GaussDB(for MySQL)

Kernels

Issue 01

Date 2024-12-23

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 GaussDB(for MySQL) Kernel Version Release History

This section describes the kernel version updates of GaussDB(for MySQL).

2.0.54.240900

Table 1-1 Version 2.0.54.240900

Date	Description
2024-10-18	• New features and optimized features – Partition-level MDL: In MySQL Community Edition, you cannot perform both data manipulation language (DML) operations for accessing data of partitioned tables and data definition language (DDL) operations for maintaining partitions at the same time. This means that DDL operations can only be done during off-peak hours. To resolve such an issue, this version introduces partition-level metadata lock (MDL) to refine the lock granularity of a partitioned table from the table level to the partition level. After partition-level MDL is enabled, DML operations and specific DDL operations (such as adding and deleting partitions) on different partitions can be both performed, greatly improving concurrency between partitions. – Table recycle bin: After this function is enabled, the DROP TABLE statement that meets conditions does not directly delete a specified table. Instead, the table is temporarily stored in the recycle bin. When the maximum retention period expires, the table is automatically deleted in the background. You can change the retention period of deleted tables in the recycle bin. You can also restore or permanently delete tables from the recycle bin at any time. • Fixed issues – Fixed the issue that CPU resources of each tenant are not strictly allocated based on the configured ratio in resource preemption scenarios. – Allowed Statement Outline to support views and EXPLAIN ANALYZE statements.

2.0.54.240600

Table 1-2 Version 2.0.54.240600

Date	Description
2024-07-19	• New features and optimized features – Optimized hot row update: Hot rows are frequently updated for flash sales, concert ticket reservations, and train ticket bookings for popular routes. This version enhances hot row update, which can be enabled automatically or manually. After hot row update is enabled, hot rows can be upgraded efficiently. – Non-blocking DDL: When you perform a DDL operation, if the target table has uncommitted long transactions or large queries, the DDL operation continuously waits for obtaining the MDL-X lock. As a result, service connections are stacked and blocked. This version supports non-blocking DDL, which allows new transactions to enter the target table even if the MDL-X lock cannot be obtained, ensuring the stability of the entire service system. – Multi-tenant management: This feature enables a database to serve multiple tenants, maximizing database resource utilization. – Binlog pull for read replicas: You can use read replicas as the data source to establish a binlog replication link and synchronize the binlogs in real time, which helps reduce the load on the primary node. – Column compression: GaussDB(for MySQL) introduces fine-grained column compression to reduce data page storage and save costs. Two compression algorithms, ZLIB and ZSTD, are provided. You can select either of them to compress infrequently accessed large columns based on the compression ratio and compression and decompression performance. – INTERVAL RANGE partitioned tables: In previous versions, if the data to be inserted into an existing RANGE partitioned table exceeds the range of existing partitions, the data cannot be inserted and an error is returned. With the support for INTERVAL RANGE partitioned tables in this version, the database can now create partitions based on rules specified by the INTERVAL clause when new data exceeds the range of existing partitions. – LIST DEFAULT HASH partitioned tables: This feature supports two types of partitions at the same level: LIST and HASH. Data is first inserted into LIST partitions. Data that does not comply with the LIST

Date	Description
	partitioning rules is placed in the DEFAULT partition. If the DEFAULT partition has multiple partitions, HASH rules are used. LIST DEFAULT HASH partitioned tables are usually used in scenarios where LIST VALUES are unevenly distributed and cannot be fully enumerated. ● Fixed issues – Optimized the table-level restoration performance. – Optimized the execution performance of read replicas of a high-spec instance in high-concurrency scenarios.

2.0.51.240300

Table 1-3 Version 2.0.51.240300

Date	Description
2024-03-30	● New features and optimized features – Added global consistency, which provides strongly consistent reads at the cluster level with low performance loss. – Added the SHOW BINARY LOGS NO BLOCK syntax, which prevents transaction commits from being blocked during the execution of SHOW BINARY LOGS. – Optimized the UNDO TRUNCATE capability, which solves the issue of undo space expansion caused by a large number of writes. – Enhanced the degree of parallelism for full restoration, which optimizes the backup and restoration efficiency. ● Fixed issues – The query results of window functions are incorrect, or errors occur when window functions are executed. – Database nodes break down when specific PREPARE statements are repeatedly executed after plan cache is enabled. – An error is reported due to inconsistent character sets when stored procedures are executed in sequence. – Query results do not meet the expectation when an on-disk hash join is performed after PQ is enabled. – An error is reported due to duplicate primary keys when a query involves performing a GROUP BY operation on temporary table fields.

2.0.48.231200

Table 1-4 Version 2.0.48.231200

Date	Description
2024-01-30	• New features and optimized features – Enhanced composite partitioning: In addition to RANGE-HASH and LIST-HASH of MySQL Community Edition, added RANGE-RANGE, RANGE-LIST, LIST-RANGE, LIST-LIST, HASH-HASH, HASH-KEY, HASH-RANGE, HASH-LIST, KEY-HASH, KEY-KEY, KEY-RANGE, and KEY-LIST. – Added the forward compatibility with GROUP BY implicit/explicit sorting in MySQL 5.7. – Added the forward compatibility with the **max_length_for_sort_data** parameter in MySQL 5.7, which optimizes the file sorting performance in specific scenarios. – Optimized the issue that accessing views in information_schema is slow due to incorrect execution plan selection. – Added the EXIST subquery in PQ. – Optimized restoration of database tables or instances to a specific point in time. • Fixed issues – OpenSSL is upgraded. – The default value **SYSTEM** of the **time_zone** parameter impacts the efficiency of concurrent SQL statement execution in some scenarios. – SQL query results are incorrect when conditions are partially pushed down to a materialized derived table. – Performance suffers after PQ is enabled for on-disk hash joins in some scenarios. – The permissions page is not updated accordingly after a user is granted permissions on a database through the console and the database is later deleted in non-console mode.

2.0.45.230900

Table 1-5 Version 2.0.45.230900

Date	Description

2023-11-24	• New features and optimized features – Added forward compatibility of **datatime**, **timestamp**, and **time** field behaviors. – Added on-disk hash joins in PQ. – Added INSERT and REPLACE SELECT functions in PQ. – Added log printing mechanism for connection and disconnection, which helps you locate connection-related issues quickly. – Added some useful information in slow query logs, which helps you locate slow SQL statements. – Allowed you to dynamically enable binlog. – Optimized the NDP bloom filter. – Allowed you to use the CAST (... AS INT) syntax. – Optimized the Nested Loop Join + Distinct performance. – Identified slice ID corresponding to the slow I/O quickly. – Added the sal_init log, which helps you to locate storage API timeout issues. • Fixed issues – There are **trx_id** and **cpu_time** fields in full SQL statements. – Character strings can be converted into INT in WHERE conditions of PREPARE statements. – No crash issue occurs when DDL operations and queries are concurrently executed on a read replica. – The binlogs that are sharply generated in a short period of time can be cleared in a timely manner. – Execution results are consistent after PQ is enabled for multi-table JOIN SQL statements. – Backward Index Scan is compatible with ICP. – weight_string functions support LEVEL clauses. – The results of the same SQL statement using different indexes are consistent. – When NDP and PQ are enabled at the same time, recycle LSN is correct.

2.0.42.230600

Table 1-6 Version 2.0.42.230600

Date	Description

2023-08-31	• New features and optimized features – Added support for storing full and incremental backups on read replicas, which reduces the memory and CPU usage of the primary node. – Optimized UNDO damage location: When the undo damage occurs during startup, the undo damage log and the corresponding table name are printed. – Improved the query performance of read replicas. – Added the conversion of IN predicates to subqueries. – Supported large-scale commercial use of the NDP feature. – Optimized execution plans using statement outline. – Supported round functions in PQ. • Fixed issues – The ORDER BY LIMIT and ORDER LIMIT result sets do not overlap when fast sorting and priority queue sorting algorithms are used. – Returned results are correct for PQ statements. – No errors are reported when PREPARE statements are executed. – No PQ assertion errors are reported on UNION queries. – The results of full-text index queries are correct after a read replica is promoted to the primary while a large amount of data is being inserted into the primary node. – When read replicas use the **general_log** and **slow_log** tables, warning logs will not be displayed. – After the value of the parameter **innodb_lock_wait_timeout** is changed, the actual timeout wait time is correct. – When a read replica is promoted to primary, there is no the error "Failed to find page in slice manager". – The percentage for the PWAL scanning progress in the SALSQL log cannot exceed 100%. – When the sqlsmith tool is executed, there is no the error "mysqld coredump" in the EXPLAIN phase of query statements. – In SELECT DISTINCT CAST functions, datetime can be converted to the float type correctly.

2.0.39.230300

Table 1-7 Version 2.0.39.230300

Date	Description
2023-05-11	• New features and optimized features – Supported small-scale instances. – Optimized the solution when DDL statements on standby nodes fail. – Optimized the capacity calculation of salsql. – Supported the restriction on resources of a single SQL statement. – Supported the use of per thread for admin port and local socket. – Optimized the memory of pwalScanner. – Supported the modification of **default_collation_for_utf8mb4** parameter. – Supported diagnosis on large transactions. – Supported the killing of idle transactions. – Accelerated incremental restoration. – Added database and account descriptions. – Supported the acceleration of buffer pool resize. • Fixed issues – Ptrc does not lead to inconsistent execution results of Nestedloop join. – No crash issue occurs when subqueries are sorted using Windows functions. – When using rewrites view, tables are not evaluated to turn left joins into inner joins. – Execution results are returned from decimal data that meets specified filter criteria. – Memory is aligned. – Scan_row is correctly recorded in full logs.

2.0.28.18

Table 1-8 Version 2.0.28.18

Date	Description
2023-05-17	Errors of exceeded sorting memory are not reported for columns containing large JSON data.

2.0.28.17

Table 1-9 Version 2.0.28.17

Date	Description
2023-04-02	Character sets are not used in combination in prepared statements.

2.0.28.16

Table 1-10 Version 2.0.28.16

Date	Description
2023-03-14	• New features Reduced primary/standby latency. • Fixed issues – No error occurs when JSON-related functions are used in prepare statements. – Query results are returned when filter criteria are specified. – No null pointer error is reported after Windows functions generate a temporary disk table. – The crash issue caused by the use of null pointers in Windows functions is resolved. – Prepared statements are executed successfully.

2.0.28.15

Table 1-11 Version 2.0.28.15

Date	Description
2023-01-11	• New features – Supported SQL statement concurrency control. – Optimized read flow control. – Optimized the consistency of primary/standby execution plan. – Pre-created slices asynchronously. • Fixed issues – No crash issue occurs when the system variable **INNODB_VALIDATE_TABLESPACE_PATHS** is disabled and the **undo space truncate** command is executed. – The query of **information_schema.innodb_trx** is fast. – The issue of inconsistent results is resolved: left joins now are turned into inner joins. – The crash issue caused by subquery optimization is resolved. – Values of the Instant field are correctly obtained under concurrent instant DDL and DML operations. – No OOM issue occurs when two InnoDB tables with FTS indexes are loaded. – No OOM issue occurs when the data dictionary of millions of tables is being updated.

2.0.28.12

Table 1-12 Version 2.0.28.12

Date	Description
2022-12-07	Scan errors triggered by Skip Scans are not displayed when a table with virtual columns is updated.

2.0.28.10

Table 1-13 Version 2.0.28.10

Date	Description
2022-11-16	During a primary/standby switchover, databases will not break down when connecting to the standby instance times out.

2.0.28.9

Table 1-14 Version 2.0.28.9

Date	Description
2022-09-23	• The If(...) statement in **Condition_pushdown::replace_columns_in_cond** is modified. • The database does not break down when: – Storage functions are invoked recursively. – Multiple tables are deleted or full-text search is performed. – SQL query statements of multiple window functions are executed. • Users with global permission can successfully run **SHOW CREATE DATABASE**.

2.0.28.7

Table 1-15 Version 2.0.28.7

Date	Description
2022-08-25	The ptrc crash problem in stored procedure is resolved.

2.0.28.4

Table 1-16 Version 2.0.28.4

Date	Description
2022-07-22	• Databases will not break down due to empty accounts. • When a temporary table used for aggregation is updated, BLOB points to the latest data.

2.0.28.1

Table 1-17 Version 2.0.28.1

Date	Description
2022-05-16	• New features – You can enable or disable **orphaned definer check control**. – GaussDB(for MySQL) supports transparent transmission of proxy IP addresses. – You can set the consistency level of your proxy instances to session consistency. • Fixed issues – The data dictionary on standby nodes is updated if DDL statements on the primary node are not submitted. – During a failover, the auto increment of the primary node is not rolled back. – The performance issue of standby nodes is resolved.

2.0.31.220700

Table 1-18 Version 2.0.31.220700

Date	Description
2022-08-12	• New features and performance optimized – Supported SQL statement concurrency control. – Added a limit to concurrent numbers of Faster DDL. – Supported all Faster DDL operations in row format. – Extended full SQL fields. – Optimized flow control. – Supported the quick timeout of **ALTER TABLE**. – Supported the query of plan cache. – Optimized statistics on standby nodes. • Fixed issues – Standby nodes do not break down after partition-table on the primary node is renamed. – The default buffer size of SQL tracer is modified. – When the truncate lsn of standby nodes lags behind, the standby nodes can start successfully. – The execution plan error is not displayed when SQL queries with the same range are executed. – The crash issue caused by empty accounts is resolved. – The crash issue caused by database dropping is resolved.

2 Common Kernel Functions

2.1 Parallel Query

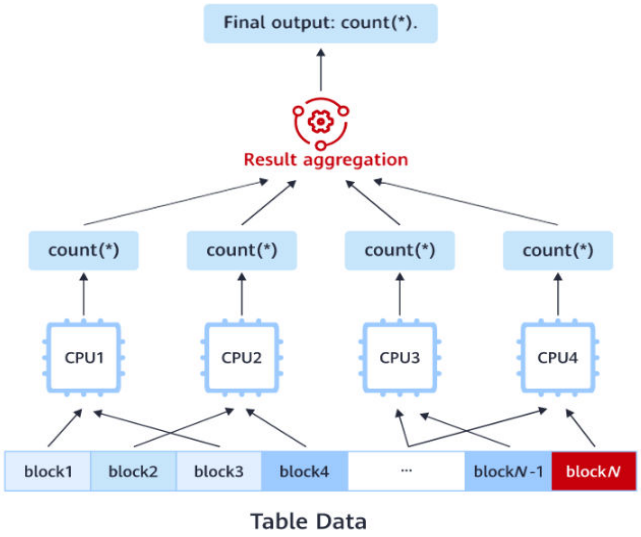
2.1.1 Overview

What Is Parallel Query?

Parallel query (PQ) reduces the processing time of analytical queries to satisfy the low latency requirements of enterprise-class applications. It distributes a query task to multiple CPU cores for computation to shorten the query time. Theoretically, the performance improvement of parallel query is positively correlated with the number of CPU cores. The more CPU cores are used, the higher the performance improvement is.

The following figure shows the count(*) process for a table based on parallel query. Table data is divided into blocks and distributed to multiple cores for parallel computing. Each core processes some data to obtain an intermediate count(*) result, and all the intermediate results are aggregated to obtain the final result.

Figure 2-1 How PQ works



Scenarios

Parallel query is mainly suitable for SELECT statements to query large tables, multiple tables, and a large amount of data. This feature does not benefit extremely short queries.

- Lightweight analysis

  The SQL statements for report queries are complex and time-consuming. Parallel query can improve the efficiency of a single query.
- More available system resources

  Parallel query requires more system resources. You can enable parallel query to improve resource utilization and query efficiency only when the system has a large number of CPUs, low I/O loads, and sufficient memory resources.
- Frequent data queries

  For data-intensive queries, you can use parallel query to improve query processing efficiency, ease network traffic, and reduce pressure on compute nodes.

2.1.2 Precautions

- Parallel query is in the open beta test (OBT) phase. You are advised to use it in the test environment.
- The GaussDB(for MySQL) engine version must be MySQL 8.0.22 or later.
- Both read replicas and primary nodes support parallel query. Parallel query consumes a lot of compute resources (such as CPU and memory). To ensure instance stability, parallel query is disabled by default on primary nodes of GaussDB(for MySQL) instances whose kernel version is 2.0.42.230600 or later. To use parallel query, contact customer service.
- **Parallel query is suitable for the following scenarios:**
  - Full table scans, index scans, index range scans, index reverse scans, index point queries, and index pushdown
  - Single-table queries, multi-table joins, views, subqueries, and partial CTE queries

  - Multiple JOIN algorithms, including BNL JOIN, BKA JOIN, HASH JOIN, NESTED LOOP JOIN, SEMI JOIN, ANTI JOIN, and OUTER JOIN
  - Multiple subqueries, including conditional subqueries, scalar subqueries, some correlated subqueries, non-correlated subqueries, and derived tables
  - Multiple data types, including Integer, Character, Floating Point, and Time
  - Arithmetic expressions (+, -, *, %, /, |, and &), conditional expressions (<, <=, >, >=, <>, BETWEEN/AND, and IN), logical operations (OR, AND, and NOT), and common functions (Character, Integer, and Time), and aggregation functions (COUNT/SUM/AVG/MIN/MAX)

    NOTE

    The COUNT aggregate function can only be executed concurrently when **innodb_parallel_select_count** is disabled.
  - Non-partitioned table queries, and queries for a single partition of partitioned tables
  - ORDER BY, GROUP BY/DISTINCT, LIMIT/OFFSET, WHERE/HAVING, and column projection
  - UNION/UNION ALL queries
  - EXPLAIN statements to view execution plans, including traditional Explain statements, EXPLAIN FORMAT=TREE, and EXPLAIN FORMAT=JSON
- **Parallel query is not suitable for the following scenarios:**
  - Non-query statements
  - Window functions
  - Triggers
  - Prepared statements
  - Spatial indexes
  - System tables, temporary tables, and non-InnoDB tables
  - Full-text indexes
  - Stored procedures
  - Subqueries that cannot be converted to semi-joins
  - Statements that do not meet the **ONLY_FULL_GROUP_BY** rules
  - Index Merge statements
  - HASH JOIN operations, during which data overflows to disks
  - Lock queries, such as SERIALIZABLE isolation level, FOR UPDATE or SHARE LOCK
  - Recursive queries
  - WITH ROLLUP
  - Statements with keyword HIGH_PRIORITY
  - No line of data returned in the execution result. (The execution plan shows: Zero limit, Impossible WHERE, Impossible HAVING, No matching min/max row, Select tables optimized away, Impossible HAVING noticed after reading const tables, or no matching row in const table)
  - Columns with type ZEROFILL. Its column values can be optimized to constants.

  - Generated columns, BLOB, TEXT, JSON, and GEOMETRY
  - Spatial functions (such as SP_WITHIN_FUNC)
  - DISTINCT clauses in aggregate functions, such as SUM(DISTINCT), AVG(DISTINCT), and COUNT(DISTINCT)
  - GROUP_CONCAT
  - JSON_ARRAYAGG and JSON_OBJECTAGG
  - User-defined functions
  - STD, STDDEV, and STDDEV_POP
  - VARIANCE, VAR_POP, and VAR_SAMP
  - BIT_AND, BIT_OR, and BIT_XOR
  - set_user_var
  - RAND functions with parameters
  - json_* (such as json_length and json_type)
  - st_distance
  - get_lock
  - is_free_lock, is_used_lock, release_lock, and release_all_locks
  - sleep
  - xml_str
  - weight_string
  - REF functions (VIEW_REF, OUTER_REF, and AGGREGATE_REF)
  - SHA, SHA1, SHA2, and MD5
  - row_count
  - User-related functions (such as user, current_user, session_user, and system_user)
  - extractvalue
  - GeomCollection, GeometryCollection, LineString, MultiLineString, MultiPoint, MultiPolygon, and Polygon
  - MASTER_POS_WAIT
  - Spatial relationship functions, such as MBRContains, MBRCoveredBy, MBRCovers, MBRDisjoint, MBREquals, MBRIntersects, MBROverlaps, MBRTouches, and MBRWithin
  - Point
  - PS_CURRENT_THREAD_ID()
  - PS_THREAD_ID(CONNECTION_ID())
  - WAIT_FOR_EXECUTED_GTID_SET
  - WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS
  - UNCOMPRESS (COMPRESS ())
  - STATEMENT_DIGEST_TEXT
  - Functions BINARY and CONVERT
  - Functions starting with ST_
- **The execution results of parallel queries may be incompatible with that of serial queries.**

– Number of errors or alarms

If an error or alarm message is displayed during serial queries, the error or alarm message will be displayed in each worker thread during the parallel queries. As a result, the total number of error or alarm messages increases.

```
mysql> SELECT dt1 = 99991231235959.999999 AS a, dt2 = 99991231235959.999999 AS b FROM t7;
+------+------+
| a    | b    |
+------+------+
|    0 |    0 |
|    0 |    0 |
|    0 |    1 |
+------+------+
3 rows in set, 2 warnings (0.00 sec)

mysql> show warnings;
+---------+------+--------------------------------------------------------------------------+
| Level   | Code | Message                                                                  |
+---------+------+--------------------------------------------------------------------------+
| Warning | 1441 | Datetime function: datetime field overflow                               |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
+---------+------+--------------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> set force_parallel_execute=ON;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT dt1 = 99991231235959.999999 AS a, dt2 = 99991231235959.999999 AS b FROM t7;
+------+------+
| a    | b    |
+------+------+
|    0 |    0 |
|    0 |    0 |
|    0 |    1 |
+------+------+
3 rows in set, 12 warnings (0.01 sec)

mysql> show warnings;
+---------+------+--------------------------------------------------------------------------+
| Level   | Code | Message                                                                  |
+---------+------+--------------------------------------------------------------------------+
| Warning | 1441 | Datetime function: datetime field overflow                               |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                               |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                               |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                               |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                               |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
| Warning | 1441 | Datetime function: datetime field overflow                               |
| Warning | 1292 | Incorrect datetime value: '99991231235959.999999' for column 'dt1' at row 1 |
+---------+------+--------------------------------------------------------------------------+
12 rows in set (0.00 sec)
```

– Precision

During the parallel queries, if there is a function type in a SELECT statement, additional stored procedures will be generated in the intermediate results. As a result, compared with serial queries, the precision of the floating point part in parallel queries may be different, and the final result may be slightly different.

```
mysql> create table tb(double_col double);
Query OK, 0 rows affected (0.08 sec)

mysql> insert into tb values (-1.7976931348623157e308),(-1.7976931348623157e308);
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select sum(double_col) from tb;
+-----------------+
| sum(double_col) |
+-----------------+
|               0 |
+-----------------+
1 row in set (0.00 sec)

mysql> set force_parallel_execute=ON;
Query OK, 0 rows affected (0.00 sec)

mysql> select sum(double_col) from tb;
+-------------------------+
| sum(double_col)         |
+-------------------------+
| -1.7976931348623157e308 |
+-------------------------+
1 row in set (0.02 sec)
```


- Truncation

  During the parallel queries, if there is a function type in a SELECT statement, additional stored procedures will be generated in the intermediate results. In this process, the calculation result of the function needs to be cached, and data truncation may occur (generally due to data type conversion, for example, covering a floating-point value to a character string). As a result, the final result is different from the serial queries.

- Sequence of result sets

  Because tasks are executed by multiple worker threads during parallel queries, the sequence of the returned result set may not be consistent with that of serial queries. In the case of a query with LIMIT, this problem is more likely to occur. If fields of GROUP BY are invisible characters, the sequence of the returned result set is also different.

```
mysql> select a,count(*) from t group by a;
+------+----------+
| a    | count(*) |
+------+----------+
|    0 |    32768 |
|    1 |    32768 |
|    2 |    32768 |
|    3 |    32768 |
|    4 |    32768 |
|    5 |    32768 |
|    6 |    32768 |
|    7 |    32768 |
|    8 |    32768 |
|    9 |    32768 |
+------+----------+
10 rows in set (6.37 sec)

mysql> set force_parallel_execute=ON;
Query OK, 0 rows affected (0.00 sec)

mysql> select a,count(*) from t group by a;
+------+----------+
| a    | count(*) |
+------+----------+
|    4 |    32768 |
|    5 |    32768 |
|    6 |    32768 |
|    7 |    32768 |
|    8 |    32768 |
|    9 |    32768 |
|    0 |    32768 |
|    1 |    32768 |
|    2 |    32768 |
|    3 |    32768 |
+------+----------+
10 rows in set (4.35 sec)
```

- UNION ALL result sets

  UNION ALL ignores sort operators. The sequence of the returned result set in parallel execution may be different from that in non-parallel execution. In the case of a query with LIMIT, the result sets are different.

2.1.3 Enabling Parallel Query

System Parameters and Status Variables

- Table 2-1 lists the supported system parameters.

Table 2-1 System parameters

Parameter	Level	Description
force_parallel_execute	Global, Session	Enables or disables parallel query. If this parameter is set to **ON**, parallel query is enabled. • Value range: **ON** and **OFF** • Default value: **OFF**
parallel_max_threads	Global	Maximum number of active threads allowed for parallel execution. If the number of active threads in the current system exceeds the value of this parameter, parallel execution cannot be enabled for new queries. • Value range: 0 to 4294967295 • Default value: **64**
parallel_default_dop	Global, Session	Default parallelism degree for parallel execution. If the parallelism degree is not specified in query statements, this parameter value is used. • Value range: 0 to 1024 • Default value: **4**
parallel_cost_threshold	Global, Session	Cost threshold for enabling parallel execution. If the parallel execution cost of query statements exceeds the value of this parameter, parallel execution is enabled. • Value range: 0 to 4294967295 • Default value: **1000**
parallel_queue_timeout	Global, Session	Waiting time of the parallel execution. If the waiting time exceeds the value of this parameter, new queries will be executed in single-thread mode. • Value range: 0 to 4294967295 • Default value: **0**

Parameter	Level	Description
parallel_memory_limit	Global	Maximum available memory for parallel execution. If the amount of memory used for parallel execution exceeds the value of this parameter, new queries will not be executed in parallel mode. • Value range: 0 to 4294967295 • Default value: **104857600**

- Table 2-2 lists the supported status variables.

Table 2-2 Status variables

Variable	Level	Description
PQ_threads_running	Global	Total number of concurrent threads that are running.
PQ_memory_used	Global	Total memory used for parallel execution.
PQ_threads_refused	Global	Total number of queries that fail to be executed in parallel due to the limit on the total number of threads.
PQ_memory_refused	Global	Total number of queries that fail to be executed in parallel due to the limit on the total memory.

Enabling Parallel Query

You can enable or disable parallel query by configuring system parameters in the console or using hints in SQL statements.

- **Method 1: Configuring system parameters in the console**

  Log in to the console and go to the **Parameters** page to configure the following **system parameters**:

  force_parallel_execute: determines whether to forcibly enable parallel execution.

  parallel_default_dop: indicates the parallelism degree for parallel execution. It controls the number of concurrent threads.

  parallel_cost_threshold: indicates the cost threshold for enabling parallel execution.

  Theses parameters can be modified at any time. The modifications will take effect immediately and you do not need to reboot the instance.

  For example, if you want to forcibly enable parallel execution, set the parallelism degree to **4**, and set the minimum execution cost to **0**, configure the parameters as follows:


```
SET force_parallel_execute=ON
SET parallel_default_dop=4
SET parallel_cost_threshold=0
```

- **Method 2: Using hints in SQL statements**

  Hints can be used to control whether a single statement is executed in parallel. If parallel execution is disabled by default, uses hints to enable parallel execution for specific SQL statements. You can also use hints to disable parallel execution for specified SQL statements.

  Enabling parallel execution:

  Enabling parallel execution: **SELECT /*+ PQ() */... FROM...**

  Enabling parallel execution and setting the parallelism degree to **8**: **SELECT /*+ PQ(8) */... FROM...**

  Enabling parallel execution and set the parallel-executed table to **t1**: **SELECT /*+ PQ(t1) */... FROM...**

  Enabling parallel execution, set the parallel-executed table to **t1**, and set the parallelism degree to **8**: **SELECT /*+ PQ(t1 8) */... FROM...**

  NOTE

  SELECT is followed by PQ (*Hints*). Otherwise, the hints do not take effect. **dop** indicates the parallelism degree of a parallel query and its value ranges from 1 to min(parallel_max_threads, 1024).

  When the **dop** value exceeds the normal range, parallel query does not take effect.

  Disabling parallel execution: When parallel query is enabled, use the **NO_PQ** to disable parallel execution of a single SQL statement.

  SELECT /*+ NO_PQ */ … FROM …

  NOTE

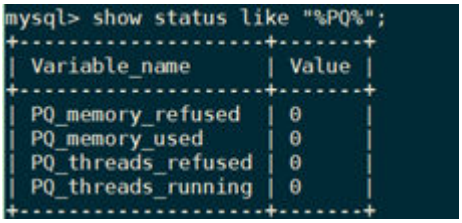
  NO_PQ (*Hints*) takes precedence over PQ (*Hints*). If an SQL statement contains NO_PQ (*Hints*), the SQL statement will not be executed concurrently even if PQ (*Hints*) is configured.

Checking the Statuses of Query Statements Executed in Parallel

Run the following SQL statement to display the statuses of query statements executed in parallel, as shown in **Figure 2-2**.

show status like "%PQ%"

Figure 2-2 Status



Use EXPLAIN to display the parallel execution plans of the query statements, as shown in **Figure 2-3**.

Figure 2-3 Parallel execution plan

```
mysql> explain select/*+ PQ(4) */  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_order
key and l_suppkey = s_suppkey and c_nationkey = s_nationkey and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = 'MIDDLE EAST' and o_orderdate >= date '1994-01-01' and o_orderdate < da
te '1994-01-01' + interval '1' year group by n_name order by revenue desc;
+----+-------------+------------+------------+--------+----------------------+--------------+---------+---------------------------+------+----------+----------------------------------------------+
| id | select_type | table      | partitions | type   | possible_keys        | key          | key_len | ref                       | rows | filtered | Extra                                        |
+----+-------------+------------+------------+--------+----------------------+--------------+---------+---------------------------+------+----------+----------------------------------------------+
|  1 | SIMPLE      | <gather2>  | NULL       | ALL    | NULL                 | NULL         | NULL    | NULL                      |   10 |   100.00 | Parallel execute (4 workers, tpch.supplier)  |
|  2 | SIMPLE      | region     | NULL       | ALL    | PRIMARY              | NULL         | NULL    | NULL                      |    5 |    20.00 | Using where; Using temporary; Using filesort |
|  2 | SIMPLE      | supplier   | NULL       | index  | PRIMARY,SUPPLIER_FK1 | SUPPLIER_FK1 | 4       | NULL                      |   10 |   100.00 | Using index; Using join buffer (Block Nested Loop) |
|  2 | SIMPLE      | nation     | NULL       | eq_ref | PRIMARY,NATION_FK1   | PRIMARY      | 4       | tpch.supplier.S_NATIONKEY |    1 |    20.00 | Using where                                  |
|  2 | SIMPLE      | customer   | NULL       | ref    | PRIMARY,CUSTOMER_FK1 | CUSTOMER_FK1 | 4       | tpch.supplier.S_NATIONKEY |    6 |   100.00 | Using index                                  |
|  2 | SIMPLE      | orders     | NULL       | ref    | PRIMARY,ORDERS_FK1   | ORDERS_FK1   | 4       | tpch.customer.c_custkey   |   15 |    11.11 | Using where                                  |
|  2 | SIMPLE      | lineitem   | NULL       | ref    | PRIMARY              | PRIMARY      | 4       | tpch.orders.o_orderkey    |    4 |    10.00 | Using where                                  |
+----+-------------+------------+------------+--------+----------------------+--------------+---------+---------------------------+------+----------+----------------------------------------------+
7 rows in set, 1 warning (0.02 sec)
```

NOTE

Compared with a traditional execution plan, a parallel execution plan has one more row of records. In the first row of the query result, the parallel-executed tables and parallelism degree are displayed.

2.1.4 Testing Parallel Query Performance

This section describes how to use the TPC-H test tool to test the performance of 22 parallel queries.

The test instance information is as follows:

- Instance specifications: 32 vCPUs | 256 GB
- Kernel version: 2.0.26.1
- Concurrent threads: 16
- Data volume: 100 GB

Procedure

Step 1 Generate test data.

1. Download the shared source code in the TPC-H test from https://github.com/electrum/tpch-dbgen.
2. Find the **makefile.suite** file, modify its contents as follows, and save the modifications:
```
CC      = gcc
# Current values for DATABASE are: INFORMIX, DB2, TDAT (Teradata)
#                                  SQLSERVER, SYBASE, ORACLE
# Current values for MACHINE are:  ATT, DOS, HP, IBM, ICL, MVS,
#                                  SGI, SUN, U2200, VMS, LINUX, WIN32
# Current values for WORKLOAD are:  TPCH
DATABASE= SQLSERVER
MACHINE = LINUX
WORKLOAD = TPCH
```
3. In the root directory of the source code, run the following command to compile and generate the data tool dbgen:
make -f makefile.suite
4. Run the following command to generate 100 GB data:
./dbgen -s 100

Step 2 Log in to the target GaussDB(for MySQL) instance, create a database, and run the following command to create a table:

```
CREATE TABLE nation  ( N_NATIONKEY  INTEGER NOT NULL,
                       N_NAME       CHAR(25) NOT NULL,
                       N_REGIONKEY  INTEGER NOT NULL,
                       N_COMMENT    VARCHAR(152));
```



```
CREATE TABLE region  ( R_REGIONKEY  INTEGER NOT NULL,
                       R_NAME       CHAR(25) NOT NULL,
                       R_COMMENT    VARCHAR(152));
CREATE TABLE part    ( P_PARTKEY     INTEGER NOT NULL,
                       P_NAME        VARCHAR(55) NOT NULL,
                       P_MFGR        CHAR(25) NOT NULL,
                       P_BRAND       CHAR(10) NOT NULL,
                       P_TYPE        VARCHAR(25) NOT NULL,
                       P_SIZE        INTEGER NOT NULL,
                       P_CONTAINER   CHAR(10) NOT NULL,
                       P_RETAILPRICE DECIMAL(15,2) NOT NULL,
                       P_COMMENT     VARCHAR(23) NOT NULL );
CREATE TABLE supplier  ( S_SUPPKEY     INTEGER NOT NULL,
                         S_NAME        CHAR(25) NOT NULL,
                         S_ADDRESS     VARCHAR(40) NOT NULL,
                         S_NATIONKEY   INTEGER NOT NULL,
                         S_PHONE       CHAR(15) NOT NULL,
                         S_ACCTBAL     DECIMAL(15,2) NOT NULL,
                         S_COMMENT     VARCHAR(101) NOT NULL);
CREATE TABLE partsupp  ( PS_PARTKEY     INTEGER NOT NULL,
                         PS_SUPPKEY     INTEGER NOT NULL,
                         PS_AVAILQTY    INTEGER NOT NULL,
                         PS_SUPPLYCOST  DECIMAL(15,2)  NOT NULL,
                         PS_COMMENT     VARCHAR(199) NOT NULL );
CREATE TABLE customer  ( C_CUSTKEY     INTEGER NOT NULL,
                         C_NAME        VARCHAR(25) NOT NULL,
                         C_ADDRESS     VARCHAR(40) NOT NULL,
                         C_NATIONKEY   INTEGER NOT NULL,
                         C_PHONE       CHAR(15) NOT NULL,
                         C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
                         C_MKTSEGMENT  CHAR(10) NOT NULL,
                         C_COMMENT     VARCHAR(117) NOT NULL);
CREATE TABLE orders  ( O_ORDERKEY       INTEGER NOT NULL,
                       O_CUSTKEY        INTEGER NOT NULL,
                       O_ORDERSTATUS    CHAR(1) NOT NULL,
                       O_TOTALPRICE     DECIMAL(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);
CREATE TABLE 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);
```

Step 3 Run the following command to import the generated data to the table:

```
load data INFILE '/path/customer.tbl' INTO TABLE customer FIELDS TERMINATED BY '|';
load data INFILE '/path/region.tbl' INTO TABLE region FIELDS TERMINATED BY '|';
load data INFILE '/path/nation.tbl' INTO TABLE nation FIELDS TERMINATED BY '|';
load data INFILE '/path/supplier.tbl' INTO TABLE supplier FIELDS TERMINATED BY '|';
load data INFILE '/path/part.tbl' INTO TABLE part FIELDS TERMINATED BY '|';
load data INFILE '/path/partsupp.tbl' INTO TABLE partsupp FIELDS TERMINATED BY '|';
load data INFILE '/path/orders.tbl' INTO TABLE orders FIELDS TERMINATED BY '|';
load data INFILE '/path/lineitem.tbl' INTO TABLE lineitem FIELDS TERMINATED BY '|';
```

Step 4 Create an index for the table.


```
alter table region add primary key (r_regionkey);
alter table nation add primary key (n_nationkey);
alter table part add primary key (p_partkey);
alter table supplier add primary key (s_suppkey);
alter table partsupp add primary key (ps_partkey,ps_suppkey);
alter table customer add primary key (c_custkey);
alter table lineitem add primary key (l_orderkey,l_linenumber);
alter table orders add primary key (o_orderkey);
```

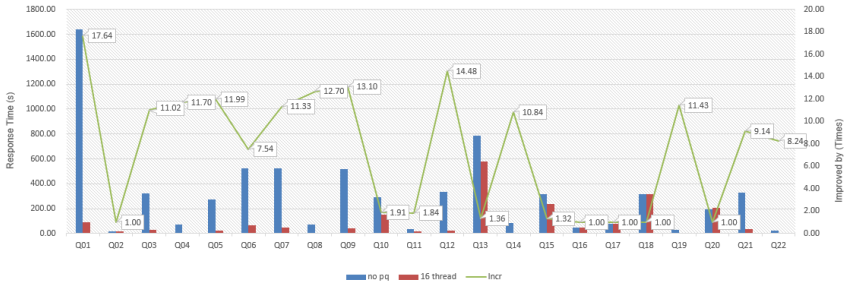
Step 5 Obtain 22 query statements from https://github.com/dragansah/tpch-dbgen/tree/master/tpch-queries and perform corresponding operations.

----End

Test Results

Based on 16-thread parallel execution, the performance of 17 query statements is greatly improved. The query speed of all statements is improved by more than 10 times on average. The following figure shows the TPC-H performance test results.

Figure 2-4 Test results



2.2 Near Data Processing

2.2.1 Overview

What Is Near Data Processing?

Near Data Processing (NDP) is a compute pushdown solution to improve data query efficiency. For data-intensive queries, operations such as column extraction, aggregation calculation, and condition filtering are pushed down to multiple nodes on a distributed storage layer for parallel execution. This reduces query processing pressure on compute nodes, improves parallel processing capabilities, and saves network traffic.

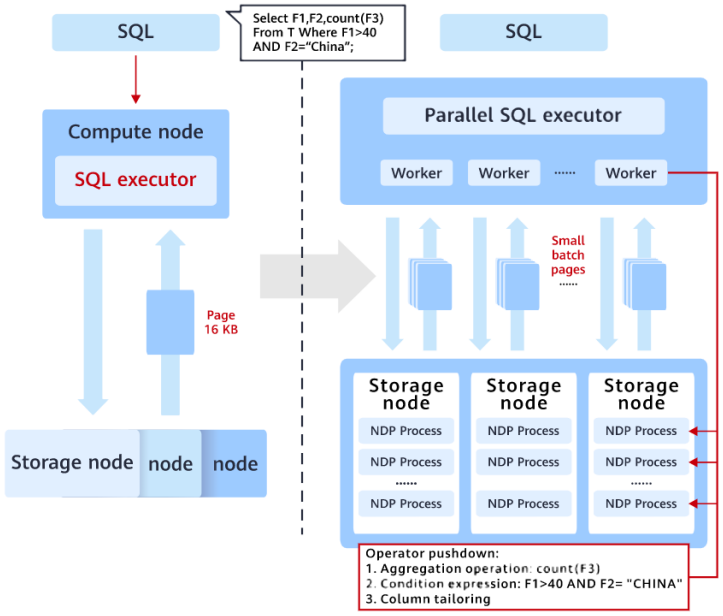
How It Works

GaussDB(for MySQL) uses an architecture with decoupled storage and compute to reduce network traffic. Based on this architecture, NDP is used to accelerate data queries. Without NDP, all raw data needs to be transmitted from storage nodes to compute nodes for query processing. NDP pushed the most I/O-intensive and CPU-intensive query tasks down to storage nodes. Only the required columns and

filtered rows or aggregated results are sent back to compute nodes, greatly reducing network traffic. Additionally, parallel processing across storage nodes reduces the CPU usage of compute nodes and improves the query efficiency.

NDP is integrated with parallel query. Pages are prefetched in batches to realize the entire process in parallel. The query execution efficiency is greatly improved.

Figure 2-5 How NDP works



Scenarios

NDP is suitable for the following scenarios:

- **Projection**

  Column pruning: Only the fields required by a query statement are sent to the compute node.

- **Aggregate**

  Typical aggregation operations include COUNT, SUM, AVG, MAX, MIN, and GROUP BY. Only the aggregated results (not all tuples) are sent to the query engine. COUNT (*) is the most common.

- **SELECT - WHERE clause for filtering**

  Common condition expressions are COMPARE(>=,<=,<,>,==), BETWEEN, IN, AND/OR, and LIKE.

  A filter expression is executed on the storage nodes. Only the rows that meet the conditions are sent to the compute node.

Application Constraints

1. InnoDB tables.
2. Tables with rows in the COMPACT or DYNAMIC format.
3. Primary keys or B-tree indexes. Hash and full-text indexes are not supported.

4. SELECT statements among the DML statements. INSERT INTO SELECT statements and SELECT statements that will lock rows (such as SELECT FOR SHARE/UPDATE) are not supported.
5. Expressions with numeric, log, time, or partial string types (CHAR and VARCHAR). The utf8mb4 and utf8 character sets are supported.
6. Expression predicates with comparison operators (<,>,=,<=,>=,!=), IN, NOT IN, LIKE, NOT LIKE, BETWEEN AND, and AND/OR.

Parameters

Table 2-3 Parameter description

Parameter	Level	Description
ndp_mode	Global **NOTE** • To enable NDP at the global level, contact technical support. • NDP is in the test phase. There are 10 test users in total.	Enables or disables NDP. Value: **off** or **on** Default value: **off**

2.3 DDL Optimization

2.3.1 Parallel DDL

Traditional DDL is designed based on a single core and traditional disks. It takes a long time to perform DDL operations on large tables and the latency is too high. For example, when creating secondary indexes, DDL operations with high latency block subsequent DML queries that depend on new indexes.

GaussDB(for MySQL) supports parallel DDL. When database hardware resources are idle, you can use parallel DDL to accelerate DDL execution, preventing subsequent DML operations from being blocked and shortening the DDL operation window.

Constraints

- This function is supported when the kernel version is 2.0.45.230900 or later.
- This function is only suitable for BTREE secondary indexes.
- This function is not suitable for primary key indexes, spatial indexes, and fulltext indexes. If an SQL statement for concurrently creating indexes contains a primary key index, spatial index, or fulltext index, the client will receive an alarm indicating that the operation does not support concurrent index creation. The statement is executed in single-thread index creation mode. Assume that multiple threads are specified when a primary key index is modified. An alarm will also be reported and the index is created through a single thread.

Enabling Parallel DDL

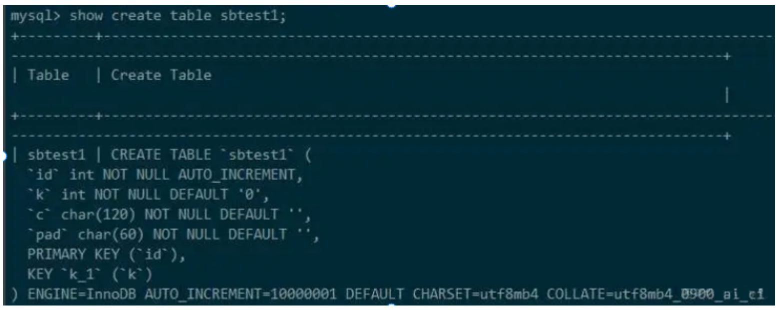
Table 2-4 Parameter description

Parameter	Level	Description
innodb_rds_parallel_index_creation_threads	Global, Session	• Number of threads for concurrently creating indexes. • If the value is greater than 1, concurrent creation is performed. Otherwise, single-thread creation is performed. • Default value: **8**. You advised to set the value to be half of the number of CPU cores and be at most the value of **innodb_rds_parallel_index_creation_threads_max**.

Example

1. Prepare a sysbench table with 100 million data records.

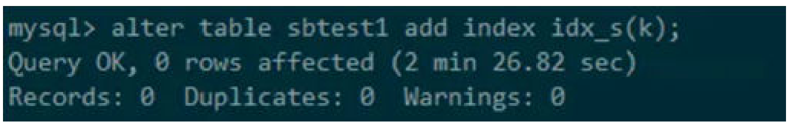
Figure 2-6 Viewing table information



2. Create an index in the **k** field of the table.

   Create an index for the **k** field in the table. If a single thread is used to create the index by default, it should take 146.82 seconds.

Figure 2-7 Creating an index using a single thread



3. Set **innodb_rds_parallel_index_creation_threads** = **4** to use four threads to create the index.

   It should take 38.72 seconds to create the index, 3.79 times faster than with a single thread.

Figure 2-8 Creating an index using multiple threads

```
mysql> set innodb_rds_parallel_index_creation_threads = 4;
Query OK, 0 rows affected (0.00 sec)

mysql> alter table sbtest1 add index idx_p4(k);
Query OK, 0 rows affected (38.72 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

4. Assume that a primary key index needs to be modified. Even if multiple threads are specified, a warning will be received and the index is created using just a single thread.

Figure 2-9 Modifying a primary key index

```
mysql> set innodb_rds_parallel_index_creation_threads = 4;
Query OK, 0 rows affected (0.00 sec)

mysql> alter table sbtest1 add primary key(id,k);
Query OK, 0 rows affected, 1 warning (10 min 11.36 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> show warnings;
+---------+------+-----------------------------------------------------------+
| Level   | Code | Message                                                   |
+---------+------+-----------------------------------------------------------+
| Warning | 7519 | InnoDB: Creating or rebuilding PK in parallel is disallowed. |
+---------+------+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

2.3.2 DDL Fast Timeout

For some specific DDL operations, you can configure their metadata lock (MDL) waiting time, preventing subsequent DML operations from being blocked.

Constraints

- The kernel version is 2.0.45.230900 or later.
- Currently, the following DDL operations are supported: ALTER TABLE, CREATE INDEX, and DROP INDEX.

Enabling DDL Fast Timeout

Table 2-5 Parameter description

Parameter	Level	Description

rds_ddl_lock_wait_timeout	Global, Session	Defines how long that a DDL operation waits for a lock in the current session or global sessions. • Value range: **1** to **31536000** (s). Default value: **31536000**, indicating that the function is disabled. • The actual lock wait timeout for DDL operations is the smaller value between **lock_wait_timeout** and this parameter value. • The actual table lock timeout during DDL execution at the InnoDB layer is the minimum value of **innodb_lock_wait_timeout** and this parameter value. Row locks are not considered.

Example

1. Start a client and add a lock for tables.

   Figure 2-10 Adding a lock

   ```
   mysql>
   mysql>
   mysql> flush tables with read lock;
   Query OK, 0 rows affected (0.01 sec)

   mysql>
   ```

2. Run the following command to check the status of the DDL fast timeout function

   show variables like "%rds_ddl_lock_wait_timeout%";

   Figure 2-11 Querying the status of the DDL fast timeout function

   ```
   mysql> show variables like "rds_ddl_lock_wait_timeout";
   +---------------------------+----------+
   | Variable_name             | Value    |
   +---------------------------+----------+
   | rds_ddl_lock_wait_timeout | 31536000 |
   +---------------------------+----------+
   1 row in set (0.02 sec)
   ```

   As shown in the preceding figure, the value of **rds_ddl_lock_wait_timeout** is **31536000** (default value). The function is disabled. The subsequent operations will wait for a long time.


```
mysql> set rds_ddl_lock_wait_timeout=31536000;
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql>
mysql> alter table lzk.t_lzk drop index indexa;
```

To enable function, referring to **3**.

3. Run the following command to set **rds_ddl_lock_wait_timeout**.

   set rds_ddl_lock_wait_timeout=1;

   Figure 2-12 Configuring parameters

```
mysql> show variables like "%rds_ddl_lock_wait_timeout%";
+----------------------------+----------+
| Variable_name              | Value    |
+----------------------------+----------+
| rds_ddl_lock_wait_timeout | 31536000 |
+----------------------------+----------+
1 row in set (0.01 sec)

mysql> set rds_ddl_lock_wait_timeout=1;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like "%rds_ddl_lock_wait_timeout%";
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| rds_ddl_lock_wait_timeout | 1     |
+----------------------------+-------+
1 row in set (0.01 sec)
```

4. Run the following command to create an index. It is found that the DDL operation times out quickly.

   alter table lzk.t_lzk drop index indexa;

   Figure 2-13 Creating an index

```
mysql> alter table lzk.t_lzk drop index indexa;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

2.3.3 Non-blocking DDL

When a user executes a DDL statement on a table with uncommitted long transactions or large queries, the DDL statement keeps waiting for an MDL-X lock. GaussDB(for MySQL) gives MDL-X locks the highest priority. When a DDL statement is waiting for an MDL-X lock, all new transactions on the table are blocked. As a result, connections are congested, which may even cause the entire service system to break down. Non-blocking DDL allows new transactions to enter the table even if the MDL-X lock cannot be acquired, ensuring the stability of the entire service system.

Prerequisites

The kernel version is 2.0.54.240600 or later.

Constraints

- Enabling non-blocking DDL lowers the priority of DDL statements, and increases the chance of DDL statement execution failure if an MDL-X lock cannot be acquired.
- Non-blocking DDL is only supported for ALTER TABLE, RENAME TABLE, CREATE INDEX, DROP INDEX, and OPTIMIZE TABLE statements.

Parameters

You can set **rds_nonblock_ddl_enable** to enable non-blocking DDL, and then set **rds_nonblock_ddl_retry_times**, **rds_nonblock_ddl_retry_interval**, and **rds_nonblock_ddl_lock_wait_timeout** to specify the maximum number, interval, and timeout period of retries for acquiring an MDL-X lock, respectively.

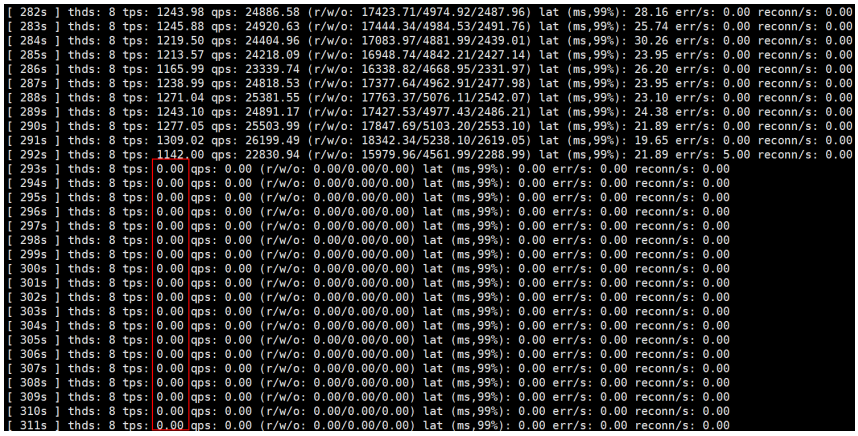
Table 2-6 Parameter description

Parameter	Level	Description
rds_nonblock_ddl_enable	Global, Session	Enables or disables non-blocking DDL. Value range: • **ON**: Non-blocking DDL is enabled. • **OFF**: Non-blocking DDL is disabled. Default value: **OFF**
rds_nonblock_ddl_lock_wait_timeout	Global, Session	Controls how long a statement waits to acquire the MDL-X lock before giving up. Value range: 1 to 31536000, in seconds Default value: **1**
rds_nonblock_ddl_retry_interval	Global, Session	Controls the amount of time between retry attempts for acquiring the MDL-X lock. Value range: 1 to 31536000, in seconds Default value: **6**

Parameter	Level	Description
rds_nonblock_ddl_retry_times	Global, Session	Controls the maximum number of times to retry for acquiring the MDL-X lock. Value range: 0 to 31536000 Default value: **0** If this parameter is set to **0**, the value is calculated based on the smaller value of the **lock_wait_timeout** and **rds_ddl_lock_wait_timeout** parameters. For statements that do not support the **rds_ddl_lock_wait_timeout** parameter, the value is calculated based on the **lock_wait_timeout** parameter.

Example

1. Use sysbench to create a test table **sbtest1** and insert one million rows of data into the table.
   ```
   ./oltp_read_write.lua --mysql-host="cluster_address" --mysql-port="port" --mysql-user="username" --mysql-password="password" --mysql-db="sbtest" --tables=1 --table-size=1000000 --report-interval=1 --percentile=99 --threads=8 --time=6000 prepare
   ```
2. Use **oltp_read_write.lua** in sysbench to simulate user services.
   ```
   ./oltp_read_write.lua --mysql-host="cluster_address" --mysql-port="port" --mysql-user="username" --mysql-password="password" --mysql-db="sbtest" --tables=1 --table-size=1000000 --report-interval=1 --percentile=99 --threads=8 --time=6000 run
   ```
3. Start a new transaction on table **sbtest1** but do not commit the transaction. The transaction holds the MDL lock of table **sbtest1**.
   ```
   begin;
   select * from sbtest1;
   ```
4. Start a new session, add columns to table **sbtest1** when non-blocking DDL is enabled and disabled, and observe the TPS changes.
   ```
   alter table sbtest1 add column d int;
   ```
5. Check the test results.
   - When non-blocking DDL is disabled, the TPS keeps decreasing to zero. The default timeout period is 31,536,000 seconds, which severely affects user services.



- When non-blocking DDL is enabled, the TPS periodically decreases but does not decrease to zero, which has little impact on user services.

```
[ 446s ] thds: 8 tps: 1382.19 qps: 27619.78 (r/w/o: 19329.65/5526.76/2763.38) lat (ms,99%): 20.37 err/s: 0.00 reconn/s: 0.00
[ 447s ] thds: 8 tps: 1474.77 qps: 29473.85 (r/w/o: 20625.22/5899.09/2949.54) lat (ms,99%): 16.71 err/s: 0.00 reconn/s: 0.00
[ 448s ] thds: 8 tps: 1472.88 qps: 29487.17 (r/w/o: 20644.79/5895.60/2946.78) lat (ms,99%): 17.32 err/s: 0.00 reconn/s: 0.00
[ 449s ] thds: 8 tps: 1374.07 qps: 27468.49 (r/w/o: 19230.04/5490.29/2748.15) lat (ms,99%): 21.11 err/s: 0.00 reconn/s: 0.00
[ 450s ] thds: 8 tps: 1500.13 qps: 30034.56 (r/w/o: 21024.79/6009.51/3000.26) lat (ms,99%): 16.12 err/s: 0.00 reconn/s: 0.00
[ 451s ] thds: 8 tps: 715.01 qps: 14305.30 (r/w/o: 10021.21/2847.06/1437.03) lat (ms,99%): 21.50 err/s: 7.00 reconn/s: 0.00
[ 452s ] thds: 8 tps: 725.83 qps: 14619.61 (r/w/o: 10257.62/2910.32/1451.66) lat (ms,99%): 1013.60 err/s: 0.00 reconn/s: 0.00
[ 453s ] thds: 8 tps: 1370.32 qps: 27368.38 (r/w/o: 19152.46/5476.28/2739.64) lat (ms,99%): 20.37 err/s: 0.00 reconn/s: 0.00
[ 454s ] thds: 8 tps: 1325.85 qps: 26544.90 (r/w/o: 18582.83/5309.38/2652.69) lat (ms,99%): 20.37 err/s: 0.00 reconn/s: 0.00
[ 455s ] thds: 8 tps: 1262.04 qps: 25234.86 (r/w/o: 17663.60/5048.17/2523.09) lat (ms,99%): 19.65 err/s: 0.00 reconn/s: 0.00
[ 456s ] thds: 8 tps: 1383.16 qps: 27693.36 (r/w/o: 19381.34/5544.69/2767.33) lat (ms,99%): 19.29 err/s: 0.00 reconn/s: 0.00
[ 457s ] thds: 8 tps: 1527.61 qps: 30491.10 (r/w/o: 21341.47/6094.42/3055.21) lat (ms,99%): 14.46 err/s: 0.00 reconn/s: 0.00
[ 458s ] thds: 8 tps: 705.90 qps: 14121.98 (r/w/o: 9886.58/2819.60/1415.80) lat (ms,99%): 18.95 err/s: 4.00 reconn/s: 0.00
[ 459s ] thds: 8 tps: 753.10 qps: 15166.09 (r/w/o: 10634.46/3025.42/1506.21) lat (ms,99%): 24.83 err/s: 0.00 reconn/s: 0.00
[ 460s ] thds: 8 tps: 1428.37 qps: 28534.57 (r/w/o: 19970.31/5709.51/2854.75) lat (ms,99%): 17.63 err/s: 0.00 reconn/s: 0.00
[ 461s ] thds: 8 tps: 1387.34 qps: 27740.77 (r/w/o: 19423.75/5540.34/2776.68) lat (ms,99%): 22.28 err/s: 0.00 reconn/s: 0.00
[ 462s ] thds: 8 tps: 1429.64 qps: 28642.76 (r/w/o: 20044.93/5738.55/2859.28) lat (ms,99%): 19.29 err/s: 0.00 reconn/s: 0.00
[ 463s ] thds: 8 tps: 1547.19 qps: 30931.86 (r/w/o: 21656.70/6180.77/3094.39) lat (ms,99%): 14.73 err/s: 0.00 reconn/s: 0.00
[ 464s ] thds: 8 tps: 1484.16 qps: 29654.08 (r/w/o: 20756.15/5929.62/2968.31) lat (ms,99%): 18.28 err/s: 0.00 reconn/s: 0.00
[ 465s ] thds: 8 tps: 721.01 qps: 14451.30 (r/w/o: 10123.21/2879.06/1449.03) lat (ms,99%): 20.00 err/s: 7.00 reconn/s: 0.00
[ 466s ] thds: 8 tps: 716.98 qps: 14446.66 (r/w/o: 10128.76/2883.93/1433.97) lat (ms,99%): 995.51 err/s: 0.00 reconn/s: 0.00
[ 467s ] thds: 8 tps: 1381.13 qps: 27611.68 (r/w/o: 19330.88/5518.54/2762.27) lat (ms,99%): 17.95 err/s: 0.00 reconn/s: 0.00
[ 468s ] thds: 8 tps: 1391.96 qps: 27836.19 (r/w/o: 19482.43/5569.84/2783.92) lat (ms,99%): 18.61 err/s: 0.00 reconn/s: 0.00
[ 469s ] thds: 8 tps: 1372.91 qps: 27476.13 (r/w/o: 19237.69/5492.63/2745.81) lat (ms,99%): 17.95 err/s: 0.00 reconn/s: 0.00
[ 470s ] thds: 8 tps: 1271.28 qps: 25417.51 (r/w/o: 17793.86/5081.10/2542.55) lat (ms,99%): 25.74 err/s: 0.00 reconn/s: 0.00
[ 471s ] thds: 8 tps: 1416.82 qps: 28335.37 (r/w/o: 19833.46/5668.27/2833.64) lat (ms,99%): 15.00 err/s: 0.00 reconn/s: 0.00
```

2.3.4 Progress Queries for Creating Secondary Indexes

When PFS is disabled, creating indexes in a production environment can take a lot of time. To help you track DDL progress, this feature displays progress for time-consuming index creation operations even after performance schema has been disabled.

Constraints

- The kernel version of your GaussDB(for MySQL) instance is 2.0.51.240300 or later.
- This feature only displays progress for creating secondary indexes, but not for creating spatial indexes, creating full-text indexes, or other DDL operations.

Functions

This feature is enabled by default. When an index is being created for a table, you can obtain the index creation progress by querying the **INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS** table.

Figure 2-14 Table structure

```
+--------------------+------------------+------+-----+---------+-------+
| Field              | Type             | Null | Key | Default | Extra |
+--------------------+------------------+------+-----+---------+-------+
| THREAD_ID          | bigint unsigned  | NO   |     |         |       |
| QUERY              | varchar(1024)    | NO   |     |         |       |
| START_TIME         | datetime         | NO   |     |         |       |
| ELAPSED_TIME(s)    | int unsigned     | NO   |     |         |       |
| ALTER_TABLE_PHASE  | varchar(128)     | NO   |     |         |       |
| WORK_COMPLETED     | bigint unsigned  | NO   |     |         |       |
| WORK_ESTIMATED     | bigint unsigned  | NO   |     |         |       |
| TIME_REQUIRED(s)   | bigint unsigned  | NO   |     |         |       |
+--------------------+------------------+------+-----+---------+-------+
```

- **THREAD_ID**: the thread ID
- **QUERY**: the statement delivered by the client to create an index
- **START_TIME**: the time when the command for creating an index is delivered

- **ELAPSED_TIME**: the amount of time that has already been used
- **ALTER_TABLE_PHASE**: the current phase
- **WORK_COMPLETED**: the amount of work that has been completed so far
- **WORK_ESTIMATED**: an estimate of the total amount of work required for the entire index creation process
- **TIME_REQUIRED**: an estimate of how much more time is needed
- **WORK_ESTIMATED** and **TIME_REQUIRED** will be adjusted continuously throughout the index creation process, so they do not change linearly.

Example

Step 1 Run the following SQL statement to query the structure of a table:

desc *table_name*;

Example:

Query the structure of table **test_stage**.

desc test_stage;

Figure 2-15 Viewing the table structure

```
mysql> desc test_stage;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| a     | int          | YES  |     | NULL    |       |
| b     | varchar(100) | YES  |     | NULL    |       |
| c     | varchar(100) | YES  |     | NULL    |       |
| d     | varchar(100) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```

Table **test_stage** does not have a secondary index, as indicated by its structure.

Step 2 Run the following SQL statement to add an index for a column in the table:

ALTER TABLE *table_name* **ADD INDEX idxa(***field_name***);**

Example:

Add an index to column **a** in table **test_stage**.

ALTER TABLE test_stage ADD INDEX idxa(a);

Step 3 Run the following SQL statement to query the index creation progress:

SELECT QUERY, ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;

Figure 2-16 Querying the index creation progress

```
mysql> SELECT QUERY,ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;
+-----------------------------------------------------------------+---------------------------------------+
| QUERY                                                           | ALTER_TABLE_PHASE                     |
+-----------------------------------------------------------------+---------------------------------------+
| alter table test_stage add index indexa(a), ALGORITHM=INPLACE | alter table (read PK and internal sort) |
+-----------------------------------------------------------------+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT QUERY,ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;
+-----------------------------------------------------------------+-------------------------+
| QUERY                                                           | ALTER_TABLE_PHASE       |
+-----------------------------------------------------------------+-------------------------+
| alter table test_stage add index indexa(a), ALGORITHM=INPLACE | alter table (merge sort) |
+-----------------------------------------------------------------+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT QUERY,ALTER_TABLE_PHASE FROM INFORMATION_SCHEMA.INNODB_ALTER_TABLE_PROGRESS;
+-----------------------------------------------------------------+---------------------+
| QUERY                                                           | ALTER_TABLE_PHASE   |
+-----------------------------------------------------------------+---------------------+
| alter table test_stage add index indexa(a), ALGORITHM=INPLACE | alter table (insert) |
+-----------------------------------------------------------------+---------------------+
1 row in set (0.00 sec)
```

----End

2.4 Backward Index Scan

Backward Index Scan is a feature that allows for reverse scanning of an index, eliminating the need for sorting. However, it is not compatible with other features like Index Condition Pushdown (ICP), which can lead to decreased performance once the optimizer selects Backward Index Scan.

To address this issue, GaussDB (for MySQL) has added a switch to enable or disable Backward Index Scan dynamically.

Constraints

This feature is only available when the kernel version is 2.0.48.231200 or later.

Enabling Backward Index Scan

Table 2-7 Parameter description

Parameter	Level	Description
optimizer_switch	Global, Session	Enables or disables query optimization. The **backward_index_scan** parameter controls whether the optimizer can use Backward Index Scan. Its default value is **ON**. ● **ON**: The optimizer can use Backward Index Scan. ● **OFF**: The optimizer cannot use Backward Index Scan.

You can also use hints to enable or disable Backward Index Scan. The syntax is as follows:

- Enabling Backward Index Scan during SQL statement execution
  /*+ set_var(optimizer_switch='backward_index_scan=on') */ :
- Disabling Backward Index Scan during SQL statement execution
  /*+ set_var(optimizer_switch='backward_index_scan=off') */ :

Example

1. Enable Backward Index Scan.
   - Set the switch value in the **optimizer_switch** parameter.
     ```
     mysql> set optimizer_switch='backward_index_scan=on';
     Query OK, 0 rows affected (0.00 sec)

     mysql> set optimizer_switch='backward_index_scan=off';
     Query OK, 0 rows affected (0.00 sec)
     ```
   - Use hints to set the switch value in SQL statements.
     ```
     mysql> explain select /*+ set_var(optimizer_switch='backward_index_scan=on') */
     c13,c16 from tt where c10=10 and c7=7 and c12=12 and to_days(c13)=547864 and
     c16 is not null order by c13 desc;

     mysql> explain select /*+ set_var(optimizer_switch='backward_index_scan=off') */
     c13,c16 from tt where c10=10 and c7=7 and c12=12 and to_days(c13)=547864 and
     c16 is not null order by c13 desc;
     ```
2. Check the control effect.

   Run the EXPLAIN statement to check whether the execution plan contains Backward Index Scan.

a. Prepare data.

```
create table tt(
  id int not null primary key,
  a int,
  b int,
  c int,
  key idx_a_b(a, b));
insert into tt values(1,1,1,1),(2,1,2,1),(3,2,3,2),(4,2,4,3),(5,2,4,4);
```

b. When Backward Index Scan is enabled, the optimizer selects this feature to eliminate sorting. Query the **optimizer_switch** parameter to determine whether this feature is enabled.

```
mysql> select @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch:
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=o
n,engine_condition_pushdown=on,index_condition_pushdown=on,mrr=on,mrr_cost_based=on,blo
ck_nested_loop=on,batched_key_access=off,materialization=on,semijoin=on,loosescan=on,firstma
tch=on,duplicateweedout=on,subquery_materialization_cost_based=on,use_index_extensions=on,
condition_fanout_filter=on,derived_merge=on,use_invisible_indexes=off,skip_scan=on,hash_join=o
n,subquery_to_derived=off,prefer_ordering_index=on,hypergraph_optimizer=off,derived_condition
_pushdown=on,derived_merge_no_subquery_check=off,gen_col_partition_prune=off,partial_result
_cache=off,offset_pushdown=off,backward_index_scan=on
1 row in set (0.00 sec)

mysql> explain select * from tt where a = 2 order by b desc;
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+---------------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered |
Extra                |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+---------------------+
|  1 | SIMPLE      | tt    | NULL       | ref  | idx_a_b       | idx_a_b | 5       | const |    3 |   100.00 |
Backward index scan |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+---------------------+
1 row in set, 1 warning (0.00 sec)
```

c. When Backward Index Scan is disabled, the optimizer adds the Sort operator for sorting. Check the following execution plan.

```
mysql> set optimizer_switch='backward_index_scan=off';
Query OK, 0 rows affected (0.00 sec)

mysql> explain select * from tt where a = 2 order by b desc;
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered |
Extra          |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+----------------+
|  1 | SIMPLE      | tt    | NULL       | ref  | idx_a_b       | idx_a_b | 5       | const |    3 |   100.00 |
Using filesort |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------
+----------+----------------+
1 row in set, 1 warning (0.00 sec)
```

Performance Test

When an SQL statement is executed, the optimizer uses Backward Index Scan. The query takes about 4.54s.

```
mysql> explain analyze select detail_record_id, record_id, business_id, business_detail_id, unique_code,
create_time, creator, last_updater, last_update_time, tenant_id, is_usable, operation_time,
detail_operation_type, work_time, operator_id from detail_record d where d.tenant_id=554008 and d.creator
= 585764 and operation_type = 3 and to_days(operation_time) = to_days(now()) and detail_operation_type
is not null order by operation_time desc limit 1\G
*************************** 1. row ***************************
```



```
EXPLAIN: -> Limit: 1 row(s)  (cost=151707.54 rows=1) (actual time=4539.137..4539.137 rows=0 loops=1)
    -> Filter: ((d.creator = 585764) and (to_days(d.operation_time) = <cache>(to_days(now())))) and
(d.detail_operation_type is not null))  (cost=151707.54 rows=263565) (actual time=4539.135..4539.135
rows=0 loops=1)
        -> Index lookup on d using idx_time (tenant_id=554008, operation_type=3; iterate backwards)
(cost=151707.54 rows=2928502) (actual time=0.089..4449.445 rows=1562755 loops=1)
1 row in set (4.54 sec)
```

After hints are used to prevent the optimizer from using Backward Index Scan, there is no incompatibility issue between Backward Index Scan and index condition pushdown in this scenario. As a result, the query time is reduced to approximately 0.37s, and the execution efficiency is significantly improved.

```
mysql> explain analyze select /*+ set_var(optimizer_switch='backward_index_scan=off') */ detail_record_id,
record_id, business_id, business_detail_id, unique_code, create_time, creator, last_updater, last_update_time,
tenant_id, is_usable, operation_time, detail_operation_type, work_time, operator_id from detail_record d
where d.tenant_id=554008 and d.creator = 585764 and operation_type = 3 and to_days(operation_time) =
to_days(now()) and detail_operation_type is not null order by operation_time desc limit 1\G
*************************** 1. row ***************************
EXPLAIN: -> Limit: 1 row(s)  (cost=209431.59 rows=1) (actual time=370.208..370.208 rows=0 loops=1)
    -> Sort: d.operation_time DESC, limit input to 1 row(s) per chunk  (cost=209431.59 rows=2928502)
(actual time=370.207..370.207 rows=0 loops=1)
        -> Filter: ((d.creator = 585764) and (d.detail_operation_type is not null))  (actual
time=370.189..370.189 rows=0 loops=1)
            -> Index lookup on d using idx_time (tenant_id=554008, operation_type=3), with index condition:
(to_days(d.operation_time) = <cache>(to_days(now())))  (actual time=370.188..370.188 rows=0 loops=1)
1 row in set (0.37 sec)
```

2.5 Statement Outline

During the runtime of a MySQL DB instance, the execution plan of a SQL statement often changes, causing database instability. To resolve the issue, GaussDB(for MySQL) provides the Statement Outline function, which uses MySQL optimizer and index hints to stabilize plan execution. GaussDB(for MySQL) also provides a group of management interfaces (**dbms_outln package**) for easy use.

Prerequisites

The kernel version of GaussDB(for MySQL) is 2.0.42.230600 or later.

Precautions

1. Statement Outline is disabled by default. To enable it, see **Enabling Statement Outline**.
2. If Statement Outline is disabled, the performance is not affected. If there are a large number of rules after Statement Outline is enabled, the performance deteriorates.

Description

Statement Outline supports the optimizer hints and index hints of MySQL 8.0.

- Optimizer hints

  Optimizer hints are classified into Global-Level Hint, Table-Level Hint, Index-Level Hint and Join-Order Hints based on the scope (query blocks) and Hint objects. For details, see **Optimizer Hints**.

- Index hints

  Index hints provide the optimizer with information about how to select indexes during query processing without changing the optimizer's policy. There are three common index hints: USE INDEX hint, IGNORE INDEX hint, and FORCE INDEX hint. For details, see **Index Hints**.

Enabling Statement Outline

Step 1 **Log in to the management console.**

Step 2 Click  in the upper left corner and select a region and project.

Step 3 Click  in the upper left corner of the page and choose **Databases** > **GaussDB(for MySQL)**.

Step 4 On the **Instances** page, click the instance name to go to the **Basic Information** page.

Step 5 In the navigation pane, choose **Parameters**.

Step 6 Search for **rds_opt_outline_enabled** in the search box and change its value to **ON**.

Table 2-8 Parameter description

Parameter	Description
rds_opt_outline_enabled	Controls whether to enable Statement Outline. • **ON**: Statement Outline is enabled. • **OFF**: Statement Outline is disabled.

Step 7 Click **Save**.

----End

outline Table

GaussDB(for MySQL) has a built-in system table (**outline**) to store hints. This table is automatically created when the system is started. The SQL statements for creating the table are as follows.

```
CREATE TABLE `mysql`.`outline` (
  `Id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Schema_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Digest` varchar(64) COLLATE utf8_bin NOT NULL,
  `Digest_text` longtext COLLATE utf8_bin,
  `Type` enum('IGNORE INDEX','USE INDEX','FORCE INDEX','OPTIMIZER') CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `Scope` enum('','FOR JOIN','FOR ORDER BY','FOR GROUP BY') CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT '',
  `State` enum('N','Y') CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL DEFAULT 'Y',
  `Position` bigint(20) NOT NULL,
  `Hint` text COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`Id`)
) ENGINE=InnoDB
 DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0 COMMENT='Statement outline'
```


For details about the parameter description, see the following table.

Table 2-9 Parameter description

Parameter	Description
Id	ID of the outline table.
Schema_name	Database name.
Digest	64-byte hash string calculated from **Digest_text** during the hash calculation.
Digest_text	Digest of the SQL statement.
Type	In optimizer hints, the value is **OPTIMIZER**. In index hints, the value can be **USE INDEX**, **FORCE INDEX**, or **IGNORE INDEX**.
Scope	This field is required only for index hints. Its value can be: • **FOR GROUP BY** • **FOR ORDER BY** • **FOR JOIN** • An empty string **NOTE** If this field is set to an empty string, it indicates all types of index hints.
State	Whether Statement Outline is enabled. Its value can be: • **N** • **Y** (default value)
Position	• Optimizer hints Sequence number of the keyword in query blocks on which the hint is applied. Its value starts from **1**. All optimizer hints must be applied to the query block. • Index hints Sequence number of the table on which the hint is applied. Its value starts from **1**.
Hint	• Optimizer hints A complete hint string, for example, **/*+ MAX_EXECUTION_TIME(1000) */** • Index hints A list of index names, for example, **ind_1,ind_2**

Statement Outline Management

There are six local storage rules to manage Statement Outline.

- **add_optimizer_outline**

  Adding optimizer hints

  - **Syntax**

    dbms_outln.add_optimizer_outline(<*Schema_name*>,<*Digest*>,<*Query_block*>,<*Hint*>,<*Query*>);

    NOTE

    You can set either *Digest* or *Query* (original SQL statement). If you set *Query*, DBMS_OUTLN calculates **Digest** and **Digest_text**. You are advised to set *Query* directly.

  - **Description**

Parameter	Mandatory	Type	Description
Schema_name	Yes	VARCHAR	Name of the database to which the statement belongs. This parameter can be set to **NULL** or left blank, the statement cannot be matched.
Digest	No	VARCHAR	Hash value of the statement. You can set this parameter or **Query**. If you do not want to set it to a specific value, set it to an empty string.
Query_block	Yes	INT	Position of the object to which the hint applies. Value range: Greater than or equal to 1
Hint	Yes	VARCHAR	Hint name.

Query	No	VARCHAR	SQL statement. ● You can set either this parameter or **Digest**. If you do not want to set it to a specific value, set it to an empty string. ● If both of them are set, check whether **Digest** and **Query** match. If they do not match, the parameter verification fails and the execution fails.

– **Example**

```
call dbms_outln.add_optimizer_outline("outline_db", '', 1, '/*+ MAX_EXECUTION_TIME(1000) */', "select * from t1 where id = 1");
call dbms_outln.add_optimizer_outline("outline_db", '', 1, '/*+ SET_VAR(foreign_key_checks=OFF) */', "select * from t1 where id = 1");

mysql> explain select * from t1 where id = 1;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+--------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+--------------------------------+
|  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | no matching row in const table |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+--------------------------------+
1 row in set, 1 warning (0.01 sec)

mysql> show warnings;
+-------+------+------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                                                                                    |
+-------+------+------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select /*+ MAX_EXECUTION_TIME(1000) SET_VAR(foreign_key_checks='OFF') */ NULL AS `id`,NULL AS `col1`,NULL AS `col2` from `outline_db`.`t1` where multiple equal(1, NULL) |
+-------+------+------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

- **add_index_outline**

  Adding index hints

  – **Syntax**

  dbms_outln.add_index_outline(<*Schema_name*>,<*Digest*>,<*Position*>,<*Type*>,<*Hint*>,<*Scope*>,<*Query*>);

  NOTE

  You can set either *Digest* or *Query* (original SQL statement). If you set *Query*, DBMS_OUTLN calculates *Digest* and *Digest_text*. You are advised to set *Query* directly.

  – **Description**

Parameter	Mandatory	Type	Description
Schema_name	Yes	VARCHAR	Name of the database to which the statement belongs. This parameter can be set to **NULL** or left blank, the statement cannot be matched.

Digest	No	VARCHAR	Hash value of the statement. Set either this parameter or *Query*. If you do not want to set it to a specific value, set it to an empty string.
Position	Yes	INT	Position of the table to which the index hint applies in the statement. The value must be greater than or equal to **1**.
Type	Yes	ENUM	Hint type. Its value can be: ● **OPTIMIZER** ● **USE INDEX** ● **FORCE INDEX** ● **IGNORE INDEX**
Hint	Yes	VARCHAR	Hint name or index name set. Use commas (,) to separate multiple index names.
Scope	Yes	ENUM	Hint scope. Its value can be: ● **FOR GROUP BY** ● **FOR ORDER BY** ● **FOR JOIN** ● An empty string
Query	No	VARCHAR	SQL statement. ● You can select either or *Digest*. If you do not want to set it to a specific value, set it to an empty string. ● If both of them are set, check whether **Digest** and **Query** match. If they do not match, the parameter verification fails and the execution fails.

– **Example**


```
call dbms_outln.add_index_outline('outline_db', '', 1, 'USE INDEX', 'ind_1', '',"select * from t1
where t1.col1 =1 and t1.col2 ='xpchild'");
```

- **preview_outline**

  Querying the status of the SQL statement matching the statement outline, which can be used for manual verification.

  – **Syntax**

    dbms_outln.preview_outline(<*Schema_name*>,<*Query*>);

  – **Description**

Parameter	Mandatory	Data Type	Description
Schema_name	Yes	VARCHAR	Database name.
Query	Yes	VARCHAR	SQL statement.

  – **Example**

```
mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 and t1.col2 ='xpchild'");
+------------+------------------------------------------------------------------+------------+------------+-------+------------------+
| SCHEMA     | DIGEST                                                           | BLOCK_TYPE | BLOCK_NAME | BLOCK | HINT             |
+------------+------------------------------------------------------------------+------------+------------+-------+------------------+
| outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE      | t1         |     1 | USE INDEX (`ind_1`) |
+------------+------------------------------------------------------------------+------------+------------+-------+------------------+
1 row in set (0.00 sec)
```

- **show_outline**

  Displaying the in-memory hit rate of statement outline

  – **Syntax**

    dbms_outln.show_outline();

  – **Example**

```
mysql> call dbms_outln.show_outline();
+------+------------+------------------------------------------------------------------+------------+-------+------+-------------------------------------------------------+------+----------+------------------------------------------------------------------------------+
| ID   | SCHEMA     | DIGEST                                                           | TYPE       | SCOPE | POS  | HINT                                                  | HIT  | OVERFLOW | DIGEST_TEXT                                                                  |
+------+------------+------------------------------------------------------------------+------------+-------+------+-------------------------------------------------------+------+----------+------------------------------------------------------------------------------+
|   33 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 | OPTIMIZER  |       |    1 | /*+ SET_VAR(foreign_key_checks=OFF) */                |    1 |        0 | SELECT * FROM `t1` WHERE `id` = ?                                            |
|   32 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 | OPTIMIZER  |       |    1 | /*+ MAX_EXECUTION_TIME(1000) */                       |    2 |        0 | SELECT * FROM `t1` WHERE `id` = ?                                            |
|   34 | outline_db | d4dcef634a4a664518e5fb8a21c6ce9b79fccb44b773e86431eb67840975b649 | OPTIMIZER  |       |    1 | /*+ BNL(t1,t2) */                                     |    1 |        0 | SELECT `t1` . `id` , `t2` . `id` FROM `t1` , `t2`                            |
|   35 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f | OPTIMIZER  |       |    2 |  /*+ QB_NAME(subq1) */                                |    2 |        0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )        |
|   36 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f | OPTIMIZER  |       |    1 | /*+ SEMIJOIN(@subq1 MATERIALIZATION, DUPSWEEDOUT) */  |    2 |        0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )        |
|   30 | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | USE INDEX  |       |    1 | ind_1                                                 |    3 |        0 | SELECT * FROM `t1` WHERE `t1` . `col1` = ? AND `t1` . `col2` = ?             |
|   31 | outline_db | 33c71541754093f78a1f2108795cfb45f8b15ec5d6bff76884f4461fb7f33419 | USE INDEX  |       |    2 | ind_2                                                 |    1 |        0 | SELECT * FROM `t1` , `t2` WHERE `t1` . `col1` = `t2` . `col1` AND `t2` . `col2` = ? |
+------+------------+------------------------------------------------------------------+------------+-------+------+-------------------------------------------------------+------+----------+------------------------------------------------------------------------------+
7 rows in set (0.00 sec)
```

    HIT and OVERFLOW description

    i. **HIT** indicates the number of times that the statement outline finds the destination query block or table.

    ii. **OVERFLOW** indicates the number of times that statement outline does not find the destination query block or table.

- **del_outline**

  Deleting a statement outline from the memory and table.

  – **Syntax**

    dbms_outln.del_outline(<*id*>);

  – **Description**

Parameter	Mandatory	Type	Description

id	Yes	INT	Statement outline ID, which is the value in the **id** column in the **mysql.outline** table. The value cannot be left blank.

– **Example**

```
mysql> call dbms_outln.del_outline(1000);
Query OK, 0 rows affected, 2 warnings (0.00 sec)

mysql> show warnings;
+---------+------+----------------------------------------------+
| Level   | Code | Message                                      |
+---------+------+----------------------------------------------+
| Warning | 7521 | Statement outline 1000 is not found in table |
| Warning | 7521 | Statement outline 1000 is not found in cache |
+---------+------+----------------------------------------------+
2 rows in set (0.00 sec)
```

Note: If the statement outline to be deleted does not exist, the system reports a warning. You can run the **show warnings;** command to view the warning content.

- **flush_outline**

  If you modify the statement outline in the **outline** table, you need to make the statement outline take effect again.

  – **Syntax**

  dbms_outln.flush_outline();

  – **Example**

```
update mysql.outline set Position = 1 where Id = 18;
call dbms_outln.flush_outline();
```

Function Verification

To check whether the statement outline takes effect, perform the following steps:

- Use the preview_outline interface.

```
mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 and t1.col2 ='xpchild'");
+------------+------------------------------------------------------------------+------------+------------+-------+---------------------+
| SCHEMA     | DIGEST                                                           | BLOCK_TYPE | BLOCK_NAME | BLOCK | HINT                |
+------------+------------------------------------------------------------------+------------+------------+-------+---------------------+
| outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE      | t1         |     1 | USE INDEX (`ind_1`) |
+------------+------------------------------------------------------------------+------------+------------+-------+---------------------+
1 row in set (0.00 sec)
```

- Run the **EXPLAIN** command.

```
mysql> explain select * from t1 where t1.col1 =1 and t1.col2 ='xpchild';
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref   | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ref  | ind_1         | ind_1 | 5       | const |    1 |    50.00 | Using where |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> show warnings;
+-------+------+------------------------------------------------------------------------------------------+
| Level | Code | Message                                                                                  |
+-------+------+------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select `outline_db`.`t1`.`id` AS `id`,`outline_db`.`t1`.`col1` AS `col1`,`outline_db`.`t1`.`col2` AS `col2` from `outline_db`.`t1` USE INDEX (`ind_1`) USE INDEX (`ind_1`) USE INDEX (`ind_1`) where ((`outline_db`.`t1`.`col1` = 1) and (`outline_db`.`t1`.`col2` = 'xpchild')) |
+-------+------+------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

2.6 Idle Transaction Disconnection

2.6.1 Function

If an idle transaction is not committed for a long time, its rollback will consume database resources and performance. If a large number of idle transactions are not committed and not rolled back for a long time, the performance loss to a database is severe especially during peak hours. GaussDB(for MySQL) can proactively terminate idle transactions. Different parameters are used to control different types of transactions. When idle transactions timed out, they are automatically rolled back and disconnected.

NOTE

This function is supported when the kernel version is 2.0.39.230300 or later.

2.6.2 Parameter Description

```
mysql> show variables like '%idle%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| idle_readonly_transaction_timeout | 0     |
| idle_transaction_timeout          | 0     |
| idle_write_transaction_timeout    | 0     |
+-----------------------------------+-------+
```

Table 2-10 Parameter description

Parameter	Level	Description
idle_readonly_transaction_timeout	global, session	Time in seconds that the server waits for idle read-only transactions before killing the connection. If this parameter is set to **0**, there is not timeout threshold for idle read-only transactions.
idle_transaction_timeout	global, session	Time in seconds that the server waits for common idle transactions before killing the connection. If this parameter is set to **0**, there is not timeout threshold for common idle transactions.
idle_write_transaction_timeout	global, session	Time in seconds that the server waits for idle read/write transactions before killing the connection. If this parameter is set to **0**, there is not timeout threshold for idle read/write transactions.

The parameters **idle_readonly_transaction_timeout** and **idle_write_transaction_timeout** have higher priorities than the parameter **idle_transaction_timeout**.

Figure 2-17 Read-only transactions

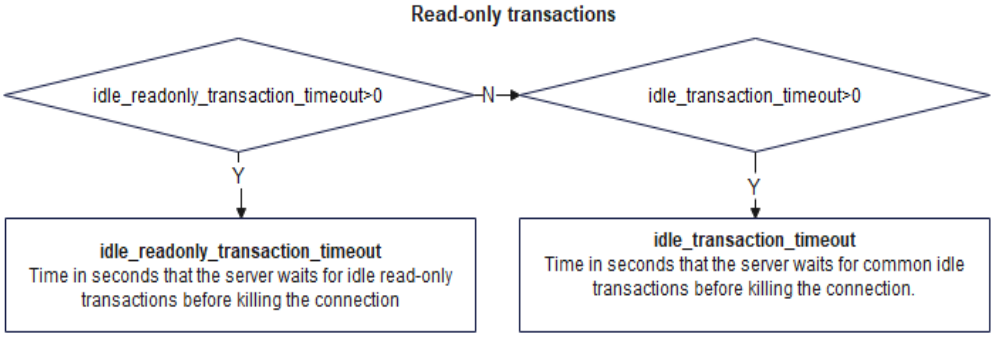
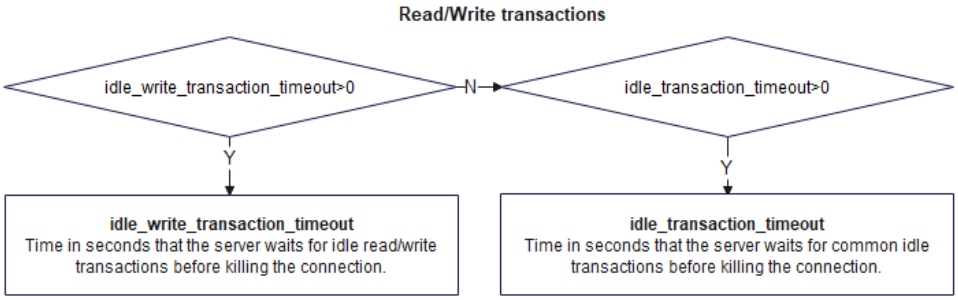


Figure 2-18 Read/Write transactions



2.6.3 Example

1. Set **idle_transaction_timeout** to **10**, **idle_readonly_transaction_timeout** to **0**, and **idle_write_transaction_timeout** to **0**.
   - Read-only transactions

     ```
     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)
     ```

     Wait for 10 seconds and run a query statement again. The following information is displayed.

     ```
     mysql> select * from t1;
     ERROR 2013 (HY000): Lost connection to MySQL server during query
     ```

   - Read/Write Transaction

     Run the **begin** statement to start a transaction and run a query statement. The following information is displayed.

     ```
     mysql> select * from t1;
     +----------+
     | col_int |
     +----------+
     |       1 |
     +----------+
     1 row in set (0.00 sec)
     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)

     mysql> insert into t1 values(2);
     Query OK, 1 row affected (0.00 sec)
     ```

     Wait for 10 seconds and run a query statement again. The following information is displayed.


```
mysql> select * from t1;
ERROR 2013 (HY000): Lost connection to MySQL server during query
```

Reconnect the transaction to the database and run a query statement. If the following information is displayed, the transaction has been rolled back.

```
mysql> select * from t1;
+---------+
| col_int |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)
```

2. Set **idle_write_transaction_timeout** to **15**.
   - Read/Write transactions

     Run the **begin** statement to start a transaction and run a query statement. The following information is displayed.

     ```
     mysql> select * from t1;
     +---------+
     | col_int |
     +---------+
     |       2 |
     +---------+
     1 row in set (0.00 sec)

     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)

     mysql> insert into t1 values(3);
     Query OK, 1 row affected (0.00 sec)
     ```

     Wait for 15 seconds and run a query statement again. The following information is displayed.

     ```
     mysql> select * from t1;
     ERROR 2013 (HY000): Lost connection to MySQL server during query
     ```

     Reconnect the transaction to the database and run a query statement. If the following information is displayed, the transaction has been rolled back.

     ```
     mysql> select * from t1;
     +---------+
     | col_int |
     +---------+
     |       2 |
     +---------+
     1 row in set (0.01 sec)
     ```

3. Set **idle_readonly_transaction_timeout** to **15**.
   - Read-only transactions

     ```
     mysql> begin;
     Query OK, 0 rows affected (0.00 sec)
     ```

     Wait for 15 seconds and run a query statement again. The following information is displayed.

     ```
     mysql> select * from t1;
     ERROR 2013 (HY000): Lost connection to MySQL server during query
     ```

2.7 LIMIT...OFFSET Pushdown

2.7.1 Function

In MySQL Community Edition, If you use LIMIT(*N*) and OFFSET(*P*) in a SELECT statement, the engine layer returns all rows that meet the WHERE condition to the SQL layer for processing. The SQL layer skip *P* rows of data and returns *N* rows of data. When a secondary index needs to access the columns in the primary table, the engine layer returns the table to obtain all required column information. If OFFSET value (*P*) is much greater than the LIMIT value (*N*), the engine layer sends a large amount of data to the SQL layer for processing.

In GaussDB(for MySQL), If you use LIMIT(*N*) and OFFSET(*P*) in a SELECT statement, data is pushed down to the engine layer for processing, speeding up queries.

2.7.2 Usage

Table 2-11 Parameter description

Parameter	Level	Description
optimizer_switch	Global, Session	Enables or disables query optimization. After this parameter is enabled, you can configure **offset_pushdown** to enable or disable LIMIT OFFSET pushdown. • **ON**: enabled • **OFF**: (default value): disabled

You can also add a HINT clause to enable or disable LIMIT OFFSET pushdown.

- **OFFSET_PUSHDOWN(***table_name***)**: enabled
- **NO_OFFSET_PUSHDOWN(***table_name***)**: disabled

Example:

Take a schema as an example in a TPC-H test. After LIMIT OFFSET pushdown is enabled using the parameter or the HINT clause, **Using limit-offset pushdown** is displayed in the **Extra** column when you run EXPLAIN SQL to view an execution plan.

- Enabling LIMIT OFFSET pushdown by configuring **offset_pushdown**

```
mysql> EXPLAIN SELECT * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| id | select_type | table    | partitions | type | possible_keys | key | key_len | ref | rows     | filtered |
Extra                 |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL          | NULL | NULL    | NULL | 59281262 |   100.00 |
Using offset pushdown |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

- Enabling LIMIT OFFSET pushdown by adding a HINT clause


```
mysql> EXPLAIN SELECT /*+ OFFSET_PUSHDOWN() */ * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+----------------------+
| id | select_type | table    | partitions | type | possible_keys | key | key_len | ref | rows     | filtered |
Extra                |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+----------------------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL          | NULL | NULL    | NULL | 59281262 |   100.00 |
Using offset pushdown |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT /*+ NO_OFFSET_PUSHDOWN() */ * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-------+
| id | select_type | table    | partitions | type | possible_keys | key | key_len | ref | rows     | filtered | Extra |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL          | NULL | NULL    | NULL | 59281262 |   100.00 |
NULL |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+-------+
1 row in set, 1 warning (0.00 sec)
```

2.7.3 Performance Tests

- Run following SQL statement (**Q1**) with no predicate conditions to access the primary table.

```
mysql> EXPLAIN SELECT * FROM lineitem LIMIT 10000000,10;
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+----------------------+
| id | select_type | table    | partitions | type | possible_keys | key | key_len | ref | rows     | filtered |
Extra                |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+----------------------+
| 1 | SIMPLE     | lineitem | NULL       | ALL | NULL          | NULL | NULL    | NULL | 59281262 |   100.00 |
Using offset pushdown |
+----+-------------+----------+------------+------+---------------+------+---------+------+----------+----------
+----------------------+
1 row in set, 1 warning (0.00 sec)
```

- Run following SQL statement (**Q2**) with predicate conditions to access the secondary index (including the index range conditions). Information about other columns needs to be obtained from the table.

```
mysql> EXPLAIN SELECT * FROM lineitem WHERE l_partkey > 10 AND l_partkey < 200000 LIMIT
5000000, 10;
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-----------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys                 | key         | key_len | ref |
rows     | filtered | Extra                                         |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-----------------------------------------------+
| 1 | SIMPLE     | lineitem | NULL       | range | i_l_partkey_suppkey,i_l_partkey | i_l_partkey | 4       |
NULL | 10949662 |   100.00 | Using offset pushdown; Using index condition |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-----------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

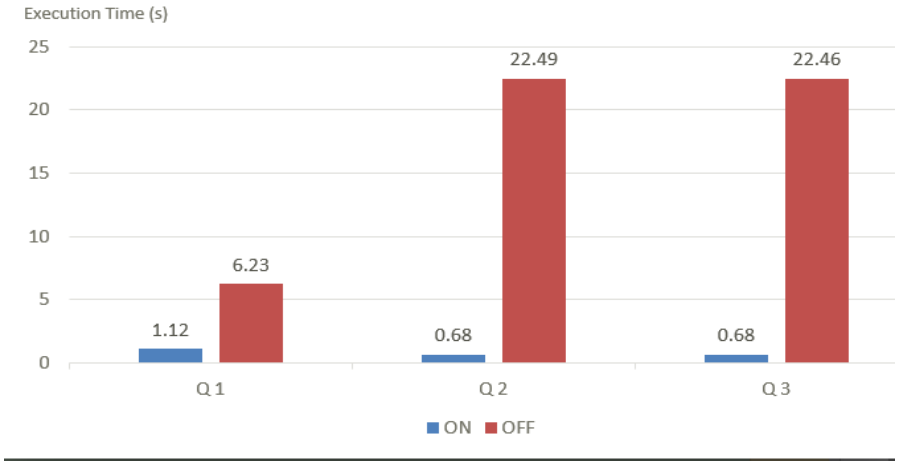
- Run following SQL statement (**Q3**) with predicate conditions and ORDER BY to sort data by index.

```
mysql> EXPLAIN SELECT * FROM lineitem WHERE l_partkey > 10 AND l_partkey < 200000 ORDER BY
l_partkey LIMIT 5000000, 10;
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+-----------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys                 | key         | key_len | ref |
```

```
rows     | filtered | Extra                                      |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+--------------------------------------------+
| 1 | SIMPLE     | lineitem | NULL       | range | i_l_partkey_suppkey,i_l_partkey | i_l_partkey | 4       |
NULL | 10949662 |   100.00 | Using offset pushdown; Using index condition |
+----+-------------+----------+------------+-------+-------------------------------+-------------+---------
+------+----------+----------+--------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

The following figure describes the performance of Q1, Q2, and Q3 when LIMIT OFFSET pushdown is enabled and disabled in the TPC-H benchmark (10 scale).

Figure 2-19 Performance comparison



2.8 Conversion of IN Predicates Into Subqueries

2.8.1 Function

To execute complex queries efficiently, the GaussDB(for MySQL) optimizer can convert some big IN predicates into IN subqueries. The conversion happens if the following conditions are met:

- The GaussDB(for MySQL) kernel version is 2.0.42.230600 or later.
- The number of elements in the **IN** list exceeds the value of **rds_in_predicate_conversion_threshold**.

Overview

In MySQL Community Edition, if column IN (const1, const2, ...) is executed and there is an index on the column, the optimizer usually performs a range scan. The parameter **range_optimizer_max_mem_size** controls the memory available to the range optimizer. If there are many elements in the IN list and the used memory exceeds the parameter value, the range scan will fail and the query performance deteriorates. To solve this problem, you can increase the parameter value to expand the memory that can be used. However, the memory is at the session level. It means that each session occupies the same memory, so the instance may be out of memory. Even if the range optimizer can be used, if the number of

elements in the IN list exceeds the **eq_range_index_dive_limit** value, index statistics, instead of index dive is used. This may cause inaccurate estimation and performance rollback. After IN predicates into subqueries, the optimizer will continue to consider whether to convert the IN clause into a semijoin to improve performance. A specific conversion process is as follows.

```
select ... from lineitem where l_partkey in (...)
```

====>

```
select ... from lineitem where l_partkey in
 (select tb._col_1 from (values (9628136),(19958441),...) tb)
```

2.8.2 Precautions

Supported Query Statements

- SELECT
- INSERT ... SELECT
- REPLACE ... SELECT
- PREPARED STMT and views

Constraints

- Only the constant IN LIST (including statements that do not involve table queries, such as NOW() and ?) is supported.
- Stored procedures, functions, and triggers are not supported.
- NOT IN is not supported. Statements where indexes cannot be used are not supported.

2.8.3 Usage

You can use the **rds_in_predicate_conversion_threshold** parameter to convert IN predicates into subqueries.

NOTE

The default value is **0**, indicating the conversion is disabled. To configure this parameter, contact customer service.

Table 2-12 Parameter description

Parameter	Level	Description
rds_in_predicate_conversion_threshold	Global	Controls the minimum number of elements in the value list of an IN predicate that triggers its conversion to an IN subquery.

Example:

- Query before conversion:

```
mysql> explain select * from t where a in (1,2,3,4,5);
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------
```

```
+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered |
Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------
+-------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | idx1          | NULL | NULL    | NULL |    5 |   100.00 | Using
where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain format=tree select * from t where a in (1,2,3,4,5);
+---------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                     |
+---------------------------------------------------------------------------------------------+
| -> Filter: (t.a in (1,2,3,4,5))  (cost=0.75 rows=5)
    -> Table scan on t  (cost=0.75 rows=5)
 |
+---------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

- Query after conversion:

```
mysql> set rds_in_predicate_conversion_threshold=3;
Query OK, 0 rows affected (0.00 sec)

mysql> explain select * from t where a in (1,2,3,4,5);
+----+-------------+------------------+------------+--------+--------------------+--------------------+---------
+----------+------+----------+-------------------+
| id | select_type | table            | partitions | type   | possible_keys      | key                | key_len | ref
| rows | filtered | Extra             |
+----+-------------+------------------+------------+--------+--------------------+--------------------+---------
+----------+------+----------+-------------------+
|  1 | SIMPLE      | t                | NULL       | ALL    | idx1               | NULL               | NULL    | NULL
|    5 |   100.00 | Using where       |
|  1 | SIMPLE      | <in_predicate_2> | NULL       | eq_ref | <auto_distinct_key> | <auto_distinct_key> |
5       | test.t.a |    1 |   100.00 | IN-list converted |
+----+-------------+------------------+------------+--------+--------------------+--------------------+---------
+----------+------+----------+-------------------+
2 rows in set, 1 warning (0.00 sec)

mysql> explain format=tree select * from t where a in (1,2,3,4,5);
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-----------------+
|
EXPLAIN
                                                                                                   |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-----------------+
| -> Nested loop inner join  (cost=2.50 rows=5)
    -> Filter: (t.a is not null)  (cost=0.75 rows=5)
        -> Table scan on t  (cost=0.75 rows=5)
    -> Single-row index lookup on <in_predicate_2> using <auto_distinct_key> (a=t.a)  (cost=0.27
rows=1)
 |
+-------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------
-----------------+
```

EXPLAIN returns the execution plan. There is **<in_predicate_*>** (*indicates a number) in the table column. It means that the table is a temporary table that stores all data in the IN query.

You can also view **in_to_subquery_conversion** information in the optimize trace.

```
| explain format=tree select * from t where a in (1,2,3,4,5) | {
  "steps": [
```

```
{
  "join_preparation": {
    "select#": 1,
    "steps": [
      {
        "IN_uses_bisection": true
      },
      {
        "in_to_subquery_conversion": {
          "item": "(`t`.`a` in (1,2,3,4,5))",
          "steps": [
            {
              "creating_tmp_table": {
                "tmp_table_info": {
                  "table": "intermediate_tmp_table",
                  "columns": 1,
                  "row_length": 5,
                  "key_length": 5,
                  "unique_constraint": false,
                  "makes_grouped_rows": false,
                  "cannot_insert_duplicates": true,
                  "location": "TempTable"
                }
              }
            },
```

2.8.4 Performance Tests

sysbench is used to perform a benchmark test.

1. Prepare 10 million data records.
   ```
   sysbench /usr/share/sysbench/oltp_read_only.lua --tables=1 --report-interval=10 --table-size=10000000  --mysql-user=root --mysql-password=123456 --mysql-host=127.0.0.1 --mysql-port=3306 --mysql-db=sbtest --time=300 --max-requests=0  --threads=200   prepare
   ```
2. Run a statement where there are 10,000 elements in IN list.
   ```
   select count(*) from sbtest1 where id/k in (... ...);
   ```

The following table lists the performance comparison.

Table 2-13 Performance data

Method	Function Enabled	Function Disabled (Not Suitable for range_opt)	Performance Comparison
Statements using indexes	0.09	2.48	Improved by 26.5 times

2.9 DISTINCT Optimization for Multi-Table Joins

When using multi-table joins with DISTINCT, MySQL 8.0 needs to scan the table join results. When there is a large amount of data in base tables or when there are many table joins, a large amount of data needs to be scanned. As a result, the execution efficiency is low.

To improve DISTINCT query efficiency, particularly in the case of multi-table joins, GaussDB(for MySQL) adds the pruning function to the optimizer to remove unnecessary scanning branches.

Scenarios

- Nested Loop Inner Join + Distinct
- Nested Loop Outer Join + Distinct

Constraints

This feature is only available when the kernel version is 2.0.51.240300 or later.

Enabling DISTINCT Optimization for Multi-Table Joins

Table 2-14 Parameter description

Parameter	Level	Description
rds_nlj_distinct_optimize	Global, Session	Enables or disables DISTINCT optimization. The default value is **OFF**. • **ON**: DISTINCT optimization is enabled. • **OFF**: DISTINCT optimization is disabled.

You can also use hints to enable or disable DISTINCT optimization. The syntax is as follows:

- Enabling DISTINCT optimization

  /*+ SET_VAR(rds_nlj_distinct_optimize=ON) */
- Disabling DISTINCT optimization

  /*+ SET_VAR(rds_nlj_distinct_optimize=OFF) */

Example

1. Use either of the following methods to enable DISTINCT optimization:
   - Run the **SET** command to set the switch value.

     ```
     mysql> SET rds_nlj_distinct_optimize=ON;
     Query OK, 0 rows affected (0.00 sec)

     mysql> SET rds_nlj_distinct_optimize=OFF;
     Query OK, 0 rows affected (0.00 sec)
     ```
   - Use hints to set the switch value in SQL statements.

     ```
     mysql> EXPLAIN ANALYZE SELECT/*+ SET_VAR(rds_nlj_distinct_optimize=ON) */
     DISTINCT tt1.a FROM t1 AS tt1 JOIN t1 AS tt2 JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a;

     mysql> EXPLAIN ANALYZE SELECT/*+ SET_VAR(rds_nlj_distinct_optimize=OFF) */
     DISTINCT tt1.a FROM t1 AS tt1 JOIN t1 AS tt2 JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a;
     ```
2. Check the DISTINCT optimization effect in the multi-table join scenario.

Run the **Explain Analyze/Explain Format=tree** statement to check whether the optimization is applied. If the execution plan contains keyword **with distinct optimization**, the optimization is applied.

The detailed procedure is as follows:

a. Prepare data.

```
CREATE TABLE t1(a INT, KEY(a));
INSERT INTO t1 VALUES(1),(2),(5),(6),(7),(8),(9),(11);
ANALYZE TABLE t1;
```

b. Disable the feature and run the following SQL statements. The optimizer chooses the default execution plan.

```
mysql> SET rds_nlj_distinct_optimize=OFF;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN FORMAT=TREE SELECT DISTINCT tt1.a FROM t1 AS tt1 LEFT JOIN t1
AS tt2 ON TRUE LEFT JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a\G
*************************** 1. row ***************************
EXPLAIN: -> Table scan on <temporary>
    -> Temporary table with deduplication  (cost=29.18 rows=64)
        -> Nested loop left join  (cost=29.18 rows=64)
            -> Left hash join (no condition)  (cost=6.78 rows=64)
                -> Index scan on tt1 using a  (cost=1.05 rows=8)
                -> Hash
                    -> Index scan on tt2 using a  (cost=0.13 rows=8)
            -> Filter: ((tt2.a + 3) = tt3.a)  (cost=0.25 rows=1)
                -> Index lookup on tt3 using a (a=(tt2.a + 3))  (cost=0.25 rows=1)
```

c. Enable the feature and run the following SQL statements. The execution plan contains keyword **with distinct optimization**, which indicates that the optimization is applied.

```
mysql> SET rds_nlj_distinct_optimize=ON;
Query OK, 0 rows affected (0.00 sec)

mysql> EXPLAIN FORMAT=TREE SELECT DISTINCT tt1.a FROM t1 AS tt1 LEFT JOIN t1
AS tt2 ON TRUE LEFT JOIN t1 AS tt3 ON tt2.a + 3 = tt3.a\G
*************************** 1. row ***************************
EXPLAIN: -> Table scan on <temporary>
    -> Temporary table with deduplication  (cost=29.18 rows=64)
        -> Nested loop left join with distinct optimization  (cost=29.18 rows=64)
            -> Left hash join (no condition)  (cost=6.78 rows=64)
                -> Index scan on tt1 using a  (cost=1.05 rows=8)
                -> Hash
                    -> Index scan on tt2 using a  (cost=0.13 rows=8)
            -> Filter: ((tt2.a + 3) = tt3.a)  (cost=0.25 rows=1)
                -> Index lookup on tt3 using a (a=(tt2.a + 3))  (cost=0.25 rows=1)
```

Performance Test

GaussDB(for MySQL) completed the execution in 2.7s and scanned only about 610,000 rows of data. This is a significant improvement in execution efficiency compared to MySQL 8.0, which completed the execution in 186s and scanned 44 million rows of data.

In the following example, when performing a DISTINCT operation on the results after 7 tables were joined, MySQL 8.0.30 took 186s to execute and scanned about 44 million rows of data,

while GaussDB(for MySQL) only took 2.7s and scanned about 610,000 rows of data.

Query statement:

```
select distinct ed.code,et.*
from ele_template et
left join ele_template_tenant ett on ett.template_id = et.id
left join ele_relation tm on tm.tom_id = et.id and tm.jerry_type = 'chapter'
left join ele_relation mv on mv.tom_id = tm.jerry_id and mv.jerry_type = 'variable'
left join ele_relation cv on cv.jerry_id = mv.jerry_id and cv.tom_type = 'column'
left join ele_doc_column edc on edc.id = cv.tom_id
left join ele_doc ed on ed.id = edc.doc_id
where ett.uctenantid = 'mmo0l3f8'
and ed.code = 'contract'
and et.billtype = 'contract'
order by ifnull(et.utime,et.ctime)
desc limit 0,10;
```

Execution plan:

```
+----+-------------+-------+------------+--------+-------------------------+-----------------+---------
+----------------------+------+----------+-----------------------------------------------+
| id | select_type | table | partitions | type   | possible_keys           | key             | key_len | ref                 |
rows | filtered | Extra                                         |
+----+-------------+-------+------------+--------+-------------------------+-----------------+---------
+----------------------+------+----------+-----------------------------------------------+
|  1 | SIMPLE      | ed    | NULL       | ref    | PRIMARY,idx_code        | idx_code        | 203     | const
|    1 |   100.00 | Using index; Using temporary; Using filesort |
|  1 | SIMPLE      | ett   | NULL       | ref    | PRIMARY,idx_uctenanatid | idx_uctenanatid | 203     |
const               |  352 |   100.00 | Using index                                   |
|  1 | SIMPLE      | et    | NULL       | eq_ref | PRIMARY,idx_billtype    | PRIMARY         | 8       |
test.ett.template_id |    1 |    94.57 | Using where                                   |
|  1 | SIMPLE      | tm    | NULL       | ref    | idx_tom_id,idx_jerry_id | idx_tom_id      | 9       |
test.ett.template_id |   59 |    10.00 | Using index condition; Using where; Distinct |
|  1 | SIMPLE      | mv    | NULL       | ref    | idx_tom_id,idx_jerry_id | idx_tom_id      | 9       |
test.tm.jerry_id     |   59 |    10.00 | Using where; Distinct                         |
|  1 | SIMPLE      | cv    | NULL       | ref    | idx_tom_id,idx_jerry_id | idx_jerry_id    | 9       | test.mv.jerry_id
|   47 |    10.00 | Using where; Distinct                         |
|  1 | SIMPLE      | edc   | NULL       | eq_ref | PRIMARY,idx_doc_id      | PRIMARY         | 8       |
test.cv.tom_id       |    1 |    50.00 | Using where; Distinct                         |
+----+-------------+-------+------------+--------+-------------------------+-----------------+---------
+----------------------+------+----------+-----------------------------------------------+
```

Figure 2-20 comparison of execution duration

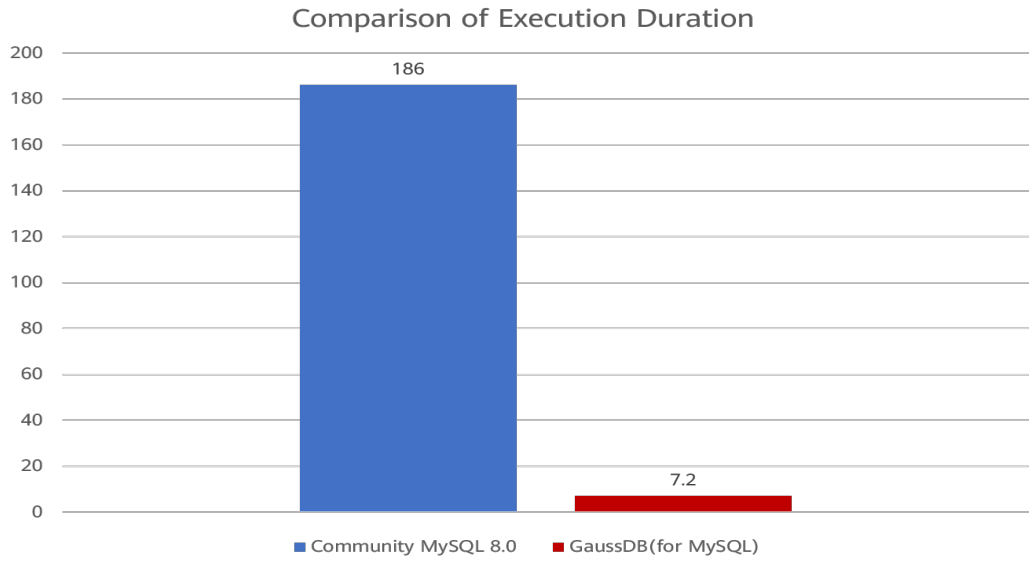
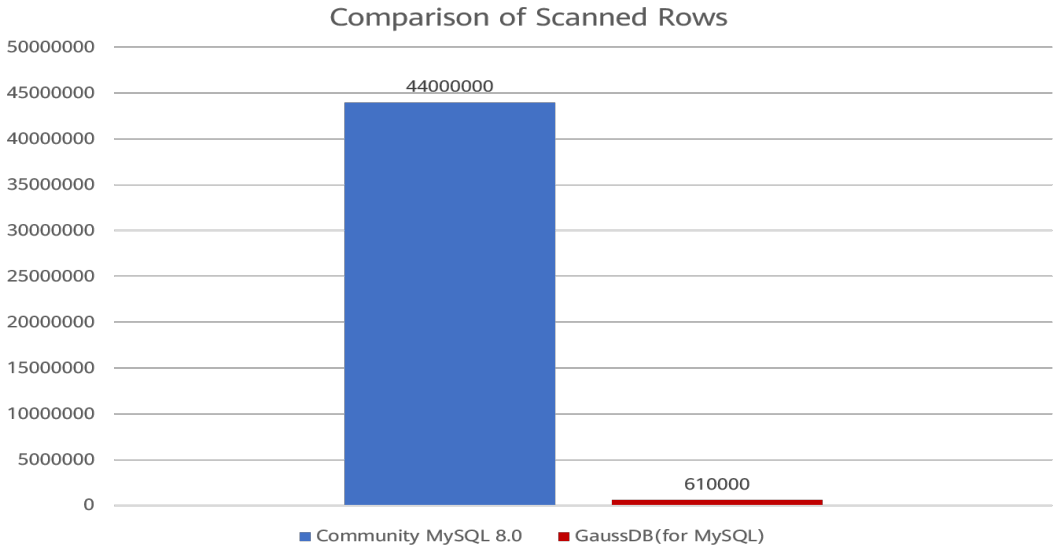


Figure 2-21 comparison of scanned rows



2.10 Diagnosis on Large Transactions

Large transactions affect the health and stability of DB instances. In typical scenarios, long rollbacks of large transactions prolong the upgrade and specification change time. GaussDB(for MySQL) provides diagnosis on large transactions. When there is a large transaction, an alarm is generated to notify you to submit the transaction in a timely manner.

Prerequisites

- The kernel version is 2.0.39.230300 or later.
- The related parameter is configured based on the following conditions:
  - If the kernel version is earlier than 2.0.45.230900, set the value of **log-bin** is **ON**.
  - If the kernel version is 2.0.45.230900 or later, set the value of **rds_global_sql_log_bin** to **ON**.

Usage

1. Configure the parameter **rds_warn_max_binlog_cache_size** as required.

Table 2-15 Parameter description

Parameter	Level	Description
rds_warn_max_binlog_cache_size	global	Controls the maximum binlog cache size for a transaction. If the size in a transaction exceeds the parameter value, a WARNING message is reported. Default value: **18446744073709547520** Value range: **4096** to **18446744073709547520**

To prevent multiple WARNING messages from being sent to the client, a WARNING message can be sent to the client once for each statement in a transaction.

In this example, **rds_warn_max_binlog_cache_size** is set to **40960** (40 KB).

```
mysql> CREATE TABLE t1 (
    -> a longtext
    -> ) DEFAULT CHARSET=latin1;
Query OK, 0 rows affected (0.12 sec)

mysql> show variables like 'rds_warn_max_binlog_cache_size';
+--------------------------------+-------+
| Variable_name                  | Value |
+--------------------------------+-------+
| rds_warn_max_binlog_cache_size | 40960 |
+--------------------------------+-------+
1 row in set (0.01 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> show warnings;
+---------+------
+-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
----+
| Level   | Code | Message                          |
+---------+------
+-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
----+
| Warning | 4008 | Recommend you to INSERT/UPDATE/DELETE rows in batches by multiple
transactions. The current transaction required more than 'rds_warn_max_binlog_cache_size' (40960)
bytes of storage. Which shall cause replication latency. Please commit it. |
+---------+------
+-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
----+
1 row in set (0.00 sec)
```



```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.01 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',50000));
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> show warnings;
+---------+------
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Level   | Code | Message           |
+---------+------
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Warning | 4008 | Recommend you to INSERT/UPDATE/DELETE rows in batches by multiple transactions. The current transaction required more than 'rds_warn_max_binlog_cache_size' (40960) bytes of storage. Which shall cause replication latency. |
+---------+------
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

2. Check the binlog cache size of the transactions in the current connection.

```
mysql> CREATE TABLE t1 (
->   a longtext
-> ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
Query OK, 0 rows affected (0.10 sec)

mysql> SHOW STATUS LIKE 'Rds_binlog_trx_cache_size';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| Rds_binlog_trx_cache_size | 0     |
+---------------------------+-------+
1 row in set (0.04 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 VALUES (REPEAT('a',20000));
Query OK, 1 row affected (0.01 sec)

mysql> SHOW STATUS LIKE 'Rds_binlog_trx_cache_size';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| Rds_binlog_trx_cache_size | 20150 |
+---------------------------+-------+
1 row in set (0.05 sec)
mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW STATUS LIKE 'Rds_binlog_trx_cache_size';
```

```
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Rds_binlog_trx_cache_size | 0     |
+----------------------------+-------+
1 row in set (0.09 sec)
```

3. Check the binlog cache size of transactions in all connections.

```
mysql> SHOW GLOBAL STATUS LIKE 'rds_binlog_trx_cache_size';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Rds_binlog_trx_cache_size | 40300 |
+----------------------------+-------+
1 row in set (0.05 sec)
```

2.11 Enhanced Partitioned Tables

2.11.1 Subpartitioning

2.11.1.1 Overview

GaussDB(for MySQL) partitioned tables are fully compatible with the syntax and functions of MySQL Community Edition. Compared with MySQL Community Edition, GaussDB(for MySQL) provides more functions in terms of partitioned tables. It supports more partition types and combinations, and allows you to use partitioned tables easily and efficiently.

GaussDB(for MySQL) is compatible with the following MySQL partition types:

- HASH
- KEY
- RANGE
- LIST
- RANGE-HASH
- RANGE-KEY
- LIST-HASH
- LIST-KEY

A composite partition comprises both partitions and subpartitions.

GaussDB(for MySQL) supports the following composite partition types:

- RANGE-RANGE
- RANGE-LIST
- LIST-RANGE
- LIST-LIST
- HASH-HASH
- HASH-KEY
- HASH-RANGE
- HASH-LIST

- KEY-KEY
- KEY-HASH
- KEY-RANGE
- KEY-LIST

2.11.1.2 Precautions

- The kernel version of GaussDB(for MySQL) must be 2.0.48.231200 or later.
- To use extended partition types, submit an application by choosing **Service Tickets > Create Service Ticket** in the upper right corner of the management console.

2.11.1.3 RANGE-RANGE

Constraints

- The RANGE type requires that the partition key **value** or **value_list** defined for each partition be monotonically increasing.
- **MAXVALUE** must be at the end.
- The NULL value is considered to be infinitely small. It is always inserted into the first partition definition.
- A subpartition in each partition can be considered as a new RANGE partition. All rules and constraints are the same as those of RANGE partitions.

Syntax

The following statement is used to create one or more RANGE-RANGE partitioned tables where each partition may contain one or more RANGE subpartitions:

```
CREATE TABLE ... PARTITION BY RANGE {(expr) | COLUMNS(column_list)}
  SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
[(partition_definition [, partition_definition] ...)];
```

partition_definition is:

```
PARTITION partition_name
    VALUES LESS THAN {(value | MAXVALUE | value_list) | MAXVALUE}
[(subpartition_definition [, subpartition_definition] ...)]
```

subpartition_definition is:

```
SUBPARTITION subpartition_name
    VALUES LESS THAN {value | value_list | MAXVALUE}
```

Table 2-16 Parameters

Parameter	Description
expr	The expression of the partition. Currently, only the INT type is supported.

Parameter	Description
column_list	The list of partition key columns. It is used in RANGE COLUMNS(). Expressions are not supported. Multiple columns are supported.
value	The boundary value of the partition.
value_list	The list of the values of the partition key columns. It is used in RANGE COLUMNS().
MAXVALUE	The maximum value of the partition.
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Examples

- **Create a RANGE-RANGE partitioned table:**

```
CREATE TABLE tbl_range_range (col1 INT, col2 INT, col3 varchar(20))
PARTITION BY RANGE(col1)
SUBPARTITION BY RANGE(col2)
(
 PARTITION p0 VALUES LESS THAN (1000) (
  SUBPARTITION s0 VALUES LESS THAN(100),
  SUBPARTITION s1 VALUES LESS THAN(MAXVALUE)
 ),
 PARTITION p1 VALUES LESS THAN (2000)
 (
  SUBPARTITION s2 VALUES LESS THAN(100),
  SUBPARTITION s3 VALUES LESS THAN(200)
 ),
 PARTITION p2 VALUES LESS THAN (MAXVALUE)
 (
  SUBPARTITION s4 VALUES LESS THAN(200),
  SUBPARTITION s5 VALUES LESS THAN(400)
 )
);
```

- **Create a RANGE COLUMNS-RANGE partitioned table:**

```
CREATE TABLE tbl_range_col_range (col1 INT, col2 INT, col3 INT)
PARTITION BY RANGE COLUMNS(col1, col2)
SUBPARTITION BY RANGE(col3)
(
 PARTITION p1 VALUES LESS THAN(1000, MAXVALUE)(
  SUBPARTITION s0 VALUES LESS THAN(100),
  SUBPARTITION s1 VALUES LESS THAN(MAXVALUE)
 ),
 PARTITION p2 VALUES LESS THAN(2000, MAXVALUE)(
  SUBPARTITION s2 VALUES LESS THAN(100),
  SUBPARTITION s3 VALUES LESS THAN(200)
 ),
 PARTITION p3 VALUES LESS THAN(MAXVALUE, MAXVALUE)(
  SUBPARTITION s4 VALUES LESS THAN(200),
  SUBPARTITION s5 VALUES LESS THAN(400)
```



```
  )
);
```

2.11.1.4 RANGE-LIST

Constraints

- The LIST type requires that **value** or **value_list** in the same or different partition definitions be unique.
- You can only insert or query the NULL value when it is contained in **value**. Otherwise, the NULL value does not comply with definitions and cannot be inserted.
- A subpartition in each partition can be considered as a new LIST partition. All rules and constraints are the same as those of LIST partitions. The definitions of subpartitions in different partitions can be different.

Syntax

The following statement is used to create one or more RANGE-LIST partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE ... PARTITION BY RANGE {(expr) | COLUMNS(column_list)}
  SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
[(partition_definition [, partition_definition] ...)];
```

partition_definition is:

```
PARTITION partition_name
    VALUES LESS THAN {(value | value_list) | MAXVALUE}
[(subpartition_definition [, subpartition_definition] ...)]
```

subpartition_definition is:

```
SUBPARTITION subpartition_name
    VALUES IN {(value | value_list)}
```

Table 2-17 Parameters

Parameter	Description
expr	The expression of the partition. Currently, only the INT type is supported.
column_list	The list of partition key columns. It is used in RANGE COLUMNS(). Expressions are not supported. Multiple columns are supported.
value	The boundary value of the partition.
value_list	The list of the values of the partition key columns. It is used in RANGE COLUMNS().
MAXVALUE	The maximum value of the partition.
partition_name	The name of the partition. The name must be unique within the table.

Parameter	Description
subpartition_name	The name of the subpartition. The name must be unique within the table.

Examples

- **Create a RANGE-LIST partitioned table:**

```
CREATE TABLE tbl_range_list (col1 INT, col2 INT, col3 varchar(20))
PARTITION BY RANGE(col1)
   SUBPARTITION BY LIST(col2)
(
 PARTITION m1 VALUES LESS THAN(1000) (
   SUBPARTITION p0 VALUES in (1, 2),
   SUBPARTITION p1 VALUES in (3, 4),
   SUBPARTITION p2 VALUES in (5, 6)
 ),
 PARTITION m2 VALUES LESS THAN(2000) (
   SUBPARTITION p3 VALUES in (1, 2),
   SUBPARTITION p4 VALUES in (3, 4),
   SUBPARTITION p5 VALUES in (5, 6)
 ),
 PARTITION m3 VALUES LESS THAN(MAXVALUE) (
   SUBPARTITION p6 VALUES in (1, 2),
   SUBPARTITION p7 VALUES in (3, 4),
   SUBPARTITION p8 VALUES in (5, 6)
 )
);
```

- **Create a RANGE COLUMNS-LIST partitioned table:**

```
CREATE TABLE tbl_range_columns_list
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY RANGE COLUMNS(col4)
   SUBPARTITION BY LIST(col1)
(
 PARTITION dp1 VALUES LESS THAN('2023-01-01')(
   SUBPARTITION p0 VALUES in (1, 2),
   SUBPARTITION p1 VALUES in (3, 4),
   SUBPARTITION p2 VALUES in (5, 6)
 ),
 PARTITION dp2 VALUES LESS THAN('2024-01-01')(
   SUBPARTITION p3 VALUES in (1, 2),
   SUBPARTITION p4 VALUES in (3, 4),
   SUBPARTITION p5 VALUES in (5, 6)
 ),
 PARTITION dp3 VALUES LESS THAN('2025-01-01')(
   SUBPARTITION p6 VALUES in (1, 2),
   SUBPARTITION p7 VALUES in (3, 4),
   SUBPARTITION p8 VALUES in (5, 6)
 )
);
```


2.11.1.5 LIST-RANGE

Constraints

- The LIST type requires that **value** or **value_list** in the same or different partition definitions be unique.
- You can only insert or query the NULL value when it is contained in **value**. Otherwise, the NULL value does not comply with definitions and cannot be inserted.
- A subpartition in each partition can be considered as a new LIST partition. All rules and constraints are the same as those of LIST partitions. The definitions of subpartitions in different partitions can be different.

Syntax

The following statement is used to create one or more LIST-RANGE partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY LIST {(expr) | COLUMNS(column_list)}
   SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
   (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name VALUES IN (value_list)
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name VALUES LESS THAN {value | value_list | MAXVALUE}
```

Table 2-18 Parameters

Parameter	Description
expr	The expression of the partition. Currently, only the INT type is supported.
column_list	The list of partition key columns. It is used in LIST COLUMNS(). Expressions are not supported.
value	The boundary value of the partition.
value_list	The list of the values of the partition key columns. It is used in LIST COLUMNS().
MAXVALUE	The maximum value of the partition.
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Examples

- **Create a LIST-RANGE partitioned table:**

```
CREATE TABLE tbl_list_range
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY LIST (col1)
SUBPARTITION BY RANGE(col2)
(
 PARTITION p0 VALUES in (1, 2)(
   SUBPARTITION s0 VALUES LESS THAN(1000),
   SUBPARTITION s1 VALUES LESS THAN(2000)
 ),
 PARTITION p1 VALUES in (3, 4)(
   SUBPARTITION s2 VALUES LESS THAN(1000),
   SUBPARTITION s3 VALUES LESS THAN(MAXVALUE)
 ),
 PARTITION p2 VALUES in (5, 6)(
   SUBPARTITION s4 VALUES LESS THAN(3000),
   SUBPARTITION s5 VALUES LESS THAN(MAXVALUE)
 )
);
```

- **Create a LIST COLUMNS-RANGE partitioned table:**

```
CREATE TABLE tbl_list_columns_range
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY LIST COLUMNS(col3)
SUBPARTITION BY RANGE(month(col4))
(
 PARTITION europe VALUES in ('FRANCE', 'ITALY')(
   SUBPARTITION q1_2012 VALUES LESS THAN(4),
   SUBPARTITION q2_2012 VALUES LESS THAN(7)
 ),
 PARTITION asia VALUES in ('INDIA', 'PAKISTAN')(
   SUBPARTITION q1_2013 VALUES LESS THAN(4),
   SUBPARTITION q2_2013 VALUES LESS THAN(7)
 ),
 PARTITION americas VALUES in ('US', 'CANADA')(
   SUBPARTITION q1_2014 VALUES LESS THAN(4),
   SUBPARTITION q2_2014 VALUES LESS THAN(7)
 )
);
```

2.11.1.6 LIST-LIST

Syntax

The following statement is used to create one or more LIST-LIST partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
```



```
PARTITION BY LIST {(expr) | COLUMNS(column_list)}
SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
(partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name VALUES IN (value_list)
   (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name VALUES IN (value_list)
```

Table 2-19 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
column_list	The list of partition key columns. It is used in LIST COLUMNS(). Expressions are not supported.
value_list	The list of the values of the partition key columns. It is used in LIST COLUMNS().
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Examples

- **Create a LIST-LIST partitioned table:**

```
CREATE TABLE tbl_list_list
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY LIST (col1)
SUBPARTITION BY LIST (col2)
(
 PARTITION p0 VALUES in (1, 2)(
   SUBPARTITION partno0 VALUES in (1, 2),
   SUBPARTITION partno1 VALUES in (3, 4),
   SUBPARTITION partno2 VALUES in (5, 6)
 ),
 PARTITION p1 VALUES in (3, 4)(
   SUBPARTITION partno3 VALUES in (1, 2),
   SUBPARTITION partno4 VALUES in (3, 4),
   SUBPARTITION partno5 VALUES in (5, 6)
 ),
 PARTITION p2 VALUES in (5, 6)(
   SUBPARTITION partno6 VALUES in (1, 2),
```



```
    SUBPARTITION partno7 VALUES in (3, 4),
    SUBPARTITION partno8 VALUES in (5, 6)
  )
);
```

- **Create a LIST COLUMNS-LIST partitioned table:**

```
CREATE TABLE tbl_list_columns_list
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY LIST COLUMNS(col3)
SUBPARTITION BY LIST (col1)
(
   PARTITION europe VALUES in ('FRANCE', 'ITALY')(
    SUBPARTITION p0 VALUES in (1, 2),
        SUBPARTITION p1 VALUES in (3, 4),
        SUBPARTITION p2 VALUES in (5, 6)
   ),
   PARTITION asia VALUES in ('INDIA', 'PAKISTAN')(
    SUBPARTITION p3 VALUES in (1, 2),
        SUBPARTITION p4 VALUES in (3, 4),
        SUBPARTITION p5 VALUES in (5, 6)
   ),
   PARTITION americas VALUES in ('US', 'CANADA')(
    SUBPARTITION p6 VALUES in (1, 2),
        SUBPARTITION p7 VALUES in (3, 4),
        SUBPARTITION p8 VALUES in (5, 6)
   )
);
```

2.11.1.7 HASH-HASH

Constraints

- The definitions of a HASH partitioned table can be omitted. If **PARTITIONS num** is specified, that exact number of partition definitions are created. Otherwise, one partition definition is created by default.
- If you want to omit definitions of subpartitions, ensure that no definition is provided for any of the subpartitions. Otherwise, you need to specify the partition definition for each subpartition.

Syntax

The following statement is used to create one or more HASH-HASH partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] HASH(expr) [PARTITIONS num]
   SUBPARTITION BY [LINEAR] HASH(expr) [SUBPARTITIONS sub_num]
   [partition_definition [, partition_definition] ...];
```

partition_definition is:

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name
```

Table 2-20 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
num	The number of partitions. It is only valid for HASH or KEY partitions.
sub_num	The number of subpartitions. It is only valid for HASH or KEY subpartitions.
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Example

Create a HASH-HASH partitioned table:

```
CREATE TABLE tbl_hash_hash
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY HASH(col1) PARTITIONS 9
  SUBPARTITION BY HASH(col2) SUBPARTITIONS 3;
```

2.11.1.8 HASH-KEY

Constraints

- The definitions of a KEY partitioned table can be omitted. If **PARTITIONS num** is specified, that exact number of partition definitions are created. Otherwise, one partition definition is created by default.
- If you want to omit definitions of subpartitions, ensure that no definition is provided for any of the subpartitions. Otherwise, you need to specify the partition definition for each subpartition.

Syntax

The following statement is used to create one or more HASH-KEY partitioned tables where each partition may contain one or more subpartitions:


```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] HASH(expr) [PARTITIONS num]
   SUBPARTITION BY [LINEAR] KEY(expr) [SUBPARTITIONS sub_num]
   (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name
```

Table 2-21 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Example

Create a HASH-KEY partitioned table:

```
CREATE TABLE tbl_hash_key
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY HASH(col1) PARTITIONS 3
   SUBPARTITION BY KEY(col3) SUBPARTITIONS 2;
```

2.11.1.9 HASH-RANGE

Syntax

The following statement is used to create one or more HASH-RANGE partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] HASH(expr)
   SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
   (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name
   (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name VALUES LESS THAN {value | valuse_list | MAXVALUE}
```

Table 2-22 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
column_list	The list of partition key columns. It is used in LIST COLUMNS(). Expressions are not supported.
value	The boundary value of the partition.
value_list	The list of the values of the partition key columns. It is used in LIST COLUMNS().
MAXVALUE	The maximum value of the partition.
partition_name subpartition_name	The name of the partition. The name must be unique within the table. The name of the subpartition. The name must be unique within the table.

Example

Create a HASH-RANGE partitioned table:

```
CREATE TABLE tbl_hash_range
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY HASH(col1)
SUBPARTITION BY RANGE(col2)
(
 PARTITION p0 (
   SUBPARTITION s0 VALUES LESS THAN(4),
   SUBPARTITION s1 VALUES LESS THAN(7),
   SUBPARTITION s2 VALUES LESS THAN(10),
   SUBPARTITION s3 VALUES LESS THAN(13)
 ),
 PARTITION p1
 (
   SUBPARTITION s4 VALUES LESS THAN(4),
   SUBPARTITION s5 VALUES LESS THAN(7),
   SUBPARTITION s6 VALUES LESS THAN(10),
   SUBPARTITION s7 VALUES LESS THAN(13)
 ),
```



```
 PARTITION p2
 (
   SUBPARTITION s8 VALUES LESS THAN(4),
   SUBPARTITION s9 VALUES LESS THAN(7),
   SUBPARTITION s10 VALUES LESS THAN(10),
   SUBPARTITION s11 VALUES LESS THAN(13)
 )
);
```

2.11.1.10 HASH-LIST

Syntax

The following statement is used to create one or more HASH-LIST partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
 PARTITION BY [LINEAR] HASH(expr)
   SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
   (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name VALUES IN (value_list)
```

Table 2-23 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
column_list	The list of partition key columns. It is used in LIST COLUMNS(). Expressions are not supported.
value_list	The list of the values of the partition key columns. It is used in LIST COLUMNS().
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Example

Create a HASH-LIST partitioned table:

```
CREATE TABLE tbl_hash_list
(
```



```
  col1 INT,
  col2 INT,
  col3 varchar(20),
  col4 DATE
)
PARTITION BY HASH(col1)
SUBPARTITION BY LIST(col2)
(
 PARTITION dp0 (
  SUBPARTITION p0 VALUES in (1, 2),
  SUBPARTITION p1 VALUES in (3, 4),
  SUBPARTITION p2 VALUES in (5, 6)
 ),
 PARTITION dp1
 (
  SUBPARTITION p3 VALUES in (1, 2),
  SUBPARTITION p4 VALUES in (3, 4),
  SUBPARTITION p5 VALUES in (5, 6)
 ),
 PARTITION dp2
 (
  SUBPARTITION p6 VALUES in (1, 2),
  SUBPARTITION p7 VALUES in (3, 4),
  SUBPARTITION p8 VALUES in (5, 6)
 )
);
```

2.11.1.11 KEY-HASH

Syntax

The following statement is used to create one or more KEY-HASH partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
  PARTITION BY [LINEAR] KEY(expr) [PARTITIONS num]
  SUBPARTITION BY [LINEAR] HASH(expr) [SUBPARTITIONS sub_num]
  (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name
   (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name
```

Table 2-24 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.

Parameter	Description
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Example

Create a KEY-HASH partitioned table:

```
CREATE TABLE tbl_key_hash
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY KEY(col1) PARTITIONS 3
  SUBPARTITION BY HASH(col2) SUBPARTITIONS 2;
```

2.11.1.12 KEY-KEY

Syntax

The following statement is used to create one or more KEY-KEY partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] KEY(expr) [PARTITIONS num]
   SUBPARTITION BY [LINEAR] KEY(expr) [SUBPARTITIONS sub_num]
   (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name
```

Table 2-25 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
partition_name	The name of the partition. The name must be unique within the table.

Parameter	Description
subpartition_name	The name of the subpartition. The name must be unique within the table.

Example

Create a KEY-KEY partitioned table:

```
CREATE TABLE tbl_key_key
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY KEY(col1) PARTITIONS 3
  SUBPARTITION BY KEY(col2) SUBPARTITIONS 2;
```

2.11.1.13 KEY-RANGE

Syntax

The following statement is used to create one or more KEY-RANGE partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
   PARTITION BY [LINEAR] KEY (column_list)
   SUBPARTITION BY RANGE {(expr) | COLUMNS(column_list)}
   (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name
  VALUES LESS THAN {value | value_list | MAXVALUE}
```

Table 2-26 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
column_list	The list of partition key columns. It is used in RANGE COLUMNS(). Expressions are not supported.
value	The boundary value of the partition.

Parameter	Description
value_list	The list of the values of the partition key columns. It is used in LIST COLUMNS().
MAXVALUE	The maximum value of the partition.
partition_name subpartition_name	The name of the partition. The name must be unique within the table. The name of the subpartition. The name must be unique within the table.

Example

Create a KEY-RANGE partitioned table:

```
CREATE TABLE tbl_key_range
(
   col1 INT,
   col2 INT,
   col3 varchar(20),
   col4 DATE
)
PARTITION BY KEY(col1)
   SUBPARTITION BY RANGE COLUMNS(col4)
(
 PARTITION  p0(
   SUBPARTITION p0_q1_2023 VALUES LESS THAN('2023-04-01'),
   SUBPARTITION p0_q2_2023 VALUES LESS THAN('2023-07-01'),
   SUBPARTITION p0_q3_2023 VALUES LESS THAN('2023-10-01'),
   SUBPARTITION p0_q4_2023 VALUES LESS THAN('2024-01-01')
 ),
 PARTITION  p1(
   SUBPARTITION p1_q1_2023 VALUES LESS THAN('2023-04-01'),
   SUBPARTITION p1_q2_2023 VALUES LESS THAN('2023-07-01'),
   SUBPARTITION p1_q3_2023 VALUES LESS THAN('2023-10-01'),
   SUBPARTITION p1_q4_2023 VALUES LESS THAN('2024-01-01')
   ),
 PARTITION  p2(
   SUBPARTITION p2_q1_2023 VALUES LESS THAN('2023-04-01'),
   SUBPARTITION p2_q2_2023 VALUES LESS THAN('2023-07-01'),
   SUBPARTITION p2_q3_2023 VALUES LESS THAN('2023-10-01'),
   SUBPARTITION p2_q4_2023 VALUES LESS THAN('2024-01-01')
 )
);
```

2.11.1.14 KEY-LIST

Syntax

The following statement is used to create one or more KEY-LIST partitioned tables where each partition may contain one or more subpartitions:

```
CREATE TABLE [ schema. ]table_name
 table_definition
 PARTITION BY [LINEAR] KEY(expr)
```



```
  SUBPARTITION BY LIST {(expr) | COLUMNS(column_list)}
  (partition_definition [, partition_definition] ...);
```

partition_definition is:

```
PARTITION partition_name
    (subpartition_definition [, subpartition_definition] ...)
```

subpartition_definition is:

```
SUBPARTITION subpartition_name VALUES IN (value_list)
```

Table 2-27 Parameters

Parameter	Description
table_name	The name of the table to be created.
expr	The expression of the partition. Currently, only the INT type is supported.
column_list	The list of partition key columns. It is used in LIST COLUMNS(). Expressions are not supported.
value_list	The values of the partition.
partition_name	The name of the partition. The name must be unique within the table.
subpartition_name	The name of the subpartition. The name must be unique within the table.

Example

Create a KEY-LIST partitioned table:

```
CREATE TABLE tbl_key_list
(
    col1 INT,
    col2 INT,
    col3 varchar(20),
    col4 DATE
)
PARTITION BY KEY(col1)
SUBPARTITION BY LIST(col2)
(
  PARTITION dp0 (
    SUBPARTITION p0 VALUES in (1, 2),
    SUBPARTITION p1 VALUES in (3, 4),
    SUBPARTITION p2 VALUES in (5, 6)
  ),
  PARTITION dp1
  (
    SUBPARTITION p3 VALUES in (1, 2),
    SUBPARTITION p4 VALUES in (3, 4),
    SUBPARTITION p5 VALUES in (5, 6)
  ),
  PARTITION dp2
  (
```



```
  SUBPARTITION p6 VALUES in (1, 2),
  SUBPARTITION p7 VALUES in (3, 4),
  SUBPARTITION p8 VALUES in (5, 6)
 )
);
```

2.11.2 LIST DEFAULT HASH

LIST DEFAULT HASH supports two partition types at the same level: LIST and HASH. Data is first inserted into LIST partitions. Data that does not comply with the LIST partitioning rules is placed in the default partition. If the default partition has multiple partitions, HASH rules are used. LIST DEFAULT HASH partitioned tables are usually used in scenarios where LIST VALUES are unevenly distributed and cannot be fully enumerated.

Prerequisites

- The kernel version of GaussDB(for MySQL) must be 2.0.54.240600 or later.
- **rds_list_default_partition_enabled** has been set to **ON**.

Constraints

- You can create one or more DEFAULT partitions.
- You can create LIST and DEFAULT subpartitions together, but each partition can only have one DEFAULT subpartition.
- If there is only one DEFAULT partition, subpartitions can be of any types.
- If there are multiple DEFAULT partitions, only HASH or KEY subpartitions are supported.

Parameters

On the **Parameters** page, set **rds_list_default_partition_enabled** to enable or disable LIST DEFAULT HASH.

Table 2-28 Parameter description

Parameter	Level	Description
rds_list_default_partition_enabled	Global	Enables or disables LIST DEFAULT HASH. Value: • **ON**: LIST DEFAULT HASH is enabled. • **OFF**: LIST DEFAULT HASH is disabled.

Creating a LIST DEFAULT HASH Partitioned Table

- Syntax

```
CREATE TABLE [ schema. ]table_name
 table_definition
```



```
PARTITION BY LIST [COLUMNS] (expr)
SUBPARTITION BY ...
(list_partition_definition[, ..., list_partition_definition],
 default_partition_definition
)
```

default_partition_definition is:

```
PARTITION partition_name DEFAULT [PARTITIONS number]
```

The definition of each partition can also contain subpartitions. Subpartitions can also use LIST DEFAULT. The definition is as follows:

```
SUBPARTITION subpartition_name DEFAULT
```

Table 2-29 Parameter description

Parameter	Description
table_name	The name of the table to be created.
partition_name	• The name of the partition if there is only one DEFAULT partition. The name must be different from those in other partitioned tables. • The prefix of the partition name if there are multiple DEFAULT partitions. The partition name is in the format of partition_name+sequence number.
subpartition_name	The name of the subpartition. The name must be unique within a table. Only one DEFAULT subpartition is supported.
number	The number of DEFAULT partitions if the DEFAULT partition is divided into multiple DEFAULT partitions based on HASH rules. This parameter is optional. If you do not specify it, a DEFAULT partition is created.

- Examples

  Create a single DEFAULT partition:

  ```
  CREATE TABLE list_default_tbl (
    a INT,
    b INT
  )
  PARTITION BY LIST (a)
  (PARTITION p0 VALUES IN (1,2,3,4,5),
   PARTITION p1 VALUES IN (6,7,8,9,10),
   PARTITION pd DEFAULT);
  ```

  Create multiple DEFAULT partitions:

  ```
  CREATE TABLE list_default_hash (
    a INT,
    b INT
  )
  PARTITION BY LIST (a)
  (PARTITION p0 VALUES IN (1,2,3,4,5),
   PARTITION p1 VALUES IN (6,7,8,9,10),
   PARTITION pd DEFAULT PARTITIONS 3);
  ```

  Use LIST COLUMNS:


```
CREATE TABLE t_goods
(
  country   VARCHAR(30),
  year      VARCHAR(60),
  goods     TEXT
) PARTITION BY LIST COLUMNS(country)
(
  PARTITION p1 VALUES IN ('China'),
  PARTITION p2 VALUES IN ('USA'),
  PARTITION p3 VALUES IN ('Asia'),
  PARTITION p3 VALUES IN ('India'),
  PARTITION p_deft DEFAULT PARTITIONS 5
);
```

Execute the **EXPLAIN** statement to view partitions:

```
EXPLAIN SELECT * FROM list_default_hash;
```

The following information is displayed:

```
+----+-------------+-------------------+-------------------+------+---------------+------+---------+------+------
+----------+-------+
| id | select_type | table             | partitions        | type | possible_keys | key  | key_len | ref  | rows |
filtered | Extra |
+----+-------------+-------------------+-------------------+------+---------------+------+---------+------+------
+----------+-------+
|  1 | SIMPLE      | list_default_hash | p0,p1,pd0,pd1,pd2 | ALL  | NULL          | NULL | NULL    | NULL |
1 |   100.00 | NULL  |
+----+-------------+-------------------+-------------------+------+---------------+------+---------+------+------
+----------+-------+
1 row in set (0.04 sec)
```

Create a LIST DEFAULT HASH partitioned table which supports List Default subpartitions:

```
CREATE TABLE test (a int, b int)
PARTITION BY RANGE(a)
SUBPARTITION BY LIST(b) (
PARTITION part0 VALUES LESS THAN (10)
( SUBPARTITION sub0 VALUES IN (1,2,3,4,5),
  SUBPARTITION sub1 DEFAULT),
PARTITION part1 VALUES LESS THAN (20)
( SUBPARTITION sub2 VALUES IN (1,2,3,4,5),
  SUBPARTITION sub3 DEFAULT),
PARTITION part2 VALUES LESS THAN (30)
( SUBPARTITION sub4 VALUES IN (1,2,3,4,5),
  SUBPARTITION sub5 DEFAULT));
```

Create a LIST DEFAULT HASH partitioned table which supports only HASH or KEY subpartitions when there are multiple LIST DEFAULT HASH partitions:

```
CREATE TABLE list_default_hash_sub (
  a INT,
  b INT
)
PARTITION BY LIST (a)
SUBPARTITION BY HASH (b) SUBPARTITIONS 20
(PARTITION p0 VALUES IN (1,2,3,4,5),
 PARTITION p1 VALUES IN (6,7,8,9,10),
 PARTITION pd DEFAULT PARTITIONS 3);
```

Modifying a LIST DEFAULT HASH Partitioned Table

LIST DEFAULT HASH partitions support all the statements for modifying partitioned tables, including ALTER TABLE ADD PARTITION, ALTER TABLE DROP PARTITION, ALTER TABLE REORGANIZE PARTITION, ALTER TABLE TRUNCATE

PARTITION, ALTER TABLE EXCHANGE PARTITION, ALTER TABLE OPTIMIZE PARTITION, ALTER TABLE REBUILD PARTITION, ALTER TABLE REPAIR PARTITION, ALTER TABLE ANALYZE PARTITION, and ALTER TABLE CHECK PARTITION. This topic describes how to modify a LIST DEFAULT HASH partitioned table by executing the ALTER TABLE ADD PARTITION, ALTER TABLE DROP PARTITION, and ALTER TABLE REORGANIZE PARTITION statements.

- ALTER TABLE ADD PARTITION
  - ADD DEFAULT PARTITION

    If a partitioned table contains only LIST partitions, run **ADD PARTITION** to add a DEFAULT partition so that the table becomes a LIST DEFAULT HASH partitioned table.

    ```
    ALTER TABLE table_name ADD PARTITION(default_partition_definition)
    ```

    Add a DEFAULT partition:

    ```
    CREATE TABLE list_tab (
      a INT,
      b INT
    )
    PARTITION BY LIST (a)
    (PARTITION p0 VALUES IN (1,2,3,4,5),
     PARTITION p1 VALUES IN (6,7,8,9,10)
    );
    ALTER TABLE list_tab ADD PARTITION(PARTITION pd DEFAULT);
    ```

    Add two DEFAULT partitions:

    ```
    CREATE TABLE list_tab (
      a INT,
      b INT
    )
    PARTITION BY LIST (a)
    (PARTITION p0 VALUES IN (1,2,3,4,5),
     PARTITION p1 VALUES IN (6,7,8,9,10)
    );
    ALTER TABLE list_tab ADD PARTITION(PARTITION pd DEFAULT PARTITIONS 2);
    ```

  - ADD LIST PARTITION

    You can add **WITHOUT VALIDATION** to the ALTER TABLE ADD PARTITION statement to add LIST partitions.

    ```
    ALTER TABLE table_name ADD PARTITION(
      list_partition_definition[, ..., list_partition_definition])
    WITHOUT VALIDATION
    ```

    Add a LIST partition:

    ```
    CREATE TABLE list_default_hash (
      a INT,
      b INT
    )
    PARTITION BY LIST (a)
    (PARTITION p0 VALUES IN (1,2,3,4,5),
     PARTITION p1 VALUES IN (6,7,8,9,10),
     PARTITION pd DEFAULT PARTITIONS 3);

    ALTER TABLE list_default_hash ADD PARTITION(
      PARTITION p2 VALUES IN (11,12,13)
    )WITHOUT VALIDATION;
    ```

    After the statement is executed, a LIST partition named **p2** is added to table **list_default_hash**. There is no data in **p2**.

> **NOTE**
>
> If you use **WITHOUT VALIDATION** to add a LIST partition, you need to manually execute **ALTER TABLE ... REBUILD ALL** to reallocate data. Otherwise, data will not be reallocated. Data that meets the new partition definition will be still stored in the DEFAULT partition. During query, all DEFAULT partitions will be marked and not pruned. As a result, the query performance deteriorates. You are advised to use the ALTER TABLE REORGANIZE PARTITION statement to separate some data from the DEFAULT partition and create a new LIST partition.

- ALTER TABLE DROP PARTITION

  The DROP PARTITION statement deletes all DEFAULT partitions at a time. You cannot execute this statement to delete only some DEFAULT partitions.

  Execute the DROP PARTITION statement to delete all partitions:

  ```
  ALTER TABLE list_default_hash DROP PARTITION pd0,pd1,pd2;
  Query OK, 0 rows affected (0.33 sec)
  Records: 0  Duplicates: 0  Warnings: 0
  ```

  > **NOTE**
  >
  > An error is reported when you delete only some DEFAULT partitions.
  >
  > ```
  > ALTER TABLE list_default_hash DROP PARTITION pd0;
  > ```
  >
  > The error message is as follows:
  >
  > ERROR 8078 (HY000): DROP PARTITION cannot be used on default partitions of LIST DEFAULT, except once dropping all default partitions

- ALTER TABLE REORGANIZE PARTITION

  The REORGANIZE PARTITION statement modifies all DEFAULT partitions at a time. You cannot execute this statement to modify only some DEFAULT partitions.

  - Execute the REORGANIZE PARTITION statement to change the number of DEFAULT partitions:

    ```
    ALTER TABLE list_default_hash
    REORGANIZE PARTITION
      pd0,pd1
    INTO(
      PARTITION pd DEFAULT PARTITIONS 3);
    ```

    After the statement is executed, the number of DEFAULT partitions changes from 2 to 3.

  - Execute the REORGANIZE PARTITION statement to split a LIST partition from a DEFAULT partition:

    ```
    ALTER TABLE list_default_hash
    REORGANIZE PARTITION
      pd0,pd1
    INTO (
     PARTITION p2 VALUES IN (20,21),
     PARTITION pd DEFAULT PARTITIONS 2);
    ```

    After the statement is executed, a LIST partition named **p2** is added to the **list_default_hash** partitioned table. **p2** contains data that meets the VALUES IN (20,21) rule and is separated from the DEFAULT partition.

  - Execute the REORGANIZE PARTITION statement to merge a LIST partition into a DEFAULT partition:

    ```
    ALTER TABLE list_default_hash
    REORGANIZE PARTITION
      p2, pd0, pd1
    ```



```
INTO (
 PARTITION pd DEFAULT PARTITIONS 2);
```

After the statement is executed, the LIST partition **p2** is merged into the DEFAULT partition.

- Execute the REORGANIZE PARTITION statement to split some values from a DEFAULT partition and add them to a LIST partition:

```
ALTER TABLE list_default
REORGANIZE partition
  p2, pd0, pd1
INTO (
  PARTITION p2 VALUES IN (20,21,22,23,24),
  PARTITION pd DEFAULT PARTITIONS 4);
```

After the statement is executed, the definition of **p2** is changed from PARTITION p2 VALUES IN (20,21) to PARTITION p2 VALUES IN (20,21,22,23,24). Any data that matches the VALUES IN (20,21,22,23,24) rule is then transferred from the DEFAULT partition to **p2**.

2.11.3 INTERVAL RANGE

An INTERVAL RANGE partitioned table is an extension of a RANGE partitioned table. If data to be inserted into a RANGE partitioned table falls outside the range of an existing partition, it cannot be inserted and an error will be returned.

INTERVAL RANGE partitioned tables allow a database to create a partition based on rules specified by the INTERVAL clause when data to be inserted exceeds the range of an existing partition.

Prerequisites

- The kernel version of GaussDB(for MySQL) must be 2.0.54.240600 or later.
- **rds_interval_range_enabled** has been set to **ON**.

Constraints

- INTERVAL RANGE partitioned tables support only HASH or KEY subpartitions.
- If an INTERVAL RANGE rule is in **RANGE COLUMNS(column_list) INTERVAL([type], value)** format:
  - **column_list** specifies only a single partition key, which must be an integer or of the DATE/TIME/DATETIME type.
  - If the partition key is an integer, the interval type (**type**) can be left blank.
  - If the partition key is of the DATE type,the interval type (**type**) can only be YEAR, QUARTER, MONTH, WEEK, or DAY.
  - If the partition key is of the TIME type, the interval type (**type**) can only be HOUR, MINUTE, or SECOND.
  - If the partition key is of the DATETIME type, the interval type (**type**) can be YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, or SECOND.
  - The interval value (**value**) must be a positive integer.
  - If the interval type (**type**) is SECOND, the interval cannot be less than 60.
- If an INTERVAL RANGE rule is in **RANGE(expr) INTERVAL(value)** format, the result of the expr expression must be an integer, and the interval value must be a positive integer.

- You cannot execute the INSERT ... SELECT, INSERT ... ON DUPLICATE KEY UPDATE, and UPDATE statements to add partitions.
- When you execute the LOAD DATA statement to import data, partition creation will not be triggered. (If the range of the partition covers all data, data can be imported. If the range of the partition does not cover all data, partition creation is not triggered, and data fails to be imported.)
- Once partitions are automatically created, they cannot be rolled back.
- Prefix **_p** is reserved for automatically created partitions. If you use this prefix for custom partitions, automatic partition creation may fail.
- The **SET INTERVAL([type], value)** clause applies only to INTERVAL RANGE and RANGE partitioned tables. If these tables have subpartitions, the subpartitions must be of the HASH or KEY type.
- The values of **type** and **value** in the **SET INTERVAL([type], value)** clause must be restricted by the partition expression **expr** or the partition key **column_list** of the original table.

Parameters

Table 2-30 Parameter description

Parameter	Level	Description
rds_interval_range_enabled	Global	Enables or disables INTERVAL RANGE. Value: • **ON**: INTERVAL RANGE is enabled. • **OFF**: INTERVAL RANGE is disabled.

Creating an INTERVAL RANGE Partitioned Table

The definition format of an INTERVAL RANGE partitioned table is similar to that of a RANGE partitioned table, with the addition of an INTERVAL clause.

Syntax:

```
CREATE TABLE [IF NOT EXISTS] [schema.]table_name
table_definition
partition_options;
```

partition_options is:

```
PARTITION BY
    RANGE {(expr) | COLUMNS(column_list)}
    {INTERVAL(value) | INTERVAL(type, expr)}
    (partition_definition [, partition_definition] ...)
```

partition_definition is:

```
PARTITION partition_name
    [VALUES LESS THAN {expr | MAXVALUE}]
```



```
[[STORAGE] ENGINE [=] engine_name]
[COMMENT [=] 'string' ]
[DATA DIRECTORY [=] 'data_dir']
[INDEX DIRECTORY [=] 'index_dir']
[MAX_ROWS [=] max_number_of_rows]
[MIN_ROWS [=] min_number_of_rows]
[TABLESPACE [=] tablespace_name]
```

The INTERVAL clause supports only the interval value (**value**) and interval type (**type**).

Description of parameters associated with the INTERVAL clause:

Table 2-31 Parameter description

Parameter	Description
INTERVAL(value)	The format of the INTERVAL clause when **RANGE COLUMNS(column_list)** with an integer column or **RANGE(expr)** is used. **value** indicates the interval value and must be a positive integer.
expr	The expression of the partition. It is used in **RANGE()** and must be of the integer type.
column_list	The list of partitions. It is used in **RANGE COLUMNS()**. In an INTERVAL RANGE partitioned table, **column_list** can only be a single column.
INTERVAL(type, value)	The format of the INTERVAL clause when **RANGE COLUMNS(column_list)** is used and **column_list** is of the DATE, TIME, or DATETIME type. **type** indicates the interval type. Eight time types (YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, and SECOND) are supported. **value** indicates the interval value, which must be a positive integer. When **type** is set to **SECOND**, the interval value cannot be less than 60.

Further description of interval values (**value**) and interval types (**type**):

- Interval values (expr)

  Add 1,000 consecutive numbers to a partition.

  Example:

  ```
  INTERVAL(1000)
  ```

- Time types
  - YEAR

    Set the interval type to YEAR and add the data of one year to a partition.

    Example:

    ```
    INTERVAL(YEAR, 1)
    ```

  - QUARTER

    Set the interval type to QUARTER and add the data of one quarter to a partition.

Example:
```
INTERVAL(QUARTER, 1)
```

- MONTH

  Set the interval type to MONTH and add the data of one month to a partition.

  Example:
  ```
  INTERVAL(MONTH, 1)
  ```

- WEEK

  Set the interval type to WEEK and add the data of one week to a partition.

  Example:
  ```
  INTERVAL(WEEK, 1)
  ```

- DAY

  Set the interval type to DAY and add the data of one day to a partition.

  Example:
  ```
  INTERVAL(DAY, 1)
  ```

- HOUR

  Set the interval type to HOUR and add the data of one hour to a partition.

  Example:
  ```
  INTERVAL(HOUR, 1)
  ```

- MINUTE

  Set the interval type to MINUTE and add the data of one minute to a partition.

  Example:
  ```
  INTERVAL(MINUTE, 1)
  ```

- SECOND

  Set the interval type to SECOND and add the data of one second to a partition.

  Example:
  ```
  INTERVAL(SECOND, 60)
  ```

The following example uses **order_time** as the partition key to partition the **sales** table by interval.

Create an INTERVAL RANGE partitioned table in the database and insert data into the table. Example:

```
CREATE TABLE sales
(
  id BIGINT,
  uid BIGINT,
  order_time DATETIME
)
PARTITION BY RANGE COLUMNS(order_time) INTERVAL(MONTH, 1)
(
  PARTITION p0 VALUES LESS THAN('2021-9-1')
);
```

Insert data into the INTERVAL RANGE partitioned table. Example:

```
INSERT INTO sales VALUES(1, 1010101010, '2021-11-11');
```


After data is inserted, execute the SHOW CREATE TABLE statement to query the **sales** table definition. The new table definition is as follows:

```
CREATE TABLE `sales` (
  `id` bigint DEFAULT NULL,
  `uid` bigint DEFAULT NULL,
  `order_time` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
/*!50500 PARTITION BY RANGE  COLUMNS(order_time) */ /*!99990 800220201
INTERVAL(MONTH, 1) */
/*!50500 (PARTITION p0 VALUES LESS THAN ('2021-9-1') ENGINE = InnoDB,
 PARTITION _p20211001000000 VALUES LESS THAN ('2021-10-01 00:00:00') ENGINE = InnoDB,
 PARTITION _p20211101000000 VALUES LESS THAN ('2021-11-01 00:00:00') ENGINE = InnoDB,
 PARTITION _p20211201000000 VALUES LESS THAN ('2021-12-01 00:00:00') ENGINE = InnoDB)
*/
```

In the preceding example, three partitions **_p20211001000000**, **_p20211101000000**, and **_p20211201000000** are automatically added to the INTERVAL RANGE partition. **Note that partition names prefixed with _p are reserved by the system. Such partition names cannot be used when you create or rename partitions.**.

INTERVAL RANGE partitioned tables support HASH or KEY subpartitions. Example:

```
CREATE TABLE sales_ir_key
(
  dept_no     INT,
  part_no     INT,
  country     varchar(20),
  date        DATE,
  amount      INT
)
PARTITION BY RANGE(month(date)) INTERVAL(1)
SUBPARTITION BY KEY(date) SUBPARTITIONS 2
(
  PARTITION q1_2012 VALUES LESS THAN(4)
    (SUBPARTITION sp_001,
     SUBPARTITION sp_002),
  PARTITION q2_2012 VALUES LESS THAN(7)
    (SUBPARTITION sp_003,
     SUBPARTITION sp_004)
);

CREATE TABLE sales_ir_hash
(
  dept_no     INT,
  part_no     INT,
  country     varchar(20),
  date        DATE,
  amount      INT
)
PARTITION BY RANGE COLUMNS(date) INTERVAL(YEAR, 1)
SUBPARTITION BY HASH(TO_DAYS(date)) SUBPARTITIONS 2
(
  PARTITION q1_2012 VALUES LESS THAN('2021-01-01')
    (SUBPARTITION sp_001,
     SUBPARTITION sp_002),
  PARTITION q2_2012 VALUES LESS THAN('2022-01-01')
    (SUBPARTITION sp_003,
     SUBPARTITION sp_004)
);
```


Conversion Between INTERVAL RANGE Partitioned Tables and Other Types of Tables

Syntax:

Convert other types of tables to INTERVAL RANGE partitioned tables.

```
ALTER TABLE table_name table_definition
partition_options;

partition_options:
    PARTITION BY
    { RANGE{(expr) | COLUMNS(column_list)} }
    { INTERVAL(type, value) | INTERVAL(value) }
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
        [VALUES LESS THAN {expr | MAXVALUE}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'string' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
```

Description of parameters associated with the INTERVAL clause:

Table 2-32 Parameter description

Parameter	Description
INTERVAL(value)	The format of the INTERVAL clause when **RANGE COLUMNS(column_list)** with an integer column or **RANGE(expr)** is used. **value** indicates the interval value and must be a positive integer.
expr	The expression of the partition. It is used in **RANGE()** and must be of the integer type.
column_list	The list of partitions. It is used in **RANGE COLUMNS()**. In an INTERVAL RANGE partitioned table, **column_list** can only be a single column.
INTERVAL(type, value)	The format of the INTERVAL clause when **RANGE COLUMNS(column_list)** is used and **column_list** is of the DATE, TIME, or DATETIME type. **type** indicates the interval type. Eight time types (YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, and SECOND) are supported. **value** indicates the interval value, which must be a positive integer. When **type** is set to **SECOND**, the interval value cannot be less than 60.

Convert an INTERVAL RANGE partitioned table to any other type of table. **partition_options** is optional.


```
ALTER TABLE table_name table_definition
[partition_options];
```

Examples:

Convert other types of tables to INTERVAL RANGE partitioned tables.

```
CREATE TABLE orders(
  orderkey BIGINT NOT NULL,
  custkey BIGINT NOT NULL,
  orderdate DATE NOT NULL
);

ALTER TABLE orders
PARTITION BY RANGE COLUMNS(orderdate) INTERVAL(MONTH, 1) (
  PARTITION p0 VALUES LESS THAN('2021-10-01')
);
```

Convert an INTERVAL RANGE partitioned table to another type of table.

```
CREATE TABLE orders (a INT, b DATETIME)
PARTITION BY RANGE (a) INTERVAL(10)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);

ALTER TABLE orders PARTITION BY LIST COLUMNS (a)
(
   PARTITION p0 VALUES IN (1, 11, 25)
);
```

Modify the INTERVAL clause in the INTERVAL RANGE partitioned table.

```
CREATE TABLE orders (a INT, b DATETIME)
PARTITION BY RANGE (a) INTERVAL(10)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);

ALTER TABLE orders PARTITION BY RANGE (a) INTERVAL(20)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);

# Delete the INTERVAL clause.
ALTER TABLE orders PARTITION BY RANGE (a)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);

# Add the INTERVAL clause.
ALTER TABLE orders PARTITION BY RANGE (a) INTERVAL(100)
(
   PARTITION p0 VALUES LESS THAN(10),
   PARTITION p2 VALUES LESS THAN(20)
);
```


Support for the SET INTERVAL Clause

You can use the SET INTERVAL clause to modify the interval type and value of the INTERVAL clause defined in the INTERVAL RANGE partitioned table, or eliminate or add the INTERVAL clause.

Syntax:

```
ALTER TABLE table_name SET INTERVAL {() | (type, value) | (value)};
```

Table 2-33 Parameter description

Parameter	Description
type	The type of the interval. Eight time types (YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, and SECOND) are supported. If you do not specify this parameter, the numeric type is used by default.
value	The value of the interval. When **type** is set to SECOND, the interval value cannot be less than 60.

Example:

Modify the interval type and value in the INTERVAL RANGE partitioned table.

```
CREATE TABLE orders(
  orderkey BIGINT NOT NULL,
  custkey BIGINT NOT NULL,
  orderdate DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(orderdate) INTERVAL(MONTH, 1) (
  PARTITION p0 VALUES LESS THAN('2021-10-01')
);

ALTER TABLE orders SET INTERVAL(YEAR, 1);
```

Convert a RANGE partitioned table to an INTERVAL RANGE partitioned table.

```
CREATE TABLE orders(
  orderkey BIGINT NOT NULL,
  custkey BIGINT NOT NULL,
  orderdate DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(orderdate) INTERVAL(MONTH, 1) (
  PARTITION p0 VALUES LESS THAN('2021-10-01')
);

# Delete the INTERVAL clause.
ALTER TABLE sales SET INTERVAL();

# Add the INTERVAL clause.
ALTER TABLE sales SET INTERVAL(DAY, 60);
```


> **CAUTION**
>
> The ALTER TABLE table_name SET INTERVAL() statement can be used even if **rds_interval_range_enabled** is disabled. This statement is used to eliminate the definition of the INTERVAL clause in an INTERVAL RANGE partitioned table and convert the partitioned table to a RANGE partitioned table.

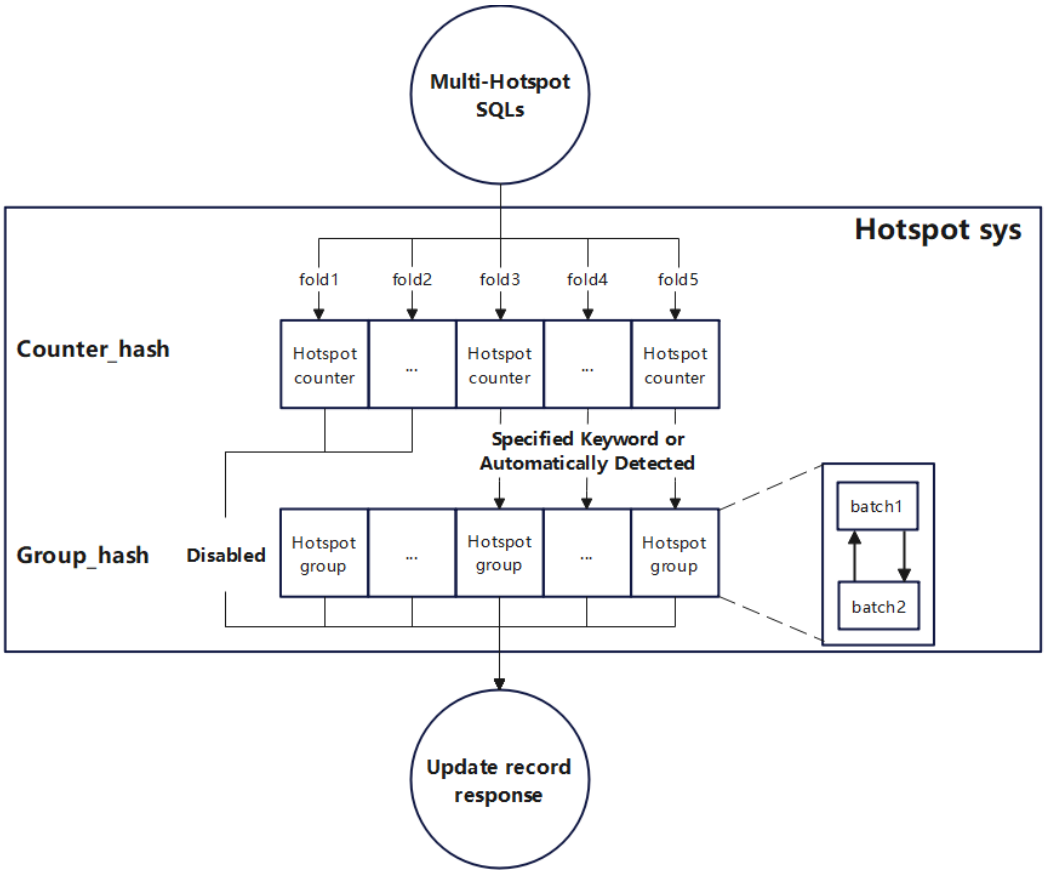
2.12 Hot Row Update

Hot rows indicate the rows that are frequently added, deleted, modified, and queried in a database in the following scenarios, such as flash sales, concert ticket booking, and train ticket booking for hot routes. When a transaction updates data in a row, the row needs to be locked. Only one transaction can update a row at a time, and other transactions can continue to be executed only after the row lock is released, so the performance of existing hot row update is poor. Traditional sharding policies are ineffective to improve the processing performance.

GaussDB(for MySQL) optimizes hot row update, which can be automatically or manually enabled. After hot row update is enabled, hot rows can be updated efficiently.

Principles

The following figure shows the architecture of GaussDB(for MySQL) hot row update. There are two parts: Counter_hash and Group_hash. Counter_hash is used to automatically determine which rows are hot rows. Group_hash consists of multiple hotspot groups and is used to update hot rows. Each hotspot group corresponds to a hot row. Each hotspot group consists of multiple batches to ensure that the statements that update hot rows can be committed alternately.



Constraints

- The kernel version of your GaussDB(for MySQL) instance must be 2.0.54.240600 or later.
- Usage constraints:
  - In a WHERE condition, only a primary key or unique index can be used for equality matching, and only one record can be updated.
  - Index columns cannot be modified.
  - The modifications apply only for the columns of the integer type.
  - Only two elements in a hot row record can be added or subtracted. The first element is the same as the left side of the equal sign (=) and meets the constraints such as unique indexes. Value assignment is not allowed. Assuming that c is the column to be modified and d is a common column, only operations similar to **c=c+1** or **c=c-1** are allowed. Operations such as **c=d+1**, **c=1+c**, **c=c+1+1**, and **c=1+c+1** are not allowed.
  - This function applies only for implicit transactions. That is, **AUTOCOMMIT** must be set to **ON** and cannot be used in BEGIN and COMMIT transactions.
  - You need to use HOTSPOT to explicitly mark hot row update transactions, or set **rds_hotspot_auto_detection_threshold** to a value other than 0 to enable automatic hot row update identification. For details about how to use **rds_hotspot_auto_detection_threshold**, see the parameter description.
  - The isolation level of transactions in a database must be READ COMMITTED (RC).

  - This function cannot be used in stored functions, triggers, or events. Otherwise, the following error is reported on the client:
    ```
    HOTSPOT hints can not be used in stored function, trigger or event
    ```

- Behavior change: In a hotspot transaction group, except the transactions that failed to be executed or that were killed in the update phase, other transactions are committed in batches and recorded in redo logs and undo logs. These transactions can only be committed or rolled back in batches and cannot be rolled back separately. Dozens to hundreds of transactions can be committed in each batch.

Parameter Description

Table 2-34 Parameter description

Parameter	Description
rds_hotspot	Whether to enable hot row update. ON: The function is enabled.
rds_hotspot_follower_wait_commit_interval	Sleep time in microseconds before a follower transaction is blocked when waiting for the leader transaction logs to be persisted. For instances with slow log persistence, you are advised to increase the value. For instances with fast log persistence, you are advised to set this parameter to **0** so that follower transactions are blocked without sleeping.
rds_hotspot_leader_wait_follower_interval	Time interval, in microseconds, that the leader transaction in a hot row update waits for the follower transaction to update records. In low concurrency, you are advised to set this parameter to a smaller value to avoid performance deterioration. In high concurrency, you are advised to set this parameter to a larger value to improve performance. If queries per second (QPS) exceeds 200,000, you are advised to set this parameter to **100** or a larger value.
rds_hotspot_auto_detection_threshold	Whether to enable automatic identification for hot rows. The value **0** indicates that the function is disabled. If the value is not **0**, it indicates the threshold for identifying hot rows. When the number of row updates per second exceeds the threshold, hot row update is enabled.

Parameter	Description
rds_hotspot_batch_size_lower_limit	Recommended minimum size for each batch of hot transactions. Each batch should strive to reach this size as much as possible. However, this is not strictly guaranteed. When the leader finds that all followers to be waited for have arrived, the batch of transactions enters the commit state.
rds_hotspot_max_memory_size	Maximum memory occupied by groups and counters during a hot row update. When the memory occupied by a group exceeds the threshold, the memory occupied by the group is cleared. When the memory occupied by a counter exceeds the threshold, the memory occupied by the counter is cleared. The system attempts to clear the old memory only when a new memory is applied for.
rds_hotspot_enable_time_statistics	Whether to enable status statistics related to the update time of hot rows. The value **ON** indicates this function is enabled.

Status Description

Table 2-35 Status description

Status	Description
Hotspot_total_trx	Total transactions using the hot row upgrade function.
Hotspot_update_errors	Transactions that failed to update hot rows. These transactions do not affect the commit of other transactions that update how rows.
Hotspot_trx_rollbacked	Number of transactions that are successfully updated but are finally rolled back. When the leader transaction decides to roll back, all follower transactions roll back together.
Hotspot_trx_committed	Number of transactions that are successfully committed to update hot rows.
Hotspot_batch_size	Number of transactions that are to update hot rows at a time. These transactions are committed in batches.

Status	Description
Hotspot_batch_wait_time	Time in microseconds that the next batch of transactions that are to update hot rows waits for the previous batch of transaction to release the lock. When a batch of transactions to update rows is committed, the rows are locked.
Hotspot_leader_wait_follower_time	Time in microseconds for the leader to wait for the followers in the current batch to complete record update.
Hotspot_leader_total_time	Total time spent by the leader transaction in updating hot rows in the current batch, in microseconds.
Hotspot_follower_total_time	Total time spent by a follower transaction in updating hot rows in the current batch, in microseconds.
Hotspot_follower_wait_commit_time	Time for a follower to wait for the leader to persist logs in the current batch, in microseconds.
Hotspot_group_counts	Number of groups. Each hot row update corresponds to a group, and transactions in the group are committed in batches.
Hotspot_counter_counts	Number of counters. Counters are used to automatically determine whether a hot row is updated. When the statistical value in a counter meets the requirement, a group is created for hot row update.

New Keywords

The following table lists new keywords.

Table 2-36 New keywords

Keyword	Description
HOTSPOT	Indicates that hot row update is enabled.
NOT_MORE_THAN	(Optional) Indicates that the target value is not greater than a certain value.
NOT_LESS_THAN	(Optional) Indicates that the target value is not less than a certain value.

The preceding keywords are placed at the end of a SQL statement. **HOTSPOT** must be placed at the beginning. **NOT_MORE_THAN** and **NOT_LESS_THAN** can be placed at any position.

For example, if **id** is a primary key column and **c** is an INT column, the following syntax is supported:

```
UPDATE c=c+1 where id=10 HOTSPOT;
UPDATE c=c+1 where id=10 HOTSPOT NOT_MORE_THAN 100; // The value of the c column is not greater than 100.
UPDATE c=c-1 where id=10 HOTSPOT NOT_LESS_THAN 0; // The value of the c column is not less than 0.
UPDATE c=c+1 where id=10 HOTSPOT NOT_MORE_THAN 100 NOT_LESS_THAN 0; // The value of the c column is not greater than 100 and not less than 0.
UPDATE c=c+1 where id=10 HOTSPOT NOT_LESS_THAN 0 NOT_MORE_THAN 100; // The value of the c column is not greater than 100 and not less than 0.
```

When any value exceeds the value of **NOT_MORE_THAN** or **NOT_LESS_THAN**, the following error is reported to the client:

```
HOTSPOT field value exceeds limit
```

Example

1. Create a table and prepare data.
   ```
   CREATE TABLE test.hotspot1 (
     `id` int NOT NULL primary key,
     `c` int NOT NULL DEFAULT '0'
   ) ENGINE=InnoDB;
   INSERT INTO test.hotspot1 VALUES (1, 1);
   ```
2. Enable hot row update.
   ```
   SET GLOBAL rds_hotspot = ON;
   ```
3. Change the isolation level to AUTOCOMMIT.
   ```
   SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
   SET SESSION AUTOCOMMIT = ON;
   ```
4. Initiate an update with HOTSPOT keyword.
   ```
   UPDATE test.hotspot1 SET c=c+1 WHERE id=1 HOTSPOT;
   ```
5. Check the update status of hot rows.
   ```
   SHOW STATUS like "%hotspot%";
   ```

   Performance test

   - Test environments

     Instance specifications: 8 vCPUs | 32 GB, 32 vCPUs | 128 GB

     ECS specifications: 32 vCPUs | 64 GB

     Region: CN North-Beijing4

     Test tool: sysbench-1.0.18

     Data models:

     - One table with one data record
     - Eight tables, with each table containing one data record
   - Parameter settings:

     rds_hotspot=ON

     transaction_isolation=READ-COMMITTED

     max_prepared_stmt_count=1048576

rds_global_sql_log_bin=OFF

- Test method

  Definition of the data tables required for the test:

  CREATE TABLE sbtest (id int NOT NULL AUTO_INCREMENT,k int NOT NULL DEFAULT '0',PRIMARY KEY (id));

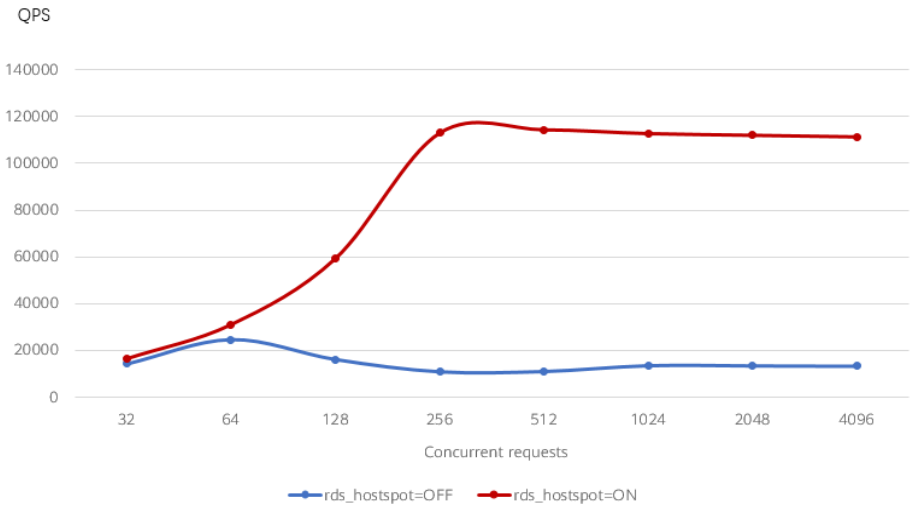
  Test statement:

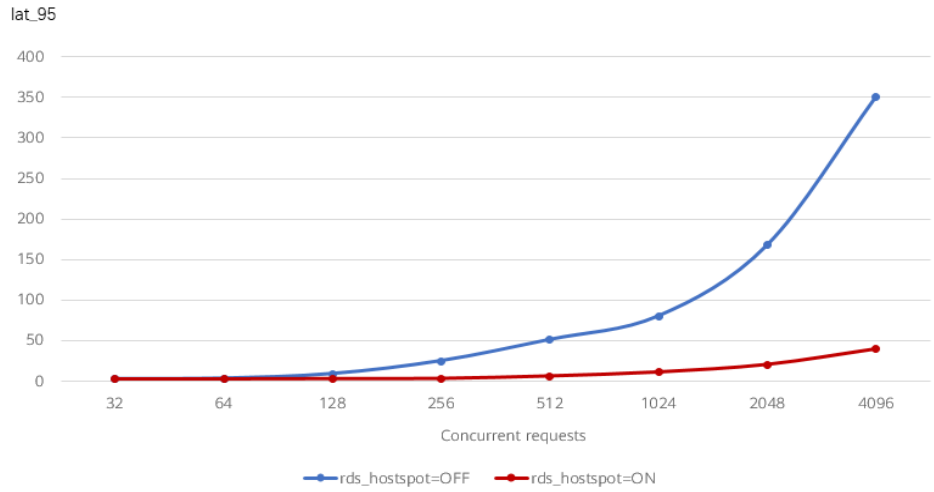
  UPDATE sbtest%u SET k=k+1 WHERE id=1 hotspot;

- Test scenarios and results

  Test scenario 1: updating a single hot row of an instance with 8 vCPUs and 32 GB

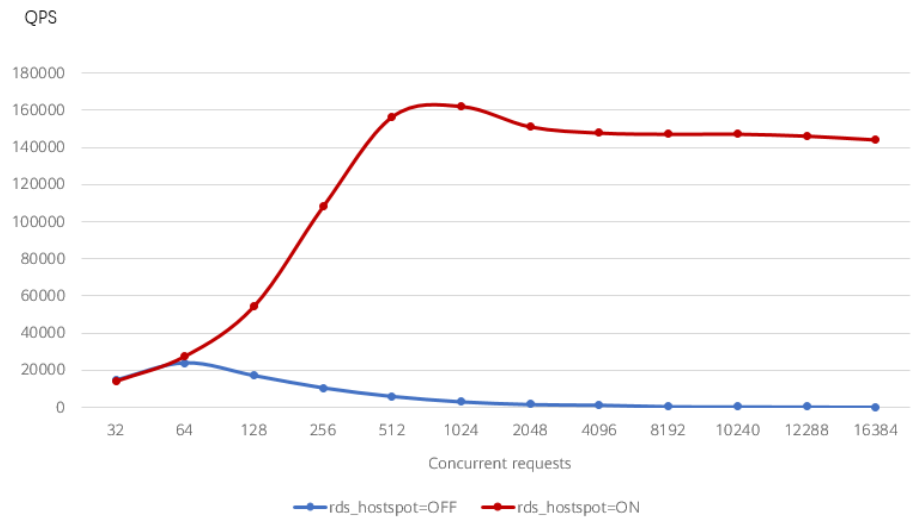
  Test result: The performance of all concurrent requests was improved to different degrees. The performance of 64 or less concurrent requests was not improved significantly, but the performance of 128 or more concurrent requests was improved significantly (up to 9.26 times).

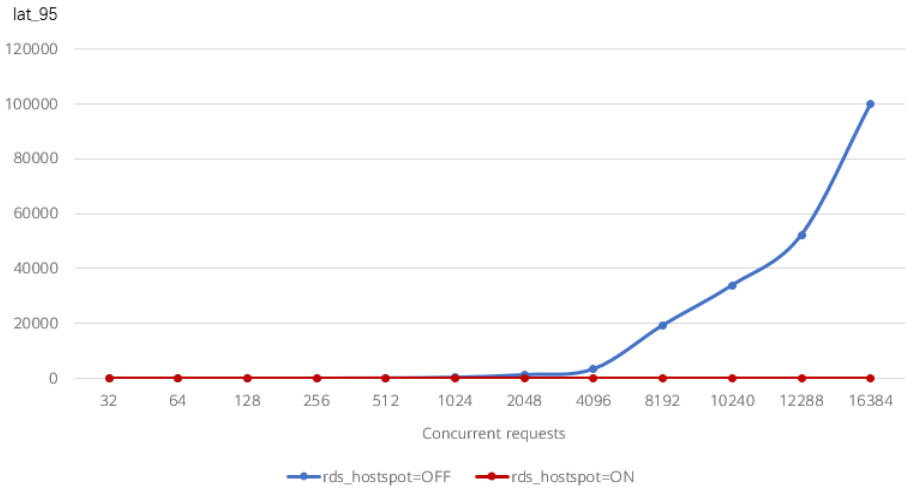




  Test scenario 2: Updating a single hot row of an instance with 32 vCPUs and 128 GB

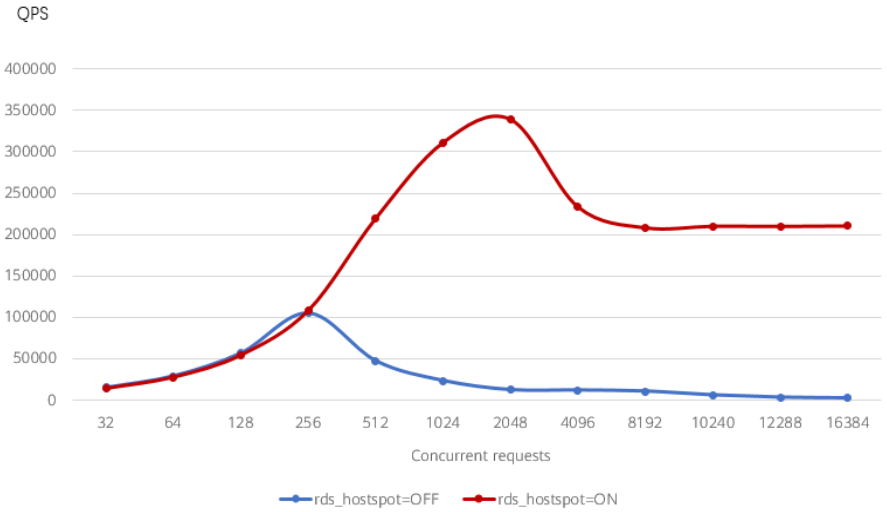
  Test result: The performance of 128 or more concurrent requests was improved significantly, by 639 times.

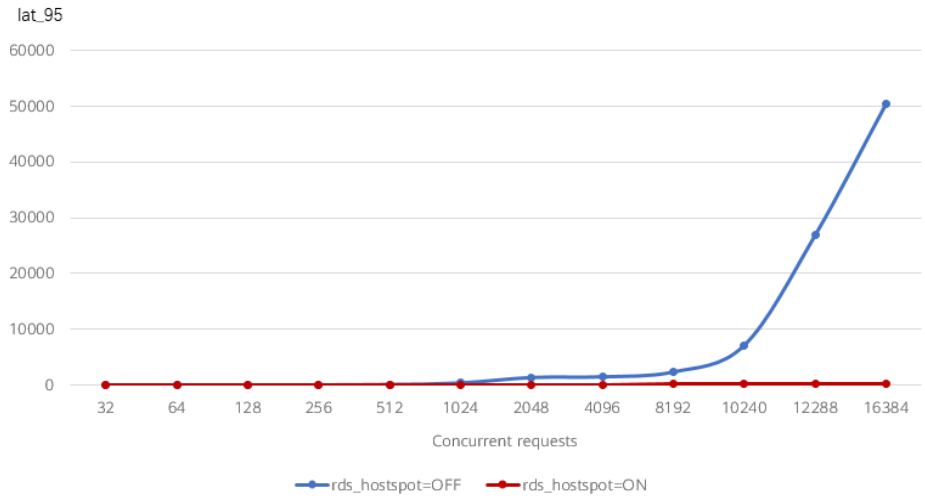




Test scenario 3: Updating eight hot rows of an instance with 32 vCPUs and 128 GB

Test result: The performance of 256 or less concurrent requests was not improved, but the performance of 512 or more concurrent requests was improved significantly, by 78 times.





2.13 Multi-tenant Management and Resource Isolation

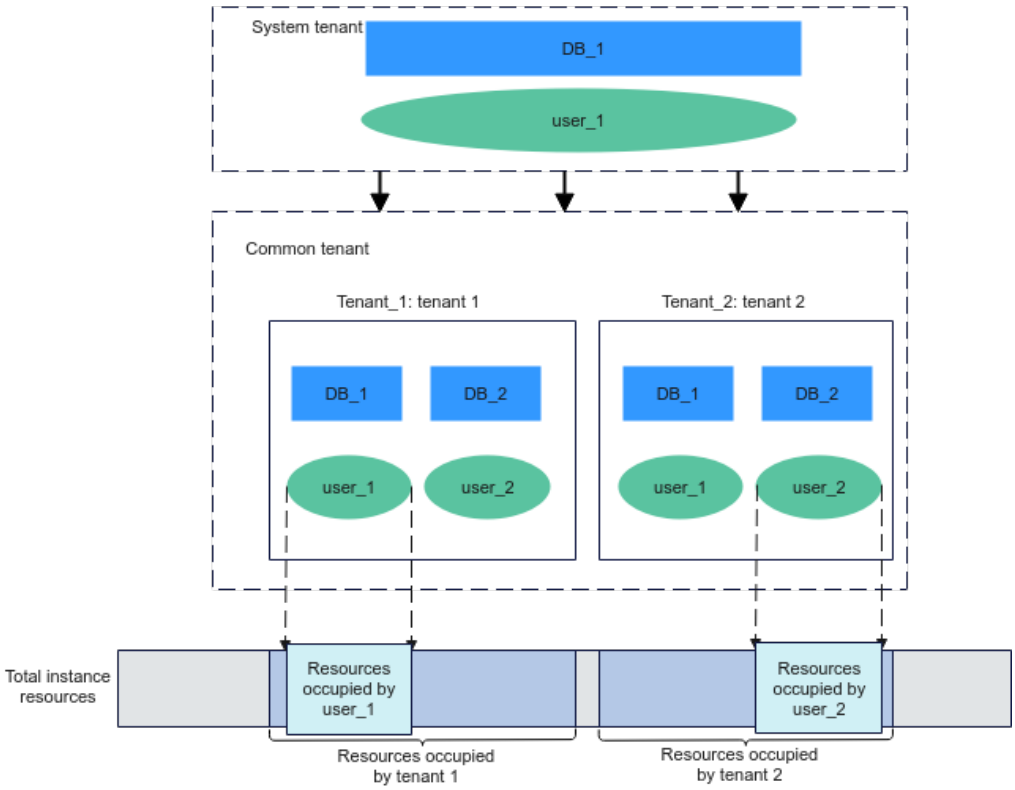
This section describes the syntax and usage of multi-tenant data isolation and resource isolation provided by GaussDB(for MySQL).

Overview

GaussDB(for MySQL) provides multi-tenant management to maximize database resource utilization. Data is isolated among tenants. Different tenants can only access their own data. There are tenant-level resource isolation and user-level resource isolation to avoid resource waster and improve performance. Resources can be dynamically adjusted to process workload peaks and troughs of different tenants or users in a timely manner.

The following figure shows the principle of multi-tenant management.

Figure 2-22 Multi-tenant management principle

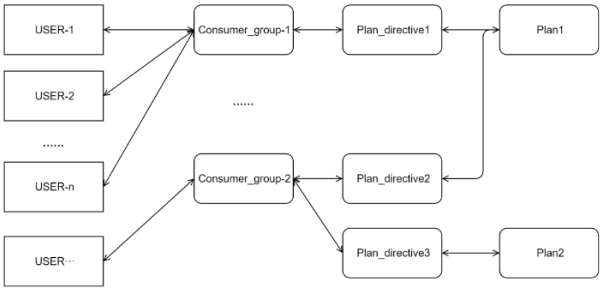


Basic Concepts

Table 2-37 describes the terms for tenant-level and user-level resource isolation.

Table 2-37 Term description

Level	Term
Tenant-level	**Tenant** Tenants are under a DB instance and above databases and users. Tenants are used for data isolation and resource isolation. However, databases can be accessed as a user. There are system tenants and common tenants. • **System tenant (sys_tenant)** System tenants are designed to accommodate users that exist before the implementation of the multi-tenant management. By default, these existing users belong to system tenants and are also referred to as system users. If a user of a system tenant can connect to and access the DB instance, the user can access the databases of all tenants. • **Common tenant (user_tenant)** Common tenants need to be created in the system tenants. The users and databases of each common tenant are completely isolated from those of the other common tenants. In addition, common tenants cannot access databases of the system tenants. During vCPU scheduling, common tenants are classified into dedicated tenants and shared tenants based on whether the value of **min_cpu** is greater than 0. – Dedicates tenants: **min_cpu** > **0**. vCPUs allocated to dedicated tenants at any time are not less than the value of **min_cpu**. – Shared tenants: **min_cpu** = **0**. The system preferentially ensures the resource requests of dedicated tenants and then allocates the remaining resources to shared tenants. In addition, the system reserves some vCPUs (specified by **mt_shared_cpu_reserved**) for shared tenants. You can change the value of **min_cpu** to change the roles between dedicated and shared tenants. **Resource configuration (resource_config)** A resource configuration indicates the resources that can be used by a tenant, enabling tenant-level resource isolation. Currently, only vCPUs can be isolated and scheduled.

Level	Term
User-level	Users are under DB instances and tenants. A tenant can have multiple users. **Figure 2-23** User-level resource configuration relationship  **Resource consumer group (consumer_group)** A set of users under the same resource configuration. Multiple users can belong to the same resource consumer group. Users in the same resource consumer group share resource configurations. **Resource plan directive (plan_directive)** Specific resource configuration. One resource plan directive is bound to one resource consumer group to describe the resource configuration of the consumer group. **Resource plan (plan)** Resource configuration set. One resource plan is bound to one or more resource plan directives. Enabling or disabling a resource plan can make the corresponding resource plan directive valid or invalid.

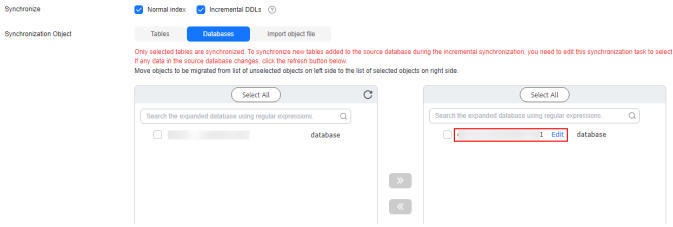
Constraints

- Multi-tenant management and resource isolation can only isolate tenant data, tenant vCPUs, and user vCPUs.
- The kernel version of your GaussDB(for MySQL) instance must be 2.0.54.240600 or later.
- Thread Pool must be enabled.
- Database names and usernames in a DB instance do not contain the at sign (@).
- Serverless instances do not support multi-tenant management and resource isolation.
- Tenant migration across instances:

  After the multi-tenant management and resource isolation is enabled, DRS can migrate data of all tenants. However, DRS does not synchronize multi-tenant metadata, so tenant information is not synchronized to the destination instance. To migrate a tenant from an instance to another instance, perform the following steps:

a. Select an instance that support multi-tenant management as the destination instance and manually create a tenant for the destination instance.

b. Use DRS data synchronization to create a database-level synchronization task. (If the tenant names at the source and destination ends are changed, you need to change the destination database name.)



c. Synchronize data.

- Binlog:

  Currently, tenant isolation cannot be implemented using binlogs. If common tenants pull binlogs, data among tenants will not be isolated. Users of common tenants are not allowed to pull binlogs.

  GaussDB(for MySQL) uses APIs related to tenants and resource isolation to record binlogs in row format. If **binlog_format** is not set to **row**, tenant and resource configuration information cannot be synchronized using binlogs.

- Proxy:
  - The **show processlist** command can be used with a proxy. When the command is used with a proxy, the command output displays node information.
  - The HTAP standard and lightweight editions do not support databases containing at signs (@). When a database of a common tenant is migrated to the HTAP engine, the destination database name will be changed. After **Auto Assign Requests to Column Store or Row Store Nodes** is enabled, the proxy requires that the source and destination database names be the same, so this function must be disabled for migrating databases of common tenants.
- Backup and restoration:

  When you restore data of an instance with multi-tenancy enabled to a new instance with multi-tenancy disabled, users and databases with at signs (@) in their names cannot be created for the new instance.

- Compatibility:
  - If multi-tenancy is enabled and then disabled, the names of databases or users cannot contain at signs (@).
  - For a common tenant, the maximum length of a database name is reduced from 64 characters to 50 characters, and the maximum length of a user name is reduced from 32 characters to 20 characters.
  - Currently, common tenants can only access **information_schema**. Other system databases cannot be accessed by common tenants.
  - Common tenants do not support tablespace-related syntax.

Enabling the Multi-Tenant Mode

Step 1 Log in to the management console.

Step 2 Click  in the upper left corner and select a region and project.

Step 3 Click  in the upper left corner of the page and choose **Databases** > **GaussDB(for MySQL)**.

Step 4 On the **Instances** page, click the instance name to go to the **Basic Information** page.

Step 5 In the **Instance Information** area, click  under **Multi-tenant**. In the displayed dialog box, click **OK**.

Before enabling multi-tenancy, ensure that the existing database names and usernames do not contain at signs (@), or the function fails to be enabled.

----End

Resource Management

Resource configurations and tenants are in one-to-many relationship. When a tenant is bound to a resource configuration, the vCPUs used by users of the tenant is restricted.

- Create a resource configuration.
  ```
  CREATE resource_config config_name MAX_CPU [=] {max_cpu_value} [MIN_CPU [=] {min_cpu_value}];
  ```
- Alter a resource configuration.
  ```
  ALTER resource_config config_name MAX_CPU [=] {max_cpu_value} [MIN_CPU [=] {min_cpu_value}];
  ```
- Drop a resource configuration.
  ```
  DROP resource_config config_name;
  ```
- Query a resource configuration.
  ```
  SELECT * FROM information_schema.DBA_RSRC_TENANT_RESOURCE_CONFIGS;
  ```


NOTE

- Only the **root** user can call the APIs for creating, altering, and dropping resource configurations.
- Parameter description:
  - **config_name**: resource configuration name. The value can contain up to 64 characters. Only uppercase letters, lowercase letters, digits, and underscores (_) are allowed.
  - **MAX_CPU**: maximum number of vCPUs that can be used by the resource configuration. The value ranges from **0.1** to the value of **mt_flavor_cpu**. The granularity is 0.1U.
  - **MIN_CPU**: minimum number of vCPUs that can be used by the resource configuration. This parameter is optional. The default value is the same as the value of **MAX_CPU**. Value range: **0** to the value of **MAX_CPU**. The granularity is 0.1U. If the value is set to **0**, the tenant is a shared tenant. If the value is greater than **0**, the tenant is a dedicated tenant.
- When a resource configuration is updated, if it has been bound to a tenant and the updated **MIN_CPU** value is greater than the original **MIN_CPU** value, check whether the new value meets the resource constraints, or the resource constraints are not verified.
- If a tenant is using a resource configuration, the resource configuration cannot be deleted.
- In actual use, the resource allocation of shared tenants does not depend on **MAX_CPU**. Resources are randomly contested.
- During vCPU contention, resources are allocated to tenants based on the value of **MIN_CPU** specified for each tenant. However, there is a certain error, which is usually within 1U.
- In give specifications, the performance of an instance with multi-tenancy enabled is lower than that of an instance with multi-tenancy disabled. In actual use, when multiple tenants preempt resources, the instance performance may be even lower.

Tenant Management

When creating a tenant, you need to bind it to a resource configuration to restrict the vCPUs used by users of the tenant.

- Create a tenant.
  ```
  CREATE TENANT tenant_name RESOURCE_CONFIG config_name [COMMENT [=] 'comment_string'];
  ```
- Alter a tenant.
  ```
  ALTER TENANT tenant_name RESOURCE_CONFIG config_name [COMMENT [=] 'comment_string'];
  ```
- Drop a tenant.
  ```
  DROP TENANT tenant_name;
  ```
- Query a tenant.
  ```
  SELECT * FROM information_schema.DBA_RSRC_TENANT;
  ```


NOTE

- Only the **root** user can call the APIs for creating, altering, and dropping tenants.
- Creating a tenant:
  - The value of **tenant_name** can contain up to 10 characters. Only lowercase letters, digits, and underscores (_) are allowed.
  - When a tenant is created, the system checks resource constraints to ensure that the sum of the **MIN_CPU** values in the resource configurations of all tenants meets the resource constraints.
- Altering a tenant:
  - If the **MIN_CPU** value of the newly bound resource configuration is at least that of the original resource configuration, the system checks the resource constraints. Ensure that the sum of the **MIN_CPU** values in the resource configurations of all tenants meets the resource constraints.
  - If the **MIN_CPU** value of a resource configuration bound to a dedicated tenant is **0**, the tenant becomes a shared tenant and the metadata related to user-level resource isolation associated with the tenant is deleted.
- Dropping a tenant:
  - Before dropping a tenant, you need to ensure that the databases and users of the tenant have been deleted, or the tenant cannot be deleted.
  - After a tenant is deleted, the user-level resource isolation configuration associated with the tenant will be deleted.

User Management

After the multi-tenant mode is enabled, there are users of system tenants and users of common tenants. Existing users belong to system tenants. New users can belong to system tenants or common tenants based on the interface semantics.

- Managing users under a system tenant

  Creating a user

  Create a user for a system tenant.

  ```
  CREATE user [IF NOT EXISTS] user_name@host;
  ```

  Create a user for a common tenant.

  ```
  CREATE user [IF NOT EXISTS] 'user_name@tenant_name'@host;
  ```

  Renaming a user

  Rename a user under a system tenant.

  ```
  RENAME USER user_from@host1 TO user_to@host2;
  ```

  Rename a user under a common tenant.

  ```
  RENAME USER 'user_from@tenant_name'@host1 TO 'user_to@tenant_name'@host2;
  ```

  Dropping a user

  Drop a user from a system tenant.

  ```
  DROP USER [IF EXISTS] user_name@host;
  ```

  Drop a user from a common tenant.

  ```
  DROP USER [IF EXISTS] 'user_name@tenant_name'@host;
  ```

  Authorizing a user

  Grant the priv_type permissions of **tenant_1** to **user1@tenant_1**.

  ```
  GRANT priv_type ON *.* to 'user_1@tenant_1'@'%' with grant option;
  ```

  View permissions.


```
SHOW grants for 'user_1@tenant_1';
```

- Managing users under a common tenant

  Creating a user

  Create a user for the current tenant.

  ```
  CREATE user [IF NOT EXISTS] user_name@host;
  ```

  Renaming a user

  ```
  RENAME USER user_from@host1 TO user_to@host2;
  ```

  Dropping a user

  ```
  DROP USER [IF EXISTS] user_name@host;
  ```

  Authorizing a user

  Grant the priv_type permissions of the current tenant to **user1**.

  ```
  GRANT priv_type ON *.* to 'user_1'@'%' with grant option;
  ```

  View permissions.

  ```
  SHOW grants for 'user_1';
  ```

  NOTE

  - When creating or dropping a user of a common tenant under a system tenant, you need to use the *user_name***@tenant_name** format.
  - The username of a common tenant can contain up to 20 characters.
  - Some special users cannot be created in a tenant, including **mysql.sys**, **mysql.session**, **mysql.infoschema**, and users reserved in the **rds_reserved_users** parameter.
  - When renaming a user under a common tenant, ensure that the values of **tenane_name** in **user_from** and **user_to** are the same, or an error is returned.
  - When multi-tenancy is disabled, users under a common tenant cannot be renamed.

Database Management

There are databases of system tenants and databases of common tenants. System tenants can access all databases, and common tenants can only access their own databases.

- Managing databases under a system tenant

  Creating a database

  Create a database for a system tenant.

  ```
  CREATE DATABASE [IF NOT EXISTS] `db_name`;
  ```

  Create a database for a common tenant.

  ```
  CREATE DATABASE [IF NOT EXISTS] `db_name@tanent_name`;
  ```

  Dropping a database

  Drop a database from a system tenant.

  ```
  DROP DATABASE [IF EXISTS] `db_name`;
  ```

  Drop a database from a common tenant.

  ```
  DROP DATABASE [IF EXISTS] `db_name@tanent_name`;
  ```

- Managing databases under a common tenant

  Create a database for the current tenant.

  ```
  CREATE DATABASE [IF NOT EXISTS] 'db_name';
  ```

  Drop a database from the current tenant.


```
DROP DATABASE [IF EXISTS] 'db_name';
```

NOTE

- When creating or dropping a database of a common tenant under a system tenant, you need to perform operations on the database in db_name@tenant_name mode.
- Currently, common tenants can only access the INFORMATION_SCHEMA system database. They cannot access the PERFORMANCE_SCHEMA, SYS, and MySQL system databases.
- Some special databases, such as **INFORMATION_SCHEMA**, **PERFORMANCE_SCHEMA**, **MYSQL**, **SYS**, and **__recyclebin**, cannot be created in a tenant.

- Allocating tenants to existing databases

  To ensure compatibility, after the upgrade or migration to a multi-tenant instance, the existing databases are under system tenants by default. You can use the following syntax to allocate the existing databases to specified tenants. After multi-tenancy is enabled, you can use the following syntax to allocate databases that are created by system tenants and are not allocated to common tenants to corresponding tenants.

  Allocating a database

  Allocate a database to common tenant **tenant_name**.

  ```
  ALTER DATABASE db_name TENANT = `tenant_name`;
  ```

  Reclaim a database to a system tenant.

  ```
  ALTER DATABASE db_name TENANT = ``;
  ```

  Querying the mappings

  ```
  SELECT * FROM information_schema.DBA_RSRC_TENANT_DB;
  ```

  NOTE

  - Only the **root** user can call the API for allocating a database.
  - If a database is created after multi-tenancy is enabled, the database named in the format of *db_name*@*tenant_name* cannot be allocated and the API for allocating a database cannot be called. Otherwise, an error is returned.
  - If the tenant is not found and is not empty, the API for allocating a database returns an error.

- Connecting to a database as a user of a tenant

  Under a system tenant, the original connection mode remains unchanged.

  Under a common tenant, the user must be in the format of *user_name*@*tenant_name*. The database must be in the format of *db_name* or *db_name*@*tenant_name*.

  ```
  mysql --host=**** -u user1@tenant_1 -D db1 -p pwssword;
  mysql --host=**** -u user1@tenant_1 -D db1@tenant_1 -p pwssword;
  ```

  After the connection is successful, the user is restricted by the resources of the corresponding tenant.

User-level Resource Configurations

By default, users under a tenant share the resources of the tenant. To restrict user-level resources, you can call the API in this section.

- **Managing resource consumer groups (consumer_group)**

  Connect to a database as a user of a tenant to manage resource consumer groups.

Creating a consumer group

```
dbms_resource_manager.create_consumer_group (
  consumer_group     CHAR(128),
  comment            CHAR(2000));
```

Binding a consumer group

If the value of **consumer_group** is not **''**, the user is bound to a consumer group.

```
dbms_resource_manager.set_consumer_group_mapping (
  attribute        CHAR(128),
  value            varbinary(128),
  consumer_group   CHAR(128));
```

Unbinding a consumer group

If the value of **consumer_group** is **''**, the user is unbound from the consumer group.

```
dbms_resource_manager.set_consumer_group_mapping (
  attribute        CHAR(128),
  value            varbinary(128),
  '');
```

Deleting a consumer group

```
dbms_resource_manager.delete_consumer_group (
   consumer_group   CHAR(128));
```

Querying a consumer group

The **DBA_RSRC_CONSUMER_GROUPS** view records the association between tenants and consumer groups.

```
select * from information_schema.DBA_RSRC_CONSUMER_GROUPS;
```

Viewing consumer group mappings

The **DBA_RSRC_GROUP_MAPPINGS** view records the association between tenants, users, and consumer groups.

```
select * from information_schema.DBA_RSRC_GROUP_MAPPINGS;
```


NOTE

- Sharded tenants cannot use the APIs for creating, binding, unbinding, or deleting consumer groups.
- Parameter description:

  consumer_group: consumer group name. Only uppercase letters, lowercase letters, digits, and underscores (_) are allowed. You can set this parameter to '' when unbinding a consumer group.

  comment: description of the resource consumer group. The value can be ''.

  attribute: mapping attribute to be added or modified. The current version supports only **USER**.

  value: mapping attribute to be added or modified. The current version supports only usernames.
- When a consumer group is deleted, the plan directive and consumer group mapping entries corresponding to the resource consumer group are also deleted.
- If multi-tenancy is enabled, when a user is deleted, the consumer group items associated with the user are also deleted.
- If multi-tenancy is enabled, when the user is renamed, the consumer group items associated with a user are also updated.
- If multi-tenancy is enabled again, if the corresponding user is invalid, the invalid items associated with the user are automatically deleted.
- After multi-tenancy is disabled, if a user is deleted and then a user with the same name is created, the items corresponding to the user will be retained when multi-tenancy is enabled later.

Resource Plan Management

Connect to a database as a user of a tenant to manage resource plans.

- Create a resource plan.

```
dbms_resource_manager.create_plan (
  plan_name         VARCHAR(128),
  comment           VARCHAR(2000));
```

- Enable a resource plan.

```
dbms_resource_manager.set_resource_manager_plan(
   plan_name     VARCHAR(128));
```

- Disable a resource plan.

```
dbms_resource_manager.set_resource_manager_plan ('');
```

- Delete a resource plan.

```
dbms_resource_manager.delete_plan (
   plan_name     VARCHAR(128));
```

- Query a resource plan.

  The **DBA_RSRC_PLANS** view records details about the current plan.

```
SELECT * FROM information_schema.DBA_RSRC_PLANS;
```


NOTE

- Sharded tenants cannot use the APIs for creating, enabling, disabling, or deleting resource plans.
- Parameter description:

  plan_name: resource plan name. Only uppercase letters, lowercase letters, digits, and underscores (_) are allowed.

  comment: description of the resource plan. The value can be ''.
- If you delete an enabled resource plan, it will be left empty and the corresponding resource plan directive will be deleted.
- **mt_resource_plan_num**: number of plans. By default, there are up to 128 plans.

Resource Plan Directive Management

Connect to a database as a user of a tenant to manage resource plan directives.

- **Creating a resource plan directive**

```
dbms_resource_manager.create_plan_directive (
  plan                  CHAR(128),
  group_or_subplan      CHAR(128),
  comment               VARCHAR(2000),
  mgmt_p1               bigint(20),
  utilization_limit     bigint(20));
```

- **Updating a resource plan directive**

```
dbms_resource_manager.update_plan_directive (
  plan                  CHAR(128),
  group_or_subplan      CHAR(128),
  new_comment           VARCHAR(2000),
  new_mgmt_p1           bigint(20),
  new_utilization_limit bigint(20));
```

- **Deleting a resource plan directive**

```
dbms_resource_manager.delete_plan_directive (
   plan                 CHAR(128),
   group_or_subplan     VARCHAR(128));
```

- **Querying a resource plan directive**

  DBA_RSRC_PLAN_DIRECTIVES records the association between the plans and consumer groups as well as the resource configurations of the consumer groups.

```
SELECT * FROM information_schema.DBA_RSRC_PLAN_DIRECTIVES;
```


NOTE

- Sharded tenants cannot use the APIs for creating, updating, or deleting resource plan directives.
- Parameter description:

  plan: plan name.

  group_or_subplan: consumer group name.

  comment: description of the resource plan directive. The value can be **''**.

  mgmt_p1: committed vCPU percentage allocated to the consumer group when the system is fully loaded. The value range is [0, 100] (**100**: 100% vCPUs of the tenant are used).

  utilization_limit: upper limit of vCPUs used by the consumer group. The value range is [1, 100]. The value **100** indicates that a maximum of all vCPUs of a tenant can be used. The value **70** indicates that a maximum of 70% vCPUs of a tenant can be used.
- If you delete a plan directive that has been enabled, the resource configuration of the corresponding user will become invalid.
- The total resources used by all users in a consumer group cannot exceed the resource limit of the current consumer group. For example, if a tenant has users **user1** and **user2** (they belong to **consumer_group1**) and **UTILIZATION_LIMIT** of **consumer_group1** is **70**, **user1** and **user2** can use up to 70% of the vCPU resources of the current tenant.

User-level Configuration Clearing

Connect to a database as a user of a tenant and clear the resource configuration data of the tenant, including data in the **DBA_RSRC_CONSUMER_GROUPS**, **DBA_RSRC_GROUP_MAPPINGS**, **DBA_RSRC_PLAN_DIRECTIVES**, and **DBA_RSRC_PLANS** tables.

```
dbms_resource_manager.clear_all_configs();
```

vCPU Usage Statistics

- **User-level vCPU usage**

  The **information_schema.cpu_summary_by_user** table is added to display the vCPU usage of each user under a tenant.

  ```
  SELECT * FROM information_schema.cpu_summary_by_user;
  ```

  NOTE

  - A consumer group must be configured for the current user in advance.
  - The column names in the query result are described as follows:

    TENANT_NAME: name of the tenant the user belongs to.

    USER_NAME: username.

    CPU_USAGE: vCPU usage of the user, that is, the ratio of the vCPUs used by the user to the **MAX_CPU** value configured for the tenant. For example, if the **MAX_CPU** value configured for the current tenant is **4** and the user actually uses 2 vCPUs, the value of **CPU_USAGE** is **50%**.

- **Tenant-level vCPU usage**

  The **information_schema.cpu_summary_by_tenant** table is added to display the vCPU usage of each tenant.

  ```
  SELECT * FROM information_schema.cpu_summary_by_tenant;
  ```


> **NOTE**
>
> The column names in the query result are described as follows:
>
> **TENANT_NAME**: tenant name.
>
> **CPU_USAGE**: vCPU usage of the tenant, that is, the ratio of the vCPUs used by the tenant to the **MAX_CPU** value. For example, if the **MAX_CPU** value configured for the current tenant is **4** and the tenant actually uses 2 vCPUs, the value of **CPU_USAGE** is **50%**.

2.14 Column Compression

To reduce the storage occupied by data pages and costs, GaussDB(for MySQL) provides algorithms ZLIB and ZSTD for fine-grained column compression. You can select either of them to compress large columns that are not frequently accessed based on the compression ratio and performance. Automatic column compression is also supported.

Application scenario: There are large columns that are not frequently accessed in tables and users want to compress these columns to reduce costs.

Constraints

- The kernel version of your GaussDB(for MySQL) instance must be 2.0.54.240600 or later.
- Partitioned tables, temporary tables, and non-InnoDB engine tables are not supported.
- A column to be compressed cannot contain an index (primary key, unique index, secondary index, foreign key, or full-text index).
- Only the following data types are supported: BLOB (including TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB), TEXT (including TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT). VARCHAR, and VARBINARY.
- This feature cannot be used on generated columns.
- EXCHANGE PARTITION cannot be executed between a partitioned table and a table with compressed columns.
- IMPORT TABLESPACE is not supported.
- This feature can be used only in CREATE TABLE, ALTER TABLE ADD, ALTER TABLE CHANGE, and ALTER TABLE MODIFY statements.
- ALTER TABLE ADD COLUMN does not support the INSTANT algorithm. The INSTANT algorithm cannot be used when the ALTER TABLE {CHANGE| MODIFY} syntax involves data changes.
- In automatic compression scenarios (**rds_column_compression** = **2**), compression attributes can be added only when the maximum length of a column is at least the compression threshold (**rds_column_compression_threshold**). In explicit compression scenarios (**rds_column_compression** = **1**), if the maximum length of a column is less than the compression threshold, compression attributes can be added but a warning message is received.
- If a table contains compressed columns, NDP is not supported.
- When you manually perform binlog synchronization, ALTER statements are incompatible. You are advised to use HINT.

- When you use DRS to migrate data from one instance to another that does not support column compression, the compression attribute is eliminated. The full migration task can be performed. During incremental migration, if ALTER statements contain compressed columns, the migration task fails.
- When physical backups are used to restore data, the related versions must support column compression.
- If column compression has been used after the version upgrade, the version cannot be rolled back to a version without this feature.

Syntax

The **column_definition** definition is extended to support compression when column attributes are defined in CREATE TABLE, ALTER TABLE ADD, ALTER TABLE CHANGE, and ALTER TABLE MODIFY statements.

```
create_definition: {
    col_name column_definition
  | {INDEX | KEY} [index_name] [index_type] (key_part,...)
      [index_option] ...
  | {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] PRIMARY KEY
      [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
      [index_name] [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (col_name,...)
      reference_definition
  | check_constraint_definition
}

alter_option: {
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST | AFTER col_name]
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  ...
```

column_definition is as follows:

```
column_definition: {
    data_type [NOT NULL | NULL] [DEFAULT {literal | (expr)} ]
      [VISIBLE | INVISIBLE]
      [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
      [COMMENT 'string']
      [COLLATE collation_name]
      [COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
      [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}|COMPRESSED[={ZLIB|ZSTD}**]]
      [ENGINE_ATTRIBUTE [=] 'string']
      [SECONDARY_ENGINE_ATTRIBUTE [=] 'string']
      [STORAGE {DISK | MEMORY}]
      [reference_definition]
      [check_constraint_definition]
  | data_type
```

```
    [COLLATE collation_name]
    [GENERATED ALWAYS] AS (expr)
    [VIRTUAL | STORED] [NOT NULL | NULL]
    [VISIBLE | INVISIBLE]
    [UNIQUE [KEY]] [[PRIMARY] KEY]
    [COMMENT 'string']
    [reference_definition]
    [check_constraint_definition]
}
```

Parameter Description

Table 2-38 Parameter description

Parameter	Description	Value Range	Default Value	Level	Dynamic Validation
rds_column_compression	• If this parameter is set to **0**, column compression is disabled. Compressed columns cannot be created explicitly or automatically. • If this parameter is set to **1**, compressed columns can only be created explicitly. • If this parameter is set to **2**, compressed columns can be created explicitly or automatically.	[0,2]	0	GLOBAL	Yes

Parameter	Description	Value Range	Default Value	Level	Dynamic Validation
rds_default_column_compression_algorithm	Controls default compression algorithm for column compression. The algorithm is used when • You explicitly create a compressed column without specifying a specific compression algorithm. • A compressed column is created automatically.	ZLIB or ZSTD	ZLIB	GLOBAL	Yes
rds_column_compression_threshold	Controls the threshold for triggering column compression. • When the maximum length of a column is less than this threshold, a compressed column can be explicitly created, but a message is displayed indicating that the compressed column cannot be automatically created. • When the maximum length of a column is at least to the threshold, a compressed column can be created explicitly or automatically.	[20-4294967295]	100	GLOBAL	Yes

Parameter	Description	Value Range	Default Value	Level	Dynamic Validation
rds_zlib_column_compression_level	Specifies the compression level of the ZLIB column compression algorithm. ● If this parameter is set to **0**, columns are not compressed. ● Setting this parameter to a value other than 0 will affect the compression speed and effectiveness. A smaller value indicates faster compression but a poorer effect, while a larger value indicates slower compression but a better effect.	[0,9]	6	GLOBAL	Yes
rds_zstd_column_compression_level	Specifies the compression level of the ZSTD column compression algorithm. A smaller value indicates faster compression but a poorer effect, while a larger value indicates slower compression but a better effect.	[1,22]	3	GLOBAL	Yes

Example

1. Explicitly create a compressed column.

```
mysql> show variables like 'rds_column_compression';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| rds_column_compression | 1     |
+------------------------+-------+
1 row in set (0.00 sec)
```



```
mysql> show variables like 'rds_default_column_compression_algorithm';
+-----------------------------------------+-------+
| Variable_name                           | Value |
+-----------------------------------------+-------+
| rds_default_column_compression_algorithm | ZLIB  |
+-----------------------------------------+-------+
1 row in set (0.00 sec)

mysql> create table t1(c1 varchar(100) compressed, c2 varchar(100) compressed=zlib, c3
varchar(100) compressed=zstd) default charset=latin1;
Query OK, 0 rows affected (0.06 sec)

mysql> show create table t1\G
*************************** 1. row ***************************
Table: t1
Create Table: CREATE TABLE `t1` (
`c1` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
`c2` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
`c3` varchar(100) /*!99990 800220201 COMPRESSED=ZSTD */ DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

2. Automatically create a compressed column.

```
mysql> set global rds_column_compression = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'rds_column_compression';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| rds_column_compression | 2     |
+------------------------+-------+
1 row in set (0.01 sec)

mysql> show variables like 'rds_column_compression_threshold';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| rds_column_compression_threshold | 100   |
+----------------------------------+-------+
1 row in set (0.01 sec)

mysql> show variables like 'rds_default_column_compression_algorithm';
+-----------------------------------------+-------+
| Variable_name                           | Value |
+-----------------------------------------+-------+
| rds_default_column_compression_algorithm | ZLIB  |
+-----------------------------------------+-------+
1 row in set (0.01 sec)

mysql> create table t2(c1 varchar(99), c2 varchar(100)) default charset=latin1;
Query OK, 0 rows affected (0.05 sec)

mysql> show create table t2\G
*************************** 1. row ***************************
Table: t2
Create Table: CREATE TABLE `t2` (
`c1` varchar(99) DEFAULT NULL,
`c2` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

3. Disable column compression.

```
mysql> set global rds_column_compression = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'rds_column_compression';
+------------------------+-------+
| Variable_name          | Value |
```



```
+------------------------+-------+
| rds_column_compression | 0     |
+------------------------+-------+
1 row in set (0.01 sec)

mysql> show variables like 'rds_column_compression_threshold';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| rds_column_compression_threshold | 100   |
+----------------------------------+-------+
1 row in set (0.01 sec)

mysql> create table t3(c1 varchar(100) compressed, c2 varchar(100) compressed=zlib, c3
varchar(100) compressed=zstd) default charset=latin1;
Query OK, 0 rows affected, 3 warnings (0.04 sec)

mysql> show create table t3\G
*************************** 1. row ***************************
Table: t3
Create Table: CREATE TABLE `t3` (
`c1` varchar(100) DEFAULT NULL,
`c2` varchar(100) DEFAULT NULL,
`c3` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

Result Verification

1. Run the **show create table** statement to display the table structure information. If the information contains "/*! If the content in "99990 800220201 COMPRESSED=xxxx */", column compression is used.

   Example

   ```
   mysql> show create table t1\G
   *************************** 1. row ***************************
         Table: t1
   Create Table: CREATE TABLE `t1` (
     `c1` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
     `c2` varchar(100) /*!99990 800220201 COMPRESSED=ZLIB */ DEFAULT NULL,
     `c3` varchar(100) /*!99990 800220201 COMPRESSED=ZSTD */ DEFAULT NULL
   ) ENGINE=InnoDB DEFAULT CHARSET=latin1
   1 row in set (0.00 sec)
   ```

2. Use the system view **information_schema.columns** to query compressed columns.

   Example

   ```
   mysql> select TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, EXTRA from
   information_schema.columns where extra like '%compressed%';
   +--------------+------------+-------------+-----------------+
   | TABLE_SCHEMA | TABLE_NAME | COLUMN_NAME | EXTRA           |
   +--------------+------------+-------------+-----------------+
   | test         | t1         | c1          | COMPRESSED=ZLIB |
   | test         | t1         | c2          | COMPRESSED=ZLIB |
   | test         | t1         | c3          | COMPRESSED=ZSTD |
   | test         | t2         | c2          | COMPRESSED=ZLIB |
   +--------------+------------+-------------+-----------------+
   4 rows in set (0.50 sec)
   ```

3. Query the status information to determine the actual number of times that the column compression or decompression API is called.

   ```
   mysql> show global status like '%column%compress%';
   +---------------------------------+-------+
   | Variable_name                   | Value |
   +---------------------------------+-------+
   | Innodb_column_compress_count    | 243   |
   ```



```
| Innodb_column_uncompress_count | 34    |
+--------------------------------+-------+
```

4. Run the following statement or view the information on the monitoring page to compare the table sizes before and after compression and check the compression effect.

```
SELECT table_name AS Table, round(((data_length + index_length) / 1024 / 1024), 2) AS Size in MB
FROM information_schema.TABLES WHERE table_schema = "***" and table_name='***'
```

Compression Ratio and Performance Impact Verification

1. Insert 10,000 rows of data in a table randomly. Each row consists of 32 character strings returned by 400 MD5 functions.

```
CREATE TABLE `random_data` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`data` longtext,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

DELIMITER $$
CREATE PROCEDURE `generate_random_data`()
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE j INT DEFAULT 1;
  DECLARE str longtext;
  WHILE i <= 10000 DO
    SET j = 1;
    SET str = '';
    WHILE j <= 400 DO
      SET str = CONCAT(str, MD5(RAND()));
      SET j = j + 1;
    END WHILE;
    INSERT INTO `random_data` (`data`) VALUES (str);
    SET i = i + 1;
  END WHILE;
END$$
DELIMITER ;
```

   Set **rds_column_compression** to **0** first and then set it to **2**. Retain the default values for other parameters. Import the preceding table structure and invoke a stored procedure to insert data. Use the ZLIB or ZTSD algorithm to compress data. The ratio of the data file size before and after compression is 1.8.

2. Use sysbench to import 64 tables. Each table contains 10 million rows of data. The types of the **c** and **pad** columns are changed to varchar. The modified table structure is as follows:

```
CREATE TABLE `sbtest1` (
  `id` int NOT NULL AUTO_INCREMENT,
  `k` int NOT NULL DEFAULT '0',
  `c` varchar(120) COLLATE utf8mb4_0900_bin NOT NULL DEFAULT '',
  `pad` varchar(60) COLLATE utf8mb4_0900_bin NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB AUTO_INCREMENT=10000001 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_bin
```

   - Set **rds_column_compression** to **0** first and then set it to **2**, and retain the default values for other parameters. Import the table structure and data. After the calculation, only column **c** is compressed using ZLIB or ZSTD, the ratio of the data file size before and after compression is 1.2.
   - Theoretically, a higher compression level has a greater impact on performance. After compression, the performance loss is about 10%.

