (
  "dynamic_partition.enable" = "true",
  "dynamic_partition.time_unit" = "DAY",
  "dynamic_partition.end" = "3",
  "dynamic_partition.prefix" = "p",
  "dynamic_partition. buckets" = "32"
);
```

5. Modify the in_memory attribute of the table, only can set value ‘false’

```
ALTER TABLE example_db.my_table set ("in_memory" = "false");
```

6. Enable batch delete function

```
ALTER TABLE example_db.my_table ENABLE FEATURE "BATCH_DELETE";
```

Note:

- Only support unique tables
- Batch deletion is supported for old tables, while new tables are already supported when they are created

7. Enable the function of ensuring the import order according to the value of the sequence column

```
ALTER TABLE example_db.my_table ENABLE FEATURE "SEQUENCE_LOAD" WITH PROPERTIES (
  "function_column.sequence_type" = "Date"
);
```

Note:

- Only support unique tables
- The sequence_type is used to specify the type of the sequence column, which can be integral and time type
- Only the orderliness of newly imported data is supported. Historical data cannot be changed

8. Change the default number of buckets for the table to 50

```
ALTER TABLE example_db.my_table MODIFY DISTRIBUTION DISTRIBUTED BY HASH(k1) BUCKETS 50;
```

Note:

- Only support non colocate table with RANGE partition and HASH distribution

9. Modify table comments

```
ALTER TABLE example_db.my_table MODIFY COMMENT "new comment";
```

10. Modify column comments

```
ALTER TABLE example_db.my_table MODIFY COLUMN k1 COMMENT "k1", MODIFY COLUMN k2 COMMENT "k2";
```

11. Modify the engine type

Only the MySQL type can be changed to the ODBC type. The value of driver is the name of the driver in the odbc.init configuration.

```
ALTER TABLE example_db.mysql_table MODIFY ENGINE TO odbc PROPERTIES("driver" = "MySQL");
```

12. Modify the number of copies

```
ALTER TABLE example_db.mysql_table SET ("replication_num" = "2");
ALTER TABLE example_db.mysql_table SET ("default.replication_num" = "2");
ALTER TABLE example_db.mysql_table SET ("replication_allocation" = "tag.location.default: 1");
ALTER TABLE example_db.mysql_table SET ("default.replication_allocation" = "tag.location.default: 1");
```

Note: 1. The property with the default prefix indicates the default replica distribution for the modified table. This modification does not modify the current actual replica distribution of the table, but only affects the replica distribution of newly created partitions on the partitioned table. 2. For non-partitioned tables, modifying the replica distribution property without the default prefix will modify both the default replica distribution and the actual replica distribution of the table. That is, after the modification, through the `show create table` and `show partitions from tbl` statements, you can see that the replica distribution has been modified. changed. 3. For partitioned tables, the actual replica distribution of the table is at the partition level, that is, each partition has its own replica distribution, which can be viewed through the `show partitions from tbl` statement. If you want to modify the actual replica distribution, see `ALTER TABLE PARTITION`.

13. **[Experimental]** turn on `light_schema_change`

For tables that were not created with light_schema_change enabled, you can enable it by using the following statement.

```
ALTER TABLE example_db.mysql_table SET ("light_schema_change" = "true");
```

Examples

1. Modify the bloom filter column of the table

```
ALTER TABLE example_db.my_table SET ("bloom_filter_columns"="k1,k2,k3");
```

Can also be incorporated into the schema change operation above (note that the syntax for multiple clauses is slightly different)

```
ALTER TABLE example_db.my_table
DROP COLUMN col2
PROPERTIES ("bloom_filter_columns"="k1,k2,k3");
```

2. Modify the Colocate property of the table

```
ALTER TABLE example_db.my_table set ("colocate_with" = "t1");
```

3. Change the bucketing method of the table from Hash Distribution to Random Distribution

```
ALTER TABLE example_db.my_table set ("distribution_type" = "random");
```

4. Modify the dynamic partition attribute of the table (support adding dynamic partition attribute to the table without dynamic partition attribute)

```
ALTER TABLE example_db.my_table set ("dynamic_partition.enable" = "false");
```

If you need to add dynamic partition attributes to tables without dynamic partition attributes, you need to specify all dynamic partition attributes (Note: adding dynamic partition attributes is not supported for non-partitioned tables)

```
ALTER TABLE example_db.my_table set ("dynamic_partition.enable" = "true", "dynamic_partition.time_unit" = "DAY",
"dynamic_partition.end" = "3", "dynamic_partition.prefix" = "p", "dynamic_partition. buckets" = "32");
```

5. Modify the in_memory attribute of the table, only can set value ‘false’

```
ALTER TABLE example_db.my_table set ("in_memory" = "false");
```

6. Enable batch delete function

```
ALTER TABLE example_db.my_table ENABLE FEATURE "BATCH_DELETE";
```

7. Enable the function of ensuring the import order according to the value of the sequence column

```
ALTER TABLE example_db.my_table ENABLE FEATURE "SEQUENCE_LOAD" WITH PROPERTIES ("function_column.sequence_type" = "Date");
```

8. Change the default number of buckets for the table to 50

```
ALTER TABLE example_db.my_table MODIFY DISTRIBUTION DISTRIBUTED BY HASH(k1) BUCKETS 50;
```

9. Modify table comments

```
ALTER TABLE example_db.my_table MODIFY COMMENT "new comment";
```

10. Modify column comments

```
ALTER TABLE example_db.my_table MODIFY COLUMN k1 COMMENT "k1", MODIFY COLUMN k2 COMMENT "k2";
```

11. Modify the engine type

```
ALTER TABLE example_db.mysql_table MODIFY ENGINE TO odbc PROPERTIES("driver" = "MySQL");
```

12. Add a cold and hot separation data migration strategy to the table

```
 ALTER TABLE create_table_not_have_policy set ("storage_policy" = "created_create_table_alter_policy");
```

NOTE: The table can be successfully added only if it hasn’t been associated with a storage policy. A table just can have one storage policy.

13. Add a hot and cold data migration strategy to the partition of the table

```
ALTER TABLE create_table_partition MODIFY PARTITION (*) SET("storage_policy"="created_create_table_partition_alter_policy");
```

NOTE: The table’s partition can be successfully added only if it hasn’t been associated with a storage policy. A table just can have one storage policy.

Keywords

```
ALTER, TABLE, PROPERTY, ALTER TABLE
```

Best Practice

ALTER TABLE COMMENT

Description

This statement is used to modify the comment of an existing table. The operation is synchronous, and the command returns to indicate completion.

Syntax

```
ALTER TABLE [database.]table alter_clause;
```

1. Modify table comment

Syntax

```
MODIFY COMMENT "new table comment";
```

2. Modify column comment

Syntax

```
MODIFY COLUMN col1 COMMENT "new column comment";
```

Examples

1. Change the table1's comment to table1_comment

```
ALTER TABLE table1 MODIFY COMMENT "table1_comment";
```

2. Change the table1's col1 comment to table1_comment

```
ALTER TABLE table1 MODIFY COLUMN col1 COMMENT "table1_col1_comment";
```

Keywords

```
ALTER, TABLE, COMMENT, ALTER TABLE
```

Best Practice

CANCEL ALTER TABLE

Description

This statement is used to cancel (revoke) an ongoing ALTER TABLE operation. You can use this command to terminate an ALTER TABLE operation while it is being executed.

Syntax

```
CANCEL ALTER TABLE { COLUMN | MATERIALIZED VIEW | ROLLUP } FROM <db_name>.<table_name> [ <job_id1> [ , <job_id2> ... ]]
```

Required Parameters

1. `{ COLUMN | MATERIALIZED VIEW | ROLLUP }` >Specify the type of modification to be canceled, one of which must be selected >- `COLUMN`: Cancel the modification operation on the table column >- `ROLLUP`: Cancel the modification operation on the view >- `MATERIALIZED VIEW`: Cancel the modification operation on the materialized view

2.`<db_name>` > Specifies the identifier (that is, the name) of the database. > > Identifier must begin with an alphabet character (if Unicode name support is enabled, any language character is allowed), and must not contain spaces or special characters, except for the entire identifier string enclosed in quotes (e.g., `My Database`). > > Identifier cannot use reserved keywords. > > For more information, see Identifier Requirements and Reserved Keywords.

3.`<table_name>` > Specifies the identifier (that is, the name) of the table, within its database (Database). > > Identifier must begin with an alphabet character (if Unicode name support is enabled, any language character is allowed), and must not contain spaces or special characters, except for the entire identifier string enclosed in quotes (e.g., `My Object`). > > Identifier cannot use reserved keywords. > > For more information, see Identifier Requirements and Reserved Keywords.

Optional Parameters

1. `<job_id>` > The specific job ID to cancel. > > If a job ID is specified, only the specified job is canceled; if not specified, all ongoing modifications of the specified type (COLUMN or ROLLUP) on the table are canceled. > > You can specify multiple job IDs, separated by commas. > > You can obtain job IDs by using the `SHOW ALTER TABLE COLUMN` or `SHOW ALTER TABLE ROLLUP` command.

Permission Control

Users who execute this SQL command must have at least the following permissions:

Privilege	Object	Notes
ALTER_PRIV	Table	CANCEL ALTER TABLE belongs to table ALTER operation

Notes

- This command is an asynchronous operation, and the actual execution result needs to be confirmed by using `SHOW ALTER TABLE COLUMN` or `SHOW ALTER TABLE ROLLUP` to check the status of the task.

Example

1. Cancel ALTER TABLE COLUMN operation

```
CANCEL ALTER TABLE COLUMN
FROM db_name.table_name
```

2. Cancel ALTER TABLE ROLLUP operation

```
CANCEL ALTER TABLE ROLLUP
FROM db_name.table_name
```

3. Cancel ALTER TABLE ROLLUP operation in batches based on job ID

```
CANCEL ALTER TABLE ROLLUP
FROM db_name.table_name (jobid,...)
```

SHOW ALTER

Description

This statement is used to display the execution status of various ongoing modification tasks.

```
SHOW ALTER [TABLE [COLUMN | ROLLUP] [FROM db_name]];
```

Notes:

1. TABLE COLUMN: Displays ALTER tasks for modifying columns.
2. Supported syntax: [WHERE TableName|CreateTime|FinishTime|State] [ORDER BY] [LIMIT].
3. TABLE ROLLUP: Displays tasks for creating or deleting ROLLUP.
4. If db_name is not specified, the current default database is used.

Result

SHOW ALTER TABLE COLUMN

Field Name	Description
JobId	Unique ID for each Schema Change job.
TableName	The name of the base table corresponding to the Schema Change.
CreateTime	Job creation time.
FinishedTime	Job completion time. If not completed, shows “N/A”.
IndexName	The name of a base table/synchronized materialized view involved in this modification.
IndexId	ID of the new base table/synchronized materialized view.
OriginIndexId	ID of a base table/synchronized materialized view involved in this modification.
SchemaVersion	Displays in M:N format. M represents the version of the Schema Change, and N represents the corresponding hash value. Each Schema Change increments the version.
TransactionId	Transaction ID for converting historical data.
State	The phase of the job.
	- PENDING: The job is waiting to be scheduled in the queue.
	- WAITING_TXN: Waiting for import tasks before the boundary transaction ID to complete.
	- RUNNING: Currently performing historical data conversion.
	- FINISHED: The job has successfully completed.
	- CANCELLED: The job has failed.
Msg	If the job fails, displays the failure message.

Field Name	Description
Progress	Job progress. Only displayed in RUNNING state. Progress is shown in M/N format. N is the total number of replicas involved in the Schema Change. M is the number of replicas for which historical data conversion has been completed.
Timeout	Job timeout in seconds.

Examples

1. Display the execution status of all modification column tasks for the default database.

```
SHOW ALTER TABLE COLUMN;
```

2. Display the execution status of the most recent modification column task for a specific table.

```
SHOW ALTER TABLE COLUMN WHERE TableName = "table1" ORDER BY CreateTime DESC LIMIT 1;
```

3. Display the execution status of creating or deleting ROLLUP tasks for a specified database.

```
SHOW ALTER TABLE ROLLUP FROM example_db;
```

Keywords

```
SHOW, ALTER
```

Best Practices

TRUNCATE TABLE

Description

This statement is used to clear the data of the specified table and partition.

Syntax

```
TRUNCATE TABLE [<db_name>.]<table_name>[ PARTITION ( <partition_name1> [, <partition_name2> ... ] ) ];
```

Required Parameters

1.<db_name> > Specifies the identifier (name) for the database. > > Identifiers must begin with an alphabetic character (or any character in a given language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Database`). > > Identifiers cannot use reserved keywords. > > See Identifier Requirements and Reserved Keywords for more details.

2.<table_name> > Specifies the table identifier (name), which must be unique within the database in which it is located. > > Identifiers must begin with an alphabetic character (or any character in a language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g. `My Object`). > > Identifiers cannot use reserved keywords. > > For more details, see Identifier Requirements and Reserved Keywords.

Optional Parameters

1.<partition_name> > Specifies the identifier (name) of the partition. > > Identifiers must begin with an alphabetic character (or any character in a script if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g. `My Object`). > > Identifiers cannot use reserved keywords. > > See Identifier Requirements and Reserved Keywords for more details.

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
Drop_priv	Table	TRUNCATE TABLE belongs to the table DROP operation

Usage Notes

- This statement clears data but retains the table or partition.
- Unlike DELETE, this statement can only clear the specified table or partition as a whole and cannot add filtering conditions.
- Unlike DELETE, clearing data in this way will not affect query performance.
- The data deleted by this operation cannot be recovered.

- When using this command, the table status must be NORMAL, that is, operations such as SCHEMA CHANGE are not allowed.
- This command may cause the import in progress to fail.

Examples

1. Clear the table tbl under example_db

```
TRUNCATE TABLE example_db.tbl;
```

2. Clear the p1 and p2 partitions of table tbl

```
TRUNCATE TABLE tbl PARTITION(p1, p2);
```

DROP TABLE

Description

This statement is used to delete a Table. ### Syntax

```
DROP TABLE [IF EXISTS] [<db_name>.]<table_name> [FORCE];
```

Required Parameters

1.`<table_name>` > Specifies the table identifier (name), which must be unique within the database in which it is located. > > Identifiers must begin with an alphabetic character (or any character in a language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g. `My Object`). > > Identifiers cannot use reserved keywords. > > For more details, see Identifier Requirements and Reserved Keywords.

Optional Parameters

1.`<db_name>` > Specifies the identifier (name) for the database. > > Identifiers must begin with an alphabetic character (or any character in a given language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Database`). > > Identifiers cannot use reserved keywords. > > See Identifier Requirements and Reserved Keywords for more details.

2.`FORCE` > If specified, the system will not check whether there are any unfinished transactions in the table. The table will be deleted directly and cannot be recovered. This operation is generally not recommended.

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
Drop_priv	Table	DROP TABLE belongs to the table DROP operation

Usage Notes

- After executing `DROP TABLE` for a period of time, the deleted table can be restored by using the RECOVER statement. For details, see the RECOVER statement.
- If you execute `DROP TABLE FORCE`, the system will not check whether there are unfinished transactions for the table. The table will be deleted directly and cannot be restored. Generally, this operation is not recommended.

Examples

1. Deleting a Table

```
DROP TABLE my_table;
```

2. If it exists, delete the Table of the specified Database.

```
DROP TABLE IF EXISTS example_db.my_table;
```

3. If it exists, delete the Table of the specified Database, forcibly delete it

```
DROP TABLE IF EXISTS example_db.my_table FORCE;
```

SHOW CREATE TABLE

Description

This statement is used to display the creation statement of the data table.

Syntax

```
SHOW [BRIEF] CREATE TABLE [<db_name>.]<table_name>
```

Required Parameters

1.`<table_name>` > Specifies the table identifier (name), which must be unique within the database in which it is located. > > Identifiers must begin with an alphabetic character (or any character in a language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g. `My Object`). > > Identifiers cannot use reserved keywords. > > For more details, see Identifier Requirements and Reserved Keywords.

Optional Parameters

1.`BRIEF` > Display only basic information about the table, excluding column definitions.

2.`<db_name>` > Specifies the identifier (i.e., name) for the database. > > Identifiers must begin with an alphabetic character (or any character in a given language if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Database`). > > Identifiers cannot use reserved keywords. > > See Identifier Requirements and Reserved Keywords for more details.

Return Value

column name	description
Table	Table name
Create Table	Create table statement

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
Select_priv	Table	SHOW CREATE TABLE belongs to table SELECT operation

Examples

1. View the creation statement of a table

```
SHOW CREATE TABLE demo.test_table;
```

2. View the simplified table creation statement for a table

```
SHOW BRIEF CREATE TABLE demo.test_table;
```

SHOW TABLES

Description

This statement is used to display all tables and views under the current db.

Syntax

```
SHOW [ FULL ] TABLES [ FROM [ <catalog_name>.]<db_name> ][ LIKE <like_condition> ]
```

Optional parameters

1. `FULL` > If this parameter is added to the statement, the returned result will have three more columns, namely Table_type (table type), Storage_format (storage format), and Inverted_index_storage_format (inverted index storage format).

2. `FROM [ <catalog_name>.]<db_name>` > In the FROM clause, you can specify the catalog name and database name to be queried.

2. `LIKE <like_condition>` > In the LIKE clause, you can perform fuzzy queries based on table names.

Return value

Column name (Column)	Type (DataType)	Notes (Notes)
Tables_in_	String	All tables and views under the database where `<db_name>` is located.
Table_type	String	Table and view type.
Storage_format	String	Table and view storage format.
Inverted_index_storage_format	String	Table and view inverted index storage format.

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege (Privilege)	Object (Object)	Notes (Notes)
SELECT_PRIV	Table (Table), View (View)	Only tables and views with query permissions can be displayed.

Usage Notes

- If the FROM clause is not specified in the statement, the corresponding database needs to be used for execution.

Examples

- View all tables under DB

```
SHOW TABLES;
```

```
+---------------------------------+
| Tables_in_demo                  |
+---------------------------------+
| ads_client_biz_aggr_di_20220419 |
| cmy1                            |
| cmy2                            |
| intern_theme                    |
| left_table                      |
+---------------------------------+
```

- Fuzzy query by table name

```
SHOW TABLES LIKE '%cm%'
```

```
+---------------+
| Tables_in_demo |
+---------------+
| cmy1           |
| cmy2           |
+---------------+
```

- Use FULL to query the tables and views under db

```
SHOW FULL TABLES
```

```
+---------------+------------+---------------+------------------------------+
| Tables_in_demo | Table_type | Storage_format | Inverted_index_storage_format |
+---------------+------------+---------------+------------------------------+
| test_table     | BASE TABLE | V2             | V1                           |
| test_view      | VIEW       | NONE           | NONE                         |
+---------------+------------+---------------+------------------------------+
```

SHOW TABLE ID

Description

This statement is used to find the corresponding database name, table name according to table id.

Syntax

```
SHOW TABLE <table_id>
```

Required parameters

1. <table_id> > Need to find <table_id> of database name, table name table.

Return value

Column name (Column)	Type (DataType)	Notes (Notes)
DbName	String	Database name
TableName	String	Table name
DbId	String	Database ID

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege (Privilege)	Object (Object)	Notes (Notes)
ADMIN_PRIV	Table (table)	Currently only supports **ADMIN** permissions to perform this operation

Examples

- Find the corresponding database name, table name according to the table id

```
SHOW TABLE 2261121
```

```
+--------+------------+---------+
| DbName | TableName  | DbId    |
+--------+------------+---------+
| demo   | test_table | 2261034 |
+--------+------------+---------+
```

SHOW TABLE STATUS

Description

This statement is used to display some information about a table or view.

Syntax

```
SHOW TABLE STATUS [ FROM [ <catalog_name>.]<db_name> ] [ LIKE <like_condition> ]
```

Optional parameters

1. `FROM [ <catalog_name>.]<db_name>` > The catalog name and database name to be queried can be specified in the FROM clause.

2. `LIKE <like_condition>` > The LIKE clause can perform fuzzy queries based on the table name.

Return value

Column	Data Type	Notes
Name	String	Table name
Engine	String	Storage engine for the table
Version	String	Version
Row_format	String	Row format. For the MyISAM engine, this may be Dynamic, Fixed, or Compressed. Dynamic rows have variable length, such as Varchar or Blob type fields. Fixed rows have fixed length, such as Char and Integer type fields.
Rows	String	Number of rows in the table. For non-transactional tables, this value is exact, for transactional engines, this value is usually estimated.
Avg_row_length	Integer	Average number of bytes per row
Data_length	Integer	The amount of data in the entire table (in bytes)
Max_data_length	Integer	The maximum amount of data that a table can hold

Column	Data Type	Notes
Index_length	Integer	The amount of disk space occupied by an index
Data_free	Integer	For the MyISAM engine, identifies the space that has been allocated but is now unused, and includes the space for deleted rows.
Auto_increment	Integer	The value of the next Auto_increment
Create_time	Datetime	The creation time of the table
Update_time	Datetime	The last update time of the table
Check_time	Datetime	The last time to check the table using the check table or myisamchk tool
Collation	String	The default character set of the table, currently only supports utf-8
Checksum	String	If enabled, the checksum calculated for the entire table content
Create_options	String	Refers to all other options when the table is created
Comment	String	Table comment

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Table, View	Currently only supports **ADMIN** permissions to perform this operation

Usage Notes

- This statement is mainly used for compatibility with MySQL syntax. Currently, only a small amount of information such as Comment is displayed.

Examples

- View information about all tables in the current database

```
SHOW TABLE STATUS
```

```
+------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+------
-----+----------------+---------------------+---------------------+------------+-----------+----------+----------------+---
------+
| Name       | Engine | Version | Row_format | Rows | Avg_row_length | Data_length | Max_data_length | Index_length |
Data_free | Auto_increment | Create_time         | Update_time         | Check_time | Collation | Checksum | Create_options
| Comment |
+------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+------
-----+----------------+---------------------+---------------------+------------+-----------+----------+----------------+---
------+
| test_table | Doris  |    NULL | NULL       |    0 |              0 |           0 |            NULL |         NULL |
NULL |           NULL | 2025-01-22 11:45:36 | 2025-01-22 11:45:36 | NULL       | utf-8     |     NULL | NULL           |
|
| test_view  | View   |    NULL | NULL       |    0 |              0 |           0 |            NULL |         NULL |
NULL |           NULL | 2025-01-22 11:46:32 | NULL                | NULL       | utf-8     |     NULL | NULL           |
|
+------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+------
-----+----------------+---------------------+---------------------+------------+-----------+----------+----------------+---
------+
```

- View information about tables whose names contain example under the specified database

```
SHOW TABLE STATUS FROM db LIKE "%test%"
```

```
+------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+------
-----+----------------+---------------------+---------------------+------------+-----------+----------+----------------+---
------+
| Name       | Engine | Version | Row_format | Rows | Avg_row_length | Data_length | Max_data_length | Index_length |
Data_free | Auto_increment | Create_time         | Update_time         | Check_time | Collation | Checksum | Create_options
| Comment |
+------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+------
-----+----------------+---------------------+---------------------+------------+-----------+----------+----------------+---
------+
| test_table | Doris  |    NULL | NULL       |    0 |              0 |           0 |            NULL |         NULL |
NULL |           NULL | 2025-01-22 11:45:36 | 2025-01-22 11:45:36 | NULL       | utf-8     |     NULL | NULL           |
|
| test_view  | View   |    NULL | NULL       |    0 |              0 |           0 |            NULL |         NULL |
NULL |           NULL | 2025-01-22 11:46:32 | NULL                | NULL       | utf-8     |     NULL | NULL           |
|
+------------+--------+---------+------------+------+----------------+-------------+-----------------+--------------+------
-----+----------------+---------------------+---------------------+------------+-----------+----------+----------------+---
------+
```

SHOW CONVERT LIGHT SCHEMA CHANGE PROCESS

Description

Used to view the conversion process of non-light schema change OLAP tables to light schema change tables.

Syntax

```
SHOW CONVERT_LIGHT_SCHEMA_CHANGE_PROCESS [ FROM <db_name> ]
```

Optional parameters

1. `<db_name>` > The name of the database to be queried can be specified in the FROM clause.

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Database	Currently only supports **ADMIN** permissions to perform this operation

Usage Notes

- To execute this statement, you need to enable the configuration `enable_convert_light_weight_schema_change`.

Examples

- View the conversion on database test

  ```
  SHOW CONVERT_LIGHT_SCHEMA_CHANGE_PROCESS FROM test;
  ```

- View global conversion status

  ```
  SHOW CONVERT_LIGHT_SCHEMA_CHANGE_PROCESS;
  ```

SHOW PARTITION

Description

SHOW PARTITION is used to display detailed information about a specified partition. This includes the name and ID of the associated database, the name and ID of the associated table, and the partition name.

Syntax

```
SHOW PARTITION <partition_id>
```

Required Parameters

> The ID of the partition. The partition ID can be obtained through methods such as SHOW PARTITIONS. For more information, please refer to the “SHOW PARTITIONS” section.

Access Control Requirements

The user executing this SQL command must have at least `ADMIN_PRIV` permissions.

Examples

Query partition information for partition ID 13004:

```
SHOW PARTITION 13004;
```

Results:

```
+--------+-----------+---------------+-------+---------+
| DbName | TableName | PartitionName | DbId  | TableId |
+--------+-----------+---------------+-------+---------+
| ods    | sales     | sales         | 13003 | 13005   |
+--------+-----------+---------------+-------+---------+
```

SHOW PARTITION ID

Description

This statement is used to find the corresponding database name, table name, and partition name based on the partition ID.

Syntax

```
SHOW PARTITION <partition_id>
```

Required Parameters

1. `<partition_id>`

> partition id

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV		

Examples

1. To find the corresponding database name, table name, and partition name based on the partition ID.

    ```
    SHOW PARTITION 10002;
    ```

SHOW PARTITIONS

Descriptions

This statement is used to display partition information. It supports both Internal catalog and Hive Catalog.

For `Hive Catalog`:

It supports returning all partitions, including multi-level partitions. ### Syntax

```
SHOW [ TEMPORARY ] PARTITIONS
    FROM [ <db_name>. ] <table_name>
    [ <where_clause> ]
    [ ORDER BY <order_by_key> ]
    [ LIMIT <limit_rows> ];
```

Required Parameters

1. `<table_name>`

The name of the table for which partition information needs to be viewed must be specified.

Optional Parameters

1. `TEMPORARY`

Whether to query the information of temporary partitions.

2. `<db_name>`

The name of the database for which partition information needs to be viewed must be specified.

3. `<where_clause>`

Filter conditions, supporting filtering on columns such as PartitionId, PartitionName, State, Buckets, ReplicationNum, LastConsistencyCheckTime, etc.

Please note that: 1. Currently, the where clause only supports the `=`, `!=`, `like` operator for string type PartitionName, State. For the other columns only support operators such as `=`, `!=`, `>`, `<`, `>=`, `<=`. 2. When using the above operators in the where clause, the column name needs to be on the left side. 3. Where clause can contains `AND`.

4. `<order_by_key>`

Sorting conditions, supporting sorting on columns such as PartitionId, PartitionName, State, Buckets, ReplicationNum, LastConsistencyCheckTime, etc.

5. `<limit_rows>`

The maximum number of rows returned.

Return Value

Columns	DataType	Note
PartitionId	bigint	Partition ID
PartitionName	varchar	Partition Name
VisibleVersion	int	The maximum VisibleVersion of the tablets in this partition.
VisibleVersionTime	datetime	The time of the most recent VisibleVersion in this partition.
State	varchar	The State of this partition
PartitionKey	datetime	The partition key of this partition
Range	datetime	The Range of this partition
DistributionKey	varchar	The distribution key of this partition
Buckets	int	The bucket num of this partition
ReplicationNum	int	The replica num of this partition
StorageMedium	varchar	The storage medium of this partition
CooldownTime	datetime	The cooldown time of this partition. If there is no hot-cold separation, the value of this field is [9999-12-31 23:59:59], which means it is always hot data.
RemoteStoragePolicy	varchar	The remote storage policy of this partition.
LastConsistencyCheckTime	datetime	The time of the last partition consistency check for this partition.
DataSize	int	The data size under this partition.
IsInMemory	boolean	Whether it is an in-memory partition, the default is false.
ReplicaAllocation	varchar	The replica distribution strategy of this partition.
IsMutable	boolean	Whether the partition is mutable, the default is true.
SyncWithBaseTables	boolean	Whether the partition is synchronized with the data of the base table.
UnsyncTables	varchar	Whether the partition is a partition of an unsynchronized table.

Access Control Requirements

The SHOW permission for the table to be viewed is required.

Examples

1. Show all non-temporary partition information for a specified table in a specified database.

```
SHOW PARTITIONS FROM t_agg;
```

```
+-------------+---------------+----------------+---------------------+--------+--------------+-------+-----------------+---------+--
-------------+---------------+---------------------+---------------------+--------------------------+-----------+------------+-----
--------------------+-----------+--------------------+--------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range | DistributionKey | Buckets |
ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime | DataSize  | IsInMemory |
ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+---------------+----------------+---------------------+--------+--------------+-------+-----------------+---------+--
-------------+---------------+---------------------+---------------------+--------------------------+-----------+------------+-----
--------------------+-----------+--------------------+--------------+
| 170307      | t_agg         | 4              | 2024-11-05 16:13:40 | NORMAL |              |       | k1              | 1       | 1
| HDD           | 9999-12-31 23:59:59 |                     | NULL                     | 806.000 B | false      |
tag.location.default: 1 | true      | true               | NULL         |
+-------------+---------------+----------------+---------------------+--------+--------------+-------+-----------------+---------+--
-------------+---------------+---------------------+---------------------+--------------------------+-----------+------------+-----
--------------------+-----------+--------------------+--------------+
```

2. Show all temporary partition information for a specified table in a specified database.

```
SHOW TEMPORARY PARTITIONS FROM t_temp;
```

```
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------------------------------+-----------------+---------+---------------+--------------+--
------------------+--------------------+--------------------------+----------+------------+-------------------------+-----------+-
------------------+--------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------------------------------+-----------------+---------+---------------+--------------+--
------------------+--------------------+--------------------------+----------+------------+-------------------------+-----------+-
------------------+--------------+
| 828863      | tp2020        | 1              | 2025-01-22 16:19:50 | NORMAL | create_time  | [types: [DATETIMEV2]; keys: [2020-01-
01 00:00:00]; ..types: [DATETIMEV2]; keys: [2021-01-01 00:00:00]; ) | reference_no    | 1       | 1              | SSD           |
9999-12-31 23:59:59 |                     | NULL                     | 0.000    | false      | tag.location.default: 1 | true      |
true               | NULL         |
+-------------+---------------+----------------+---------------------+--------+--------------+-----------------------------------
-----------------------------------------------------------------+-----------------+---------+---------------+--------------+--
------------------+--------------------+--------------------------+----------+------------+-------------------------+-----------+-
------------------+--------------+
```

3. Show the information of a specified non-temporary partition for a specified table in a specified database, and filter the results.

```
SHOW PARTITIONS FROM t_agg WHERE PartitionName = "p2024";
```

```
+-------------+---------------+---------------+---------------------+--------+----------------+-----------------------------------
-------------------------------------------------------------------------+----------------+---------+---------------+--------------
+--------------------+--------------------+-------------------------+-----------+------------+-------------------------+---------
--+-------------------+-------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey    | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize  | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+---------------+---------------+---------------------+--------+----------------+-----------------------------------
-------------------------------------------------------------------------+----------------+---------+---------------+--------------
+--------------------+--------------------+-------------------------+-----------+------------+-------------------------+---------
--+-------------------+-------------+
| 169851      | p2024         | 2              | 2024-11-05 14:14:29 | NORMAL | idp_create_time | [types: [DATETIMEV2]; keys: [2024-
01-01 00:00:00]; ..types: [DATETIMEV2]; keys: [2025-01-01 00:00:00]; ) | idp_es_id       | 3       | 1              | HDD
| 9999-12-31 23:59:59 |                     | NULL                    | 27.396 KB | false      | tag.location.default: 1 | true
| true               | NULL         |
+-------------+---------------+---------------+---------------------+--------+----------------+-----------------------------------
-------------------------------------------------------------------------+----------------+---------+---------------+--------------
+--------------------+--------------------+-------------------------+-----------+------------+-------------------------+---------
--+-------------------+-------------+
```

4. Show the information of the latest non-temporary partition for a specified table in a specified database.

```
SHOW PARTITIONS FROM t_agg ORDER BY PartitionId DESC LIMIT 1;
```

```
+-------------+---------------+---------------+---------------------+--------+----------------+-----------------------------------
-------------------------------------------------------------------------+----------------+---------+---------------+--------------
+--------------------+--------------------+-------------------------+-----------+------------+-------------------------+---------
-+-------------------+-------------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  | State  | PartitionKey    | Range
| DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime        | RemoteStoragePolicy | LastConsistencyCheckTime
| DataSize | IsInMemory | ReplicaAllocation       | IsMutable | SyncWithBaseTables | UnsyncTables |
+-------------+---------------+---------------+---------------------+--------+----------------+-----------------------------------
-------------------------------------------------------------------------+----------------+---------+---------------+--------------
+--------------------+--------------------+-------------------------+-----------+------------+-------------------------+---------
-+-------------------+-------------+
| 169866      | p2025         | 1              | 2024-11-05 14:13:56 | NORMAL | idp_create_time | [types: [DATETIMEV2]; keys: [2025-
01-01 00:00:00]; ..types: [DATETIMEV2]; keys: [2026-01-01 00:00:00]; ) | idp_es_id       | 3       | 1              | HDD
| 9999-12-31 23:59:59 |                     | NULL                    | 0.000    | false      | tag.location.default: 1 | true
| true               | NULL         |
+-------------+---------------+---------------+---------------------+--------+----------------+-----------------------------------
-------------------------------------------------------------------------+----------------+---------+---------------+--------------
+--------------------+--------------------+-------------------------+-----------+------------+-------------------------+---------
-+-------------------+-------------+
```

SHOW DYNAMIC PARTITION

Descriptions

This statement is used to display the status of all dynamic partition tables in the current database.

Syntax

```
SHOW DYNAMIC PARTITION TABLES [FROM <db_name>];
```

Required Parameters

1. `<db_name>`

Specify the `DB` name to display the status of dynamic partition tables. If not specified, the status of all dynamic partition tables in the current `DB` will be displayed by default. ### Return Value

Column	DataType	Note
TableName	varchar	The name of the table in the current DB or the specified DB.
Enable	varchar	Whether the dynamic partition property of the table is enabled
TimeUnit	varchar	The partition granularity of the dynamic partition table, including `HOUR`, `DAY`, `WEEK`, `MONTH`, `YEAR`.

Column	DataType	Note
Start	varchar	The starting offset of the dynamic partition, which is a negative number. The default value is -2147483648, which means that historical partitions will not be deleted. Depending on the time_unit attribute, partitions with a range before this offset will be deleted based on the current day (week/month).
End	varchar	The ending offset of the dynamic partition, which is a positive number. Depending on the time_unit attribute, partitions within the corresponding range are created in advance based on the current day (week/month).
Prefix	varchar	The prefix of the dynamically created partition name.
Buckets	varchar	The number of buckets corresponding to the dynamically created partition.
ReplicationNum	varchar	The number of replicas corresponding to the dynamically created partition. If not specified, it defaults to the number of replicas specified when the table was created.
ReplicaAllocation	varchar	The replica distribution strategy corresponding to the dynamically created partition. If not specified, it defaults to the replica distribution strategy specified when the table was created.

Column	DataType	Note
StartOf	varchar	The starting point of each partition granularity for dynamic partitioning. When time_unit is WEEK, this field represents the starting point of each week, with values ranging from MONDAY to SUNDAY. When time_unit is MONTH, it represents the starting date of each month, with values ranging from 1st to 28th. When time_unit is MONTH, this value defaults to NULL.
LastUpdateTime	datetime	The last update time of the dynamic partition, which defaults to NULL.
LastSchedulerTime	datetime	The last scheduling time of the dynamic partition.
State	varchar	The state of the dynamic partition.
LastCreatePartitionMsg	varchar	The error message from the last execution of the dynamic partition addition scheduling.
LastDropPartitionMsg	varchar	The error message from the last execution of the dynamic partition deletion scheduling.

Column	DataType	Note
ReservedHistoryPeriods	varchar	The partition range of the historical partitions retained by the dynamic partition, which indicates which historical partitions should be retained in the dynamic partition table instead of being automatically deleted.

Access Control Requirements

1. If the parameter db_name is not specified, the status of all dynamic partition tables in the current DB will be displayed, and it is assumed that the user has the `SHOW_PRIV` privilege for the current DB by default.
2. If the parameter db_name is specified, the status of all dynamic partition tables in the specified DB will be displayed, and the user needs to have the `SHOW_PRIV` privilege for that DB.

Examples

1. View the status of all dynamic partition tables in the current database:

```
SHOW DYNAMIC PARTITION TABLES;
```

```
+-----------+--------+----------+-------------+------+--------+---------+-----------+----------------+---------------------+--------
+-----------------------+---------------------+-------------------------+
| TableName | Enable | TimeUnit | Start       | End  | Prefix | Buckets | StartOf   | LastUpdateTime | LastSchedulerTime   | State
| LastCreatePartitionMsg | LastDropPartitionMsg | ReservedHistoryPeriods  |
+-----------+--------+----------+-------------+------+--------+---------+-----------+----------------+---------------------+--------
+-----------------------+---------------------+-------------------------+
| d3        | true   | WEEK     | -3          | 3    | p      | 1       | MONDAY    | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                 | [2021-12-01,2021-12-31] |
| d5        | true   | DAY      | -7          | 3    | p      | 32      | N/A       | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                 | NULL                    |
| d4        | true   | WEEK     | -3          | 3    | p      | 1       | WEDNESDAY | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                 | NULL                    |
| d6        | true   | MONTH    | -2147483648 | 2    | p      | 8       | 3rd       | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                 | NULL                    |
| d2        | true   | DAY      | -3          | 3    | p      | 32      | N/A       | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                 | NULL                    |
| d7        | true   | MONTH    | -2147483648 | 5    | p      | 8       | 24th      | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                 | NULL                    |
+-----------+--------+----------+-------------+------+--------+---------+-----------+----------------+---------------------+--------
+-----------------------+---------------------+-------------------------+
```

2. View the status of all dynamic partition tables in the specified database:

```
SHOW DYNAMIC PARTITION TABLES FROM test;
```

```
+-----------+--------+----------+-------------+------+--------+---------+-----------+---------------+--------------------+--------
+-----------------------+----------------------+------------------------+
| TableName | Enable | TimeUnit | Start       | End  | Prefix | Buckets | StartOf   | LastUpdateTime | LastSchedulerTime   | State
| LastCreatePartitionMsg | LastDropPartitionMsg | ReservedHistoryPeriods  |
+-----------+--------+----------+-------------+------+--------+---------+-----------+---------------+--------------------+--------
+-----------------------+----------------------+------------------------+
| test1     | true   | WEEK     | -30          | 3    | p      | 8       | MONDAY    | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | [2021-12-01,2021-12-31] |
| test2     | true   | DAY      | -7           | 3    | p      | 32      | N/A       | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | NULL                    |
| test3     | true   | WEEK     | -3           | 3    | p      | 1       | WEDNESDAY | N/A            | 2020-05-25 14:29:24 | NORMAL
| N/A                   | N/A                  | NULL                    |
+-----------+--------+----------+-------------+------+--------+---------+-----------+---------------+--------------------+--------
+-----------------------+----------------------+------------------------+
```

SHOW COLUMNS

Description

This statement is used to specify the column information of a table.

Syntax

```
SHOW [ FULL ] COLUMNS FROM <tbl>;
```

Required Parameters

1. `<tbl>`

The name of the table for which column information needs to be viewed must be specified.

Optional Parameters

1. `FULL`

If the `FULL` keyword is specified, detailed information about the columns will be returned, including the aggregation type, permissions, comments, etc. of the columns.

Return Value

Column	DataType	Note
Field	varchar	Column Name
Type	varchar	Column Data Type
Collation	varchar	Column Collation
Null	varchar	Whether NULL is Allowed
Key	varchar	Table's Primary Key
Default	varchar	Default Value
Extra	varchar	Extra Info
Privileges	varchar	Column Privileges
Comment	varchar	Column Comment

Access Control Requirements

Requires the `SHOW` privilege for the table to be viewed.

Examples

1. View detailed column information of the specified table

```
SHOW FULL COLUMNS FROM t_agg;
```

```
+-------+-----------------+-----------+------+------+---------+---------+------------+---------+
| Field | Type            | Collation | Null | Key  | Default | Extra   | Privileges | Comment |
+-------+-----------------+-----------+------+------+---------+---------+------------+---------+
| k1    | tinyint         |           | YES  | YES  | NULL    |         |            |         |
| k2    | decimalv3(10,2) |           | YES  | YES  | 10.5    |         |            |         |
| v1    | char(10)        |           | YES  | NO   | NULL    | REPLACE |            |         |
| v2    | int             |           | YES  | NO   | NULL    | SUM     |            |         |
+-------+-----------------+-----------+------+------+---------+---------+------------+---------+
```

2. View the normal column information of the specified table

```
SHOW COLUMNS FROM t_agg;
```

```
+-------+-----------------+------+------+---------+---------+
| Field | Type            | Null | Key  | Default | Extra   |
+-------+-----------------+------+------+---------+---------+
| k1    | tinyint         | YES  | YES  | NULL    |         |
| k2    | decimalv3(10,2) | YES  | YES  | 10.5    |         |
| v1    | char(10)        | YES  | NO   | NULL    | REPLACE |
| v2    | int             | YES  | NO   | NULL    | SUM     |
+-------+-----------------+------+------+---------+---------+
```

ALTER COLOCATE GROUP

Description

This statement is used to modify the properties of a Colocation Group.

Syntax

```
ALTER COLOCATE GROUP  [<database>.] <group_name>
SET (
    <property_list>
);
```

Required Parameters

1. `<group_name>`

Specify the name of the colocate group to be modified.

2.`<property_list>`

`property_list` is a property of the `colocation group`, and currently only supports modifying `replication_num` and `replication_allocation`. After modifying these two properties of the `colocation group`, simultaneously change the properties `default.replication_allocation`, `dynamic.replication_allocation`, and replication_allocation of the existing partitions of the group's tables to be the same as it.

Optional Parameters

1. `<database>`

Specify the database to which the `colocate group` to be modified belongs.

Note: 1. If the colocate group is global, that is, its name starts with **global**, then it does not belong to any Database

Access Control Requirements

Requires `ADMIN` permissions.

Examples

1. Modify the replica number of a global group, and set `"colocate_with" = "__global__foo"` when creating the table.

```
ALTER COLOCATE GROUP __global__foo
SET (
```

```
    "replication_num"="1"
    );
```

2. Modify the replica number of a non-global group, and set “colocate_with” = “bar” when creating the table, and the table belongs to Database example_db.

```
ALTER COLOCATE GROUP example_db.bar
SET (
   "replication_num"="1"
   );
```

CREATE INDEX

Description

Create a new index on a table. The table name and index name must be specified. Optionally, you can specify the index type, properties, and comments.

Syntax

```
CREATE INDEX [IF NOT EXISTS] <index_name>
             ON <table_name> (<column_name> [, ...])
             [USING {INVERTED | NGRAM_BF}]
             [PROPERTIES ("<key>" = "<value>"[ , ...])]
             [COMMENT '<index_comment>']
```

Required Parameters

1. `<index_name>`

> Specifies the identifier (i.e., name) of the index, which must be unique within its table.
>
> The identifier must start with a letter character (if Unicode name support is enabled, it can be any character from any language) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> The identifier cannot be a reserved keyword.
>
> For more details, refer to the requirements for identifiers and reserved keywords.

2. `<table_name>`

> Specifies the identifier (i.e., name) of the table, which must be unique within its database.
>
> The identifier must start with a letter character (if Unicode name support is enabled, it can be any character from any language) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> The identifier cannot be a reserved keyword.
>
> For more details, refer to the requirements for identifiers and reserved keywords.

3. `<column_name> [, ...]`

> Specifies the columns on which the index is created (currently only one column is supported), and the column must be unique within its table.
>
> The identifier must start with a letter character (if Unicode name support is enabled, it can be any character from any language) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> The identifier cannot be a reserved keyword.
>
> For more details, refer to the requirements for identifiers and reserved keywords.

Optional Parameters

1. `USING {INVERTED | NGRAM_BF}`

> Specifies the index type. Currently, two types are supported: **INVERTED** (inverted index) and **NGRAM_BF** (ngram bloomfilter index).

2. `PROPERTIES ("<key>" = "<value>"[ , ...])`

> Specifies the parameters of the index using the general PROPERTIES format. For the parameters and semantics supported by each index, refer to the documentation for the specific index type.

3. `COMMENT '<index_comment>'`

> Specifies a comment for the index to facilitate maintenance.

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ALTER_PRIV	Table	CREATE INDEX is considered an ALTER operation on the table

Usage Notes

- The **INVERTED** inverted index takes effect immediately for newly inserted data. For historical data, the index needs to be built using the BUILD INDEX operation.
- The **NGRAM_BF** NGram BloomFilter index will perform a schema change in the background on all data to complete the index construction after creation. The progress can be checked using SHOW ALTER TABLE COLUMN.

Examples

- Create an inverted index `index1` on `table1`

```
CREATE INDEX index1 ON table1 USING INVERTED;
```

- Create an NGram BloomFilter index `index2` on `table1`

```
CREATE INDEX index2 ON table1 USING NGRAM_BF PROPERTIES("gram_size"="3", "bf_size"="1024");
```

DROP INDEX

Description

This statement is used to delete an index with a specified name from a table. Currently, only bitmap indexes are supported.

Syntax

```
DROP INDEX [ IF EXISTS ] <index_name> ON [ <db_name> . ] <table_name>;
```

Required Parameters

1. <index_name>: The name of the index.

2. <table_name>：The name of the table to which the index belongs.

Optional Parameters

1. <db_name>：The database name, optional. If not specified, the current database is used by default.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Table	DROP INDEX is an ALTER operation on the table

Examples

- drop index

  ```
  DROP INDEX IF NOT EXISTS index_name ON table1 ;
  ```

SHOW INDEX

Description

This statement is used to display information about indexes in a table. Currently, only bitmap indexes are supported.

Syntax

```
SHOW INDEX [ ES ] FROM [ <db_name>. ] <table_name> [ FROM <db_name> ];
```

Varaint Syntax

```
SHOW KEY[ S ] FROM [ <db_name>. ] <table_name> [ FROM <db_name> ];
```

Required Parameters

1. `<table_name>`: The name of the table to which the index belongs.

Optional Parameters

1. `<db_name>`: The database name, optional. If not specified, the current database is used by default.

Return Value

Column Name	Type	Description
Table	string	The name of the table where the index is located.
Non_unique	int	Indicates whether the index is unique: - `0`: Unique index - `1`: Non-unique index
Key_name	string	The name of the index.
Seq_in_index	int	The position of the column in the index. This column shows the order of the column in the index, used when multiple columns form a composite index.
Column_name	string	The name of the indexed column.
Collation	string	The sorting order of the index column: - `A`: Ascending - `D`: Descending.

Column Name	Type	Description
Cardinality	int	The number of unique values in the index. This value is used to estimate query efficiency; the higher the value, the higher the index selectivity and query efficiency.
Sub_part	int	The prefix length used by the index. If the index column is a string type, `Sub_part` represents the length of the first few characters of the index.
Packed	string	Whether the index is compressed.
Null	string	Whether `NULL` values are allowed: - `YES`: NULL values allowed - `NO`: NULL values not allowed
Index_type	string	The type of index: - `BTREE`: B+ tree index (default type in MySQL) - `HASH`: Hash index - `RTREE`: R-tree index - `INVERTED`: Inverted index (e.g., full-text index)
Comment	string	The comment or description of the index, typically custom remarks.
Properties	string	Additional properties of the index.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
SHOW_PRIV	Database	

Examples

- Display indexes for a specified `table_name`

```
SHOW INDEX FROM example_db.table_name;
```

BUILD INDEX

Description

Build an index for the entire table or table partitions. You must specify the table name and index name, and optionally specify the partition list.

Syntax

```
BUILD INDEX <index_name> ON <table_name> [partition_list]
```

Where:

```
partition_list
  : PARTITION (<partition_name1>[ , parition_name2 ][ ... ])
```

Required Parameters

Specifies the identifier (name) of the index, which must be unique within its table.

The identifier must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).

The identifier cannot use reserved keywords.

For more details, see Identifier Requirements and Reserved Keywords.

Specifies the identifier (name) of the table, which must be unique within its database.

The identifier must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).

The identifier cannot use reserved keywords.

For more details, see Identifier Requirements and Reserved Keywords.

Optional Parameters

> Specifies a list of partition identifiers (names) separated by commas, which must be unique within their table.
>
> The identifier must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> The identifier cannot use reserved keywords.
>
> For more details, see Identifier Requirements and Reserved Keywords.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Table	BUILD INDEX is an ALTER operation on a table

Usage Notes

- Currently only valid for inverted indexes, other indexes such as BloomFilter index are not valid.
- Currently only valid for compute-storage integrated mode, not valid for compute-storage separated mode.
- The progress of BUILD INDEX can be viewed through SHOW BUILD INDEX

Examples

- Build index index1 on the entire table1.

```
BUILD INDEX index1 ON table1
```

- Build index index1 on partitions p1 and p2 of table1.

```
BUILD INDEX index1 ON table1 PARTITION(p1, p2)
```

CANCEL BUILD INDEX

Description

Cancel background tasks for index building.

Syntax

```
CANCEL BUILD INDEX ON <table_name> [ job_list ]
```

Where:

```
job_list
  : (<job_id1>[ , job_id2 ][ ... ])
```

Required Parameters

Specifies the identifier (i.e., name) of the table, which must be unique within its database (Database).

The identifier must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).

Identifiers cannot use reserved keywords.

For more details, see Identifier Requirements and Reserved Keywords.

Optional Parameters

Specifies a list of identifiers for index building tasks, separated by commas and enclosed in parentheses.

Identifiers must be numbers and can be viewed via SHOW BUILD INDEX.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Table	CANCEL BUILD INDEX is considered an ALTER operation on the table

Usage Notes

- Currently only effective for inverted indexes, not for other indexes such as bloomfilter index.
- Currently only effective for the integrated storage and computing mode, not for the separated storage and computing mode.
- The progress and index building tasks of BUILD INDEX can be viewed via SHOW BUILD INDEX.

Examples

- Cancel all index building tasks on table table1

```
CANCEL BUILD INDEX ON table1
```

- Cancel index building tasks jobid1 and jobid2 on table table1

```
CANCEL BUILD INDEX ON table1(jobid1, jobid2)
```

SHOW BUILD INDEX

Description

Check the status of index build tasks.

Syntax

```
SHOW BUILD INDEX [ (FROM | IN) <database_name>
[ where_clause ] [ sort_clause ] [ limit_clause ] ]
```

Where:

```
where_clause
  : WHERE <output_column_name = value>
```

Where:

```
sort_clause
  :
   ORDER BY <output_column_name>
```

Where:

```
limit_clause
  :
   LIMIT <n>
```

Optional Parameters

`<database_name>`

> Specifies the identifier (name) of the database, which must be unique within its cluster.
>
> The identifier must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> Identifiers cannot use reserved keywords.
>
> For more details, see Identifier Requirements and Reserved Keywords.

`<WHERE output_column_name = value>`

> Specifies the output filter condition, where output_column_name must be in the output field list.

`<ORDER BY output_column_name>`

> Specifies the output sort column, where output_column_name must be in the output field list.

`LIMIT <n>`

> Specifies the limit on the number of output rows, where n must be a number.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
SHOW_PRIV	Database	

Usage Notes

- Currently only valid for inverted indexes, not other indexes such as bloomfilter index.
- Currently only valid for the integrated storage and computing mode, not valid for the separated storage and computing mode.

Examples

- View all index build tasks

```
SHOW BUILD INDEX
```

- View index build tasks for database database1

```
SHOW BUILD INDEX FROM database1
```

- View index build tasks for table table1

```
SHOW BUILD INDEX WHERE TableName = 'table1'
```

- View index build tasks for table table1, sorted by JobId and taking the first 10 rows

```
SHOW BUILD INDEX WHERE TableName = 'table1' ORDER BY JobId LIMIT 10
```

CREATE VIEW

Description

This statement is used to create a logical view using a specified query statement.

Syntax

```
CREATE VIEW [IF NOT EXISTS] [<db_name>.]<view_name>
   [(<column_definition>)]
AS <query_stmt>
```

Where:

```
column_definition:
    <column_name> [COMMENT '<comment>'] [,...]
```

Required Parameters

1. `<view_name>` > The identifier (i.e., name) of the view; it must be unique within the database where the view is created.
> The identifier must start with a letter character (if Unicode name support is enabled, it can be a character in any language) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My View`).
> The identifier cannot use reserved keywords.
> For more details, see identifier requirements and reserved keywords.

2. `<query_stmt>` > The SELECT query statement that defines the view.

Optional Parameters

1. `<db_name>` > The name of the database where the view resides. If not specified, the current database is used by default.

2. `<column_definition>` > The column definitions of the view.
> Where:
> **1.** `<column_name>`
> Column name.
> **2.** `<comment>`
> Column comment.

Access Control Requirements

Privilege	Object	Notes
CREATE_PRIV	Database	CREATE_PRIV privilege is required on the database.
SELECT_PRIV	Table, View	SELECT_PRIV privilege is required on the tables, views, or materialized views being queried.

Notes

- Views are logical and do not have physical storage. All queries on the view are equivalent to queries on the corresponding subquery.
- Creating and dropping views does not affect the data in the underlying tables.

Examples

1. Create a view `example_view` on `example_db`

```
CREATE VIEW example_db.example_view (k1, k2, k3, v1)
AS
SELECT c1 as k1, k2, k3, SUM(v1) FROM example_table
WHERE k1 = 20160112 GROUP BY k1,k2,k3;
```

2. Create a view with column definitions

```
CREATE VIEW example_db.example_view
(
    k1 COMMENT "first key",
    k2 COMMENT "second key",
    k3 COMMENT "third key",
    v1 COMMENT "first value"
)
COMMENT "my first view"
AS
SELECT c1 as k1, k2, k3, SUM(v1) FROM example_table
WHERE k1 = 20160112 GROUP BY k1,k2,k3;
```

ALTER VIEW

Description

This statement is used to modify the definition of a logical view.

Syntax

```
ALTER VIEW [<db_name>.]<view_name>
 [(<column_definition>)]
AS <query_stmt>
```

Where:

```
column_definition:
    <column_name> [COMMENT '<comment>'] [,...]
```

Required Parameters

1. `<view_name>` > The identifier (i.e., name) of the view to be modified.

2. `<query_stmt>` > The SELECT query statement that defines the view.

Optional Parameters

1. `<db_name>` > The name of the database where the view resides. If not specified, the current database is used by default.

2. `<column_definition>` > The column definitions of the view.
> Where:
> **1.** `<column_name>`
> Column name.
> **2.** `<comment>`
> Column comment.

Access Control Requirements

Privilege	Object	Notes
ALTER_PRIV	View	ALTER_PRIV privilege is required on the view being modified.
SELECT_PRIV	Table, View	SELECT_PRIV privilege is required on the tables, views, or materialized views being queried.

Example

1. Modify the view `example_view` on `example_db`

```
ALTER VIEW example_db.example_view
(
  c1 COMMENT "column 1",
  c2 COMMENT "column 2",
  c3 COMMENT "column 3"
)
AS SELECT k1, k2, SUM(v1) FROM example_table
GROUP BY k1, k2
```

DROP VIEW

Description

Delete a view in the current or specified database.

Syntax

```
DROP VIEW [ IF EXISTS ] <name>
```

Required Parameters

The name of the view to be deleted.

Optional Parameters

[IF EXISTS]

If this parameter is specified, no error will be thrown when the view does not exist, and the deletion operation will be skipped directly.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
DROP_PRIV	Table	

Usage Notes

Deleted views cannot be restored and must be recreated.

Examples

```
CREATE VIEW vtest AS SELECT 1, 'test';
DROP VIEW IF EXISTS vtest;
```

SHOW CREATE VIEW

Description

Display the CREATE VIEW statement used to create the specified view.

Syntax

```
SHOW CREATE VIEW <name>
```

Required Parameters

<name> The name of the view to view.

Result Description

- View: The name of the queried view.
- Create View: The persisted SQL statement in the database.
- character_set_client: The value of the character_set_client system variable in the session when the view was created.
- collation_connection: The value of the collation_connection system variable in the session when the view was created.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
SHOW_VIEW_PRIV	Table	

View information can also be queried via the INFORMATION_SCHEMA.VIEWS table.

Examples

```
CREATE VIEW vtest AS SELECT 1, 'test';
SHOW CREATE VIEW vtest;
```

Query result:

```
+-------+-----------------------------------------+----------------------+----------------------+
| View  | Create View                             | character_set_client | collation_connection |
+-------+-----------------------------------------+----------------------+----------------------+
| vtest | CREATE VIEW `vtest` AS SELECT 1, 'test'; | utf8mb4              | utf8mb4_0900_bin     |
+-------+-----------------------------------------+----------------------+----------------------+
```

SHOW VIEW

Description

This statement is used to display all views based on the given table

Syntax

```
SHOW VIEW { FROM | IN } table [ FROM db ]
```

Examples

1. Show all views created based on table testTbl

   ```
   SHOW VIEW FROM testTbl;
   ```

Keywords

SHOW, VIEW

CREATE SYNC MATERIALIZED VIEW

Description

Statement for creating a synchronized materialized view.

Syntax

```
CREATE MATERIALIZED VIEW <materialized_view_name> AS <query>
```

Where

```
query
    :
    SELECT <select_expr> select_expr[, select_expr ...]
    FROM <base_table>
    WHERE condition
    GROUP BY <column_name>[, <column_name> ...]
    ORDER BY <column_name>[, <column_name> ...]
```

Required Parameters

1. `<materialized_view_name>`

> Specifies the identifier (i.e., name) of the materialized view. Since a synchronized materialized view is created based on a table, the name must be unique within the same table.
>
> The identifier must start with a letter character (if Unicode name support is enabled, it can be any character from any language) and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., `My Object`).
>
> The identifier cannot be a reserved keyword.
>
> For more details, refer to the requirements for identifiers and reserved keywords.

2. `<query>`

> The query statement used to construct the materialized view, the result of which constitutes the data of the materialized view. The currently supported query format is:
>
> The syntax is consistent with the query statement syntax.
>
> - `select_expr`: All columns in the schema of the materialized view.
>   - Must include at least one single column.
> - `base_table`: The name of the base table for the materialized view, a required item.

    - Must be a single table, not a subquery.
- `where`: The filter condition of the materialized view, an optional item.
    - If not specified, no data filtering will be performed.
- `group by`: The grouping columns of the materialized view, an optional item.
    - If not specified, the data will not be grouped.
- `order by`: The sorting columns of the materialized view, an optional item.
    - The declaration order of the sorting columns must be consistent with the order of columns declared in `select_expr`.
    - If `order by` is not declared, sorting columns will be automatically supplemented according to the rules. If the materialized view is of the aggregate type, all grouping columns will be automatically added as sorting columns. If the materialized view is of the non-aggregate type, the first 36 bytes will be automatically added as sorting columns.
    - If the number of automatically supplemented sorting columns is less than 3, the first three will be used as sorting columns. If the query contains grouping columns, the sorting columns must be consistent with the grouping columns.

Access Control Requirements

Privilege	Object	Notes
ALTER_PRIV	Table	Requires ALTER_PRIV permission on the base table of the current materialized view

Notes

- Synchronized materialized views only support SELECT statements for a single table, supporting WHERE, GROUP BY, ORDER BY clauses, but not JOIN, HAVING, LIMIT clauses, or LATERAL VIEW.
- The SELECT list cannot contain auto-increment columns, constants, duplicate expressions, or window functions.
- If the SELECT list contains aggregate functions, the aggregate functions must be root expressions (e.g., `sum(a + 1)` is supported, but `sum(a) + 1` is not), and no other non-aggregate function expressions can follow the aggregate functions (for example, `SELECT x, sum(a)` is acceptable, but `SELECT sum(a), x` is not).
- Too many materialized views on a single table can affect the efficiency of data import: when importing data, the data of the materialized views and the Base table are updated synchronously. If there are too many materialized views on a table, it may slow down the import speed, similar to importing data into multiple tables simultaneously in a single import operation.
- When a materialized view targets the Unique Key data model, it can only change the order of columns and cannot perform aggregation. Therefore, on the Unique Key model, data cannot be coarsely aggregated by creating materialized views.
- When a materialized view targets Unique Key and Aggregate Key data models, if a WHERE clause is specified, it can only use Key columns and not Value columns.

Example

```
desc lineitem;
```

```
+-----------------+---------------+------+-------+---------+-------+
| Field           | Type          | Null | Key   | Default | Extra |
+-----------------+---------------+------+-------+---------+-------+
| l_orderkey      | int           | No   | true  | NULL    |       |
| l_partkey       | int           | No   | true  | NULL    |       |
| l_suppkey       | int           | No   | true  | NULL    |       |
| l_linenumber    | int           | No   | true  | NULL    |       |
| l_quantity      | decimal(15,2) | No   | false | NULL    | NONE  |
| l_extendedprice | decimal(15,2) | No   | false | NULL    | NONE  |
| l_discount      | decimal(15,2) | No   | false | NULL    | NONE  |
| l_tax           | decimal(15,2) | No   | false | NULL    | NONE  |
| l_returnflag    | char(1)       | No   | false | NULL    | NONE  |
| l_linestatus    | char(1)       | No   | false | NULL    | NONE  |
| l_shipdate      | date          | No   | false | NULL    | NONE  |
| l_commitdate    | date          | No   | false | NULL    | NONE  |
| l_receiptdate   | date          | No   | false | NULL    | NONE  |
| l_shipinstruct  | char(25)      | No   | false | NULL    | NONE  |
| l_shipmode      | char(10)      | No   | false | NULL    | NONE  |
| l_comment       | varchar(44)   | No   | false | NULL    | NONE  |
+-----------------+---------------+------+-------+---------+-------+
```

```
CREATE MATERIALIZED VIEW sync_agg_mv AS
SELECT
  l_shipdate,
  l_partkey,
  count(*),
  sum(l_discount)
FROM
  lineitem
GROUP BY
  l_shipdate,
  l_partkey;
```

DROP MATERIALIZED VIEW

Description

Drop a synchronized materialized view.

Syntax

```
DROP MATERIALIZED VIEW
[ IF EXISTS ] <materialized_view_name>
ON <table_name>
```

Required Parameters

1. `<materialized_view_name>`

> The name of the materialized view to be dropped.

2. `<table_name>`

> The table to which the materialized view belongs.

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ALTER_PRIV	Table	Requires ALTER_PRIV permission on the table to which the materialized view to be deleted belongs

Example

Drop the synchronized materialized view `sync_agg_mv` on the `lineitem` table

```
DROP MATERIALIZED VIEW sync_agg_mv on lineitem;
```

SHOW ALTER TABLE MATERIALIZED VIEW

Description

Check the status of the synchronized materialized view build task.

Since creating a synchronized materialized view is an asynchronous operation, after submitting the materialized view creation task, users need to asynchronously check the status of the synchronized materialized view build through a command.

Syntax

```
SHOW ALTER TABLE MATERIALIZED VIEW FROM <database>
```

Required Parameters

1. `<database>`

> The database to which the base table of the synchronized materialized view belongs.

Permissions

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ALTER_PRIV	Table	Requires ALTER_PRIV permission on the table to which the current materialized view belongs

Example

```
SHOW ALTER TABLE MATERIALIZED VIEW FROM doc_db;
```

SHOW CREATE SYNC MATERIALIZED VIEW

Description

View the materialized view creation statement.

Syntax

```
SHOW CREATE MATERIALIZED VIEW <materialized_view_name> ON <table_name>
```

Required Parameters

1. `<materialized_view_name>`

> The name of the materialized view.

2. `<table_name>`

> The table to which the materialized view belongs.

Return Values

Column Name	Description
TableName	Name of the table
ViewName	Name of the materialized view
CreateStmt	Statement used to create the materialized view

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
SELECT_PRIV/LOAD_PRIV/ALTER_PRIV/CREATE_PRIV/DROP_PRIV	Table	You need to have permissions for the table to which the current materialized view belongs

Examples

1. View the creation statement of a synchronized materialized view

```
SHOW CREATE MATERIALIZED VIEW sync_agg_mv on lineitem;
```

ALTER ASYNC MATERIALIZED VIEW

Description

This statement is used to modify asynchronous materialized views.

syntax

```
ALTER MATERIALIZED VIEW mvName=multipartIdentifier ((RENAME newName=identifier)
       | (REFRESH (refreshMethod | refreshTrigger | refreshMethod refreshTrigger))
       | REPLACE WITH MATERIALIZED VIEW newName=identifier propertyClause?
       | (SET  LEFT_PAREN fileProperties=propertyItemList RIGHT_PAREN))
```

illustrate

RENAME

Used to change the name of the materialized view

For example, changing the name of mv1 to mv2

```
ALTER MATERIALIZED VIEW mv1 rename mv2;
```

refreshMethod

Same as creating asynchronous materialized views

refreshTrigger

Same as creating asynchronous materialized views

SET

Modify properties unique to materialized views

For example, modifying the grace_period of mv1 to 3000ms

```
ALTER MATERIALIZED VIEW mv1 set("grace_period"="3000");
```

REPLACE

```
ALTER MATERIALIZED VIEW [db.]mv1 REPLACE WITH MATERIALIZED VIEW mv2
[PROPERTIES('swap' = 'true')];
```

Replacing atoms with two materialized views

swap default is TRUE - If the swap parameter is set to TRUE, it is equivalent to renaming the materialized view mv1 to mv2, and renaming mv2 to mv1 at the same time - If the swap parameter is set to FALSE, it is equivalent to renaming mv2 to mv1 and deleting the original mv1

For example, if you want to swap the names of mv1 and mv2

```
ALTER MATERIALIZED VIEW db1.mv1 REPLACE WITH MATERIALIZED VIEW mv2;
```

For example, if you want to rename mv2 to mv1 and delete the original mv1

```
ALTER MATERIALIZED VIEW db1.mv1 REPLACE WITH MATERIALIZED VIEW mv2
PROPERTIES('swap' = 'false');
```

Keywords

```
ALTER, ASYNC, MATERIALIZED, VIEW
```

CANCEL MATERIALIZED VIEW TASK

Description

This statement is used to cancel the task of materialized views

Syntax

```
CANCEL MATERIALIZED VIEW TASK <task_id> ON <mv_name>
```

Required Parameters

1. `<task_id>` > Specifies the task id of materialized view.

2. `<mv_name>` > Specifies the materialized view name. > > The materialized view name must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire materialized view name string is enclosed in backticks (e.g., `My Object`). > > The materialized view name cannot use reserved keywords. > > For more details, see Reserved Keywords.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Materialized View	CANCEL is an ALTER operation on a materialized view

Example

- Cancel the task with id 1 in materialized view mv1

```
CANCEL MATERIALIZED VIEW TASK 1 on mv1;
```

CREATE ASYNC MATERIALIZED VIEW

Description

Statement for creating an asynchronous materialized view. The column names and types are derived from the materialized view SQL statement. Custom column names are allowed, but column types cannot be defined.

Syntax

```
CREATE MATERIALIZED VIEW
[ IF NOT EXISTS ] <materialized_view_name>
    [ (<columns_definition>) ]
    [ BUILD <build_mode> ]
    [ REFRESH <refresh_method> [<refresh_trigger>]]
    [ [DUPLICATE] KEY (<key_cols>) ]
    [ COMMENT '<table_comment>' ]
    [ PARTITION BY (
        { <partition_col>
            | DATE_TRUNC(<partition_col>, <partition_unit>) }
        )]
    [ DISTRIBUTED BY { HASH (<distribute_cols>) | RANDOM }
        [ BUCKETS { <bucket_count> | AUTO } ]
    ]
    [ PROPERTIES (
          -- Table property
          <table_property>
          -- Additional table properties
          [ , ... ])
    ]
    AS <query>
```

Where:

```
columns_definition
  : -- Column definition
    <col_name>
      [ COMMENT '<col_comment>' ]
refresh_trigger
  : ON MANUAL
  | ON SCHEDULE EVERY <int_value> <refresh_unit> [ STARTS '<start_time>']
  | ON COMMIT
```

Required Parameters

1. `<materialized_view_name>`

> Specifies the identifier (i.e., name) of the materialized view; it must be unique within the database where the table is created.
>
> The identifier must start with a letter character (if Unicode name support is enabled, it can be any character from any language) and cannot contain spaces or special characters unless the entire identifier

string is enclosed in backticks (e.g., `My Object`).

The identifier cannot be a reserved keyword.

For more details, refer to the requirements for identifiers and reserved keywords.

2. `<query>`

A required parameter when creating a materialized view. Specifies the SELECT statement that populates the data.

Optional Parameters

1. `<key_cols>`

The key columns of the table. In Doris, key columns must be the first K columns of the table. For more information on key restrictions and how to choose key columns, refer to the respective sections in the “Data Model” chapter.

2. `<build_mode>`

Refresh timing: whether the materialized view should be refreshed immediately after creation.

IMMEDIATE: Refresh immediately. Default is IMMEDIATE.

DEFERRED: Delayed refresh.

3. `<refresh_method>`

Refresh method:

COMPLETE: Refresh all partitions.

AUTO: Try to refresh incrementally, only refreshing partitions that have changed since the last materialized view refresh. If incremental refresh is not possible, all partitions will be refreshed.

Note

If a partitioned materialized view uses COMPLETE refresh mode, it will perform a full refresh of all partition data, effectively degenerating into a non-partitioned materialized view.

4. `<refresh_trigger>`

> Trigger method:
>
> MANUAL: Manual refresh.
>
> ON SCHEDULE: Scheduled refresh.
>
> ON COMMIT: Triggered refresh, where changes to the base table data trigger a refresh of the materialized view.

5. `<refresh_unit>`

> The time unit for periodic refreshes. Currently supported units are MINUTE, HOUR, DAY, and WEEK.

6. `<partition_col>`

> If PARTITION BY is not specified, there will be only one partition by default.
>
> If a partition field is specified, it will automatically infer which base table the field comes from and synchronize the base table (currently supports internal tables and Hive tables). For internal tables, only one partition field is allowed.
>
> Materialized views can also reduce the number of partitions through partition roll-up. Currently, the partition roll-up function supports `date_trunc`.

7. `<partition_unit>`

> The aggregation granularity for partition roll-up. Currently supported units are HOUR, DAY, WEEK, QUARTER, MONTH, and YEAR.

8. `<start_time>`

> The scheduled start time must be in the future, i.e., later than the current time.

9. `<table_property>`

Properties used by internal tables, most of which can be used by materialized views, along with some properties specific to materialized views, as listed below:

Property Name	Description
grace_period	The maximum allowed delay in seconds for materialized view data during query rewriting. If partition A of the materialized view and the base table data are inconsistent, and the last refresh time of partition A was 10:15:00, with the current system time being 10:15:08, the partition will not be transparently rewritten. However, if `grace_period` is set to 10, the partition will be used for transparent rewriting.
excluded_trigger_tables	Comma-separated table names to be ignored during data refresh. For example, `table1,table2`.
refresh_partition_num	The number of partitions refreshed by a single INSERT statement, defaulting to 1. When refreshing a materialized view, it first calculates the list of partitions to be refreshed and then splits them into multiple INSERT statements for sequential execution. If an INSERT statement fails, the entire task will stop. The materialized view ensures the atomicity of a single INSERT statement, and a failed INSERT will not affect partitions that have already been refreshed successfully.
workload_group	The name of the `workload_group` used when the materialized view executes refresh tasks. This is used to limit the resources used by the materialized view during data refresh to avoid impacting other business operations. For more information on creating and using `workload_group`, refer to the WORKLOAD-GROUP documentation.
partition_sync_limit	When the base table's partition field is of type time, this property can be used to configure the range of partitions to synchronize with the base table, in conjunction with `partition_sync_time_unit`. For example, setting it to 2 with `partition_sync_time_unit` set to `MONTH` means that only the partitions and data from the last 2 months of the base table will be synchronized. The minimum value is `1`. As time progresses, the materialized view will automatically add and remove partitions during each refresh. For example, if the materialized view currently has data for months 2 and 3, next month it will automatically remove the data for month 2 and add data for month 4.
partition_sync_time_unit	The time unit for partition refresh, supporting DAY/MONTH/YEAR (default is DAY).
partition_date_format	When the base table's partition field is of type string, if you want to use the `partition_sync_limit` capability, you can set the date format to parse the partition time according to the `partition_date_format` setting.
enable_nondeterministic_function	Whether the materialized view definition SQL is allowed to contain nondeterministic functions, such as current_date(), now(), random(), etc. If set to true, it allows the inclusion; otherwise, it does not. The default is not to allow.
use_for_rewrite	Indicates whether this materialized view participates in transparent rewriting. If set to false, it does not participate in transparent rewriting. The default is true. In data modeling scenarios, if the materialized view is only used for direct queries, this property can be set so that the materialized view does not participate in transparent rewriting, thereby improving query response speed.

Note

The DISTRIBUTED BY clause is mandatory before Apache Doris 2.1.10 - omitting it will cause an error. Since Apache Doris 2.1.10, if not specified, the default distribution becomes RANDOM.

Note

> Before Apache Doris 2.1.10/3.0.6, the excluded_trigger_tables property only supported specifying base table names. Since these versions, it now supports fully qualified table names including Catalog and Database (e.g., internal.db1.table1).

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
CREATE_PRIV	Database	
SELECT_PRIV	Table, View	Requires SELECT_PRIV permission on the tables or views queried in `<query>`

Notes

- Materialized view DML and DDL limitations:

  Materialized views do not support schema change operations such as modifying column types, adding, or deleting columns, as the columns are derived from the materialized view definition SQL.

  Materialized views do not support manual insert into or insert overwrite operations.

- Conditions for creating partitioned materialized views:

  > The SQL definition of the materialized view and the partition field must meet the following conditions to perform partitioned incremental updates:
  >
  > 1. At least one of the base tables used by the materialized view must be a partitioned table.
  > 2. The partitioned base table used by the materialized view must use a List or Range partitioning strategy.
  > 3. The materialized view definition SQL can only have one partition field in the Partition By clause.
  > 4. The partition field in the Partition By clause of the materialized view SQL must appear after the Select clause.
  > 5. If the materialized view definition SQL uses Group By, the partition field must appear after the Group By clause.
  > 6. If the materialized view definition SQL uses a Window function, the partition field must appear after the Partition By clause.
  > 7. Data changes should occur on the partitioned table; if they occur on a non-partitioned table, the materialized view requires a full build.
  > 8. If the materialized view uses a field from the NULL-generating side of a Join as a partition field, it cannot perform partitioned incremental updates. For example, for a LEFT OUTER JOIN, the partition field must be on the left side, not

Examples

1. Non-Partitioned Materialized View

```
CREATE MATERIALIZED VIEW complete_mv (
orderdate COMMENT 'Order date',
orderkey COMMENT 'Order key',
partkey COMMENT 'Part key'
)
BUILD IMMEDIATE
REFRESH AUTO
ON SCHEDULE EVERY 1 DAY STARTS '2024-12-01 20:30:00'
DISTRIBUTED BY HASH (orderkey) BUCKETS 2
PROPERTIES
("replication_num" = "1")
AS
SELECT
o_orderdate,
l_orderkey,
l_partkey
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey
LEFT JOIN partsupp ON ps_partkey = l_partkey
and l_suppkey = ps_suppkey;
```

2. Partitioned Materialized View

As shown below, if a partition column is specified, the system automatically identifies which base table the column comes from and synchronizes the base table partitions. The base table is partitioned by day (partition column: o_orderdate, partition type: RANGE). The materialized view is partitioned by month, using the DATE_TRUNC function to roll up the base table partitions by month.

```
CREATE TABLE IF NOT EXISTS orders  (
o_orderkey       integer not null,
o_custkey        integer not null,
o_orderstatus    char(1) not null,
o_totalprice     decimalv3(15,2) not null,
o_orderdate      date not null,
o_orderpriority  char(15) not null,
o_clerk          char(15) not null,
o_shippriority   integer not null,
o_comment        varchar(79) not null
)
DUPLICATE KEY(o_orderkey, o_custkey)
PARTITION BY RANGE(o_orderdate)(
FROM ('2023-10-16') TO ('2023-11-30') INTERVAL 1 DAY
)
DISTRIBUTED BY HASH(o_orderkey) BUCKETS 3
PROPERTIES (
"replication_num" = "3"
);
```

```
CREATE MATERIALIZED VIEW partition_mv
BUILD IMMEDIATE
REFRESH AUTO
ON SCHEDULE EVERY 1 DAY STARTS '2024-12-01 20:30:00'
PARTITION BY (DATE_TRUNC(o_orderdate, 'MONTH'))
DISTRIBUTED BY HASH (l_orderkey) BUCKETS 2
PROPERTIES
("replication_num" = "3")
AS
SELECT
```

```
o_orderdate,
l_orderkey,
l_partkey
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey
LEFT JOIN partsupp ON ps_partkey = l_partkey
and l_suppkey = ps_suppkey;
```

The following example cannot create a partitioned materialized view because the partition column uses a non-date_trunc function. The error message is: because column to check use invalid implicit expression, invalid expression is min(o_orderdate#4)

```
CREATE MATERIALIZED VIEW partition_mv_2
BUILD IMMEDIATE
REFRESH AUTO
ON SCHEDULE EVERY 1 DAY STARTS '2024-12-01 20:30:00'
PARTITION BY (DATE_TRUNC(min_orderdate, 'MONTH'))
DISTRIBUTED BY HASH (l_orderkey) BUCKETS 2
PROPERTIES
("replication_num" = "3")
AS
SELECT
min(o_orderdate) AS min_orderdate,
l_orderkey,
l_partkey
FROM
orders
LEFT JOIN lineitem ON l_orderkey = o_orderkey
LEFT JOIN partsupp ON ps_partkey = l_partkey
and l_suppkey = ps_suppkey
GROUP BY
o_orderdate,
l_orderkey,
l_partkey;
```

3. Modifying Materialized View Properties Use the ALTER MATERIALIZED VIEW statement.

For example:

```
ALTER MATERIALIZED VIEW partition_mv
SET (
"grace_period" = "10",
"excluded_trigger_tables" = "lineitem,partsupp"
);
```

4. Changing Materialized View Refresh Mode Use the ALTER MATERIALIZED VIEW statement. For example:

```
ALTER MATERIALIZED VIEW partition_mv REFRESH COMPLETE;
After running SHOW CREATE MATERIALIZED VIEW partition_mv;, you can see that the refresh mode has been changed to COMPLETE.
```

PAUSE MATERIALIZED VIEW

Description

This statement is used to pause the scheduled scheduling of materialized views

Syntax

```
PAUSE MATERIALIZED VIEW JOB ON <mv_name>
```

Required Parameters

1. `<mv_name>` > Specifies the materialized view name. > > The materialized view name must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire materialized view name string is enclosed in backticks (e.g., `My Object`). > > The materialized view name cannot use reserved keywords. > > For more details, see Reserved Keywords.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Materialized View	PAUSE is an ALTER operation on a materialized view

Usage Notes

- After you use this statement, you can use the RESUME materialized view statement to restore.

Example

- Pause scheduled scheduling of materialized view mv1

```
PAUSE MATERIALIZED VIEW JOB ON mv1;
```

DROP ASYNC MATERIALIZED VIEW

Description

This statement is used to delete asynchronous materialized views.

syntax:

```
DROP MATERIALIZED VIEW (IF EXISTS)? mvName=multipartIdentifier
```

1. IF EXISTS: If the materialized view does not exist, do not throw an error. If this keyword is not declared and the materialized view does not exist, an error will be reported.
2. mv_name: The name of the materialized view to be deleted. Required field.

Examples

1. Delete table materialized view mv1

```
DROP MATERIALIZED VIEW mv1;
```

2.If present, delete the materialized view of the specified database

```
DROP MATERIALIZED VIEW IF EXISTS db1.mv1;
```

Keywords

```
DROP, ASYNC, MATERIALIZED, VIEW
```

Best Practice

REFRESH MATERIALIZED VIEW

Description

This statement is used to manually refresh the specified asynchronous materialized view

Syntax

```
REFRESH MATERIALIZED VIEW <mv_name> <refresh_type>
```

Where:

```
refresh_type
  : { <partitionSpec> | COMPLETE | AUTO }
```

```
partitionSpec
  : PARTITIONS (<partition_name> [, <partition_name> [, ... ] ])
```

Required Parameters

1. `<mv_name>` > Specifies the materialized view name. > > The materialized view name must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire materialized view name string is enclosed in backticks (e.g., `My Object`). > > The materialized view name cannot use reserved keywords. > > For more details, see Reserved Keywords.

2. `<refresh_type>` > Specifies the refresh type of this materialized view. > > The refresh type may be one of the partitionSpec, COMPLETE or AUTO.

Optional Parameters

1. `<partition_name>` > Specifies the partition name when refresh the partition >

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Materialized View	REFRESH is an ALTER operation on a materialized view

Usage Notes

- AUTO: The calculation will determine which partitions of the materialized view are not synchronized with the base table. (Currently, if the base table is an external table, it is considered to be always

synchronized with the materialized view. Therefore, if the base table is an external table, it is necessary to specify `COMPLETE` or designate the partitions to be refreshed), and then proceed to refresh the corresponding partitions accordingly.

- COMPLETE: It will forcibly refresh all partitions of the materialized view without checking whether the partitions are synchronized with the base table.
- partitionSpec: It will forcibly refresh the specified partitions without checking whether the partitions are synchronized with the base table.

Examples

- Refresh materialized view mv1 (automatically calculate the partition to be refreshed)

```
REFRESH MATERIALIZED VIEW mv1 AUTO;
```

- Refresh partition named p_19950801_19950901 and p_19950901_19951001

```
REFRESH MATERIALIZED VIEW mv1 partitions(p_19950801_19950901,p_19950901_19951001);
```

- Force refresh of all materialized view data

```
REFRESH MATERIALIZED VIEW mv1 complete;
```

RESUME MATERIALIZED VIEW

Description

This statement is used to temporarily restore the scheduled scheduling of materialized views

Syntax

```
RESUME MATERIALIZED VIEW JOB ON <mv_name>
```

Required Parameters

1. `<mv_name>` > Specifies the materialized view name. > > The materialized view name must start with a letter character (or any language character if unicode name support is enabled) and cannot contain spaces or special characters unless the entire materialized view name string is enclosed in backticks (e.g., `My Object`). > > The materialized view name cannot use reserved keywords. > > For more details, see Reserved Keywords.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Materialized View	RESUME is an ALTER operation on a materialized view

Usage Notes

- This statement is generally used after the pause materialized view statement

Example

- Timed scheduling for restoring materialized view mv1

```
RESUME MATERIALIZED VIEW JOB ON mv1;
```

SHOW CREATE ASYNC MATERIALIZED VIEW

Description

View the materialized view creation statement.

Syntax

```
SHOW CREATE MATERIALIZED VIEW <materialized_view_name>
```

Required Parameters

1. `<materialized_view_name>`

> The name of the materialized view.

Return Values

Column Name	Description
Materialized View	Name of the materialized view
Create Materialized View	Statement used to create the materialized view

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
SELECT_PRIV/LOAD_PRIV/ALTER_PRIV/CREATE_PRIV/DROP_PRIV	Table	

Example

1. View the creation statement of an asynchronous materialized view

```
SHOW CREATE MATERIALIZED VIEW partition_mv;
```

REBALANCE DISK

Description

The `REBALANCE DISK` statement is used to optimize the data distribution on Backend (BE) nodes. This statement has the following functionalities:

- It can perform data balancing for specified BE nodes.
- It can balance data across all BE nodes in the entire cluster.
- It prioritizes balancing the data of specified nodes, regardless of the overall balance state of the cluster.

Syntax

```
ADMIN REBALANCE DISK [ ON ( "<host>:<port>" [, ... ] ) ];
```

Optional Parameters

1. `"<host>:<port>"`

> Specifies the list of BE nodes that need to be balanced.
>
> Each node consists of a hostname (or IP address) and a heartbeat port.
>
> If this parameter is not specified, it will balance all BE nodes.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN	System	The user must have ADMIN privileges to execute this command.

Usage Notes

- The default timeout for this command is 24 hours. After this period, the system will no longer prioritize balancing the disk data of specified BEs. To continue balancing, the command needs to be executed again.
- Once the disk data balancing for a specified BE node is completed, the high-priority setting for that node will automatically become invalid.
- This command can be executed even when the cluster is in an unbalanced state.

Examples

- Balance data across all BE nodes in the cluster:

```
ADMIN REBALANCE DISK;
```

- Balance data for two specified BE nodes:

```
ADMIN REBALANCE DISK ON ("192.168.1.1:1234", "192.168.1.2:1234");
```

CANCEL REBALANCE DISK

Description

The `CANCEL REBALANCE DISK` statement is used to cancel the high-priority disk data balancing for Backend (BE) nodes. This statement has the following functionalities:

- It can cancel the high-priority disk balancing for specified BE nodes.
- It can cancel the high-priority disk balancing for all BE nodes in the entire cluster.
- After cancellation, the system will still balance the disk data of BE nodes using the default scheduling method.

Syntax

```
ADMIN CANCEL REBALANCE DISK [ ON ( "<host>:<port>" [, ... ] ) ];
```

Optional Parameters

1. `"<host>:<port>"`

> Specifies the list of BE nodes for which the high-priority disk balancing needs to be canceled.
>
> Each node consists of a hostname (or IP address) and a heartbeat port.
>
> If this parameter is not specified, it will cancel the high-priority disk balancing for all BE nodes.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN	System	The user must have ADMIN privileges to execute this command.

Usage Notes

- This statement only indicates that the system will no longer prioritize balancing the disk data of specified BEs; however, the system will still balance BE's disk data using the default scheduling method.
- After executing this command, any previously set high-priority balancing strategy will become immediately invalid.

Examples

- Cancel high-priority disk balancing for all BEs in the cluster:

```
ADMIN CANCEL REBALANCE DISK;
```

- Cancel high-priority disk balancing for specified BEs:

```
ADMIN CANCEL REBALANCE DISK ON ("192.168.1.1:1234", "192.168.1.2:1234");
```

SHOW DATA

Description

The `SHOW DATA` statement is used to display information about data volume, replica count, and row statistics. This statement has the following functionalities:

- It can display the data volume and replica count for all tables in the current database.
- It can show the data volume, replica count, and row statistics for a specified table's materialized views.
- It can display the quota usage of the database.
- It supports sorting by data volume, replica count, etc.

Syntax

```
SHOW DATA [ FROM [<db_name>.]<table_name> ] [ ORDER BY <order_by_clause> ];
```

Where:

```
order_by_clause:
    <column_name> [ ASC | DESC ] [ , <column_name> [ ASC | DESC ] ... ]
```

Optional Parameters

1. `FROM [<db_name>.]<table_name>`

> Specifies the name of the table to view. The database name can be included.
>
> If this parameter is not specified, it will display data information for all tables in the current database.

2. `ORDER BY <order_by_clause>`

> Specifies the sorting method for the result set.
>
> Any column can be sorted in ascending (ASC) or descending (DESC) order.
>
> Supports multi-column combination sorting.

Return Values

Depending on different query scenarios, the following result sets are returned:

- When the `FROM` clause is not specified (displaying database-level information):

Column Name	Description
DbId	Database ID
DbName	Database name
Size	Total data volume of the database
RemoteSize	Remote storage data volume
RecycleSize	Recycle bin data volume
RecycleRemoteSize	Recycle bin remote storage volume

- When the `FROM` clause is specified (displaying table-level information):

Column Name	Description
TableName	Table name
IndexName	Index (materialized view) name
Size	Data size
ReplicaCount	Replica count
RowCount	Row statistics (shown only when viewing a specific table)

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
SELECT	Table	SELECT permission is required for viewing the table.

Usage Notes

- The data volume statistics include the total data volume of all replicas.
- The replica count includes all partitions and replicas of all materialized views for the table.
- When counting rows, it considers the maximum row count among multiple replicas.
- The `Total` row in the result set indicates aggregated data.
- The `Quota` row in the result set indicates the current quota set for the database.
- The `Left` row in the result set indicates remaining quota.
- If you need to view the size of each partition, use the `SHOW PARTITIONS` command.

Examples

- Display data volume information for all databases:

```
SHOW DATA;
```

```
+-------+----------------------------------+--------+------------+-------------+-------------------+
| DbId  | DbName                           | Size   | RemoteSize | RecycleSize | RecycleRemoteSize |
+-------+----------------------------------+--------+------------+-------------+-------------------+
```

```
| 21009 | db1                                | 0      | 0          | 0           | 0                 |
| 22011 | regression_test_inverted_index_p0 | 72764  | 0          | 0           | 0                 |
| Total | NULL                               | 118946 | 0          | 0           | 0                 |
+-------+------------------------------------+--------+------------+-------------+-------------------+
```

- Display data volume information for all tables in the current database:

```
USE db1;
SHOW DATA;
```

```
+-----------+-------------+--------------+
| TableName | Size        | ReplicaCount |
+-----------+-------------+--------------+
| tbl1      | 900.000 B   | 6            |
| tbl2      | 500.000 B   | 3            |
| Total     | 1.400 KB    | 9            |
| Quota     | 1024.000 GB | 1073741824   |
| Left      | 1021.921 GB | 1073741815   |
+-----------+-------------+--------------+
```

- Display detailed data volume information for a specified table:

```
SHOW DATA FROM example_db.test;
```

```
+-----------+-----------+-----------+--------------+----------+
| TableName | IndexName | Size      | ReplicaCount | RowCount |
+-----------+-----------+-----------+--------------+----------+
| test      | r1        | 10.000MB  | 30           | 10000    |
|           | r2        | 20.000MB  | 30           | 20000    |
|           | test2     | 50.000MB  | 30           | 50000    |
|           | Total     | 80.000MB  | 90           |          |
+-----------+-----------+-----------+--------------+----------+
```

- Sort by replica count in descending order and by data volume in ascending order:

```
SHOW DATA ORDER BY ReplicaCount DESC, Size ASC;
```

```
+-----------+-------------+--------------+
| TableName | Size        | ReplicaCount |
+-----------+-------------+--------------+
| table_c   | 3.102 KB    | 40           |
| table_d   | .000        | 20           |
| table_b   | 324.000 B   | 20           |
| table_a   | 1.266 KB    | 10           |
| Total     | 4.684 KB    | 90           |
| Quota     | 1024.000 GB | 1073741824   |
| Left      | 1024.000 GB | 1073741734   |
+-----------+-------------+--------------+
```

SHOW DATA SKEW

Description

The `SHOW DATA SKEW` statement is used to view the data skew of a table or partition. This statement has the following functionalities:

- It can display the data distribution of the entire table.
- It can display the data distribution of specified partitions.
- It shows the row count, data volume, and percentage for each bucket.
- It supports both partitioned and non-partitioned tables.

Syntax

```
SHOW DATA SKEW FROM [<db_name>.]<table_name> [ PARTITION (<partition_name> [, ...]) ];
```

Required Parameters

1. `FROM [<db_name>.]<table_name>`

> Specifies the name of the table to be viewed. The database name can be included.
>
> The table name must be unique within its database.

Optional Parameters

1. `PARTITION (<partition_name> [, ...])`

> Specifies a list of partition names to be viewed.
>
> If this parameter is not specified, it will display the data distribution for all partitions in the table.
>
> For non-partitioned tables, the partition name is the same as the table name.

Return Values

Column Name	Description
PartitionName	Partition name
BucketIdx	Bucket index number
AvgRowCount	Average row count

Column Name	Description
AvgDataSize	Average data size (in bytes)
Graph	Visualization chart of data distribution
Percent	Percentage of this bucket's data volume relative to total data volume

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
SELECT	Table	SELECT permission is required for viewing the table.

Usage Notes

- The data distribution is displayed along two dimensions: partition and bucket.
- The Graph column uses the character > to visually represent the data distribution ratio.
- Percentages are accurate to two decimal places.
- For non-partitioned tables, the partition name in the query result is the same as the table name.

Examples

- Create a partitioned table and view its data distribution:

```
CREATE TABLE test_show_data_skew
(
    id int,
    name string,
    pdate date
)
PARTITION BY RANGE(pdate)
(
    FROM ("2023-04-16") TO ("2023-04-20") INTERVAL 1 DAY
)
DISTRIBUTED BY HASH(id) BUCKETS 5
PROPERTIES (
    "replication_num" = "1"
);
```

- View data distribution for the entire table:

```
SHOW DATA SKEW FROM test_show_data_skew;
```

```
+---------------+-----------+-------------+-------------+-----------------------------------------------------------------
-----------------------------------+---------+
| PartitionName | BucketIdx | AvgRowCount | AvgDataSize | Graph
| Percent |
+---------------+-----------+-------------+-------------+-----------------------------------------------------------------
-----------------------------------+---------+
| p_20230416    | 0         | 1           | 648         | >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
| 49.77 % |
```

```
| p_20230416    | 1         | 2           | 654         | >>>>
```

```
| 00.00 % |
| p_20230418    | 1         | 1           | 648         |
>>>
```

COMPACT TABLE

Description

In the storage-computing coupled mode, this is used to trigger a compaction for all replicas under the specified table partition.

This command is not supported in the storage-computing separation mode.

Syntax

```
ADMIN COMPACT TABLE <table_name>
PARTITION <partition_name>
WHERE TYPE={ BASE | CUMULATIVE }
```

Required Parameters

> The name of the table for which compaction is to be triggered.

> The name of the partition for which compaction is to be triggered. (Note: This line should be corrected as it repeats the table name description, it should specify the partition name.)

TYPE={ BASE | CUMULATIVE }

> Where BASE refers to triggering a base compaction, and CUMULATIVE refers to triggering a cumulative compaction. For details, refer to the COMPACTION section.

Access Control Requirements

The prerequisite for successfully executing this SQL command is to have ADMIN_PRIV permissions. Refer to the permission documentation.

Privilege	Object	Notes
ADMIN_PRIV	Entire cluster management privileges	All privileges except NODE_PRIV

Example

1. Trigger cumulative compaction for partition par01 of table tbl.

```
ADMIN COMPACT TABLE tbl PARTITION par01 WHERE TYPE='CUMULATIVE';
```

Usage Note

1. This command is not supported in the storage-computing separation mode. Executing it in this mode will result in an error, for example:

```
ADMIN COMPACT TABLE tbl PARTITION par01 WHERE TYPE='CUMULATIVE';
```

The error message is as follows:

```
ERROR 1105 (HY000): errCode = 2, detailMessage = Unsupported operation
```

REPAIR TABLE

Description

The `REPAIR TABLE` statement is used to prioritize the repair of replicas for a specified table or partition. This statement has the following functionalities:

- It can repair all replicas of an entire table.
- It can repair replicas of specified partitions.
- It performs replica repairs with high priority.
- It supports setting a repair timeout.

Syntax

```
ADMIN REPAIR TABLE <table_name> [ PARTITION (<partition_name> [, ...]) ];
```

Required Parameters

1. `<table_name>`

> Specifies the name of the table that needs to be repaired.
>
> The table name must be unique within its database.

Optional Parameters

1. `PARTITION (<partition_name> [, ...])`

> Specifies a list of partition names that need to be repaired.
>
> If this parameter is not specified, it will repair all partitions of the entire table.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN	System	The user must have ADMIN privileges to execute this command.

Usage Notes

- This statement indicates that the system will attempt to repair the specified replicas with high priority, but it does not guarantee successful repairs.
- The default timeout is set to 14,400 seconds (4 hours).
- After the timeout, the system will no longer prioritize the repair of specified replicas.
- If a repair times out, the command needs to be executed again to continue the repair process.
- The progress of repairs can be monitored using the `SHOW REPLICA STATUS` command.
- This command does not affect the normal replica repair mechanism of the system; it merely elevates the priority of repairs for the specified table or partition.

Examples

- Repair all replicas of an entire table:

```
ADMIN REPAIR TABLE tbl1;
```

- Repair replicas of specified partitions:

```
ADMIN REPAIR TABLE tbl1 PARTITION (p1, p2);
```

- Check the repair progress:

```
SHOW REPLICA STATUS FROM tbl1;
```

CANCEL REPAIR TABLE

Description

The `CANCEL REPAIR TABLE` statement is used to cancel high-priority repairs for a specified table or partition. This statement has the following functionalities:

- It can cancel high-priority repairs for an entire table.
- It can cancel high-priority repairs for specified partitions.
- It does not affect the system's default replica repair mechanism.

Syntax

```
ADMIN CANCEL REPAIR TABLE <table_name> [ PARTITION (<partition_name> [, ...]) ];
```

Required Parameters

1. `<table_name>`

> Specifies the name of the table for which the repair is to be canceled.
>
> The table name must be unique within its database.

Optional Parameters

1. `PARTITION (<partition_name> [, ...])`

> Specifies a list of partition names for which the repair is to be canceled.
>
> If this parameter is not specified, it will cancel high-priority repairs for the entire table.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN	System	The user must have ADMIN privileges to execute this command.

Usage Notes

- This statement only cancels high-priority repairs and does not stop the system's default replica repair mechanism.
- After cancellation, the system will still repair replicas using the default scheduling method.
- If there is a need to re-establish high-priority repairs, the `ADMIN REPAIR TABLE` command can be used.
- The effects of this command take place immediately after execution.

Examples

- Cancel high-priority repairs for an entire table:

```
ADMIN CANCEL REPAIR TABLE tbl;
```

- Cancel high-priority repairs for specified partitions:

```
ADMIN CANCEL REPAIR TABLE tbl PARTITION(p1, p2);
```

SET TABLE STATUS

Description

The `SET TABLE STATUS` statement is used to manually set the status of an OLAP table. This statement has the following functionalities:

- It only supports setting the status of OLAP tables.
- It can modify the table status to a specified target state.
- It is used to resolve task blocking caused by the table status.

Supported States:

State	Description
NORMAL	Indicates that the table is in a normal state.
ROLLUP	Indicates that the table is undergoing a ROLLUP operation.
SCHEMA_CHANGE	Indicates that the table is undergoing a schema change.
BACKUP	Indicates that the table is undergoing a backup operation.
RESTORE	Indicates that the table is undergoing a restore operation.
WAITING_STABLE	Indicates that the table is waiting for a stable state.

Syntax

```
ADMIN SET TABLE <table_name> STATUS PROPERTIES ("<key>" = "<value>" [, ...]);
```

Where:

```
<key>
  : "state"

<value>
  : "NORMAL"
  | "ROLLUP"
  | "SCHEMA_CHANGE"
  | "BACKUP"
  | "RESTORE"
  | "WAITING_STABLE"
```

Required Parameters

1. `<table_name>`

> Specifies the name of the table for which the status needs to be set.

> The table name must be unique within its database.

2. `PROPERTIES ("state" = "<value>")`

> Specifies the target status of the table.
>
> The “state” property must be set, and its value must be one of the supported states.

Access Control Requirements

Users executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN	System	The user must have ADMIN privileges to execute this command.

Usage Notes

- This command is intended for emergency fault recovery; please use it with caution.
- It only supports OLAP tables and does not support other types of tables.
- If the table is already in the target state, this command will be ignored.
- Improper state settings may lead to system anomalies; it is recommended to use this command under technical support guidance.
- After modifying the status, it is advisable to monitor the system’s operational status promptly.

Examples

- Set the table status to NORMAL:

```
ADMIN SET TABLE tbl1 STATUS PROPERTIES("state" = "NORMAL");
```

- Set the table status to SCHEMA_CHANGE:

```
ADMIN SET TABLE tbl2 STATUS PROPERTIES("state" = "SCHEMA_CHANGE");
```

SET TABLE PARTITION VERSION

Description

In compute-storage coupled mode. This statement is used to manually change the visible version of the specified partition. In some special cases, the version of the partition in the metadata may be inconsistent with the version of the actual replica.

This command can manually change the version of the partition in the metadata. This command is generally only used for emergency failure recovery. Please operate with caution.

Syntax

```
ADMIN SET TABLE <table_name> PARTITION VERSION PROPERTIES ("<partition_id>" = "visible_version>");
```

Required Parameters

> The name of the table to be set.

> Specify a Partition Id.

> Specify Version.

Examples

1. Set the version of the partition with partition_id 10075 to 100 in the FE metadata.

```
ADMIN SET TABLE __internal_schema.audit_log PARTITION VERSION PROPERTIES("partition_id" = "10075", "visible_version" = "100");
```

Usage Note

1. Before setting the partition version, you need to confirm the version of the actual replica on the BE machine. This command is generally only used for emergency failure recovery. Please operate with caution.
2. This command is not supported in the storage-computing separation mode. It will not take effect if set.

DIAGNOSE TABLET

Description

In the compute-storage coupled mode, this statement is used to diagnose a specified tablet. The result will display information about the tablet and some potential issues.

This command is not supported in the storage-computing separation mode.

Syntax

```
SHOW TABLET DIAGNOSIS <tablet_id>;
```

Required Parameters

> The ID of the tablet to be diagnosed

Return Value (Return Value)

Returns information about the tablet

- TabletExist

  > Whether the tablet exists

- TabletId

  > Tablet ID

- Database

  > The DB and its ID that the tablet belongs to

- Table

  > The Table and its ID that the tablet belongs to

- Partition

  > The Partition and its ID that the tablet belongs to

- MaterializedIndex

  > The materialized view and its ID that the tablet belongs to

- Replicas

  > The replicas of the tablet and their corresponding BEs

- ReplicasNum

  > Whether the number of replicas is correct

- ReplicaBackendStatus

  > Whether the BE nodes where the replicas are located are normal

- ReplicaVersionStatus

  > Whether the version numbers of the replicas are normal

- ReplicaStatus

  > Whether the status of the replicas is normal

- ReplicaCompactionStatus

  > Whether the compaction status of the replicas is normal

Examples

1. Diagnose the information of the tablet with the specified tablet id 10078

```
show tablet diagnosis 10078;
+----------------------------------+-----------------------------------------+------------+
| Item                             | Info                                    | Suggestion |
+----------------------------------+-----------------------------------------+------------+
| TabletExist                      | Yes                                     |            |
| TabletId                         | 10078                                   |            |
| Database                         | __internal_schema: 10005                |            |
| Table                            | audit_log: 10058                        |            |
| Partition                        | p20241109: 10075                        |            |
| MaterializedIndex                | audit_log: 10059                        |            |
| Replicas(ReplicaId -> BackendId) | {"10099":10003,"10116":10002,"10079":10004} |            |
| ReplicasNum                      | OK                                      |            |
| ReplicaBackendStatus             | OK                                      |            |
| ReplicaVersionStatus             | OK                                      |            |
| ReplicaStatus                    | OK                                      |            |
| ReplicaCompactionStatus          | OK                                      |            |
+----------------------------------+-----------------------------------------+------------+
```

Access Control Requirements (Access Control Requirements)

The prerequisite for successfully executing this SQL command is to have ADMIN_PRIV privileges. Please refer to the privilege documentation.

Privilege (Privilege)	Object (Object)	Notes (Notes)
ADMIN_PRIV	Entire cluster management privileges	All privileges except NODE_PRIV

Usage Note (Usage Note)

1. This command is not supported in the storage-computing separation mode. Executing it in this mode will result in an error, for example:

```
show tablet diagnosis 15177;
```

The error message is as follows:

```
ERROR 1105 (HY000): errCode = 2, detailMessage = Unsupported operation
```

ADMIN COPY TABLET

Description

This statement is used to make a snapshot for the specified tablet, mainly used to load the tablet locally to reproduce the problem.

Syntax

```
ADMIN COPY TABLET <tablet_id> PROPERTIES ("<key>"="<value>" [,...]).
```

Required Parameters

1. `<tablet_id>`

The ID of the tablet to be copied.

Optional Parameters

```
[ PROPERTIES ("<key>"="<value>" [, ... ]) ]
```

The PROPERTIES clause allows specifying additional parameters:

1. `<backend_id>`

Specifies the id of the BE node where the replica is located. If not specified, a replica is randomly selected.

2. `<version>`

Specifies the version of the snapshot. The version must be less than or equal to the largest version of the replica. If not specified, the largest version is used.

3. `<expiration_minutes>`

Snapshot retention time. The default is 1 hour. It will automatically clean up after a timeout. Unit minutes.

Return Value

Column	Data Type	Note
TabletId	string	The ID of the tablet for which the snapshot was created.
BackendId	string	The ID of the BE node where the snapshot is stored.

Column	Data Type	Note
Ip	string	The IP address of the BE node storing the snapshot.
Path	string	The storage path where the snapshot is saved on the BE node.
ExpirationMinutes	string	The duration (in minutes) after which the snapshot will be automatically deleted.
CreateTableStmt	string	The table creation statement for the table corresponding to the tablet. This statement is not the original table-building statement, but a simplified table-building statement for later loading the tablet locally.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

- Take a snapshot of the replica on the specified BE node

```
ADMIN COPY TABLET 10020 PROPERTIES("backend_id" = "10003");
```

```
         TabletId: 10020
        BackendId: 10003
               Ip: 192.168.10.1
             Path: /path/to/be/storage/snapshot/20220830101353.2.3600
ExpirationMinutes: 60
  CreateTableStmt: CREATE TABLE `tbl1` (
  `k1` int(11) NULL,
  `k2` int(11) NULL
) ENGINE=OLAP
DUPLICATE KEY(`k1`, `k2`)
DISTRIBUTED BY HASH(k1) BUCKETS 1
PROPERTIES (
"replication_num" = "1",
"version_info" = "2"
);
```

- Take a snapshot of the specified version of the replica on the specified BE node

```
ADMIN COPY TABLET 10010 PROPERTIES("backend_id" = "10003", "version" = "10");
```

```
         TabletId: 10010
        BackendId: 10003
               Ip: 192.168.10.1
             Path: /path/to/be/storage/snapshot/20220830101353.2.3600
ExpirationMinutes: 60
  CreateTableStmt: CREATE TABLE `tbl1` (
  `k1` int(11) NULL,
  `k2` int(11) NULL
) ENGINE=OLAP
DUPLICATE KEY(`k1`, `k2`)
DISTRIBUTED BY HASH(k1) BUCKETS 1
PROPERTIES (
"replication_num" = "1",
"version_info" = "2"
);
```

ADMIN CHECK TABLET

Description

This statement is used to perform the specified check operation on a set of tablets.

Syntax

```
ADMIN CHECK TABLET ( <tablet_id> [,...] ) PROPERTIES("type" = "<type_value>")
```

Required Parameters

1. `<tablet_id>`

The ID of the tablet on which the specified check operation will be performed.

Optional Parameters

1. `<type_value>`

Currently, only consistency is supported.

- consistency:

  Checks the replica data consistency of the tablet. This command is asynchronous, meaning that after sending it, Doris will start executing the consistency check job for the corresponding tablet.

Return Value

The final result of the statement will be reflected in the InconsistentNum column of the result from `SHOW PROC "/cluster_health/tablet_health";`.

Column	DataType	Note
InconsistentNum	Int	Number of tablets with inconsistency

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

- Perform a replica data consistency check on a specified set of tablets

```
admin check tablet (10000, 10001) PROPERTIES("type" = "consistency");
```

SHOW TABLET

Description

This statement is used to show details of a specific tablet (only for administrators).

Syntax

```
SHOW TABLET <tablet_id>;
```

Required Parameters

1. `<tablet_id>`

> The ID of the specific tablet to display information for.

Return Value

When using `SHOW TABLET <tablet_id>`, the following columns are returned:

Column	DataType	Note
DbName	String	The name of the database that contains the tablet.
TableName	String	The name of the table that contains the tablet.
PartitionName	String	The name of the partition that contains the tablet.
IndexName	String	The name of the index that contains the tablet.
DbId	Int	The ID of the database.
TableId	Int	The ID of the table.
PartitionId	Int	The ID of the partition.
IndexId	Int	The ID of the index.
IsSync	Boolean	Whether the tablet is in sync with its replicas.
Order	Int	The order of the tablet.
QueryHits	Int	The number of query hits on this tablet.
DetailCmd	String	The command to get more detailed information about the tablet.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

Show details of a specific tablet:

```
SHOW TABLET 10145;
```

```
+--------+-----------+---------------+-----------+-------+---------+-------------+---------+--------+-------+-----------+-----------
-----------------------------------------------+
| DbName | TableName | PartitionName | IndexName | DbId  | TableId | PartitionId | IndexId | IsSync | Order | QueryHits | DetailCmd
|
+--------+-----------+---------------+-----------+-------+---------+-------------+---------+--------+-------+-----------+-----------
-----------------------------------------------+
| test   | sell_user | sell_user     | sell_user | 10103 | 10143   | 10142       | 10144   | true   | 0     | 0         | SHOW PROC
'/dbs/10103/10143/partitions/10142/10144/10145'; |
+--------+-----------+---------------+-----------+-------+---------+-------------+---------+--------+-------+-----------+-----------
-----------------------------------------------+
```

SHOW TABLETS BELONG

Description

This statement is used to show tablets and information of their belonging table.

Syntax

```
SHOW TABLETS BELONG <tablet_id> [, <tablet_id>]...;
```

Required Parameters

1. `<tablet_id>`

> One or more tablet IDs, separated by commas. Duplicate IDs will be deduplicated in the result.

Return Value

When using `SHOW TABLETS BELONG <tablet_id> [, <tablet_id>]...`, the following columns are returned:

Column	DataType	Note
DbName	String	The name of the database that contains the tablet.
TableName	String	The name of the table that contains the tablet.
TableSize	String	The size of the table (e.g., “8.649 KB”).
PartitionNum	Int	The number of partitions in the table.
BucketNum	Int	The number of buckets in the table.
ReplicaCount	Int	The number of replicas in the table.
TabletIds	Array	The list of tablet IDs that belong to the table.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

Show information about a specific tablet:

```
SHOW TABLETS BELONG 10145;
```

```
+--------+-----------+-----------+--------------+-----------+--------------+-----------+
| DbName | TableName | TableSize | PartitionNum | BucketNum | ReplicaCount | TabletIds |
+--------+-----------+-----------+--------------+-----------+--------------+-----------+
| test   | sell_user | 8.649 KB  | 1            | 4         | 4            | [10145]   |
+--------+-----------+-----------+--------------+-----------+--------------+-----------+
```

Show information about multiple tablets:

```
SHOW TABLETS BELONG 27028,78880,78382,27028;
```

```
+---------------------+-----------+-----------+--------------+-----------+--------------+----------------+
| DbName              | TableName | TableSize | PartitionNum | BucketNum | ReplicaCount | TabletIds      |
+---------------------+-----------+-----------+--------------+-----------+--------------+----------------+
| default_cluster:db1 | kec       | 613.000 B | 379          | 604       | 604          | [78880, 78382] |
| default_cluster:db1 | test      | 1.874 KB  | 1            | 1         | 1            | [27028]        |
+---------------------+-----------+-----------+--------------+-----------+--------------+----------------+
```

SHOW TABLET STORAGE FORMAT

Description

This statement is used to display storage format information on the Backend.

Syntax

```
SHOW TABLET STORAGE FORMAT [VERBOSE]
```

Optional Parameters

** 1. `VERBOSE` **

Displays more detailed information.

Return Value

Default output (without `VERBOSE`):

Column	DataType	Note
BackendId	Int	The ID of the BE (Backend) node where the tablet replica is located.
V1Count	Int	Number of V1 version tablets.
V2Count	Int	Number of V2 version tablets.

With `VERBOSE`:

Column	DataType	Note
BackendId	Int	The ID of the BE (Backend) node where the tablet replica is located.
TabletId	Int	The unique identifier of the tablet.
StorageFormat	String	The version of the tablet, either V1 or V2.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

- Execute the statement without the verbose parameter

```
show tablet storage format;
```

```
+-----------+---------+---------+
| BackendId | V1Count | V2Count |
+-----------+---------+---------+
| 10002     | 0       | 2867    |
+-----------+---------+---------+
```

- Execute the statement with the verbose parameter

```
show tablet storage format verbose;
```

```
+-----------+----------+---------------+
| BackendId | TabletId | StorageFormat |
+-----------+----------+---------------+
| 10002     | 39227    | V2            |
| 10002     | 39221    | V2            |
| 10002     | 39215    | V2            |
| 10002     | 39199    | V2            |
+-----------+----------+---------------+
```

SHOW TABLET DIAGNOSIS

Description

This statement is used to diagnose the specified tablet. The result will display information about the tablet along with any potential issues.

Syntax

```
SHOW TABLET DIAGNOSIS <tablet_id>
```

Required Parameters

1. `<tablet_id>`

The ID of the tablet to be diagnosed.

Return Value

Column	DataType	Note
TabletExist	String	Indicates whether the tablet exists.
TabletId	String	The ID of the tablet.
Database	String	The database the tablet belongs to, along with its ID.
Table	String	The table the tablet belongs to, along with its ID.
Partition	String	The partition the tablet belongs to, along with its ID.
MaterializedIndex	String	The materialized index the tablet belongs to, along with its ID.
Replicas(ReplicaId -> BackendId)	String	The replicas of the tablet and their respective BE nodes.
ReplicasNum	String	Indicates whether the number of replicas is correct.
ReplicaBackendStatus	String	Indicates whether the BE node where the replica is located is functioning properly.
ReplicaVersionStatus	String	Indicates whether the version number of the replica is correct.
ReplicaStatus	String	Indicates whether the replica status is normal.
ReplicaCompactionStatus	String	Indicates whether the compaction status of the replica is normal.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

```
SHOW TABLET DIAGNOSIS 10145;
```

```
+---------------------------------+-----------------+------------+
| Item                            | Info            | Suggestion |
+---------------------------------+-----------------+------------+
| TabletExist                     | Yes             |            |
| TabletId                        | 10145           |            |
| Database                        | test: 10103     |            |
| Table                           | sell_user: 10143 |            |
| Partition                       | sell_user: 10142 |            |
| MaterializedIndex               | sell_user: 10144 |            |
| Replicas(ReplicaId -> BackendId) | {"10146":10009}  |            |
| ReplicasNum                     | OK              |            |
| ReplicaBackendStatus            | OK              |            |
| ReplicaVersionStatus            | OK              |            |
| ReplicaStatus                   | OK              |            |
| ReplicaCompactionStatus         | OK              |            |
+---------------------------------+-----------------+------------+
```

ADMIN SET REPLICA STATUS

Description

This statement is used to set the status of a specified replica. Currently, it is only used for manually setting certain replica statuses to `BAD`, `DROP`, or `OK`, allowing the system to automatically repair these replicas.

Syntax

```
ADMIN SET REPLICA STATUS
PROPERTIES ("tablet_id"="<tablet_id>","backend_id"="<backend_id>","status"="<status>")
```

Required Parameters

1. `<tablet_id>`

The ID of the tablet whose replica status needs to be set.

2. `<backend_id>`

The ID of the BE node where the replica is located.

3. `<status>`

Currently, only the values “drop”, “bad”, and “ok” are supported. If the specified replica does not exist or is already in a bad state, the operation will be ignored.

Notes：

- Setting a replica to BAD status

  A replica marked as BAD cannot be read or written. However, setting a replica to BAD may not always take effect. If the replica’s actual data is correct, and the BE reports the replica status as OK, the FE will automatically restore the replica back to OK. This operation may immediately delete the replica, so proceed with caution.

- Setting a replica to DROP status

  A replica marked as DROP can still be read and written. The system will first add a healthy replica on another machine before deleting the DROP replica. Compared to BAD, setting a replica to DROP is a safer operation.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

- Set the replica of tablet 10003 on BE 10001 to bad

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003", "backend_id" = "10001", "status" = "bad");
```

- Set the replica of tablet 10003 on BE 10001 to drop

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003", "backend_id" = "10001", "status" = "drop");
```

- Set the replica of tablet 10003 on BE 10001 to ok

```
ADMIN SET REPLICA STATUS PROPERTIES("tablet_id" = "10003", "backend_id" = "10001", "status" = "ok");
```

ADMIN SET REPLICA VERSION

Description

This statement is used to manually set the version, last successful version, and last failed version of a specified replica. It is primarily used for recovering replicas from an abnormal state when an issue occurs in the system.

Syntax

```
ADMIN SET REPLICA VERSION PROPERTIES ("<key>"="<value>" [,...])
```

Required Parameters

** 1. "<key>"="<value>"**

key	value type	Notes
tablet_id	Int	The ID of the tablet whose replica version needs to be modified.
backend_id	Int	The ID of the BE node where the replica is located.

Optional Parameters

** 1. "<key>"="<value>"**

key	value type	Notes
version	Int	Sets the version of the replica.
last_success_version	Int	Sets the last successful version of the replica.
last_failed_version	Int	Sets the last failed version of the replica.

Notes

- If the specified replica does not exist, the operation will be ignored.
-
- Modifying these values may cause subsequent data read and write failures, leading to data inconsistency. Please proceed with caution!
- Record the original values before making modifications. After making changes, verify table read and write operations. If they fail, revert to the original values. However, reverting may also fail.
- Do not perform this operation on a tablet that is currently writing data!

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

- Clear the failed status flag for the replica of tablet 10003 on BE 10001

```
ADMIN SET REPLICA VERSION PROPERTIES("tablet_id" = "10003", "backend_id" = "10001", "last_failed_version" = "-1");
```

- Set the replica version of tablet 10003 on BE 10001 to 1004

```
ADMIN SET REPLICA VERSION PROPERTIES("tablet_id" = "10003", "backend_id" = "10001", "version" = "1004");
```

SHOW REPLICA STATUS

Description

This statement is used to display the replica status information for a table or partition.

Syntax

```
SHOW REPLICA STATUS FROM [ <database_name>.]<table_name> [<partition_list>]
[where_clause]
```

Where:

```
partition_list
  : PARTITION (<partition_name>[ , ... ])
```

Where:

```
where_clause
: WHERE <output_column_name> = <value>
```

Required Parameters

1. `<table_name>`

> The identifier (i.e., name) of the table, which must be unique within the database (Database).
>
> The identifier must start with a letter character (or any character from supported languages if Unicode name support is enabled), and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., My Object).
>
> Identifiers cannot use reserved keywords.
>
> For more details, refer to the identifier requirements and reserved keywords.

Optional Parameters

1. `<db_name>`

> The identifier (i.e., name) of the database, which must be unique within the cluster (Cluster).
>
> The identifier must start with a letter character (or any character from supported languages if Unicode name support is enabled), and cannot contain spaces or special characters unless the entire identifier

string is enclosed in backticks (e.g., My Object).

Identifiers cannot use reserved keywords.

For more details, refer to the identifier requirements and reserved keywords.

2. `<partition_list>`

A comma-separated list of partition identifiers (i.e., names), which must be unique within the table (Table).

The identifier must start with a letter character (or any character from supported languages if Unicode name support is enabled), and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., My Object).

Identifiers cannot use reserved keywords.

For more details, refer to the identifier requirements and reserved keywords.

3. `WHERE <output_column_name> = <value>`

Specifies the filtering condition for the output. The output_column_name must be part of the output field list.

When output_column_name is STATUS, the value can be one of the following:

- DEAD: The backend where the replica resides is unavailable.
- VERSION_ERROR: The replica's data version is incomplete.
- SCHEMA_ERROR: The schema hash of the replica is incorrect.
- MISSING: The replica does not exist.

Return Value

Column	DataType	Note
TabletId	Int	Unique identifier for the tablet.
ReplicaId	Int	Unique identifier for the replica.
BackendId	Int	ID of the Backend (BE) node where the replica is located.
Version	Int	The current version of the replica.
LastFailedVersion	Int	The version when the replica last failed. A value of -1 means no failure.
LastSuccessVersion	Int	The version when the replica last succeeded.
CommittedVersion	Int	The committed version of the replica.
SchemaHash	Int	A hash value representing the schema of the replica.
VersionNum	Int	The number of versions the replica has gone through.

Column	DataType	Note
IsBad	Boolean	Indicates whether the replica is in a bad state (true/false).
IsUserDrop	Boolean	Indicates if the replica has been marked for user-driven deletion.
State	String	The current state of the replica (e.g., NORMAL).
Status	String	The health status of the replica (e.g., OK).

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

- Display the replica status for all replicas of a table

```
SHOW REPLICA STATUS FROM db1.tbl1;
```

```
+----------+-----------+-----------+---------+-------------------+--------------------+------------------+------------+----
--------+-------+------------+--------+--------+
| TabletId | ReplicaId | BackendId | Version | LastFailedVersion | LastSuccessVersion | CommittedVersion | SchemaHash |
VersionNum | IsBad | IsUserDrop | State  | Status |
+----------+-----------+-----------+---------+-------------------+--------------------+------------------+------------+----
--------+-------+------------+--------+--------+
| 10145    | 10146     | 10009     | 14      | -1                | 14                 | 14               | 182881783  | 1
| false | false      | NORMAL | OK     |
| 10147    | 10148     | 10009     | 14      | -1                | 14                 | 14               | 182881783  | 1
| false | false      | NORMAL | OK     |
| 10149    | 10150     | 10009     | 14      | -1                | 14                 | 14               | 182881783  | 1
| false | false      | NORMAL | OK     |
| 10151    | 10152     | 10009     | 14      | -1                | 14                 | 14               | 182881783  | 1
| false | false      | NORMAL | OK     |
+----------+-----------+-----------+---------+-------------------+--------------------+------------------+------------+----
--------+-------+------------+--------+--------+
```

- Display the replicas of specific partitions with a VERSION_ERROR status

```
SHOW REPLICA STATUS FROM tbl1 PARTITION (p1, p2)
WHERE STATUS = "VERSION_ERROR";
```

- Display all replicas of a table that are in unhealthy states

```
SHOW REPLICA STATUS FROM tbl1
WHERE STATUS != "OK";
```

SHOW REPLICA DISTRIBUTION

Description

This statement is used to display the replica status information for a table or partition.

Syntax

```
SHOW REPLICA DISTRIBUTION FROM [ <database_name>.]<table_name> [<partition_list>]
[where_clause]
```

Where:

```
partition_list
  : PARTITION (<partition_name>[ , ... ])
```

Where:

```
where_clause
: WHERE <output_column_name> = <value>
```

Required Parameters

1. `<table_name>`

> The identifier (i.e., name) of the table, which must be unique within the database (Database).
>
> The identifier must start with a letter character (or any character from supported languages if Unicode name support is enabled), and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., My Object).
>
> Identifiers cannot use reserved keywords.
>
> For more details, refer to the identifier requirements and reserved keywords.

Optional Parameters

1. `<db_name>`

> The identifier (i.e., name) of the database, which must be unique within the cluster (Cluster).
>
> The identifier must start with a letter character (or any character from supported languages if Unicode name support is enabled), and cannot contain spaces or special characters unless the entire identifier

string is enclosed in backticks (e.g., My Object).

Identifiers cannot use reserved keywords.

For more details, refer to the identifier requirements and reserved keywords.

2. `<partition_list>`

A comma-separated list of partition identifiers (i.e., names), which must be unique within the table (Table).

The identifier must start with a letter character (or any character from supported languages if Unicode name support is enabled), and cannot contain spaces or special characters unless the entire identifier string is enclosed in backticks (e.g., My Object).

Identifiers cannot use reserved keywords.

For more details, refer to the identifier requirements and reserved keywords.

Return Value

Column	DataType	Note
BackendId	Int	The ID of the BE (Backend) node where the replica is located.
ReplicaNum	Int	The number of replicas on this BE node.
ReplicaSize	Int	The total storage size occupied by all replicas on this BE node (in bytes).
NumGraph	String	A visual representation (using > symbols) of the replica count proportion.
NumPercent	String	The percentage representation of the replica count, e.g., 100.00%.
SizeGraph	String	A visual representation (using > symbols) of the replica storage size proportion.
SizePercent	String	The percentage representation of the replica storage size, e.g., 100.00%.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
Admin_priv	Database	Required to execute administrative operations on the database, including managing tables, partitions, and system-level commands.

Examples

- View the replica distribution of a table

```
SHOW REPLICA DISTRIBUTION FROM sell_user;
```

```
*************************** 1. row ***************************
BackendId: 10009
 ReplicaNum: 4
ReplicaSize: 8857
   NumGraph: >>>>
```

ADMIN CLEAN TRASH

Description

This statement is used to clear garbage data in backend.

Syntax

```
ADMIN CLEAN TRASH [ON ("<be_host>:<be_heartbeat_port>" [, ...])]
```

Optional Parameters

1. `[ON ("<be_host>:<be_heartbeat_port>" [, ...])]`

Specify the backend to be cleaned up. If you do not add ON, all backend is cleared by default.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	User or Role	Only users or roles with the ADMIN_PRIV privilege can perform the CLEAN TRASH operation.

Examples

```
-- Clean up the junk data of all be nodes.
ADMIN CLEAN TRASH;
```

```
-- Clean up garbage data for '192.168.0.1:9050' and '192.168.0.2:9050'.
ADMIN CLEAN TRASH ON ("192.168.0.1:9050", "192.168.0.2:9050");
```

SHOW TRASH

Description

This statement is used to view the space occupied by garbage data in backend.

Syntax

```
SHOW TRASH [ON ("<be_host>:<be_heartbeat_port>" [, ...])];
```

Optional Parameters

1. `[ON ("<be_host>:<be_heartbeat_port>" [, ...])]`

Specify the backend that you want to view. If you do not add ON, all backend is displayed by default.

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV or NODE_PRIV	User or Role	Only users or roles with the ADMIN_PRIV or NODE_PRIV privilege can perform the SHOW TRASH operation.

Examples

1. View the garbage data usage of all be nodes.

```
SHOW TRASH;
```

2. View the junk data usage of '192.168.0.1:9050' (disk information is displayed).

```
SHOW TRASH ON "192.168.0.1:9050";
```

SHOW CATALOG RECYCLE BIN

Description

This statement is used to display the recoverable metadata of databases, tables, or partitions in the recycle bin.

Syntax

```
SHOW CATALOG RECYCLE BIN [ WHERE NAME [ = "<name>" | LIKE "<name_matcher>"] ]
```

Optional Parameters

Filter by name

1. `<name>` > The name of the database, table, or partition.

Filter by pattern matching

1. `<name_matcher>` > The pattern matching for the name of the database, table, or partition.

Return Values

Column	Type	Note
Type	String	Metadata type: Database, Table, Partition
Name	String	Metadata name
DbId	Bigint	ID of the database
TableId	Bigint	ID of the table
PartitionId	Bigint	ID of the partition
DropTime	DateTime	Time when the metadata was moved to the recycle bin
DataSize	Bigint	Data size. If the metadata type is database, this value includes the data size of all tables and partitions in the recycle bin

Column	Type	Note
RemoteDataSize	Decimal	Data size on remote storage (HDFS or object storage). If the metadata type is database, this value includes the remote data size of all tables and partitions in the recycle bin

Access Control Requirements

Privilege	Object	Notes
ADMIN_PRIV		

Examples

1. Display all metadata in the recycle bin

```
SHOW CATALOG RECYCLE BIN;
```

2. Display metadata with the name ‘test’ in the recycle bin

```
SHOW CATALOG RECYCLE BIN WHERE NAME = 'test';
```

3. Display metadata with names starting with ‘test’ in the recycle bin

```
SHOW CATALOG RECYCLE BIN WHERE NAME LIKE 'test%';
```

RECOVER

Description

This statement is used to recover previously deleted databases, tables, or partitions.

It supports recovering specified metadata by name or ID and allows renaming the recovered metadata.

Syntax:

```
RECOVER { DATABASE <db_name> [<db_id>] [AS <new_db_name>]
        | TABLE [<db_name>.]<table_name> [<table_id>] [AS <new_table_name>]
        | PARTITION <partition_name> [<partition_id>] FROM [<db_name>.]<table_name> [AS <new_partition_name>] }
```

Required Parameters

Recover a database

1. <db_name> > The name of the database to recover.

Recover a table

1. <table_name> > The name of the table to recover.

Recover a partition

1. <partition_name> > The name of the partition to recover.

2. <table_name> > The name of the table where the partition resides.

Optional Parameters

Recover a database

1. <db_id> > The ID of the database to recover.

2. <new_db_name> > The new name of the recovered database.

Recover a table

1. <db_name> > The name of the database where the table resides.

2. <table_id> > The ID of the table to recover.

3. <new_table_name> > The new name of the recovered table.

Recover a partition

1. <partition_id> > The ID of the partition to recover.

2. `<db_name>` > The name of the database where the table resides.

3. `<new_partition_name>` > The new name of the recovered partition.

Access Control Requirements

Privilege	Object	Note
ADMIN_PRIV		

Usage Notes

- This operation can only recover metadata deleted within a certain period. The default is 1 day (configurable via the `catalog_trash_expire_second` parameter in `fe.conf`).
- If no ID is specified when recovering metadata, the last deleted metadata with the same name is recovered by default.
- You can query the currently recoverable metadata using `SHOW CATALOG RECYCLE BIN`.

Examples

1. Recover a database named `example_db`

```
RECOVER DATABASE example_db;
```

2. Recover a table named `example_tbl`

```
RECOVER TABLE example_db.example_tbl;
```

3. Recover a partition named `p1` from the table `example_tbl`

```
RECOVER PARTITION p1 FROM example_tbl;
```

4. Recover a database with ID `example_db_id` and name `example_db`

```
RECOVER DATABASE example_db example_db_id;
```

5. Recover a table with ID `example_tbl_id` and name `example_tbl`

```
RECOVER TABLE example_db.example_tbl example_tbl_id;
```

6. Recover a partition with ID `p1_id` and name `p1` from the table `example_tbl`

```
RECOVER PARTITION p1 p1_id FROM example_tbl;
```

7. Recover a database with ID `example_db_id` and name `example_db`, and rename it to `new_example_db`

```
RECOVER DATABASE example_db example_db_id AS new_example_db;
```

8. Recover a table named `example_tbl` and rename it to `new_example_tbl`

```
RECOVER TABLE example_db.example_tbl AS new_example_tbl;
```

9. Recover a partition with ID `p1_id` and name `p1` from the table `example_tbl`, and rename it to `new_p1`

```
RECOVER PARTITION p1 p1_id AS new_p1 FROM example_tbl;
```

CREATE FUNCTION

Description

This statement is used to create a custom function.

Syntax

```
CREATE [ GLOBAL ]
    [{AGGREGATE | TABLES | ALIAS }] FUNCTION <function_name>
    (<arg_type> [, ...])
    [ RETURNS <ret_type> ]
    [ INTERMEDIATE <inter_type> ]
    [ WITH PARAMETER(<param> [,...]) AS <origin_function> ]
    [ PROPERTIES ("<key>" = "<value>" [, ...]) ]
```

Required Parameters

1. `<function_name>`

> If `function_name` includes a database name, such as `db1.my_func`, the custom function will be created in the corresponding database. Otherwise, the function will be created in the database of the current session. The name and parameters of the new function must not be identical to an existing function in the current namespace; otherwise, the creation will fail.

2. `<arg_type>`

> The input parameter type of the function. For variable-length parameters, use ， ... to indicate them. If it is a variable-length type, the type of the variable-length parameters must be consistent with the type of the last non-variable-length parameter.

3. `<ret_type>`

> The return parameter type of the function. This is a required parameter for creating a new function. If creating an alias for an existing function, this parameter is not necessary.

Optional Parameters

1. `GLOBAL`

> If specified, the created function is effective globally.

2. `AGGREGATE`

> If specified, the created function is an aggregate function.

3. `TABLES`

> If specified, the created function is a table function.

4. `ALIAS`

> If specified, the created function is an alias function.

> If none of the above parameters representing the function type is selected, it indicates that the created function is a scalar function.

5. `<inter_type>`

> Used to indicate the data type during the intermediate stage of an aggregate function.

6. `<param>`

> Used to indicate the parameters of an alias function, with at least one parameter required.

7. `<origin_function>`

> Used to indicate the original function corresponding to the alias function.

8. `<properties>`

> - `file`: Indicates the JAR package containing the user-defined function (UDF). In a multi-machine environment, it can also be downloaded via HTTP. This parameter is mandatory.
> - `symbol`: Indicates the class name containing the UDF class. This parameter is mandatory.
> - `type`: Indicates the UDF call type. The default is Native. Use JAVA_UDF when using a Java UDF.

- `always_nullable`: Indicates whether the UDF result may contain NULL values. This is an optional parameter with a default value of true.

Access Control Requirements

To execute this command, the user must have `ADMIN_PRIV` privileges.

Example

1. Create a custom UDF function. For more details, refer to JAVA-UDF.

```
CREATE FUNCTION java_udf_add_one(int) RETURNS int PROPERTIES (
    "file"="file:///path/to/java-udf-demo-jar-with-dependencies.jar",
    "symbol"="org.apache.doris.udf.AddOne",
    "always_nullable"="true",
    "type"="JAVA_UDF"
);
```

2. Create a custom UDAF function.

```
CREATE AGGREGATE FUNCTION simple_sum(INT) RETURNS INT PROPERTIES (
    "file"="file:///pathTo/java-udaf.jar",
    "symbol"="org.apache.doris.udf.demo.SimpleDemo",
    "always_nullable"="true",
    "type"="JAVA_UDF"
);
```

3. Create a custom UDTF function.

```
CREATE TABLES FUNCTION java_udtf(string, string) RETURNS array<string> PROPERTIES (
    "file"="file:///pathTo/java-udaf.jar",
    "symbol"="org.apache.doris.udf.demo.UDTFStringTest",
    "always_nullable"="true",
    "type"="JAVA_UDF"
);
```

4. Create a custom alias function. For more information, refer to Alias Function.

```
CREATE ALIAS FUNCTION id_masking(INT) WITH PARAMETER(id) AS CONCAT(LEFT(id, 3), '****', RIGHT(id, 4));
```

5. Create a global custom alias function.

```
CREATE GLOBAL ALIAS FUNCTION id_masking(INT) WITH PARAMETER(id) AS CONCAT(LEFT(id, 3), '****', RIGHT(id, 4));
```

DROP FUNCTION

Description

Delete a custom function.

Syntax

```
DROP [ GLOBAL ] FUNCTION <function_name> ( <arg_type> )
```

Required Parameters

1. `<function_name>`

> Specifies the name of the function to delete.
>
> The function name needs to be exactly the same as the function name when the function was created.

2. `<arg_type>`

> Specifies the argument list for the function to be deleted.
>
> Parameter List Location You need to enter the data type of the location parameter.

Optional Parameters

1.`GLOBAL`

> GLOBAL is an optional parameter.
>
> If GLOBAL is set, the function is searched for globally and deleted.
>
> If GLOABL is not entered, the function is searched for in the current database and deleted.

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Custom function	DROP is an administrative operation

Usage Notes

- A function can be deleted only when its name and parameter types are identical

Examples

```
DROP FUNCTION my_add(INT, INT)
```

```
DROP GLOBAL FUNCTION my_add(INT, INT)
```

SHOW CREATE FUNCTION

Description

This statement is used to show the creation statement of a user-defined function

Syntax

```
SHOW CREATE [ GLOBAL ] FUNCTION <function_name>( <arg_type> ) [ FROM <db_name> ];
```

Required Parameters

1. `<function_name>`

> The name of the custom function that you want to query for the creation statement.

2. `<arg_type>`

> The parameter list of the custom function that needs to be queried for the creation statement.
>
> Parameter list location you need to enter the data type of the location parameter

Optional Parameters

1.`GLOBAL`

> GLOBAL is an optional parameter.
>
> If GLOBAL is set, the function is searched for globally and deleted.
>
> If GLOABL is not entered, the function is searched for in the current database and deleted.

2.`<db_name>`

> FROM db_name indicates that the custom function is queried from the specified database

Return Value

Column	Description
SYMBOL	Function package name
FILE	jar package path
ALWAYS_NULLABLE	Whether the result can be NULL
TYPE	Function type

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
SHOW_PRIV	Function	You need to have the show permission on this function

Examples

```
SHOW CREATE FUNCTION add_one(INT)
```

```
| Function Signature | Create Function
+--------------------+-------------------------------------------------------
| add_one(INT)       | CREATE FUNCTION add_one(INT) RETURNS INT PROPERTIES (
  "SYMBOL"="org.apache.doris.udf.AddOne",
  "FILE"="file:///xxx.jar",
  "ALWAYS_NULLABLE"="true",
  "TYPE"="JAVA_UDF"
  ); |
+--------------------+-------------------------------------------------------
```

DESC FUNCTION

Description

Use the desc function table_valued_function to obtain the schema information for the corresponding table-valued function.

Syntax

```
DESC FUNCTION <table_valued_function>
```

Required Parameters

> table_valued_function, the name of the table-valued function, such as CATALOGS. For a list of supported table-valued functions, please refer to the "Table-Valued Functions" section

Examples

Query the information of the table-valued function CATALOGS:

```
DESC FUNCTION catalogs();
```

The result is as follows:

```
+-------------+--------+------+-------+---------+-------+
| Field       | Type   | Null | Key   | Default | Extra |
+-------------+--------+------+-------+---------+-------+
| CatalogId   | bigint | No   | false | NULL    | NONE  |
| CatalogName | text   | No   | false | NULL    | NONE  |
| CatalogType | text   | No   | false | NULL    | NONE  |
| Property    | text   | No   | false | NULL    | NONE  |
| Value       | text   | No   | false | NULL    | NONE  |
+-------------+--------+------+-------+---------+-------+
```

SHOW FUNCTIONS

Description

View all custom and system provided functions under the database.

Syntax

```
SHOW [ FULL ] [ BUILTIN ] FUNCTIONS [ { IN | FROM } <db> ]  [ LIKE '<function_pattern>' ]
```

Varaint Syntax

```
SHOW GLOBAL [ FULL ] FUNCTIONS [ LIKE '<function_pattern>' ]
```

Required Parameters

1. `<function_pattern>`

> Matching pattern rules used to filter function names

Optional Parameters

1. `FULL`

> FULL is an optional parameter.
>
> This parameter indicates the detailed information about the function.

2. `BUILTIN`

> BUILTIN is an optional parameter.
>
> This parameter indicates that the functions provided by the system need to be displayed

3. `<db>`

> db is an optional parameter.

> This parameter indicates the query under the specified database

Return Value

Column	Description
Signature	Function name and parameter type
Return Type	The data type of the value returned by the function
Function Type	Type of function
Intermediate Type	Intermediate result type
Properties	Detailed properties of a function

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
SHOW_PRIV	Function	You need to have the show permission on this function

Examples

```
show full functions in testDb
```

```
*************************** 1. row ***************************
Signature: my_add(INT,INT)
Return Type: INT
Function Type: Scalar
Intermediate Type: NULL
Properties:
{"symbol":"_ZN9doris_udf6AddUdfEPNS_15FunctionContextERKNS_6IntValES4_","object_file":"http://host:port/libudfsample.so","md5":"cfe7
a362d10f3aaf6c49974ee0f1f878"}
*************************** 2. row ***************************
Signature: my_count(BIGINT)
Return Type: BIGINT
Function Type: Aggregate
Intermediate Type: NULL
Properties:
{"object_file":"http://host:port/libudasample.so","finalize_fn":"_ZN9doris_udf13CountFinalizeEPNS_15FunctionContextERKNS_9BigIntValE
","init_fn":"_ZN9doris_udf9CountInitEPNS_15FunctionContextEPNS_9BigIntValE","merge_fn":"_ZN9doris_udf10CountMergeEPNS_15FunctionCont
extERKNS_9BigIntValEPS2_","md5":"37d185f80f95569e2676da3d5b5b9d2f","update_fn":"_ZN9doris_udf11CountUpdateEPNS_15FunctionContextERKN
S_6IntValEPNS_9BigIntValE"}
*************************** 3. row ***************************
Signature: id_masking(BIGINT)
Return Type: VARCHAR
Function Type: Alias
Intermediate Type: NULL
Properties: {"parameter":"id","origin_function":"concat(left(`id`, 3), `****`, right(`id`, 4))"}
```

```
show builtin functions in testDb like 'year%';
```

```
+---------------+
| Function Name |
+---------------+
| year          |
| years_add     |
| years_diff    |
| years_sub     |
+---------------+
```

```
show global full functions
```

```
*************************** 1. row ***************************
        Signature: decimal(ALL, INT, INT)
      Return Type: VARCHAR
    Function Type: Alias
Intermediate Type: NULL
       Properties: {"parameter":"col, precision, scale","origin_function":"CAST(`col` AS decimal(`precision`, `scale`))"}
*************************** 2. row ***************************
        Signature: id_masking(BIGINT)
      Return Type: VARCHAR
    Function Type: Alias
Intermediate Type: NULL
       Properties: {"parameter":"id","origin_function":"concat(left(`id`, 3), `****`, right(`id`, 4))"}
```

```
show global functions
```

```
+---------------+
| Function Name |
+---------------+
| decimal       |
| id_masking    |
+---------------+
```

ANALYZE

Description

This statement is used to collect column statistics. Statistics of columns can be collected for a table (specific columns can be specified) or for the entire database.

Syntax

```
ANALYZE {TABLE <table_name> [ (<column_name> [, ...]) ] | DATABASE <database_name>}
    [ [ WITH SYNC ] [ WITH SAMPLE {PERCENT | ROWS} <sample_rate> ] ];
```

Required Parameters

1. `<table_name>`

> The specified target table. This parameter and the parameter must have and can only have one of them specified.

2. `<database_name>`

> The specified target database. This parameter and the parameter must have and can only have one of them specified.

Optional Parameters

1. `<column_name>`

> The specified target column. It must be an existing column in `table_name`. You can specify multiple column names separated by commas.

2. `WITH SYNC`

> Collect statistics synchronously. Returns after collection. If not specified, it executes asynchronously.

3. `WITH SAMPLE {PERCENT | ROWS} <sample_rate>`

> Specify to use the sampling method for collection. When not specified, full collection is the default. is the sampling parameter. When using PERCENT sampling, it specifies the sampling percentage; when using ROWS sampling, it specifies the number of sampled rows.

Return Value

Column	Note
Job_Id	Unique Job Id
Catalog_Name	Catalog name
DB_Name	database name
Columns	column name list

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
SELECT_PRIV	Table	When executing ANALYZE, the SELECT_PRIV privilege for the queried table is required.

Examples

1. Collect statistics by sampling 10% of table lineitem.

```
ANALYZE TABLE lineitem WITH SAMPLE PERCENT 10;
```

2. Collect statistics by sampling 100,000 rows from table lineitem.

```
ANALYZE TABLE lineitem WITH SAMPLE ROWS 100000;
```

ALTER STATS

Description

Manually modify the statistics of a specified column in a specified table. Please refer to the “Statistics” chapter.

Syntax

```
ALTER TABLE <table_name>
  [ INDEX <index_name> ]
  MODIFY COLUMN <column_name>
  SET STATS (<column_stats>)
```

Where:

```
column_stats
  : -- column stats value
  ("key1" = "value1", "key2" = "value2" [...])
```

Required Parameters

> Specifies the identifier (i.e., name) of the table.

> Specifies the column identifier (i.e., name). When is not specified, it is the column name of the base table.

> The statistics value to be set, given in the form of key = value, where both key and value need to be enclosed in quotation marks, and key-value pairs are separated by commas. The statistics that can be set include:

> row_count, total number of rows

> ndv, cardinality of the column

> num_nulls, number of null values in the column

> data_size, total size of the column

> min_value, minimum value of the column

> max_value, maximum value of the column

> Among them, row_count must be specified, and other attributes are optional. If not set, the corresponding statistic attribute value for that column will be empty.

Optional Parameters

> Synchronized materialized view (please refer to the “Synchronized Materialized Views” chapter) identifier (i.e., name). A table can have 0 to multiple materialized views. If you need to set the statistics of a column in a materialized view, you need to use to specify the name of the materialized view. If not specified, the properties of the column in the base table are set.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ALTER_PRIV	Table	

Usage Notes

After a user manually injects statistics into a table, the table will no longer participate in the automatic collection of statistics (please refer to the “Automatic Collection of Statistics” chapter) to avoid overwriting the statistics manually injected by the user. If the injected statistics are no longer used, the drop stats statement can be used to delete the already injected information, which allows the table to re-enable automatic collection.

Examples

- Inject statistics into the p_partkey column of the part table (base table column, as no index_name is specified).

```
alter
    table part
```

```
    modify column p_partkey
    set stats ('row_count'='2.0E7', 'ndv'='2.0252576E7', 'num_nulls'='0.0', 'data_size'='8.0E7', 'min_value'='1',
'max_value'='20000000');
```

- Inject statistics into the col1 column of the index1 materialized view of the part table (materialized view column, as index_name is specified).

```
alter
    table part index index1
    modify column col1
    set stats ('row_count'='2.0E7', 'ndv'='2.0252576E7', 'num_nulls'='0.0', 'data_size'='8.0E7', 'min_value'='1',
'max_value'='20000000');
```

DROP STATS

Description

Delete statistics information for the specified table and columns. If no column names are specified, the statistics information for all columns will be deleted.

Syntax

```
DROP STATS <table_name> [ <column_names> ]
```

Where:

```
column_names
  :
  (<column_name>, [ <column_name>... ])
```

Required Parameters

> The identifier (name) of the table.

Optional Parameters

> List of column identifiers (names).

Access Control Requirements

Users executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
DROP_PRIV	Table	

Examples

- Delete statistics information for all columns in table1

  ```
  DROP STATS table1
  ```

- Delete statistics information for col1 and col2 in table1

```
DROP STATS table1 (col1, col2)
```

SHOW TABLE STATS

Description

This statement is used to view the overview statistics of a table.

Syntax

```
SHOW TABLE STATS <table_name>;
```

Required Parameters

1. `<table_name>`

> Table name

Optional Parameters

None

Return Value

Column	Note
updated_rows	table updated row count
query_times	table query times
row_count	table row count
updated_time	table last modified time
columns	analyzed column list
trigger	last analyze trigger method
new_partition	flag of new partition first loaded
user_inject	flag of user inject statistics
enable_auto_analyze	flag of auto analyzed enabled
last_analyze_time	last analyze time

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
SELECT_PRIV	Table	When executing SHOW, the SELECT_PRIV privilege for the queried table is required.

Examples

1. Show the overview statistics of table test1.

```
SHOW TABLE STATS test1;
```

```
+--------------+-------------+-----------+---------------------+------------------------+---------+--------------+-------------+---
------------------+---------------------+
| updated_rows | query_times | row_count | updated_time        | columns                | trigger | new_partition | user_inject |
enable_auto_analyze | last_analyze_time   |
+--------------+-------------+-----------+---------------------+------------------------+---------+--------------+-------------+---
------------------+---------------------+
| 0            | 0           | 100000    | 2025-01-17 16:46:31 | [test1:name, test1:id] | MANUAL  | false         | false       |
true                | 2025-02-05 12:17:41 |
+--------------+-------------+-----------+---------------------+------------------------+---------+--------------+-------------+---
------------------+---------------------+
```

SHOW COLUMN STATS

Description

This statement is used to show the column statistics of a table.

Syntax

```
SHOW COLUMN [CACHED] STATS <table_name> [ (<column_name> [, ...]) ];
```

Required Parameters

1. `<table_name>`

> The name of the table for which column statistics needs to be displayed.

Optional Parameters

1. `CACHED`

> Show the column statistics in FE cache. When not specified, the information persisted in the statistics table is displayed by default.

2. `<column_name>`

> Specify the column names that need to be displayed. The column names must exist in the table, and multiple column names are separated by commas. If not specified, the information of all columns will be displayed by default.

Return Value

Column	Note
column_name	column name
index_name	index name
count	column row count
ndv	column distinct value
num_null	column null count

Column	Note
data_size	column total data size
avg_size_byte	column average size
min	min value
max	max value
method	collect method
type	collect type
trigger	job trigger method
query_times	query times
updated_time	update time
update_rows	update rows when last analyze
last_analyze_row_count	table row count when last analyze
last_analyze_version	table version when last analyze

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
SELECT_PRIV	Table	When executing SHOW, the SELECT_PRIV privilege for the queried table is required.

Examples

1. Show the statistics of all columns in table test1.

```
SHOW COLUMN STATS test1;
```

```
+-------------+------------+----------+---------+----------+-----------+---------------+--------+--------+--------+-------------+--
-------+-------------+---------------------+-------------+-----------------------+---------------------+
| column_name | index_name | count    | ndv     | num_null | data_size | avg_size_byte | min    | max    | method | type        |
trigger | query_times | updated_time        | update_rows | last_analyze_row_count | last_analyze_version |
+-------------+------------+----------+---------+----------+-----------+---------------+--------+--------+--------+-------------+--
-------+-------------+---------------------+-------------+-----------------------+---------------------+
| name        | test1      | 87775.0  | 48824.0 | 0.0      | 351100.0  | 4.0           | '0001' | 'ffff' | FULL   | FUNDAMENTALS |
MANUAL  | 0           | 2025-02-05 12:17:08 | 0           | 100000                | 3                     |
| id          | test1      | 100000.0 | 8965.0  | 0.0      | 351400.0  | 3.514         | 1000   | 9999   | SAMPLE | FUNDAMENTALS |
MANUAL  | 0           | 2025-02-05 12:17:41 | 0           | 100000                | 3                     |
+-------------+------------+----------+---------+----------+-----------+---------------+--------+--------+--------+-------------+--
-------+-------------+---------------------+-------------+-----------------------+---------------------+
```

2. Show the statistics of all columns in the test1 in the current FE cache.

```
SHOW COLUMN CACHED STATS test1;
```

```
+-------------+------------+----------+---------+----------+-----------+---------------+--------+--------+--------+-------------+--
-------+-------------+--------------------+-------------+-----------------------+---------------------+
| column_name | index_name | count    | ndv     | num_null | data_size | avg_size_byte | min    | max    | method | type        |
trigger | query_times | updated_time        | update_rows | last_analyze_row_count | last_analyze_version |
+-------------+------------+----------+---------+----------+-----------+---------------+--------+--------+--------+-------------+--
-------+-------------+--------------------+-------------+-----------------------+---------------------+
| name        | test1      | 87775.0  | 48824.0 | 0.0      | 351100.0  | 4.0           | '0001' | 'ffff' | FULL   | FUNDAMENTALS |
MANUAL  | 0           | 2025-02-05 12:17:08 | 0           | 100000                | 3                    |
| id          | test1      | 100000.0 | 8965.0  | 0.0      | 351400.0  | 3.514         | 1000   | 9999   | SAMPLE | FUNDAMENTALS |
MANUAL  | 0           | 2025-02-05 12:17:41 | 0           | 100000                | 3                    |
+-------------+------------+----------+---------+----------+-----------+---------------+--------+--------+--------+-------------+--
-------+-------------+--------------------+-------------+-----------------------+---------------------+
```

DROP ANALYZE JOB

Description

Delete the history of the specified statistics collection job.

Syntax

```
DROP ANALYZE JOB <job_id>
```

Required Parameters

> Specifies the id of the job. You can obtain the job_id by running SHOW ANALYZE. For detailed usage, please refer to the “SHOW ANALYZE” section.

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
SELECT_PRIV	Table	

Examples

Delete the statistics information job record with id 10036

```
DROP ANALYZE JOB 10036
```

KILL ANALYZE JOB

Description

Stop the statistical information collection job that is being executed in the background.

Syntax

```
KILL ANALYZE <job_id>
```

Required Parameters

> Specifies the id of the job. You can obtain the job_id of the job by using SHOW ANALYZE. For detailed usage, please refer to the “SHOW ANALYZE” chapter.

Optional Parameters

None

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege (Privilege)	Object (Object)	Notes (Notes)
SELECT_PRIV	Table	

Usage Notes

Jobs that have already been executed cannot be stopped.

Examples

Stop the statistical information job record with an id of 10036

```
kill analyze 10036
```

SHOW ANALYZE

Description

This statement is used to view the status of the statistics collection job.

Syntax

```
SHOW [AUTO] ANALYZE [ <table_name> | <job_id> ]
    [ WHERE STATE = {"PENDING" | "RUNNING" | "FINISHED" | "FAILED"} ];
```

Required Parameters

None

Optional Parameters

1. `AUTO`

> Show information about automatic jobs. If not specified, information about manual jobs will be displayed by default.

2. `<table_name>`

> Table name. After specifying it, you can view the job information corresponding to this table. When not specified, the job information of all tables will be returned by default.

3. `<job_id>`

> Statistics Job ID，Obtained when performing asynchronous collection with ANALYZE. When the ID is not specified, this command returns information about all jobs.

4. `WHERE STATE = {"PENDING" | "RUNNING" | "FINISHED" | "FAILED"}`

> Filter conditions of job state. If not specified, information about jobs in all states will be displayed by default.

Return Value

Column	Note
job_id	Unique statistics job id
catalog_name	Catalog name
db_name	database name
tbl_name	table name
col_name	column name list
job_type	job type
analysis_type	analysis type
message	error message
last_exec_time_in_ms	last analyze time
state	job state
progress	job progress
schedule_type	schedule type
start_time	job start time
end_time	job end time
priority	job priority
enable_partition	enable partition collection flag

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
SELECT_PRIV	Table	When executing SHOW, the SELECT_PRIV privilege for the queried table is required.

Examples

1. Show jobs by table name.

```
SHOW ANALYZE test1 WHERE STATE="FINISHED";
```

```
+---------------+--------------+---------+----------+----------------------+----------+---------------+---------+------------------
----+----------+-------------------------------------------------------+--------------+--------------------+--------------------
+----------+------------------+
| job_id        | catalog_name | db_name | tbl_name | col_name             | job_type | analysis_type | message |
last_exec_time_in_ms | state    | progress                                              | schedule_type | start_time          |
end_time            | priority | enable_partition |
+---------------+--------------+---------+----------+----------------------+----------+---------------+---------+------------------
----+----------+-------------------------------------------------------+--------------+--------------------+--------------------
```

```
+----------+-----------------+
| 1737454119144 | internal     | test    | test1    | [test1:name,test1:id] | MANUAL   | FUNDAMENTALS  |         | 2025-01-21
18:10:11  | FINISHED | 2 Finished  |  0 Failed  |  0 In Progress  |  2 Total | ONCE          | 2025-01-21 18:10:10 | 2025-01-21
18:10:11 | MANUAL   | false            |
| 1738725887879 | internal     | test    | test1    | [test1:name,test1:id] | MANUAL   | FUNDAMENTALS  |         | 2025-02-05
11:26:15  | FINISHED | 2 Finished  |  0 Failed  |  0 In Progress  |  2 Total | ONCE          | 2025-02-05 11:26:15 | 2025-02-05
11:26:15 | MANUAL   | false            |
| 1738725887890 | internal     | test    | test1    | [test1:name,test1:id] | MANUAL   | FUNDAMENTALS  |         | 2025-02-05
12:17:09  | FINISHED | 2 Finished  |  0 Failed  |  0 In Progress  |  2 Total | ONCE          | 2025-02-05 12:17:08 | 2025-02-05
12:17:09 | MANUAL   | false            |
| 1738725887895 | internal     | test    | test1    | [test1:id]            | MANUAL   | FUNDAMENTALS  |         | 2025-02-05
12:17:24  | FINISHED | 1 Finished  |  0 Failed  |  0 In Progress  |  1 Total | ONCE          | 2025-02-05 12:17:23 | 2025-02-05
12:17:24 | MANUAL   | false            |
| 1738725887903 | internal     | test    | test1    | [test1:id]            | MANUAL   | FUNDAMENTALS  |         | 2025-02-05
12:17:42  | FINISHED | 1 Finished  |  0 Failed  |  0 In Progress  |  1 Total | ONCE          | 2025-02-05 12:17:41 | 2025-02-05
12:17:42 | MANUAL   | false            |
+---------------+-------------+---------+----------+----------------------+----------+---------------+---------+------------------
----+----------+-------------------------------------------------------+--------------+--------------------+--------------------
+----------+-----------------+
```

2. Show job by job id.

```
show analyze 1738725887903;
```

```
+---------------+-------------+---------+----------+------------+----------+---------------+---------+----------------------+------
----+-------------------------------------------------------+--------------+--------------------+--------------------+----------
+-----------------+
| job_id        | catalog_name | db_name | tbl_name | col_name   | job_type | analysis_type | message | last_exec_time_in_ms | state
| progress                                              | schedule_type | start_time          | end_time            | priority |
enable_partition |
+---------------+-------------+---------+----------+------------+----------+---------------+---------+----------------------+------
----+-------------------------------------------------------+--------------+--------------------+--------------------+----------
+-----------------+
| 1738725887903 | internal     | test    | test1    | [test1:id] | MANUAL   | FUNDAMENTALS  |         | 2025-02-05 12:17:42  |
FINISHED | 1 Finished  |  0 Failed  |  0 In Progress  |  1 Total | ONCE          | 2025-02-05 12:17:41 | 2025-02-05 12:17:42 |
MANUAL   | false            |
+---------------+-------------+---------+----------+------------+----------+---------------+---------+----------------------+------
----+-------------------------------------------------------+--------------+--------------------+--------------------+----------
+-----------------+
```

ADD FOLLOWER

Description

This statement adds a node with the FOLLOWER role to the FRONTEND node (only used by administrators!)

Syntax

```
ALTER SYSTEM ADD FOLLOWER "<follower_host>:<edit_log_port>"
```

Required Parameters

1. `<follower_host>`

> Can be the hostname or IP address of the FE node

2. `<edit_log_port>`

> bdbje communication port of FE node, the default is 9010

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. Before adding a new FOLLOWER node, make sure the node is properly configured.
2. Before adding a FOLLOWER node, ensure that the number of FOLLOWER nodes in the cluster is an odd number after the addition.
3. After adding the FOLLOWER node, use the `SHOW FRONTENDS` command to verify that they have been added successfully and are in a normal state.

Examples

1. Add a FOLLOWER node

   ```
   ALTER SYSTEM ADD FOLLOWER "host_ip:9010"
   ```

This command adds a FOLLOWER node to the cluster (IP host_ip, port 9010)

DROP FOLLOWER

Description

This statement deletes the node with the FOLLOWER role of FRONTEND. (For administrators only!)

Syntax

```
ALTER SYSTEM DROP FOLLOWER "<follower_host>:<edit_log_port>"
```

Required Parameters

1. `<follower_host>`

> Can be the hostname or IP address of the FE node

2. `<edit_log_port>`

> bdbje communication port of FE node, the default is 9010

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. Before deleting a FOLLOWER node, make sure that the node that needs to be taken offline is not a Master node.
2. Before deleting a FOLLOWER node, ensure that the number of FOLLOWER nodes in the cluster is an odd number after they go offline.
3. After deleting the FOLLOWER node, use the `SHOW FRONTENDS` command to verify that they were successfully deleted.

Examples

1. Delete a FOLLOWER node

```
ALTER SYSTEM DROP FOLLOWER "host_ip:9010"
```

This command deletes a FOLLOWER node in the cluster (IP host_ip, port 9010)

ADD OBSERVER

Description

This statement adds a node with the OBSERVER role to the FRONTEND node (only used by administrators!)

Syntax

```
ALTER SYSTEM ADD OBSERVER "<observer_host>:<edit_log_port>"
```

Required Parameters

1. `<observer_host>`

> Can be the hostname or IP address of the FE node

2. `<edit_log_port>`

> bdbje communication port of FE node, the default is 9010

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. Before adding a new OBSERVER node, make sure the node is properly configured.
2. After adding the OBSERVER node, use the `SHOW FRONTENDS` command to verify that they have been added successfully and are in a normal state.

Examples

1. Add a OBSERVER node

   ```
   ALTER SYSTEM ADD OBSERVER "host_ip:9010"
   ```

   This command adds a OBSERVER node to the cluster (IP host_ip, port 9010)

DROP OBSERVER

Description

This statement deletes the node with the OBSERVER role of FRONTEND (only used by administrators!)

Syntax

```
ALTER SYSTEM DROP OBSERVER "<observer_host>:<edit_log_port>"
```

Required Parameters

1. `<observer_host>`

> Can be the hostname or IP address of the FE node

2. `<edit_log_port>`

> bdbje communication port of FE node, the default is 9010

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. After deleting the OBSERVER node, use the SHOW FRONTENDS command to verify that they were successfully deleted.

Examples

1. Deleting an OBSERVER node

   ```
   ALTER SYSTEM DROP OBSERVER "host_ip:9010"
   ```

   This command deletes an OBSERVER node in the cluster (IP host_ip, port 9010)

SET FRONTEND CONFIG

Description

This statement is used to set the configuration items of the cluster (currently only supports setting FE configuration items).

Syntax:

```
ADMIN SET {ALL FRONTENDS | FRONTEND} CONFIG ("<fe_config_key>" = "<fe_config_value>")
```

Required Parameters

`{ALL FRONTENDS | FRONTEND}` > `ALL FRONTENDS`: Represents all FE nodes in the Doris cluster > > `FRONTEND`: Represents the currently connected FE node, i.e., the FE node the user is interacting with

Optional Parameters

The `<fe_config_key>` and `<fe_config_value>` that need to be modified can be viewed through the SHOW FRONTEND CONFIG command

> **Explanation**
>
> - Starting from version 2.1.5, the `ALL` keyword is supported. When using the `ALL` keyword, the configuration parameters will be applied to all FEs (except for the `master_only` parameter).
> - This syntax does not persistently modify the configuration. After an FE restarts, the modified configuration becomes invalid. To persist the changes, the configuration items need to be synchronously added in fe.conf. :::
>
> ## Example
>
> 1. Set `disable_balance` to `true`
>
>    ```
>    ADMIN SET FRONTEND CONFIG ("disable_balance" = "true");
>    ```
>
> 2. Set `disable_balance` of all FE nodes to `true`
>
>    ```
>    ADMIN SET ALL FRONTENDS CONFIG ("disable_balance" = "true");
>    ```

SHOW FRONTEND CONFIG

Description

This statement is used to display the configuration of the current cluster (currently only the configuration items of FE are supported)

Syntax

```
SHOW FRONTEND CONFIG [LIKE "<pattern>"];
```

Optional Parameters

`<pattern>` > A string that can contain ordinary characters and wildcards.

Return Values

Column name	Describe
Value	Configuration item value
Type	Configuration item type
IsMutable	Whether it can be set by `ADMIN SET CONFIG` command
MasterOnly	Is it only applicable to Master FE
Comment	Configuration item description

Example

1. View the configuration of the current FE node

   ```
   SHOW FRONTEND CONFIG;
   ```

2. Use the like predicate to search the configuration of the current Fe node

   ```
    SHOW FRONTEND CONFIG LIKE '%check_java_version%';
   ```

   ```
   +--------------------+-------+---------+-----------+------------+---------+
   | Key                | Value | Type    | IsMutable | MasterOnly | Comment |
   +--------------------+-------+---------+-----------+------------+---------+
   | check_java_version | true  | boolean | false     | false      |         |
   +--------------------+-------+---------+-----------+------------+---------+
   ```

MODIFY FRONTEND HOSTNAME

Description

Modify the properties of the FRONTEND (hereinafter referred to as FE). Currently, this command can only modify the HOSTNAME of the FE. When the hostname of the host where a certain FE instance in the cluster is running needs to be changed, this command can be used to change the hostname registered by this FE in the cluster so that it can continue to operate normally.

This command is only used to convert the DORIS cluster to FQDN deployment. For details on FQDN deployment, please refer to the “FQDN” chapter.

Syntax

```
ALTER SYSTEM MODIFY FRONTEND "<frontend_hostname_port>" HOSTNAME "<frontend_new_hostname>"
```

Required Parameters

> The hostname and edit log port registered by the FE whose hostname needs to be changed. You can view information about all FEs in the cluster by using the SHOW FRONTENDS command. For detailed usage, please refer to the “SHOW FRONTENDS” chapter.

> The new hostname of the FE.

Access Control Requirements

The user executing this SQL command must have at least NODE_PRIV permissions.

Examples

Change the hostname of an FE instance in the cluster from 10.10.10.1 to 172.22.0.1:

```
ALTER SYSTEM
MODIFY FRONTEND "10.10.10.1:9010"
HOSTNAME "172.22.0.1"
```

SHOW FRONTENDS

Description

This statement is used to view the basic status information of the FE nodes.

Syntax

```
SHOW FRONTENDS
```

Return Value

Column	Note
Name	The name of the current FE in Doris. This name is usually a random string prefixed with `fe`.
Host	The IP address or host name of the current FE.
EditLogPort	The bdbje communication port of the current FE.
HttpPort	The http communication port of the current FE.
QueryPort	The MySQL protocol communication port of the current FE.
RpcPort	The thrift RPC communication port of the current FE.
ArrowFlightSqlPort	The ArrowFlight protocol communication port of the current FE.
Role	The role of the current FE. Possible values are FOLLOWER and OBSERVER.
IsMaster	Whether the current FE is elected as the Master.
ClusterId	The ID of the current Doris cluster, usually a randomly generated number.
Join	Used to indicate whether the current FE node has successfully joined the current Doris cluster.
Alive	Whether the current FE is alive.
ReplayedJournalId	The ID of the largest metadata log that the current FE has replayed.
LastStartTime	The timestamp when the current FE started.
LastHeartbeat	The timestamp of the last successful heartbeat sent by the current FE.
IsHelper	Whether the current FE is a helper node in bdbje.
ErrMsg	The error message when the heartbeat of the current FE fails.
Version	The version information of the current FE.
CurrentConnected	Whether the current client connection is connected to the current FE node.

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV		

Usage Notes

If you need to further filter the query results, you can use the table-valued function frontends(). SHOW FRONTENDS is equivalent to the following statement:

```
SELECT * FROM FRONTENDS();
```

Examples

```
SHOW FRONTENDS
```

```
+----------------------------------------+-----------+-------------+----------+-----------+---------+--------------------+---------
-+----------+-----------+------+-------+------------------+--------------------+--------------------+----------+--------+--------
--------------------+-----------------+
| Name                                   | Host      | EditLogPort | HttpPort | QueryPort | RpcPort | ArrowFlightSqlPort | Role
| IsMaster | ClusterId | Join | Alive | ReplayedJournalId | LastStartTime       | LastHeartbeat       | IsHelper | ErrMsg | Version
| CurrentConnected |
+----------------------------------------+-----------+-------------+----------+-----------+---------+--------------------+---------
-+----------+-----------+------+-------+------------------+--------------------+--------------------+----------+--------+--------
--------------------+-----------------+
| fe_65a0c6f0_b31f_42ac_bd20_26d851299f1a | 127.0.0.1 | 9010        | 8030     | 9030      | 9020    | 10030              | FOLLOWER
| true     | 840241689 | true | true  | 302891            | 2025-01-20 02:11:39 | 2025-01-21 09:48:36 | true     |        | doris-
2.1.7-rc03-443e87e203 | Yes              |
+----------------------------------------+-----------+-------------+----------+-----------+---------+--------------------+---------
-+----------+-----------+------+-------+------------------+--------------------+--------------------+----------+--------+--------
--------------------+-----------------+
```

SHOW FRONTENDS DISKS

Description

This statement is used to view the disk information of important directories on the FE node, such as metadata, logs, audit logs, and temporary directories.

Syntax

```
SHOW FRONTENDS DISKS;
```

Return Value

Column	Note
Name	The name of the FE node in bdbje
Host	The IP address of the FE node.
DirType	The types of directories to be displayed include the following four categories: meta, log, audit-log, temp, and deploy.
Dir	The directories for the types of directories to be displayed.
FileSystem	The file system in the Linux system where the types of directories to be displayed are located.
Capacity	The capacity of the file system.
Used	The used size of the file system.
Available	The remaining capacity of the file system.
UseRate	The usage percentage of the file system capacity.
MountOn	The mount directory of the file system.

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV or NODE_PRIV		

Usage Notes

If further filtering of the query results is required, the table-valued function frontends_disks() can be used. SHOW BACKENDS is equivalent to the following statement:

```
SELECT * FROM FRONTENDS_DISKS();
```

Examples

```
SHOW FRONTENDS DISKS;
```

```
+----------------------------------------+-------------+-----------+--------------------------------+------------+----------+-----
-+-----------+---------+------------+
| Name                                   | Host        | DirType   | Dir                            | Filesystem | Capacity | Used
| Available | UseRate | MountOn    |
+----------------------------------------+-------------+-----------+--------------------------------+------------+----------+-----
-+-----------+---------+------------+
| fe_a1daac68_5ec0_477c_b5e8_f90a33cdc1bb | 10.xx.xx.90 | meta      | /home/disk/output/fe/doris-meta | /dev/sdf1  | 7T       | 2T
| 4T        | 36%     | /home/disk |
| fe_a1daac68_5ec0_477c_b5e8_f90a33cdc1bb | 10.xx.xx.90 | log       | /home/disk/output/fe/log        | /dev/sdf1  | 7T       | 2T
| 4T        | 36%     | /home/disk |
| fe_a1daac68_5ec0_477c_b5e8_f90a33cdc1bb | 10.xx.xx.90 | audit-log | /home/disk/output/fe/log        | /dev/sdf1  | 7T       | 2T
| 4T        | 36%     | /home/disk |
| fe_a1daac68_5ec0_477c_b5e8_f90a33cdc1bb | 10.xx.xx.90 | temp      | /home/disk/output/fe/temp_dir   | /dev/sdf1  | 7T       | 2T
| 4T        | 36%     | /home/disk |
| fe_a1daac68_5ec0_477c_b5e8_f90a33cdc1bb | 10.xx.xx.90 | deploy    | /home/disk/output/fe            | /dev/sdf1  | 7T       | 2T
| 4T        | 36%     | /home/disk |
+----------------------------------------+-------------+-----------+--------------------------------+------------+----------+-----
-+-----------+---------+------------+
```

ADD BACKEND

Description

The ADD BACKEND is used to add one or more BE nodes to the Doris cluster. This command allows administrators to specify the host and port of the new BE nodes, as well as optional properties to configure their behavior.

Syntax

```
ALTER SYSTEM ADD BACKEND "<host>:<heartbeat_port>"[,"<host>:<heartbeat_port>" [, ...]] [PROPERTIES ("<key>"="<value>" [, ...] )]
```

Required Parameters

1.

> It can be the hostname or IP address of the BE node.

2.

> The heartbeat port of the BE node, the default is 9050.

Optional Parameters

1. `PROPERTIES ("<key>"="<value>" [, ... ] )`

> A set of key-value pairs used to define additional properties of the BE node. These properties can be used to customize the configuration of the BE being added. Available properties include: - `tag.location`: Used to specify the Resource Group to which the BE node belongs.

Access Control Requirements

The user executing this SQL must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. Before adding a new BE node, make sure the node is correctly configured and running.
2. Using Resource Group can help you better manage and organize the BE nodes in the cluster.
3. When adding multiple BE nodes, you can specify them in one command to improve efficiency.
4. After adding the BE nodes, use the `SHOW BACKENDS` to verify whether they have been successfully added and are in a normal state.
5. Consider adding BE nodes in different physical locations or racks to improve the availability and fault tolerance of the cluster.
6. Regularly check and balance the load in the cluster to ensure that the newly added BE nodes are properly utilized.

Examples

1. Add BE nodes without additional properties

   ```
   ALTER SYSTEM ADD BACKEND "192.168.0.1:9050,192.168.0.2:9050";
   ```

   This command adds two BE nodes to the cluster:

   - 192.168.0.1, port 9050
   - 192.168.0.2, port 9050 No additional properties are specified, so the default settings will be applied.

2. Add a BE node to a specified Resource Group

   ```
   ALTER SYSTEM ADD BACKEND "doris-be01:9050" PROPERTIES ("tag.location" = "groupb");
   ```

   This command adds a single BE node (hostname doris-be01, port 9050) to the Resource Group `groupb` in the cluster.

DROP BACKEND

Description

This statement is used to remove BE nodes from the Doris cluster.

Syntax

```
ALTER SYSTEM DROP BACKEND "<be_identifier>" [, "<be_identifier>" ... ]
```

Where:

```
be_identifier
  : "<be_host>:<be_heartbeat_port>"
  | "<backend_id>"
```

Required Parameters

1.

> It can be the hostname or IP address of the BE node.

2.

> The heartbeat port of the BE node, the default is 9050.

3.

> The ID of the BE node.

> `<be_host>`**,** `<be_heartbeat_port>`**, and** `<backend_id>` **can all be obtained by querying with the SHOW BACKENDS statement.**

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. It is not recommended to use this command to take a BE node offline. This command will directly remove the BE node from the cluster. The data on the current node will not be load-balanced to other BE nodes. Data loss may occur if there are single-replica tables in the cluster. A better approach is to use the DECOMMISSION BACKEND command to gracefully take the BE node offline.

2. Since this operation is a high-risk operation, when you directly run this command:

   ```
   ALTER SYSTEM DROP BACKEND "127.0.0.1:9050";
   ```

   ```
   ERROR 1105 (HY000): errCode = 2, detailMessage = It is highly NOT RECOMMENDED to use DROP BACKEND stmt.It is not safe to
   directly drop a backend. All data on this backend will be discarded permanently. If you insist, use DROPP instead of DROP
   ```

   The above prompt message will appear. If you understand what you are doing, you can replace the `DROP` keyword with `DROPP` and continue:

   ```
   ALTER SYSTEM DROPP BACKEND "127.0.0.1:9050";
   ```

Examples

1. Remove two nodes from the cluster based on the Host and HeartbeatPort of the BE nodes:

   ```
   ALTER SYSTEM DROPP BACKEND "192.168.0.1:9050", "192.168.0.2:9050";
   ```

2. Remove one node from the cluster based on the ID of the BE node: `sql ALTER SYSTEM DROPP BACKEND "10002";`

MODIFY BACKEND

Description

This statement is used to modify the attributes of BE nodes. After modifying the attributes of BE nodes, it will affect the query, write, and data distribution of the current node. The following are the attributes that can be modified:

Property	Effect
`tag.location`	The resource tag of the BE, with the `default` value being default. After modification, it will affect the data balancing of BEs within the same tag group and the BE nodes for data distribution during table creation. For more information, please refer toResource Group
`disable_query`	Whether to disable queries, with the default value being `false`. After setting it to `true`, no new query requests will be scheduled to this BE node.
`disable_load`	Whether to disable load, with the default value being `false`. After setting it to `true`, no new import requests will be scheduled to this BE node.

Syntax

```
ALTER SYSTEM MODIFY BACKEND <be_identifier> [, <be_identifier> ... ]
SET (
     "<key>" = "<value>"
)
```

Where:

```
be_identifier
  : "<be_host>:<be_heartbeat_port>"
  | "<backend_id>"
```

Required Parameters

1.

> It can be the hostname or IP address of the BE node.

2.

> The heartbeat port of the BE node, the default is 9050.

3.

> The ID of the BE node.

> **<be_host>, <be_heartbeat_port>, and <backend_id> can all be obtained by querying with the SHOW BACKENDS statement.**

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

Since this operation is at the entire BE level and has a wide impact, it may affect the normal querying, loading, and even table creation operations of the entire cluster if not performed carefully. Please operate with caution.

Examples

1. Modify the resource tag of the BE

```
ALTER SYSTEM MODIFY BACKEND "127.0.0.1:9050" SET ("tag.location" = "group_a");
```

2. Modify the `disable_query` attribute of the BE

```
ALTER SYSTEM MODIFY BACKEND "10002" SET ("disable_query" = "true");
```

3. Modify the `disable_load` attribute of the BE

```
ALTER SYSTEM MODIFY BACKEND "127.0.0.1:9050" SET ("disable_load" = "true");
```

SHOW BACKENDS

Description

This statement is used to view the basic status information of BE nodes.

Syntax

```
SHOW BACKENDS
```

Return Value

Column	Note
BackendId	The ID of the current BE.
Host	The IP address or host name of the current BE.
HeartbeatPort	The communication port of the heartbeat service of the current BE.
BePort	The thrift RPC communication port of the current BE.
HttpPort	The HTTP communication port of the current BE.
BrpcPort	The bRPC communication port of the current BE.
ArrowFlightSqlPort	The communication port of the ArrowFlight protocol of the current BE.
LastStartTime	The timestamp when the current BE started.
LastHeartbeat	The timestamp of the last successful heartbeat sent by the current BE.
Alive	Whether the current BE is alive.
SystemDecommissioned	When this value is true, it means that the current BE node is in the process of safe decommissioning.
TabletNum	The number of tablets stored on the current BE.
DataUsedCapacity	The disk space occupied by the data of the current BE.

Column	Note
TrashUsedCapacity	The disk space occupied by the data in the trash of the current BE.
AvailCapacity	The available disk space of the current BE.
TotalCapacity	The total disk space of the current BE. TotalCapacity = AvailCapacity + TrashUsedCapacity + DataUsedCapacity + Disk space occupied by other non-user data files.
UsedPct	The percentage of the total used disk space of the current BE.
MaxDiskUsedPct	The maximum percentage of the used disk space among all disks of the current BE.
RemoteUsedCapacity	The disk space occupied by the data uploaded to the remote storage after the hot and cold tiering function is used by the current BE.
Tag	The tag information of the current BE, displayed in JSON format. The name of the current BE resource group is saved.
ErrMsg	The error message when the heartbeat of the current BE fails.
Version	The version information of the current BE.
Status	Some status information of the current BE, displayed in JSON format, including: lastSuccessReportTabletsTime, lastStreamLoadTime, isQueryDisabled, isLoadDisabled, etc. It should be noted that the information saved in different versions may vary slightly.
HeartbeatFailureCounter	The number of consecutive failed heartbeats of the current BE. If the number exceeds the `max_backend_heartbeat_failure_tolerance_count` configured by the FE Master (the default value is 1), the `Alive` field will be set to false.
NodeRole	The role of the current BE. There are two types: `mix` is the default role, and `computation` means that the current node is only used for federated analysis queries.

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV		

Usage Notes

If further filtering of the query results is required, the table-valued function backends() can be used. SHOW BACKENDS is equivalent to the following statement:

```
SELECT * FROM BACKENDS();
```

Examples

```
SHOW BACKENDS;
```

```
+-----------+-----------+---------------+--------+----------+----------+-------------------+---------------------+-----------------
----+-------+----------------------+-----------+-----------------+------------------+--------------+--------------+---------+---
-------------+--------------------+-------------------------+--------+-----------------------------+-----------------------------
-----------------------------------------------------------------------------------------------------------+-------------------------
-+----------+
| BackendId | Host      | HeartbeatPort | BePort | HttpPort | BrpcPort | ArrowFlightSqlPort | LastStartTime       | LastHeartbeat
| Alive | SystemDecommissioned | TabletNum | DataUsedCapacity | TrashUsedCapacity | AvailCapacity | TotalCapacity | UsedPct |
MaxDiskUsedPct | RemoteUsedCapacity | Tag                     | ErrMsg | Version                     | Status
| HeartbeatFailureCounter | NodeRole |
+-----------+-----------+---------------+--------+----------+----------+-------------------+---------------------+-----------------
----+-------+----------------------+-----------+-----------------+------------------+--------------+--------------+---------+---
-------------+--------------------+-------------------------+--------+-----------------------------+-----------------------------
-----------------------------------------------------------------------------------------------------------+-------------------------
-+----------+
| 10002     | 127.0.0.1 | 9050          | 9060   | 8040     | 8060     | 10040              | 2025-01-20 02:11:39 | 2025-01-21
11:52:40 | true  | false                | 281       | 9.690 MB         | 0.000             | 10.505 GB     | 71.750 GB     | 85.36 %
| 85.36 %        | 0.000              | {"location" : "default"} |        | doris-2.1.7-rc03-443e87e203 |
{"lastSuccessReportTabletsTime":"2025-01-21
11:51:59","lastStreamLoadTime":1737460114345,"isQueryDisabled":false,"isLoadDisabled":false} | 0                       | mix      |
+-----------+-----------+---------------+--------+----------+----------+-------------------+---------------------+-----------------
----+-------+----------------------+-----------+-----------------+------------------+--------------+--------------+---------+---
-------------+--------------------+-------------------------+--------+-----------------------------+-----------------------------
-----------------------------------------------------------------------------------------------------------+-------------------------
-+----------+
```

DECOMMISSION BACKEND

Description

This statement is used to safely decommission a BE node from the cluster. This operation is asynchronous.

Syntax

```
ALTER SYSTEM DECOMMISSION BACKEND "<be_identifier>" [, "<be_identifier>" ... ]
```

Where:

```
be_identifier
  : "<be_host>:<be_heartbeat_port>"
  | "<backend_id>"
```

Required Parameters

1.

> It can be the hostname or IP address of the BE node.

2.

> The heartbeat port of the BE node, the default is 9050.

3.

> The ID of the BE node.

> `<be_host>`**,** `<be_heartbeat_port>`**, and** `<backend_id>` **can all be obtained by querying with the SHOW BACKENDS statement.**

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. After executing this command, you can use the SHOW BACKENDS statement to view the decommissioning status (the value of the `SystemDecommissioned` column is `true`) and the decommissioning progress (the value of the `TabletNum` column will slowly drop to 0).
2. Under normal circumstances, after the value of the `TabletNum` column drops to 0, this BE node will be deleted. If you do not want Doris to automatically delete the BE, you can change the configuration `drop_backend_after_decommission` of the FE Master to false.
3. If the current BE stores a relatively large amount of data, the DECOMMISSION operation may last for several hours or even days.
4. If the progress of the DECOMMISSION operation gets stuck, specifically, the `TabletNum` column in the SHOW BACKENDS statement remains fixed at a certain value, it may be due to the following situations:
    - There is no suitable other BE to migrate the tablets on the current BE. For example, in a 3-node cluster with a table having 3 replicas, if one of the nodes is to be decommissioned, this node cannot find other BEs to migrate the data (the other two BEs already have one replica each).
    - The tablets on the current BE are still in the Recycle Bin. You can empty the recycle bin and then wait for decommission.
    - The tablet on the current BE is too large, causing the migration of a single tablet to always timeout and unable to migrate this tablet away. You can adjust the configuration `max_clone_task_timeout_sec` of the FE Master to a larger value (the default is 7200 seconds).
    - There are unfinished transactions on the tablets of the current BE. You can wait for the transactions to complete or manually abort the transactions.
    - In other cases, you can filter the keyword `replicas to decommission` in the logs of the FE Master to find the abnormal tablet, use the SHOW TABLET statement to find the table to which this tablet belongs, then create a new table, migrate the data from the old table to the new table, and finally use the DROP TABLE FORCE to delete the old table.

Examples

1. Safely decommission two nodes from the cluster according to the Host and HeartbeatPort of the BE.

```
ALTER SYSTEM DECOMMISSION BACKEND "192.168.0.1:9050", "192.168.0.2:9050";
```

2. Safely decommission a node from the cluster according to the ID of the BE. `sql ALTER SYSTEM DECOMMISSION BACKEND "10002";`

CANCEL DECOMMISSION BACKEND

Description

This statement is used to cancel the decommissioning operation of a BE node.

Syntax

```
CANCEL DECOMMISSION BACKEND "<be_identifier>" [, "<be_identifier>" ... ]
```

Where:

```
be_identifier
  : "<be_host>:<be_heartbeat_port>"
  | "<backend_id>"
```

Required Parameters

> It can be the hostname or IP address of the BE node.

> The heartbeat port of the BE node, the default is 9050.

> The ID of the BE node.

> **`<be_host>`, `<be_heartbeat_port>`, and `<backend_id>` can all be obtained by querying with the SHOW BACKENDS statement.**

Access Control Requirements

The user who executes this SQL must have at least the following permissions:

Privilege	Object	Notes
NODE_PRIV		

Usage Notes

1. After executing this command, you can view the decommissioning status (the value of the `SystemDecommissioned` column is false) and the decommissioning progress (the value of the `TabletNum` column no longer decreases slowly) through the SHOW BACKENDS statement.
2. The cluster will slowly migrate the tablets from other nodes back to the current BE, so that the number of tablets on each BE will eventually tend to approach.

Examples

1. Safely decommission two nodes from the cluster according to the Host and HeartbeatPort of the BE.

```
CANCEL DECOMMISSION BACKEND "192.168.0.1:9050", "192.168.0.2:9050";
```

2. Safely decommission one node from the cluster according to the ID of the BE.

```
CANCEL DECOMMISSION BACKEND "10002";
```

ADD BROKER

Description

This statement is used to add one or more BROKER nodes.

Syntax

```
ALTER SYSTEM ADD BROKER <broker_name> "<host>:<ipc_port>" [, "<host>:<ipc_port>" [, ... ] ];
```

Required Parameters

1. `<broker_name>`

The name given to the added broker process. It is recommended to keep the broker_name consistent within the same cluster.

2. `<host>`

The IP of the node where the broker process needs to be added. If FQDN is enabled, use the FQDN of the node.

3. `<ipc_port>`

The PORT of the node where the broker process needs to be added, and the default value of this port is 8000.

Access Control Requirements

The user who executes this operation needs to have the NODE_PRIV permission.

Examples

1. Add two Brokers

   ```
   ALTER SYSTEM ADD BROKER broker1 "host1:port", "host2:port";
   ```

2. Add a Broker using FQDN

   ```
   ALTER SYSTEM ADD BROKER broker1 "broker_fqdn1:port";
   ```

DROP BROKER

Description

This statement is used to delete BROKER nodes.

Syntax

1. Drop all Brokers `sql ALTER SYSTEM DROP ALL BROKER broker_name;`
2. Drop one or more Broker nodes `sql ALTER SYSTEM DROP BROKER <broker_name> "<host>:<ipc_port>"[, "<host>:<ipc_port>" [, ...] ];` ### Required Parameters

1. `<broker_name>`

The name of the broker process to be deleted.

2. `<host>`

The IP of the node where the broker process to be deleted is located. If FQDN is enabled, use the FQDN of the node.

3. `<ipc_port>` The PORT of the node where the broker process to be deleted is located, and the default value of this port is 8000.

Access Control Requirements

The user who executes this operation needs to have the NODE_PRIV permission.

Examples

1. Delete all Brokers

```
ALTER SYSTEM DROP ALL BROKER broker_name;
```

2. Delete a specific Broker node

```
ALTER SYSTEM DROP BROKER broker_name "10.10.10.1:8000";
```

SHOW BROKER

Description

This statement is used to view the status of the currently existing broker processes.

Syntax

```
SHOW BROKER;
```

Output

Column	DateType	Note
Name	varchar	Broker Process Name
Host	varchar	Broker Process Node IP
Port	varchar	Broker Process Node Port
Alive	varchar	Broker Process Node Status
LastStartTime	varchar	Broker Process Last Start Time
LastUpdateTime	varchar	Broker Process Last Update Time
ErrMsg	varchar	Error message of the last failed startup of the Broker process

Access Control Requirements

The user executing this statement needs to have the `ADMIN/OPERATOR` permission.

Examples

1. View the status of the currently existing broker processes `sql show broker; text +-------------+---------+------+-------+--------------------+--------------------+--------+ | Name | Host | Port | Alive | LastStartTime | LastUpdateTime | ErrMsg | +-------------+-----------+------+-------+--------------------+--------------------+--------+ | broker_test | 10.10.10.1 | 8196 | true | 2025-01-21 11:30:10 | 2025-01-21 11:31:40 | | +-------------+-----------+------+-------+--------------------+--------------------+--------+`

CREATE RESOURCE

Description

This statement is used to create a resource. Only the root or admin user can create resources. Currently supports Spark, ODBC, S3 external resources. In the future, other external resources may be added to Doris for use, such as Spark/GPU for query, HDFS/S3 for external storage, MapReduce for ETL, etc.

Syntax

```
CREATE [EXTERNAL] RESOURCE "<resource_name>"
PROPERTIES (
   `<property>`
    [ , ... ]
);
```

Parameters

1.<property>

<property> format as <key> = <value>, and the specific available values for <key> are as follows:

Parameter	Description	Required
<type>	Specify the type of resource, supported: spark/odbc_catalog/s3/jdbc/hdfs/hms/es.	Y

Depending on the <type>, the parameters of PROPERTIES vary. Please refer to the example for details.

Examples

1. Create a Spark resource named spark0 in yarn cluster mode.

```
CREATE EXTERNAL RESOURCE "spark0"
PROPERTIES
(
  "type" = "spark",
  "spark.master" = "yarn",
  "spark.submit.deployMode" = "cluster",
  "spark.jars" = "xxx.jar,yyy.jar",
  "spark.files" = "/tmp/aaa,/tmp/bbb",
  "spark.executor.memory" = "1g",
  "spark.yarn.queue" = "queue0",
  "spark.hadoop.yarn.resourcemanager.address" = "127.0.0.1:9999",
  "spark.hadoop.fs.defaultFS" = "hdfs://127.0.0.1:10000",
  "working_dir" = "hdfs://127.0.0.1:10000/tmp/doris",
  "broker" = "broker0",
  "broker.username" = "user0",
  "broker.password" = "password0"
);
```

Spark related parameters are as follows: - spark.master: Required, currently supports yarn, spark://host:port. - spark.submit.deployMode: The deployment mode of the Spark program, required, supports both cluster and client. - spark.hadoop.yarn.resourcemanager.address: Required when master is yarn. - spark.hadoop.fs.defaultFS: Required when master is yarn. - Other parameters are optional, refer to here

Working_dir and broker need to be specified when Spark is used for ETL. described as follows:

- working_dir: The directory used by the ETL. Required when spark is used as an ETL resource. For example: hdfs://host:port/tmp/doris.
- broker: broker name. Required when spark is used as an ETL resource. Configuration needs to be done in advance using the `ALTER SYSTEM ADD BROKER` command.
- broker.property_key: The authentication information that the broker needs to specify when reading the intermediate file generated by ETL.

2. Create an ODBC resource

```
CREATE EXTERNAL RESOURCE `oracle_odbc`
PROPERTIES (
"type" = "odbc_catalog",
"host" = "192.168.0.1",
"port" = "8086",
"user" = "test",
"password" = "test",
"database" = "test",
"odbc_type" = "oracle",
"driver" = "Oracle 19 ODBC driver"
);
```

The relevant parameters of ODBC are as follows: - hosts: IP address of the external database - driver: The driver name of the ODBC appearance, which must be the same as the Driver name in be/conf/odbcinst.ini. - odbc_type: the type of the external database, currently supports oracle, mysql, postgresql - user: username of the foreign database - password: the password information of the corresponding user - charset: connection charset - There is also support for implementing custom parameters per ODBC Driver, see the description of the corresponding ODBC Driver

3. Create S3 resource

```
CREATE RESOURCE "remote_s3"
PROPERTIES
(
   "type" = "s3",
   "s3.endpoint" = "bj.s3.com",
   "s3.region" = "bj",
   "s3.access_key" = "bbb",
   "s3.secret_key" = "aaaa",
   -- the followings are optional
   "s3.connection.maximum" = "50",
   "s3.connection.request.timeout" = "3000",
   "s3.connection.timeout" = "1000"
);
```

If S3 resource is used for cold hot separation, we should add more required fields. `sql CREATE RESOURCE "remote_s3" PROPERTIES ( "type" = "s3", "s3.endpoint" = "bj.s3.com", "s3.region" = "bj", "s3.access_key" =`

"bbb", "s3.secret_key" = "aaaa", -- required by cooldown "s3.root.path" = "/path/to/root", "s3.bucket" = "test-bucket");

S3 related parameters are as follows: - Required parameters - `s3.endpoint`: s3 endpoint - `s3.region`:s3 region - `s3.root.path`: s3 root directory - `s3.access_key`: s3 access key - `s3.secret_key`: s3 secret key - `s3.bucket`：s3 bucket - optional parameter - `s3.connection.maximum`: the maximum number of s3 connections, the default is 50 - `s3.connection.request.timeout`: s3 request timeout, in milliseconds, the default is 3000 - `s3.connection.timeout`: s3 connection timeout, in milliseconds, the default is 1000

4. Create JDBC resource

```
CREATE RESOURCE mysql_resource PROPERTIES (
   "type"="jdbc",
   "user"="root",
   "password"="123456",
   "jdbc_url" = "jdbc:mysql://127.0.0.1:3316/doris_test?useSSL=false",
   "driver_url" = "https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.25/mysql-connector-java-8.0.25.jar",
"driver_class" = "com.mysql.cj.jdbc.Driver"
);
```

JDBC related parameters are as follows: - user：The username used to connect to the database - password：The password used to connect to the database - jdbc_url: The identifier used to connect to the specified database - driver_url: The url of JDBC driver package - driver_class: The class of JDBC driver

5. Create HDFS resource

```
CREATE RESOURCE hdfs_resource PROPERTIES (
   "type"="hdfs",
   "hadoop.username"="user",
   "dfs.nameservices" = "my_ha",
   "dfs.ha.namenodes.my_ha" = "my_namenode1, my_namenode2",
   "dfs.namenode.rpc-address.my_ha.my_namenode1" = "nn1_host:rpc_port",
   "dfs.namenode.rpc-address.my_ha.my_namenode2" = "nn2_host:rpc_port",
   "dfs.client.failover.proxy.provider.my_ha" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
);
```

HDFS related parameters are as follows: - fs.defaultFS: namenode address and port - hadoop.username: hdfs username - dfs.nameservices: if hadoop enable HA, please set fs nameservice. See hdfs-site.xml - dfs.ha.namenodes.[nameservice ID]：unique identifiers for each NameNode in the nameservice. See hdfs-site.xml - dfs.namenode.rpc-address.[nameservice ID].[name node ID]`：the fully-qualified RPC address for each NameNode to listen on. See hdfs-site.xml - dfs.client.failover.proxy.provider.[nameservice ID]：the Java class that HDFS clients use to contact the Active NameNode, usually it is org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider

6. Create HMS resource

HMS resource is used to create hms catalog `sql CREATE RESOURCE hms_resource PROPERTIES ('type'='hms', 'hive.metastore.uris' = 'thrift://127.0.0.1:7004', 'dfs.nameservices'='HANN', 'dfs.ha.namenodes.HANN'='nn1,nn2', 'dfs.namenode.rpc-address.HANN.nn1'='nn1_host:rpc_port', 'dfs.namenode.rpc-address.HANN.nn2'='nn2_host:rpc_port', 'dfs.client.failover.proxy.provider.HANN'='org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider');`

HMS related parameters are as follows: - hive.metastore.uris: hive metastore server address Optional: - dfs.*: If hive data is on hdfs, HDFS resource parameters should be added, or copy hive-site.xml into fe/conf. - s3.*: If hive data is on s3, S3 resource parameters should be added. If using Aliyun Data Lake Formation, copy hive-site.xml into fe/conf.

7. Create ES resource

```
CREATE RESOURCE es_resource PROPERTIES (
   "type"="es",
   "hosts"="http://127.0.0.1:29200",
   "nodes_discovery"="false",
   "enable_keyword_sniff"="true"
);
```

ES related parameters are as follows: - hosts: ES Connection Address, maybe one or more node, load-balance is also accepted - user: username for ES - password: password for the user - enable_docvalue_scan: whether to enable ES/Lucene column storage to get the value of the query field, the default is true - enable_keyword_sniff: Whether to probe the string segmentation type text.fields in ES, query by keyword (the default is true, false matches the content after the segmentation) - nodes_discovery: Whether or not to enable ES node discovery, the default is true. In network isolation, set this parameter to false. Only the specified node is connected - http_ssl_enabled: Whether ES cluster enables https access mode, the current FE/BE implementation is to trust all

ALTER RESOURCE

Description

This statement is used to modify an existing resource. Only the root or admin user can modify resources.

Syntax

```
ALTER RESOURCE '<resource_name>'
PROPERTIES (
  `<property>`,
  [ , ... ]
);
```

Parameters

1.<property>

The <property> format is <key> = <value>, and modifying the <value> where <key> equals 'type' is not supported.

The modified properties parameters can be referenced in the CREATE-RESOURCE section.

Examples

1. Modify the working directory of the Spark resource named spark0:

```
ALTER RESOURCE 'spark0' PROPERTIES ("working_dir" = "hdfs://127.0.0.1:10000/tmp/doris_new");
```

2. Modify the maximum number of connections to the S3 resource named remote_s3:

```
ALTER RESOURCE 'remote_s3' PROPERTIES ("s3.connection.maximum" = "100");
```

3. Modify information related to cold and hot separation S3 resources

- Support
  - s3.access_key s3 ak
  - s3.secret_key s3 sk
  - s3.session_token s3 token
  - s3.connection.maximum default 50
  - s3.connection.timeout default 1000ms
  - s3.connection.request.timeout default 3000ms
- Not Support
  - s3.region
  - s3.bucket"
  - s3.root.path
  - s3.endpoint

```
ALTER RESOURCE "showPolicy_1_resource" PROPERTIES("s3.connection.maximum" = "1111");
```

DROP RESOURCE

Description

This statement is used to delete an existing resource. Only the root or admin user can delete resources.

Syntax

```
DROP RESOURCE '<resource_name>'
```

Usage Notes

ODBC/S3 resources in use cannot be deleted.

Examples

1. Delete the Spark resource named spark0:

   ```
   DROP RESOURCE 'spark0';
   ```

SHOW RESOURCES

Description

This statement is used to display resources that the user has permission to use. Ordinary users can only display resources with permission, and root or admin users will display all resources.

Syntax

```
SHOW RESOURCES
[
  WHERE
  [NAME [ = "<your_resource_name>" | LIKE "<name_matcher>"]]
  [RESOURCETYPE = "<type>"]
] | [LIKE "<pattern>"]
[ORDER BY ...]
[LIMIT <limit>][OFFSET <offset>];
```

Usage Notes

1. If NAME LIKE is used, it will match Resource whose Name contains name_matcher in RESOURCES
2. If NAME = is used, it will match the specified Name exactly
3. If RESOURCETYPE is specified, match the corresponding Resrouce type, The supported RESOURCETYPEs can be referred to in CREATE-RESOURCE;
4. You can use ORDER BY to sort on any combination of columns
5. If LIMIT is specified, limit matching records are displayed. Otherwise show all
6. If OFFSET is specified, the query results are displayed starting at offset offset. By default the offset is 0.
7. If using LIKE, the WHERE clause will be ignored.

Example

1. Display all resources that the current user has permissions to

   ```
   SHOW RESOURCES;
   ```

2. Display the specified Resource, the name contains the string “20140102”, and display 10 attributes

   ```
   SHOW RESOURCES WHERE NAME LIKE "2014_01_02" LIMIT 10;
   ```

3. Display the specified Resource, specify the name as “20140102” and sort by KEY in descending order

   ```
   SHOW RESOURCES WHERE NAME = "20140102" ORDER BY `KEY` DESC;
   ```

4. Using LIKE to match the resource

```
SHOW RESOURCES LIKE "jdbc%";
```

CREATE WORKLOAD GROUP

Description

This statement is used to create a workload group. Workload groups enable the isolation of cpu resources and memory resources on a single be.

Syntax

```
CREATE WORKLOAD GROUP [IF NOT EXISTS] "rg_name"
PROPERTIES (
    `<property>`
    [ , ... ]
);
```

Parameters

1.<property>

<property> format as <key> = <value>, and the specific available values for <key> are as follows:

Parameter	Description	Required
<cpu_share>	used to set how much cpu time the workload group can acquire, which can achieve soft isolation of cpu resources. cpu_share is a relative value indicating the weight of cpu resources available to the running workload group. For example, if a user creates 3 workload groups rg-a, rg-b and rg-c with cpu_share of 10, 30 and 40 respectively, and at a certain moment rg-a and rg-b are running tasks while rg-c has no tasks, then rg-a can get (10 / (10 + 30)) = 25% of the cpu resources while workload group rg-b can get 75% of the cpu resources. If the system has only one workload group running, it gets all the cpu resources regardless of the value of its cpu_share.	Y

Parameter	Description	Required
<memory_limit>	set the percentage of be memory that can be used by the workload group. The absolute value of the workload group memory limit is: physical_memory * mem_limit * memory_limit, where mem_limit is a be configuration item. The total memory_limit of all workload groups in the system must not exceed 100%. Workload groups are guaranteed to use the memory_limit for the tasks in the group in most cases. When the workload group memory usage exceeds this limit, tasks in the group with larger memory usage may be canceled to release the excess memory, refer to enable_memory_overcommit.	Y
<enable_memory_overcommit>	enable soft memory isolation for the workload group, default is false. if set to false, the workload group is hard memory isolated and the tasks with the largest memory usage will be canceled immediately after the workload group memory usage exceeds the limit to release the excess memory. if set to true, the workload group is hard memory isolated and the tasks with the largest memory usage will be canceled immediately after the workload group memory usage exceeds the limit to release the excess memory. if set to true, the workload group is softly isolated, if the system has free memory resources, the workload group can continue to use system memory after exceeding the memory_limit limit, and when the total system memory is tight, it will cancel several tasks in the group with the largest memory occupation, releasing part of the excess memory to relieve the system memory pressure. It is recommended that when this configuration is enabled for a workload group, the total memory_limit of all workload groups should be less than 100%, and the remaining portion should be used for workload group memory overcommit.	Y

Examples

1. Create a workload group named g1:

```
create workload group if not exists g1
properties (
    "cpu_share"="10",
    "memory_limit"="30%",
```

```
    "enable_memory_overcommit"="true"
);
```

ALTER WORKLOAD GROUP

Description

This statement is used to modify the workload group.

Syntax

```
ALTER WORKLOAD GROUP  "<rg_name>"
PROPERTIES (
  `<property>`
  [ , ... ]
);
```

Parameters

1.<property>

<property> 格式为 <key> = <value>，<key>的具体可选值如下：

Parameter	Description	Required
<cpu_share>	used to set how much cpu time the workload group can acquire, which can achieve soft isolation of cpu resources. cpu_share is a relative value indicating the weight of cpu resources available to the running workload group. For example, if a user creates 3 workload groups rg-a, rg-b and rg-c with cpu_share of 10, 30 and 40 respectively, and at a certain moment rg-a and rg-b are running tasks while rg-c has no tasks, then rg-a can get (10 / (10 + 30)) = 25% of the cpu resources while workload group rg-b can get 75% of the cpu resources. If the system has only one workload group running, it gets all the cpu resources regardless of the value of its cpu_share.	Y

Parameter	Description	Required
<memory_limit>	set the percentage of be memory that can be used by the workload group. The absolute value of the workload group memory limit is: physical_memory * mem_limit * memory_limit, where mem_limit is a be configuration item. The total memory_limit of all workload groups in the system must not exceed 100%. Workload groups are guaranteed to use the memory_limit for the tasks in the group in most cases. When the workload group memory usage exceeds this limit, tasks in the group with larger memory usage may be canceled to release the excess memory, refer to enable_memory_overcommit.	Y
<enable_memory_overcommit>	enable soft memory isolation for the workload group, default is false. if set to false, the workload group is hard memory isolated and the tasks with the largest memory usage will be canceled immediately after the workload group memory usage exceeds the limit to release the excess memory. if set to true, the workload group is hard memory isolated and the tasks with the largest memory usage will be canceled immediately after the workload group memory usage exceeds the limit to release the excess memory. if set to true, the workload group is softly isolated, if the system has free memory resources, the workload group can continue to use system memory after exceeding the memory_limit limit, and when the total system memory is tight, it will cancel several tasks in the group with the largest memory occupation, releasing part of the excess memory to relieve the system memory pressure. It is recommended that when this configuration is enabled for a workload group, the total memory_limit of all workload groups should be less than 100%, and the remaining portion should be used for workload group memory overcommit.	Y

Examples

1. Modify the workload group named g1:

```
alter workload group g1
properties (
    "cpu_share"="30",
    "memory_limit"="30%"
);
```

DROP WORKLOAD GROUP

Description

This statement is used to delete a workload group.

Syntax

```
DROP WORKLOAD GROUP [IF EXISTS] '<rg_name>'
```

Examples

1. Delete the workload group named g1:

   ```
   drop workload group if exists g1;
   ```

SHOW WORKLOAD GROUPS

Description

This statement is used to display the resource groups for which the current user has usage_priv privileges.

Syntax

```
SHOW WORKLOAD GROUPS [LIKE "<pattern>"];
```

Usage Notes

This statement only does a simple display of workload groups, for a more complex display refer to tvf workload_groups().

Examples

1. Show all workload groups:

```
mysql> show workload groups;
+----------+--------+-------------------------+---------+
| Id       | Name   | Item                    | Value   |
+----------+--------+-------------------------+---------+
| 10343386 | normal | cpu_share               | 10      |
| 10343386 | normal | memory_limit            | 30%     |
| 10343386 | normal | enable_memory_overcommit | true    |
| 10352416 | g1     | memory_limit            | 20%     |
| 10352416 | g1     | cpu_share               | 10      |
+----------+--------+-------------------------+---------+
```

2. Show workload groups using pattern

```
mysql> show workload groups like "normal%";
+----------+--------+-------------------------+---------+
| Id       | Name   | Item                    | Value   |
+----------+--------+-------------------------+---------+
| 10343386 | normal | cpu_share               | 10      |
| 10343386 | normal | memory_limit            | 30%     |
| 10343386 | normal | enable_memory_overcommit | true    |
+----------+--------+-------------------------+---------+
```

CREATE WORKLOAD POLICY

Description

Create a Workload Policy to execute corresponding actions on a query when it meets certain conditions.

Syntax

```
CREATE WORKLOAD POLICY [ IF NOT EXISTS ] <workload_policy_name>
CONDITIONS(<conditions>) ACTIONS(<actions>)
[ PROPERTIES (<properties>) ]
```

Required Parameters

`<workload_policy_name>`

The name of the Workload Policy.

1. **be_scan_rows**: The number of rows scanned by an SQL query within a single BE process. If the SQL query is executed concurrently on multiple BEs, it is the cumulative value of these concurrent executions.
2. **be_scan_bytes**: The number of bytes scanned by an SQL query within a single BE process. If the SQL query is executed concurrently on multiple BEs, it is the cumulative value of these concurrent executions (in bytes).
3. **query_time**: The execution time of an SQL query on a single BE process, in milliseconds.
4. **query_be_memory_bytes** (supported from version 2.1.5): The amount of memory used by an SQL query within a single BE process. If the SQL query is executed concurrently on multiple BEs, it is the cumulative value of these concurrent executions (in bytes).

`<actions>`

1. **set_session_variable**: This action executes a set session variable statement. Multiple **set_session_variable** actions can be specified in the same policy, allowing multiple session variable modification statements to be executed within one policy.
2. **cancel_query**: Cancels the query.

Optional Parameters

1. **enabled**: Takes a value of true or false, with a default value of true. When set to true, the policy is enabled; when set to false, the policy is disabled.
2. **priority**: An integer value ranging from 0 to 100, with a default value of 0. This represents the priority of the policy. The higher the value, the higher the priority. When multiple policies match, the policy with the highest priority is selected.
3. **workload_group**: Currently, a policy can be bound to one workload group, indicating that this policy only applies to a specific workload group. The default is empty, meaning it applies to all queries.

Access Control Requirements

Requires at least `ADMIN_PRIV` permissions.

Examples

1. Create a new Workload Policy to kill all queries that exceed 3 seconds in query time.

```
create workload policy kill_big_query conditions(query_time > 3000) actions(cancel_query)
```

1. Create a new Workload Policy that is not enabled by default.

```
create workload policy kill_big_query conditions(query_time > 3000) actions(cancel_query) properties('enabled'='false')
```

ALTER WORKLOAD POLICY

Description

Modify the properties of a Workload Policy. Currently, only property modifications are supported; modifications to actions and conditions are not supported.

Syntax

```
ALTER WORKLOAD POLICY <workload_policy_name> PROPERTIES( <properties> )
```

Required Parameters

`<workload_policy_name>`

Workload Policy's Name

`<properties>`

1. enabled: Can be true or false, with a default value of true, indicating that the current policy is enabled. false indicates that the current policy is disabled.
2. priority: A positive integer ranging from 0 to 100, with a default value of 0. This represents the priority of the policy. The higher the value, the higher the priority. The main role of this property is to select the policy with the highest priority when multiple policies match.
3. workload_group: Currently, a policy can be bound to one workload group, which means that this policy is only effective for a specific workload group. The default is empty, which means it is effective for all queries.

Access Control Requirements

You must have at least ADMIN_PRIV permissions.

Examples

1. Disable a Workload Policy

```
alter workload policy cancel_big_query properties('enabled'='false')
```

DROP WORKLOAD POLICY

Description

Delete a Workload Policy

Syntax

```
DROP WORKLOAD POLICY [ IF EXISTS ] <workload_policy_name>
```

Required Parameters

The name of the Workload Policy

Access Control Requirements

Must have at least `ADMIN_PRIV` permissions

Examples

1. Delete a Workload Policy named cancel_big_query

```
drop workload policy if exists cancel_big_query
```

CREATE STORAGE POLICY

Description

To create a storage policy, you must first create a storage resource, and then associate the created storage resource name when creating the migration policy. For details, refer to the RESOURCE section.

Syntax

```
CREATE STORAGE POLICY <policy_name>
PROPERTIES(
    "storage_resource" = "<storage_resource_name>"
    [{, "cooldown_datetime" = "<cooldown_datetime>"
    | , "cooldown_ttl" = "<cooldown_ttl>"}]
);
```

Required Parameters

> The name of the storage policy to be created

> The name of the associated storage resource. For details on how to create it, refer to the RESOURCE section

Optional Parameters

> Specifies the cooldown time for creating the data migration policy

> Specifies the duration of hot data for creating the data migration policy

Access Control Requirements

The prerequisite for successfully executing this SQL command is to have ADMIN_PRIV privileges. Refer to the permission documentation.

Privilege	Object	Notes
ADMIN_PRIV	Entire cluster management privileges	All privileges except NODE_PRIV

Example

1. Create a data migration policy with a specified data cooldown time.

```
CREATE STORAGE POLICY testPolicy
PROPERTIES(
  "storage_resource" = "s3",
  "cooldown_datetime" = "2022-06-08 00:00:00"
);
```

2. Create a data migration policy with a specified duration of hot data

```
CREATE STORAGE POLICY testPolicy
PROPERTIES(
  "storage_resource" = "s3",
  "cooldown_ttl" = "1d"
);
```

ALTER STORAGE POLICY

Description

This statement is used to modify an existing hot-cold tiered migration policy. Only root or admin users can modify resources.

Syntax

```
ALTER STORAGE POLICY  '<policy_name>' PROPERTIES ("<key>"="<value>"[, ... ]);
```

Required Parameters

1.`<policy_name>`
> The name of the storage policy. This is the unique identifier of the storage policy you want to modify, and an existing policy name must be specified.

Optional Parameters

`PROPERTIES ("<key>"="<value>"[, ... ])`

1.`retention_days`
> Data retention period. Defines the duration for which the data is kept in storage. Data exceeding this period will be automatically deleted.

2.`redundancy_level` > Redundancy level. Defines the number of data replicas to ensure high availability and fault tolerance. For example, a value of 2 means each data block has two replicas.

3.`storage_type`
> Storage type. Specifies the storage medium used, such as SSD, HDD, or hybrid storage. This affects performance and cost.

4.`cooloff_time`
> Cool-off time. The time interval between when data is marked for deletion and when it is actually deleted. This helps prevent data loss due to accidental operations.

5.`location_policy` > Geographical location policy. Defines the geographical placement of data, such as cross-region replication for disaster recovery.

Examples

1. Modify the cooldown_datetime for hot-cold tiered data migration:

```
ALTER STORAGE POLICY has_test_policy_to_alter PROPERTIES("cooldown_datetime" = "2023-06-08 00:00:00");
```

2. Modify the cooldown_ttl for hot-cold tiered data migration countdown:

```
ALTER STORAGE POLICY has_test_policy_to_alter PROPERTIES ("cooldown_ttl" = "10000");
```

```
ALTER STORAGE POLICY has_test_policy_to_alter PROPERTIES ("cooldown_ttl" = "1h");
```

```
ALTER STORAGE POLICY has_test_policy_to_alter PROPERTIES ("cooldown_ttl" = "3d");
```

DROP STORAGE POLICY

Description

Delete a storage policy. For detailed descriptions of storage policies, please refer to the “Storage Policy” chapter.

Syntax

```
DROP STORAGE POLICY <policy_name>
```

Required Parameters

> Storage policy name

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	Global	

Examples

1. Delete a storage policy named policy1

```
DROP STORAGE POLICY policy1
```

SHOW STORAGE POLICY

Description

View tables and partitions associated with all/specified storage policies.

Syntax

```
SHOW STORAGE POLICY [ USING [ FOR <storage_policy_name> ] ]
```

Required Parameters

> The name of the storage policy to view.

The prerequisite for successfully executing this SQL command is to have ADMIN_PRIV privileges. Please refer to the privilege document.

Privilege (Privilege)	Object (Object)	Notes (Notes)
ADMIN_PRIV	Entire cluster management privileges	All privileges except NODE_PRIV

Example

1. View all objects with enabled storage policies.

```
show storage policy using;
```

```
+----------------------+----------------------------------------+--------------------------------------+------------+
| PolicyName           | Database                               | Table                                | Partitions |
+----------------------+----------------------------------------+--------------------------------------+------------+
| test_storage_policy   | regression_test_cold_heat_separation_p2 | table_with_storage_policy_1           | ALL        |
| test_storage_policy   | regression_test_cold_heat_separation_p2 | partition_with_multiple_storage_policy | p201701    |
| test_storage_policy_2 | regression_test_cold_heat_separation_p2 | partition_with_multiple_storage_policy | p201702    |
| test_storage_policy_2 | regression_test_cold_heat_separation_p2 | table_with_storage_policy_2           | ALL        |
| test_policy          | db2                                    | db2_test_1                           | ALL        |
+----------------------+----------------------------------------+--------------------------------------+------------+
```

2. View objects using the storage policy test_storage_policy.

```
show storage policy using for test_storage_policy;
```

```
+--------------------+-----------+-------------------------------+------------+
| PolicyName         | Database  | Table                         | Partitions |
+--------------------+-----------+-------------------------------+------------+
| test_storage_policy | db_1      | partition_with_storage_policy_1 | p201701    |
```

```
| test_storage_policy | db_1      | table_with_storage_policy_1     | ALL        |
+---------------------+-----------+---------------------------------+------------+
```

3. View the properties of all storage policies.

```
show storage policy;
```

```
+-------------+----------+---------+---------+-----------------+------------------+-------------+
| PolicyName  | Id       | Version | Type    | StorageResource | CooldownDatetime | CooldownTtl |
+-------------+----------+---------+---------+-----------------+------------------+-------------+
| test_policy | 14589252 | 0       | STORAGE | remote_s3       | -1               | 300         |
| dev_policy  | 14589521 | 0       | STORAGE | remote_s3       | -1               | 3000        |
+-------------+----------+---------+---------+-----------------+------------------+-------------+
```

CREATE FILE

Description

This statement is used to create and upload a file to the Doris cluster. This function is usually used to manage files that need to be used in some other commands, such as certificates, public and private keys, and so on.

Syntax

```
CREATE FILE <file_name>
        [ { FROM | IN } <database_name>] PROPERTIES ("<key>"="<value>" [ , ... ]);
```

Required Parameters

1. `<file_name>`

> Custom file name.

2. `<key>`

> File attribute key. - **url**: Required. Specifies an unauthenticated HTTP download URL. After successful execution, the file will be stored in Doris and this URL will no longer be required. - **catalog**: Required. Category name for file classification (user-defined). Used to locate files in specific commands (e.g., searches for files under 'kafka' catalog when Kafka is the data source in scheduled imports). - **md5**: Optional. MD5 checksum of the file. If provided, verification will be performed after download.

3. `<value>`

> File attribute value.

Optional Parameters

1. `<database_name>`

> Specifies the database to which the file belongs. Uses current session's database if not specified.

Access Control Requirements

The user executing this SQL command must possess at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	User / Role	The user or role must hold the ADMIN_PRIV privilege to execute this operation

Usage Notes

- File Access Rules

> Each file belongs to a specific database (Database). Users with access privileges to the database can access all files within it.

- File Size and Quantity Limits

> This feature is primarily designed for managing small files such as certificates.
> **Size limit**: Individual file size is restricted to 1MB
> **Quantity limit**: A Doris cluster supports uploading up to 100 files maximum

Example

- Create a file ca.pem , classified as kafka

```
CREATE FILE "ca.pem"
PROPERTIES
(
    "url" = "https://test.bj.bcebos.com/kafka-key/ca.pem",
    "catalog" = "kafka"
);
```

- Create a file client.key, classified as my_catalog

```
CREATE FILE "client.key"
IN my_database
PROPERTIES
(
    "url" = "https://test.bj.bcebos.com/kafka-key/client.key",
    "catalog" = "my_catalog",
    "md5" = "b5bb901bf10f99205b39a46ac3557dd9"
);
```

- Create a file client_1.key, classified as my_catalog

```
CREATE FILE "client_1.key"
  FROM my_database
  PROPERTIES
```

```
(
   "url" = "https://test.bj.bcebos.com/kafka-key/client.key",
   "catalog" = "my_catalog",
   "md5" = "b5bb901bf10f99205b39a46ac3557dd9"
);
```

DROP FILE

Description

This statement is used to delete an uploaded file.

Syntax

```
DROP FILE "<file_name>" [ { FROM | IN } <database>] PROPERTIES ("<key>"="<value>" [ , ... ])
```

Required Parameters

1. `<file_name>`

> Custom file name.

2. `<key>`

> File attribute key. - **catalog**: Required. Classification category of the file.

3. `<value>`

> File attribute value.

Optional Parameters

1. `<database>`

> Specifies the database to which the file belongs. Uses the current session's database if not specified.

Access Control Requirements

The user executing this SQL command must possess the following minimum privileges:

Privilege	Object	Notes
`ADMIN_PRIV`	User / Role	The user or role must hold the `ADMIN_PRIV` privilege to execute this operation

Example

- Delete the file ca.pem

```
DROP FILE "ca.pem" properties("catalog" = "kafka");
```

- Delete file `client.key` categorized under `my_catalog`

```
DROP FILE "client.key"
IN my_database
```

- Delete file `client_1.key` categorized under `my_catalog`

```
DROP FILE "client_1.key"
FROM my_database
```

SHOW FILE

Description

This statement is used to display a file created in a database.

Syntax

```
SHOW FILE { [ FROM | IN ] <database_name>}
```

Optional Parameters

1. `<database_name>`

> Specifies the database to which the file belongs. Uses the current session's database if not specified.

Return Value

Column	Description
`Id`	Globally unique file identifier
`DbName`	Name of the owning database
`Catalog`	User-defined classification
`FileName`	Name of the file
`FileSize`	File size in bytes
`IsContent`	Content existence flag
`MD5`	MD5 checksum of the file

Access Control Requirements

The user executing this SQL command must possess the following minimum privileges:

Privilege	Object	Notes
`ADMIN_PRIV`	User / Role	The user or role must have access privileges to the database where the file resides to execute this command

Example

- View uploaded files in the current session database

```
SHOW FILE;
```

```
+-------+--------+----------+-------------------------+----------+-----------+----------------------------------+
| Id    | DbName | Catalog  | FileName                | FileSize | IsContent | MD5                              |
+-------+--------+----------+-------------------------+----------+-----------+----------------------------------+
| 12006 | testdb | doris_be | doris_be_metadata_layout | 89349    | true      | 9a3f68160b4106b0e923a0aa2fc05599 |
+-------+--------+----------+-------------------------+----------+-----------+----------------------------------+
```

- View uploaded files in specified database

```
SHOW FILE FROM example_db;
```

```
+-------+------------+----------+-------------------------+----------+-----------+----------------------------------+
| Id    | DbName     | Catalog  | FileName                | FileSize | IsContent | MD5                              |
+-------+------------+----------+-------------------------+----------+-----------+----------------------------------+
| 12007 | example_db | doris_fe | doris_fe_metadata_layout | 569373   | true      | 10385505d3c0d03f085fea6f8d51adfa |
| 12008 | example_db | doris_be | doris_be_metadata_layout | 89349    | true      | 9a3f68160b4106b0e923a0aa2fc05599 |
+-------+------------+----------+-------------------------+----------+-----------+----------------------------------+
```

CREATE ENCRYPTKEY

Description

This statement creates a custom key.

Syntax

```
CREATE ENCRYPTKEY <key_name> AS "<key_string>"
```

Required Parameters

1. `<key_name>`

> Specifies the name of the key to be created, which may include a database identifier.
> Example: `db1.my_key`

2. `<key_string>`

> Defines the key material string for cryptographic operations.
>
> **Behavior Notes**: - When the `<key_name>` contains a database identifier, the key will be created in the specified database - If no database is specified in `<key_name>`, the current session's database will be used - Key creation will fail if duplicate key names exist in the target database

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
`ADMIN_PRIV`	Global	The user must have the `ADMIN_PRIV` privilege to create encryption keys

Example

- Create a custom key

  ```
  CREATE ENCRYPTKEY my_key AS "ABCD123456789";
  ```

- Create a custom key in the testdb database

```
CREATE ENCRYPTKEY testdb.test_key AS "ABCD123456789";
```

- Use a custom key to encrypt data

  When using custom keys, you must prefix the key name with `KEY`/`key` followed by a space.

```
SELECT HEX(AES_ENCRYPT("Doris is Great", KEY my_key));
```

```
+-----------------------------------------------+
| hex(aes_encrypt('Doris is Great', key my_key)) |
+-----------------------------------------------+
| D26DB38579D6A343350EDDC6F2AD47C6               |
+-----------------------------------------------+
```

- Use a custom key to decrypt data

```
SELECT AES_DECRYPT(UNHEX('D26DB38579D6A343350EDDC6F2AD47C6'), KEY my_key);
```

```
+------------------------------------------------------------------------+
| aes_decrypt(unhex('D26DB38579D6A343350EDDC6F2AD47C6'), key my_key)  |
+------------------------------------------------------------------------+
| Doris is Great                                                         |
+------------------------------------------------------------------------+
```

DROP ENCRYPTKEY

Description

Delete a custom key. The name of the key is exactly the same to be deleted.

Syntax

```
DROP ENCRYPTKEY [IF EXISTS] <key_name>
```

Required Parameters

1. `<key_name>`

> Specifies the name of the key to be deleted, which may include a database identifier. Example: `db1.my_key`

Optional Parameters

1. `[IF EXISTS]`

> If specified, no error will be thrown when attempting to delete a non-existent key.

Access Control Requirements

The user executing this SQL command must possess the following minimum privileges:

Privilege	Object	Notes
`ADMIN_PRIV`	User / Role	The user or role must hold the`ADMIN_PRIV` privilege to perform key deletion operations

Example

- Delete a key

  ```
  DROP ENCRYPTKEY my_key;
  ```

- Drop a key without throwing errors if it doesn't exist

  ```
  DROP ENCRYPTKEY IF EXISTS testdb.my_key;
  ```

SHOW ENCRYPTKEY

Description

View all custom keys under the database. If the user specifies a database, check the corresponding database, otherwise directly query the database where the current session is located.

Syntax

```
SHOW ENCRYPTKEYS [ { IN | FROM } <db> ] [ LIKE '<key_pattern>']
```

Optional Parameters

1. `<db>`

> Name of the target database.
> Example: `db1` (in key names like `db1.my_key`).

2. `<key_pattern>`

> Pattern parameter for filtering key names (supports wildcard matching).

Return Value

Column	Description
`EncryptKey Name`	Name of the key
`EncryptKey String`	Value of the key

Access Control Requirements

The user executing this SQL command must have the following minimum privileges:

Privilege	Object	Notes
`ADMIN_PRIV`	User / Role	Must hold the `ADMIN_PRIV` privilege on the target user/role to view encryption keys

Usage Notes

If a database is explicitly specified, the system will query encryption keys within the designated database. Otherwise, it automatically queries keys in the current session's database.

Example

- List all custom keys in the current session's database.

```
SHOW ENCRYPTKEYS;
```

```
+-----------------+------------------+
| EncryptKey Name | EncryptKey String |
+-----------------+------------------+
| testdb.test_key | ABCD123456789     |
+-----------------+------------------+
```

- List all custom keys in a specified database.

```
SHOW ENCRYPTKEYS FROM example_db ;
```

```
+---------------------+------------------+
| EncryptKey Name     | EncryptKey String |
+---------------------+------------------+
| example_db.my_key   | ABCD123456789     |
| example_db.test_key | ABCD123456789     |
+---------------------+------------------+
```

- Filter keys by name pattern in a specified database.

```
SHOW ENCRYPTKEYS FROM example_db LIKE "%my%";
```

```
+------------------+------------------+
| EncryptKey Name   | EncryptKey String |
+------------------+------------------+
| example_db.my_key | ABCD123456789     |
+------------------+------------------+
```

CREATE SQL_BLOCK_RULE

Description

This statement is used to create an SQL block rule.

SQL_BLOCK_RULE can be used to restrict users from performing certain operations, such as avoiding scanning large amounts of data.

Syntax

```
CREATE SQL_BLOCK_RULE <rule_name>
PROPERTIES (
          -- property
          <property>
          -- Additional properties
          [ , ... ]
          )
```

Required Parameters

1. `<rule_name>`

> The name of the rule.

2. `<property>`

> The properties of the rule can be divided into three categories: SQL execution, scan limitation, and switch.
>
> The SQL execution and scan limitation categories are mutually exclusive, meaning that an SQL block rule can only restrict one of them.
>
> **SQL Execution Category**

> There are two types, representing regular expression matching and exact matching, respectively. Only one of them can be chosen.
>
> - sql: The matching rule (based on regular expression matching, special characters need to be escaped, for example, `select *` should be written as `select \\*`). When a user executes an SQL statement, the system will use the SQL set here as a regular expression to match the SQL submitted by the user. If it matches, the execution of the SQL will be blocked.
> - sqlHash: The MD5 hash value of the SQL. This is mainly used in conjunction with slow logs. Users do not need to calculate the hash value themselves. For example, if a slow log shows that a

particular SQL is running slowly, you can copy the `SqlHash` from `fe.audit.log` and create an SQL_BLOCK_RULE to restrict the execution of this SQL.

Scan Limitation Category When a user initiates a query, the query optimizer will calculate the number of partitions, tablets, and rows of data that need to be scanned for each table. The following properties can be used to limit these three numbers, either all at once or just some of them. - partition_num: The maximum number of partitions that a table will scan. - tablet_num: The maximum number of tablets that a table will scan. - cardinality: The number of rows of data that a table will scan.

Switch Category

- global: Whether the rule is effective for all users. The default is false. If it is not set to true, the rule needs to be applied to a specific user through the `set property` command.
- enable: Whether the blocking rule is enabled. The default is true.

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Global	

Example

1. Create a rule that prevents all users from executing `select * from order_analysis`

```
CREATE SQL_BLOCK_RULE test_rule
PROPERTIES(
"sql"="select \\* from order_analysis",
"global"="true",
"enable"="true"
);
```

When we execute the SQL defined in the rule, it will return an exception error, as shown below:

```
mysql> select * from order_analysis;
ERROR 1064 (HY000): errCode = 2, detailMessage = sql match regex sql block rule: test_rule
```

2. Create a rule that prevents scanning more than 30 partitions of the same table and restricts the query data volume to no more than 10 billion rows

```
CREATE SQL_BLOCK_RULE test_rule2
PROPERTIES
(
    "partition_num" = "30",
    "cardinality" = "10000000000",
    "global" = "true",
    "enable" = "true"
);
```

3. SQL matching is based on regular expressions. If you want to match more patterns of SQL, you need to write the corresponding regular expressions. For example, ignoring spaces in SQL and preventing queries on tables starting with “order”, as shown below:

```
CREATE SQL_BLOCK_RULE test_rule3
PROPERTIES(
  "sql"="\\s*select\\s*\\*\\s*from order_\\w*\\s*",
  "global"="true",
  "enable"="true"
);
```

4. Create a rule that is only applicable to specific users

```
CREATE SQL_BLOCK_RULE test_rule4
PROPERTIES(
    "sql"="select \\* from order_analysis",
    "global"="false",
    "enable"="true"
);
```

Apply the rule to user jack

```
SET PROPERTY FOR 'jack' 'sql_block_rules' = 'test_rule4';
```

Others

Common regular expressions are as follows:

TextCopy

```
. : Matches any single character except the newline character `\n`.

* : Matches the preceding element zero or more times. For example, a* matches zero or more 'a's.

+ : Matches the preceding element one or more times. For example, a+ matches one or more 'a's.

? : Matches the preceding element zero or one time. For example, a? matches zero or one 'a'.

[] : Defines a character set. For example, [aeiou] matches any vowel.

[^] : When used in a character set, ^ indicates negation, matching characters not in the set. For example, [^0-9] matches any non-digit character.

() : Groups expressions, allowing quantifiers to be applied to them. For example, (ab)+ matches consecutive 'ab's.

| : Indicates logical OR. For example, a|b matches 'a' or 'b'.

^ : Matches the beginning of a string. For example, ^abc matches strings starting with 'abc'.

$ : Matches the end of a string. For example, xyz$ matches strings ending with 'xyz'.

\ : Escapes special characters, making them ordinary characters. For example, \\. matches the period character '.'.

\s : Matches any whitespace character, including spaces, tabs, newlines, etc.

\d : Matches any digit character, equivalent to [0-9].
```

\w : Matches any word character, including letters, digits, and underscores, equivalent to [a-zA-ZO-9_].

ALTER SQL_BLOCK_RULE

Description

This statement is used to modify an SQL block rule.

Syntax

```
ALTER SQL_BLOCK_RULE <rule_name>
PROPERTIES (
          -- property
          <property>
          -- Additional properties
          [ , ... ]
          )
```

Required Parameters

1. `<rule_name>`

> The name of the rule.

2. `<property>`

See the introduction of CREATE SQL_BLOCK_RULE for details.

Access Control Requirements

The user executing this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Global	

Example

1. Modify the SQL and enable the rule

```
ALTER SQL_BLOCK_RULE test_rule PROPERTIES("sql"="select \\* from test_table","enable"="true")
```

DROP SQL_BLOCK_RULE

Description

Deletes one or more SQL blocking rules. Multiple rules can be deleted at once by separating them with commas.

Syntax

```
DROP SQL_BLOCK_RULE <rule_name>[, ...]
```

Required Parameters

The name of the SQL blocking rule to be deleted. Multiple rule names can be specified, separated by commas.

Access Control Requirements

Users executing this SQL command must have at least the following privileges: | Privilege | Object | Notes | | : ——————- | :———- | :——————- | | ADMIN | User or Role | Only users or roles with the ADMIN privilege can perform the DROP operation. |

Example

Delete `test_rule1` and `test_rule2` blocking rules

```
DROP SQL_BLOCK_RULE test_rule1, test_rule2;
```

SHOW SQL_BLOCK_RULE

Description

Displays the configured SQL blocking rules. If no rule name is specified, all rules will be displayed.

Syntax

```
SHOW SQL_BLOCK_RULE [FOR <rule_name>];
```

Optional Parameters

`<rule_name>`

The name of the SQL blocking rule to display. If omitted, all rules will be shown.

Access Control Requirements

Users executing this command must have the following privileges:

Privilege	Object	Notes
ADMIN	User or Role	Required to perform this operation.

Examples

1. Display all SQL blocking rules

```
SHOW SQL_BLOCK_RULE;
```

```
+------------+------------------------------+---------+--------------+-----------+-------------+--------+--------+
| Name       | Sql                          | SqlHash | PartitionNum | TabletNum | Cardinality | Global | Enable |
+------------+------------------------------+---------+--------------+-----------+-------------+--------+--------+
| test_rule  | select * from order_analysis | NULL    | 0            | 0         | 0           | true   | true   |
| test_rule2 | NULL                         | NULL    | 30           | 0         | 10000000000 | false  | true   |
+------------+------------------------------+---------+--------------+-----------+-------------+--------+--------+
```

2. Display a specific SQL blocking rule

```
SHOW SQL_BLOCK_RULE FOR test_rule2;
```

```
+------------+------+---------+--------------+-----------+-------------+--------+--------+
| Name       | Sql  | SqlHash | PartitionNum | TabletNum | Cardinality | Global | Enable |
+------------+------+---------+--------------+-----------+-------------+--------+--------+
| test_rule2 | NULL | NULL    | 30           | 0         | 10000000000 | false  | true   |
+------------+------+---------+--------------+-----------+-------------+--------+--------+
```

CREATE ROW POLICY

Description

Creates a row-level security policy on a table. The policy restricts the rows that specified users or roles can see by attaching a filter predicate that is applied to queries against the table.

Syntax

```
CREATE ROW POLICY [ IF NOT EXISTS ] <policy_name>
ON <table_name>
AS { RESTRICTIVE | PERMISSIVE }
TO { <user_name> | ROLE <role_name> }
USING (<filter>);
```

Required Parameters

> Row security policy name

> Table name

> RESTRICTIVE combines a set of policies with AND, PERMISSIVE combines a set of policies with OR

> Equivalent to the filter condition of a query statement, for example: id=1

Optional Parameters

> User name, cannot be created for root and admin users

> Role name

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV or GRANT_PRIV	Global	

Examples

1. Create a set of row security policies

```
CREATE ROW POLICY test_row_policy_1 ON test.table1
AS RESTRICTIVE TO test USING (c1 = 'a');
CREATE ROW POLICY test_row_policy_2 ON test.table1
AS RESTRICTIVE TO test USING (c2 = 'b');
CREATE ROW POLICY test_row_policy_3 ON test.table1
AS PERMISSIVE TO test USING (c3 = 'c');
CREATE ROW POLICY test_row_policy_3 ON test.table1
AS PERMISSIVE TO test USING (c4 = 'd');
```

When we execute a query on table1, the rewritten SQL is:

```
SELECT * FROM (SELECT * FROM table1 WHERE (c1 = 'a' AND c2 = 'b') AND (c3 = 'c' OR c4 = 'd'))
```

DROP ROW POLICY

Description

Delete row security policy. For details about row security policies, please refer to the “Security Policies” chapter

Syntax

```
DROP ROW POLICY <policy_name> on <table_name>
  [ FOR { <user_name> | ROLE <role_name> } ];
```

Required Parameters

> Row security policy name

> Table name

Optional Parameters (Optional Parameters)

> User name

> Role name

Access Control Requirements (Access Control Requirements)

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV or *GRANT_PRIV*	Global	

Examples (Examples)

1. Delete the *policy1 row security policy* for *db1.table1*

```
DROP ROW POLICY policy1 ON db1.table1
```

1. Delete the policy1 row security policy for db1.table1 that applies to user1

```
DROP ROW POLICY policy1 ON db1.table1 FOR user1
```

1. Delete the policy1 row security policy for db1.table1 that applies to role1

```
DROP ROW POLICY policy1 ON db1.table1 FOR role role1
```

SHOW ROW POLICY

Description

View row security policies. For details on row security policies, refer to the “Security Policies” chapter

Syntax

```
SHOW ROW POLICY [ FOR { <user_name> | ROLE <role_name> } ];
```

Optional Parameters

> User name

> Role name

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
ADMIN_PRIV	Global	

Examples

1. View all security policies

```
SHOW ROW POLICY;
```

1. Query by specifying a user name

```
SHOW ROW POLICY FOR user1;
```

1. Query by specifying a role name

```
SHOW ROW POLICY for role role1;
```

CREATE JOB

Description

Doris Job is a task that runs according to a set plan. It is used to trigger predefined operations at a specific time or a specified time interval, so as to help us automatically perform some tasks. Functionally, it is similar to the scheduled task on the operating system (such as cron in Linux and scheduled tasks in Windows).

There are two types of jobs: `ONE_TIME` and `RECURRING`. Among them, the `ONE_TIME` type of job will be triggered at a specified time point. It is mainly used for one-time tasks, while the `RECURRING` type of job will be triggered cyclically within a specified time interval. This method is mainly used for periodic tasks. The `RECURRING` type of job can specify the start time and end time, that is, `STARTS/ENDS`. If the start time is not specified, the default first execution time is the current time + one scheduling cycle. If the end time is specified, the task execution is completed. If the end time is reached (or exceeded, or the next execution cycle will exceed the end time), it will be updated to the FINISHED state, and no more tasks will be generated at this time.

There are 4 states for a job (`RUNNING`, `STOPPED`, `PAUSED`, `FINISHED`).

The initial state is `RUNNING`. A job in the `RUNNING` state will generate a TASK for execution according to the established scheduling cycle. When the job is completed and reaches the end time, the state changes to `FINISHED`.

A job in the `PAUSED` state can be resumed through the RESUME operation and changed to the RUNNING state.

A job in the `STOPPED` state is actively triggered by the user, and the running job will be canceled and the job will be deleted.

A job in the `FINISHED` state will be retained in the system for 24 hours and will be deleted after 24 hours.

JOB only describes job information. Execution will generate TASK. TASK status is divided into `PENDING`, `RUNNING`, `SUCCEESS`, `FAILED`, `CANCELED` `PENDING` means that the trigger time has arrived but the resource run is waiting. After the resource is allocated, the status changes to `RUNNING`. Success/failure of execution changes to `SUCCESS/FAILED`. `CANCELED` means cancellation status. TASK persists the final status, i.e. `SUCCESS/FAILED`. Other statuses can be checked during operation, but will not be visible if restarted.

Syntax

```
CREATE
    JOB
    <job_name>
    ON SCHEDULE <schedule>
    [ COMMENT <string> ]
    DO <sql_body>
```

Where:

```
schedule:
  { AT <at_timestamp> | EVERY <interval> [STARTS <start_timestamp> ] [ENDS <end_timestamp> ] }
```

Where:

```
interval:
  quantity { WEEK | DAY | HOUR | MINUTE }
```

Required parameters

1. <job_name> > Job name, which identifies a unique event in a db. The job name must be globally unique. If a job with the same name already exists, an error will be reported. We reserve the **inner**_ prefix for internal use in the system, so users cannot create names starting with **inner**_.

2. <schedule> > The ON SCHEDULE clause specifies the type, triggering time and frequency of the job. It can specify a one-time job or a periodic job.

3. <sql_body> > The DO clause specifies the operation to be performed when the job is triggered, that is, a SQL statement.

Optional parameters

1. AT <at_timestamp> > Format: 'YYYY-MM-DD HH:MM:SS', used for **one-time events**, it specifies that the event is executed only once at a given date and time timestamp, and when the execution is completed, the job status will change to FINISHED.

2. EVERY <interval> > Indicates a regularly repeated operation, it specifies the execution frequency of the job, and a time interval must be specified after the keyword, which can be days, hours, minutes, seconds, or weeks.

3. STARTS <start_timestamp> > Format: 'YYYY-MM-DD HH:MM:SS', used to specify the start time of the job. If not specified, it will be executed from the next time point after the current time. The start time must be greater than the current time.

4. ENDS <end_timestamp> > Format: 'YYYY-MM-DD HH:MM:SS', used to specify the end time of the job. If not specified, it means permanent execution. The date must be greater than the current time. If the start time is specified, that is, STARTS, the end time must be greater than the start time.

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Database	Currently only supports **ADMIN** permissions to perform this operation

Usage Notes

- TASK only retains the latest 100 records.
- Currently only supports **INSERT internal table** operations, and will support more operations in the future.

- When the next scheduled task time expires, that is, when the task needs to be scheduled for execution, if the current JOB still has historical tasks being executed, the current task scheduling will be skipped. Therefore, it is very important to control a reasonable execution interval.

Examples

- Create a job named my_job, which is executed once every minute. The operation performed is to import the data in db2.tbl2 into db1.tbl1.

```
CREATE JOB my_job ON SCHEDULE EVERY 1 MINUTE DO INSERT INTO db1.tbl1 SELECT * FROM db2.tbl2;
```

- Create a one-time job that will be executed once at 2020-01-01 00:00:00 to import the data in db2.tbl2 into db1.tbl1.

```
CREATE JOB my_job ON SCHEDULE AT '2020-01-01 00:00:00' DO INSERT INTO db1.tbl1 SELECT * FROM db2.tbl2;
```

- Create a periodic Job that will start executing at 2020-01-01 00:00:00 and execute once a day. The operation performed is to import the data in db2.tbl2 into db1.tbl1.

```
CREATE JOB my_job ON SCHEDULE EVERY 1 DAY STARTS '2020-01-01 00:00:00' DO INSERT INTO db1.tbl1 SELECT * FROM db2.tbl2 WHERE
create_time >= days_add(now(),-1);
```

- Create a periodic job that will start at 2020-01-01 00:00:00 and execute once a day. The operation is to import the data in db2.tbl2 into db1.tbl1. The job ends at 2020-01-01 00:10:00.

```
CREATE JOB my_job ON SCHEDULE EVERY 1 DAY STARTS '2020-01-01 00:00:00' ENDS '2020-01-01 00:10:00' DO INSERT INTO db1.tbl1
SELECT * FROM db2.tbl2 create_time >= days_add(now(),-1);
```

Best Practices

- Manage jobs reasonably to avoid a large number of jobs being triggered at the same time, which will cause task accumulation and affect the normal operation of the system.
- The task execution interval should be set within a reasonable range, at least greater than the task execution time.

Related Documents

- Pause-JOB
- Resume-JOB
- Delete-JOB
- Query-JOB
- Query-TASKS

CONFIG

fe.conf

- job_dispatch_timer_job_thread_num, the number of threads used to distribute timed tasks, the default value is 2, if there are a large number of periodic execution tasks, you can increase this parameter.
- job_dispatch_timer_job_queue_size, the queue size for storing timed tasks when tasks are accumulated, the default value is 1024. If a large number of tasks are triggered at the same time, this parameter can be increased. Otherwise, the queue will be full, and the submitted task will enter a blocked state, which will cause subsequent tasks to fail to submit.
- finished_job_cleanup_threshold_time_hour, the time threshold for cleaning up completed tasks, in hours, the default value is 24 hours.
- job_insert_task_consumer_thread_num = 10; the number of threads used to execute Insert tasks, the value should be greater than 0, otherwise the default value is 5.

PAUSE JOB

Description

When a user pauses a job in the RUNNING state, the running task will be interrupted and the job state will be changed to PAUSED. The stopped job can be resumed by the RESUME operation.

Syntax

```
PAUSE JOB WHERE jobname = <job_name> ;
```

Required parameters

1. <job_name> > The name of the job to be paused.

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Database	Currently only supports **ADMIN** permissions to perform this operation

Examples

- Pause the job named example.

  ```
  PAUSE JOB where jobname='example';
  ```

DROP JOB

Description

The user deletes a JOB job. The job will be stopped and deleted immediately.

Syntax

```
DROP JOB where jobName = <job_name> ;
```

Required parameters

1. <job_name> > The <job_name> of the task to be deleted.

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Database	Currently only supports **ADMIN** permissions to perform this operation

Examples

- Delete the job named example.

  ```
  DROP JOB where jobName='example';
  ```

RESUME JOB

Description

Restore the PAUSED job to the RUNNING state. The RUNNING job will be executed according to the scheduled period.

Syntax

```
RESUME JOB where jobName = <job_name> ;
```

Required parameters

1. <job_name> > The <job_name> of the recovery task.

Access Control Requirements

The user who executes this SQL command must have at least the following permissions:

Privilege	Object	Notes
ADMIN_PRIV	Database	Currently only supports **ADMIN** permissions to perform this operation

Example

- Resume the job named example.

  ```
  RESUME JOB where jobName= 'example';
  ```

CANCEL TASK

Description

Cancel a running task created by the CREATE JOB statement.

- The task must be created by the CREATE JOB statement.
- It must be a running task.
- This function is supported from version 2.1.0.

Syntax

```
CANCEL TASK WHERE jobName = '<job_name>' AND taskId = '<task_id>';
```

Required Parameters

> The name of the job, of type string.

> The task ID, of integer type. It can be queried through the tasks table-valued function. For example: SELECT * FROM tasks('type'='insert'). For more information, please refer to "task table-valued function".

Access Control Requirements

The user executing this SQL command must have at least ADMIN_PRIV privileges.

Examples

Cancel a background task with jobName 'example' and taskId 378912.

```
CANCEL TASK WHERE jobName='example' AND taskId=378912
```

INSTALL PLUGIN

Description

This statement is used to install a plug-in

Syntax

```
INSTALL PLUGIN FROM <source> [PROPERTIES ("<key>"="<value>", ...)]
```

Required parameters

** 1. <source>** > The plugin path to be installed, supports three types： > 1. An absolute path to a zip file > 2. An absolute path to a plugin directory > 3. Points to a zip file download path with http or https protocol

Optional parameters

** 1. [PROPERTIES ("<key>"="<value>", ...)]** > Used to specify properties or parameters when installing a plug-in

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
ADMIN_PRIV	The entire cluster	Requires administrative privileges for the entire cluster

Precautions

Note that you need to place an md5 file with the same name as the .zip file, such as http://mywebsite.com/plugin.zip.md5 . The content is the MD5 value of the .zip file.

Example

- Install a local zip file plugin:

  ```
  INSTALL PLUGIN FROM "/home/users/doris/auditdemo.zip";
  ```

- Install the plugin in a local directory:

  ```
  INSTALL PLUGIN FROM "/home/users/doris/auditdemo/";
  ```

- Download and install a plugin:

```
INSTALL PLUGIN FROM "http://mywebsite.com/plugin.zip";
```

- Download and install a plugin, and set the md5sum value of the zip file:

```
INSTALL PLUGIN FROM "http://mywebsite.com/plugin.zip" PROPERTIES("md5sum" = "73877f6029216f4314d712086a146570");
```

UNINSTALL PLUGIN

Description

This statement is used to uninstall a plugin.

Syntax

```
UNINSTALL PLUGIN <plugin_name>;
```

Required parameters

** 1. `<plugin_name>`** > The name of the uninstalled plugin

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
ADMIN_PRIV	The entire cluster	Requires administrative privileges for the entire cluster

Precautions

Only non-builtin plugins can be uninstalled

Example

- To uninstall a plugin:

  ```
  UNINSTALL PLUGIN auditdemo;
  ```

SHOW PLUGINS

Description

This statement is used to display installed plug-ins

Syntax

```
SHOW PLUGINS
```

Return Value

Column	Description
Description	Corresponding plug-in description
Version	Plug-in corresponding version number
JavaVersion	Corresponding Java version number
ClassName	Program class name
SoName	Program shared object name
Sources	Plugin Source
Status	Installation Status
Properties	Plugin Properties

Permission Control

The user executing this SQL command must have at least the following permissions:

Permissions	Object	Notes
ADMIN_PRIV	The entire cluster	Requires administrative privileges for the entire cluster

Example

- Show installed plugins:

```
SHOW PLUGINS;
```

```
+------------------------------+---------+-----------------------------------------------------------+---------+----------
---+-----------------------------------------------------------+--------+---------+-----------+------------+
| Name                         | Type    | Description                                               | Version |
JavaVersion | ClassName                                           | SoName | Sources | Status    | Properties |
+------------------------------+---------+-----------------------------------------------------------+---------+----------
---+-----------------------------------------------------------+--------+---------+-----------+------------+
```

```
| __builtin_AuditLoader         | AUDIT   | builtin audit loader, to load audit log to internal table | 2.1.0   | 1.8.31
| org.apache.doris.plugin.audit.AuditLoader                 | NULL   | Builtin | INSTALLED | {}         |
| __builtin_AuditLogBuilder     | AUDIT   | builtin audit logger                                      | 0.12.0  | 1.8.31
| org.apache.doris.plugin.audit.AuditLogBuilder             | NULL   | Builtin | INSTALLED | {}         |
| __builtin_SqlDialectConverter | DIALECT | builtin sql dialect converter                             | 2.1.0   | 1.8.31
| org.apache.doris.plugin.dialect.HttpDialectConverterPlugin | NULL   | Builtin | INSTALLED | {}         |
+-------------------------------+---------+-----------------------------------------------------------+---------+----------
---+-----------------------------------------------------------+--------+---------+-----------+------------+
```

SHOW COLLATION

Description

In Doris, the SHOW COLLATION command is used to display the character set collations available in the database. A collation is a set of rules that determine how data is sorted and compared. These rules affect the storage and retrieval of character data.

Syntax

```
SHOW COLLATION
```

Return Value

column name	description
Collation	The collation name
Charset	The character set
Id	The collation's ID
Default	Whether this is the default collation for the character set
Compiled	Whether the collation is compiled
Sortlen	Sort length

Usage Notes

In Doris, although it is compatible with the MySQL collation setting command, it does not actually take effect. When executed, utf8mb4_0900_bin will always be used as the comparison rule.

Examples

```
SHOW COLLATION;
```

```
+--------------------+---------+------+---------+----------+---------+
| Collation          | Charset | Id   | Default | Compiled | Sortlen |
+--------------------+---------+------+---------+----------+---------+
| utf8mb4_0900_bin   | utf8mb4 |  309 | Yes     | Yes      |       1 |
| utf8mb3_general_ci | utf8mb3 |   33 | Yes     | Yes      |       1 |
+--------------------+---------+------+---------+----------+---------+
```

SHOW CHARSET

Description

The “SHOW CHARSET” command is used to display the character sets available in the current database management system and some properties associated with each character set.

These properties may include the name of the character set, default collation, maximum byte length, etc. A list of character sets supported on the system and their details can be viewed by running the “SHOW CHARSET” command.

Syntax

```
SHOW CHARSET
```

Return Value

column name	description
Charset	Character Set
Description	Description
Default Collation	Default collation name
Maxlen	Maximum byte length

Examples

```
SHOW CHARSET;
```

```
+---------+---------------+-------------------+--------+
| Charset | Description   | Default collation | Maxlen |
+---------+---------------+-------------------+--------+
| utf8mb4 | UTF-8 Unicode | utf8mb4_0900_bin  | 4      |
+---------+---------------+-------------------+--------+
```

SHOW DATA TYPES

Description

```
This statement is used to view all supported data types.
```

Syntax

```
SHOW DATA TYPES;
```

Return Value

Column Name	Description
TypeName	Name of the type
Size	Size in bytes

Access Control Requirements

The user executing this SQL command does not require specific privileges.

Example

- Display all supported data types

```
SHOW DATA TYPES;
```

```
+----------------+------+
| TypeName       | Size |
+----------------+------+
| AGG_STATE      | 16   |
| ARRAY          | 32   |
| BIGINT         | 8    |
| BITMAP         | 16   |
| BOOLEAN        | 1    |
| CHAR           | 16   |
| DATE           | 16   |
| DATETIME       | 16   |
| DATETIMEV2     | 8    |
| DATEV2         | 4    |
| DECIMAL128     | 16   |
| DECIMAL32      | 4    |
| DECIMAL64      | 8    |
| DECIMALV2      | 16   |
| DOUBLE         | 8    |
| FLOAT          | 4    |
| HLL            | 16   |
| INT            | 4    |
```

```
| IPV4           | 4    |
| IPV6           | 16   |
| JSON           | 16   |
| LARGEINT       | 16   |
| MAP            | 24   |
| QUANTILE_STATE | 16   |
| SMALLINT       | 2    |
| STRING         | 16   |
| TINYINT        | 1    |
| VARCHAR        | 16   |
+----------------+------+
```

SHOW TYPECAST

Description

View all type cast under the database. If the user specifies a database, then view the corresponding database, otherwise directly query the database where the current session is located

Syntax

```
SHOW TYPE_CAST [ { IN | FROM } <db>]
```

Required Parameters

1. `<db>`

The name of the database to query.

Return Value

Column Name	Description
Origin Type	Original Type
Cast Type	Conversion Type

Access Control Requirements

The user executing this SQL command must have at least the following privileges:

Privilege	Object	Notes
`Select_priv`	Database (DB)	The user or role must have `Select_priv` on the DB to view all type conversions under the database

Usage Notes

If a database is specified by the user, the system will query the specified database; otherwise, it will query the database associated with the current session by default.

Example

- View all type casts in database TESTDB

```
SHOW TYPE_CAST IN TESTDB;
```

```
+---------------+---------------+
| Origin Type   | Cast Type     |
+---------------+---------------+
| DATETIMEV2    | BOOLEAN       |
| DATETIMEV2    | TINYINT       |
| DATETIMEV2    | SMALLINT      |
| DATETIMEV2    | INT           |
| DATETIMEV2    | BIGINT        |
| DATETIMEV2    | LARGEINT      |
| DATETIMEV2    | FLOAT         |
| DATETIMEV2    | DOUBLE        |
| DATETIMEV2    | DATE          |
| DATETIMEV2    | DATETIME      |
| DATETIMEV2    | DATEV2        |
| DATETIMEV2    | DATETIMEV2    |
| DATETIMEV2    | DECIMALV2     |
| DATETIMEV2    | DECIMAL32     |
| DATETIMEV2    | DECIMAL64     |
| DATETIMEV2    | DECIMAL128    |
| DATETIMEV2    | DECIMAL256    |
| DATETIMEV2    | VARCHAR       |
| DATETIMEV2    | STRING        |
| DECIMAL256    | DECIMAL128    |
| DECIMAL256    | DECIMAL256    |
| DECIMAL256    | VARCHAR       |
| DECIMAL256    | STRING        |
+---------------+---------------+
```

SHOW PROC

Description

The Proc system is a unique feature of Doris. Students who have used Linux may understand this concept better. In Linux systems, proc is a virtual file system, usually mounted in the /proc directory. Users can view the internal data structure of the system through this file system. For example, you can view the details of the specified pid process through /proc/pid.

Similar to the proc system in Linux, the proc system in Doris is also organized into a directory-like structure to view different system information according to the “directory path (proc path)” specified by the user.

The proc system is designed mainly for system administrators, so that it is convenient for them to view some running states inside the system. Such as the tablet status of the table, the cluster balance status, the status of various jobs, and so on. is a very useful function

There are two ways to view the proc system in Doris.

1. View through the WEB UI interface provided by Doris, visit the address: `http://FE_IP:FE_HTTP_PORT`
2. Another way is by command

All commands supported by Doris PROC can be seen through `SHOW PROC "/";`

After connecting to Doris through the MySQL client, you can execute the SHOW PROC statement to view the information of the specified proc directory. The proc directory is an absolute path starting with “/”.

The results of the show proc statement are presented in a two-dimensional table. And usually the first column of the result table is the next subdirectory of proc.

```
mysql> show proc "/";
+---------------------------+
| name                      |
+---------------------------+
| auth                      |
| backends                  |
| bdbje                     |
| brokers                   |
| catalogs                  |
| cluster_balance           |
| cluster_health            |
| colocation_group          |
| current_backend_instances |
| current_queries           |
| current_query_stmts       |
| dbs                       |
| diagnose                  |
| frontends                 |
| jobs                      |
| load_error_hub            |
| monitor                   |
| resources                 |
| routine_loads             |
| statistic                 |
| stream_loads              |
```

```
| tasks                    |
| transactions             |
| trash                    |
+--------------------------+
23 rows in set (0.00 sec)
```

illustrate:

1. auth: User name and corresponding permission information
2. backends: Displays the node list of BE in the cluster, equivalent to SHOW BACKENDS
3. bdbje: To view the bdbje database list, you need to modify the `fe.conf` file to add `enable_bdbje_debug_mode=true`, and then start `FE` through `sh start_fe.sh --daemon` to enter the `debug` mode. After entering `debug` mode, only `http server` and `MySQLServer` will be started and the `BDBJE` instance will be opened, but no metadata loading and subsequent startup processes will be entered.
4. brokers : View cluster broker node information, equivalent to SHOW BROKER
5. catalogs : View the data catalogs that have been created currently, equivalent to SHOW CATALOGS
6. cluster_balance : To check the balance of the cluster, please refer to Data Copy Management
7. cluster_health: Run `SHOW PROC '/cluster_health/tablet_health';` statement to view the replica status of the entire cluster.
8. colocation_group : This command can view the existing Group information in the cluster. For details, please refer to the Colocation Join chapter
9. current_backend_instances : Displays a list of be nodes that are currently executing jobs
10. current_queries : View the list of queries being executed, the SQL statement currently running.
11. current_query_stmts: Returns the currently executing query.
12. dbs: Mainly used to view the metadata information of each database and the tables in the Doris cluster. This information includes table structure, partitions, materialized views, data shards and replicas, and more. Through this directory and its subdirectories, you can clearly display the table metadata in the cluster, and locate some problems such as data skew, replica failure, etc.
13. diagnose: Report and diagnose common management and control issues in the cluster, including replica balance and migration, transaction exceptions, etc.
14. frontends: Display all FE node information in the cluster, including IP address, role, status, whether it is a master, etc., equivalent to SHOW FRONTENDS
15. jobs: show statistics of all kind of jobs. If a specific `dbId` is given, will return statistics data of the database. If `dbId` is -1, will return total statistics data of all databases
16. load_error_hub: Doris supports centralized storage of error information generated by load jobs in an error hub. Then view the error message directly through the `SHOW LOAD WARNINGS;` statement. Shown here is the configuration information of the error hub.
17. monitor : shows the resource usage of FE JVM
18. resources : View system resources, ordinary accounts can only see resources that they have USAGE_PRIV permission to use. Only the root and admin accounts can see all resources. Equivalent to SHOW RESOURCES
19. routine_loads: Display all routine load job information, including job name, status, etc.
20. statistics: It is mainly used to summarize and view the number of databases, tables, partitions, shards, and replicas in the Doris cluster. and the number of unhealthy copies. This information helps us to control the size of the cluster meta-information in general. It helps us view the cluster sharding situation from an overall perspective, and can quickly check the health of the cluster sharding. This further locates problematic data shards.
21. stream_loads: Returns the stream load task being executed.

22. tasks : Displays the total number of tasks of various jobs and the number of failures.
23. transactions : used to view the transaction details of the specified transaction id, equivalent to SHOW TRANSACTION
24. trash: This statement is used to view the space occupied by garbage data in the backend. Equivalent to SHOW TRASH

Examples

1. For example, “/dbs” displays all databases, and “/dbs/10002” displays all tables under the database with id 10002

```
mysql> show proc "/dbs/10002";
+---------+----------------------+----------+---------------------+--------------+--------+------+-------------------------
-+--------------+
| TableId | TableName            | IndexNum | PartitionColumnName | PartitionNum | State  | Type | LastConsistencyCheckTime
| ReplicaCount |
+---------+----------------------+----------+---------------------+--------------+--------+------+-------------------------
-+--------------+
| 10065   | dwd_product_live     | 1        | dt                  | 9            | NORMAL | OLAP | NULL
| 18           |
| 10109   | ODS_MR_BILL_COSTS_DO | 1        | NULL                | 1            | NORMAL | OLAP | NULL
| 1            |
| 10119   | test                 | 1        | NULL                | 1            | NORMAL | OLAP | NULL
| 1            |
| 10124   | test_parquet_import  | 1        | NULL                | 1            | NORMAL | OLAP | NULL
| 1            |
+---------+----------------------+----------+---------------------+--------------+--------+------+-------------------------
-+--------------+
4 rows in set (0.00 sec)
```

2. Display information about the number of all database tables in the cluster.

```
mysql> show proc '/statistic';
+-------+----------------------+----------+--------------+----------+-----------+------------+
| DbId  | DbName               | TableNum | PartitionNum | IndexNum | TabletNum | ReplicaNum |
+-------+----------------------+----------+--------------+----------+-----------+------------+
| 10002 | default_cluster:test | 4        | 12           | 12       | 21        | 21         |
| Total | 1                    | 4        | 12           | 12       | 21        | 21         |
+-------+----------------------+----------+--------------+----------+-----------+------------+
2 rows in set (0.00 sec)
```

3. The following command can view the existing Group information in the cluster.

```
SHOW PROC '/colocation_group';

+-------------+--------------+--------------+------------+----------------+----------+----------+
| GroupId     | GroupName    | TableIds     | BucketsNum | ReplicationNum | DistCols | IsStable |
+-------------+--------------+--------------+------------+----------------+----------+----------+
| 10005.10008 | 10005_group1 | 10007, 10040 | 10         | 3              | int(11)  | true     |
+-------------+--------------+--------------+------------+----------------+----------+----------+
```

   - GroupId: The cluster-wide unique identifier of a group, the first half is the db id, and the second half is the group id.
   - GroupName: The full name of the Group.
   - TabletIds: The id list of Tables contained in this Group.

- BucketsNum: The number of buckets.
- ReplicationNum: The number of replicas.
- DistCols: Distribution columns, that is, the bucket column type.
- IsStable: Whether the Group is stable (for the definition of stability, see the `Colocation replica balance and repair` section).

4. Use the following commands to further view the data distribution of a Group:

```
SHOW PROC '/colocation_group/10005.10008';

+-------------+---------------------+
| BucketIndex | BackendIds          |
+-------------+---------------------+
| 0           | 10004, 10002, 10001 |
| 1           | 10003, 10002, 10004 |
| 2           | 10002, 10004, 10001 |
| 3           | 10003, 10002, 10004 |
| 4           | 10002, 10004, 10003 |
| 5           | 10003, 10002, 10001 |
| 6           | 10003, 10004, 10001 |
| 7           | 10003, 10004, 10002 |
+-------------+---------------------+
```

- BucketIndex: The index of the bucket sequence.
- BackendIds: The list of BE node IDs where the data shards in the bucket are located.

5. Display the total number of tasks of various jobs and the number of failures.

```
mysql> show proc '/tasks';
+--------------------------+-----------+----------+
| TaskType                 | FailedNum | TotalNum |
+--------------------------+-----------+----------+
| CREATE                   | 0         | 0        |
| DROP                     | 0         | 0        |
| PUSH                     | 0         | 0        |
| CLONE                    | 0         | 0        |
| STORAGE_MEDIUM_MIGRATE   | 0         | 0        |
| ROLLUP                   | 0         | 0        |
| SCHEMA_CHANGE            | 0         | 0        |
| CANCEL_DELETE            | 0         | 0        |
| MAKE_SNAPSHOT            | 0         | 0        |
| RELEASE_SNAPSHOT         | 0         | 0        |
| CHECK_CONSISTENCY        | 0         | 0        |
| UPLOAD                   | 0         | 0        |
| DOWNLOAD                 | 0         | 0        |
| CLEAR_REMOTE_FILE        | 0         | 0        |
| MOVE                     | 0         | 0        |
| REALTIME_PUSH            | 0         | 0        |
| PUBLISH_VERSION          | 0         | 0        |
| CLEAR_ALTER_TASK         | 0         | 0        |
| CLEAR_TRANSACTION_TASK   | 0         | 0        |
| RECOVER_TABLET           | 0         | 0        |
| STREAM_LOAD              | 0         | 0        |
| UPDATE_TABLET_META_INFO  | 0         | 0        |
| ALTER                    | 0         | 0        |
| INSTALL_PLUGIN           | 0         | 0        |
| UNINSTALL_PLUGIN         | 0         | 0        |
| Total                    | 0         | 0        |
```

```
+-------------------------+-----------+----------+
26 rows in set (0.01 sec)
```

6. Display the replica status of the entire cluster.

```
mysql> show proc '/cluster_health/tablet_health';
+----------+---------------------------+-----------+------------+-------------------+----------------------+--------------
-------+--------------+---------------------------+-------------------------+-------------------+----------------------+---
------------------+----------------------+------------------+-----------------------------+-----------------+-------------
+------------+
| DbId     | DbName                    | TabletNum | HealthyNum | ReplicaMissingNum | VersionIncompleteNum |
ReplicaRelocatingNum | RedundantNum | ReplicaMissingInClusterNum | ReplicaMissingForTagNum | ForceRedundantNum |
ColocateMismatchNum | ColocateRedundantNum | NeedFurtherRepairNum | UnrecoverableNum | ReplicaCompactionTooSlowNum |
InconsistentNum | OversizeNum | CloningNum |
+----------+---------------------------+-----------+------------+-------------------+----------------------+--------------
-------+--------------+---------------------------+-------------------------+-------------------+----------------------+---
------------------+----------------------+------------------+-----------------------------+-----------------+-------------
+------------+
| 25852112 | default_cluster:bowen     | 1920      | 1920       | 0                 | 0                    | 0
| 0            | 0                         | 0                       | 0               | 0                  | 0
| 0                    | 0               | 0                         | 0              | 0           | 0          |
| 25342914 | default_cluster:bw        | 128       | 128        | 0                 | 0                    | 0
| 0            | 0                         | 0                       | 0               | 0                  | 0
| 0                    | 0               | 0                         | 0              | 0           | 0          |
| 2575532  | default_cluster:cps       | 1440      | 1440       | 0                 | 0                    | 0
| 0            | 0                         | 0                       | 0               | 0                  | 0
| 0                    | 0               | 0                         | 0              | 16          | 0          |
| 26150325 | default_cluster:db        | 38374     | 38374      | 0                 | 0                    | 0
| 0            | 0                         | 0                       | 0               | 0                  | 0
| 0                    | 0               | 0                         | 0              | 453         | 0          |
+----------+---------------------------+-----------+------------+-------------------+----------------------+--------------
-------+--------------+---------------------------+-------------------------+-------------------+----------------------+---
------------------+----------------------+------------------+-----------------------------+-----------------+-------------
+------------+
4 rows in set (0.01 sec)
```

View the replica status under a database, such as a database with a DbId of 25852112.

```
mysql> show proc '/cluster_health/tablet_health/25852112';
```

7. Report and diagnose cluster management issues

```
MySQL > show proc "/diagnose";
+-----------------+----------+------------+
| Item            | ErrorNum | WarningNum |
+-----------------+----------+------------+
| cluster_balance | 2        | 0          |
| Total           | 2        | 0          |
+-----------------+----------+------------+

2 rows in set
```

View replica balance migration issues

```
MySQL > show proc "/diagnose/cluster_balance";
+----------------------+--------+-----------------------------------------------------------------------------------------------
--------------------+-------------------------------------------------------------------------+------------+
| Item                 | Status | Content
```

```
| Detail Cmd                                                
```

LOCALTIME,LOCALTIMESTAMP

Description

The function retrieves the current system time and returns it as a datetime value (`DATETIME`). An optional precision can be specified to adjust the number of digits in the fractional seconds part of the return value.

Syntax

```
LOCALTIME([<precision>])
LOCALTIMESTAMP([<precision>]))
```

Parameters

Parameter	Description
<precision>	Optional parameter specifying the precision of the fractional seconds part in the return value. The range is 0 to 6, and the default is 0 (no fractional seconds). Limited by the JDK implementation: if FE is built with JDK8, the precision supports up to milliseconds (3 fractional digits), and higher precision digits will be filled with 0. If higher precision is required, please use JDK11.

Return Value

- Returns the current system time as a DATETIME type.
- If the specified `<precision>` is out of range (e.g., negative or greater than 6), the function will return an error.

Example

```
select LOCALTIME(),LOCALTIME(3),LOCALTIME(6);
```

```
+---------------------+-------------------------+----------------------------+
| now()               | now(3)                  | now(6)                     |
+---------------------+-------------------------+----------------------------+
| 2025-01-23 11:50:18 | 2025-01-23 11:50:18.883 | 2025-01-23 11:50:18.883000 |
+---------------------+-------------------------+----------------------------+
```