# TDSQL MySQL版

# 开发指南

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

# 文档目录


- 开发指南
  - InnoDB 引擎
    - 概述
    - 使用限制
    - 兼容性
    - 连接数据库
    - JOIN 和子查询
    - Sequence
    - 常用 DML
    - 读写分离
    - 分布式事务
    - 建表
    - 连接保护
    - 保留关键字
    - 全局唯一字段
    - 数据导出导入
    - 数据库管理语句
    - 透传 SQL
    - 预处理
    - 二级分区
    - 错误码和错误信息
  - TDStore 引擎
    - 与 MySQL 兼容性
      - 使用说明
      - OnlineDDL 说明
      - 列存兼容性说明
        - SQL 语法兼容
        - 表达式兼容
    - Stale Read
    - SQL 参考
      - CREATE DATABASE
      - CREATE TABLE
      - DROP DATABASE
      - DROP TABLE
      - FLASHBACK TABLE
      - OPTIMIZE TABLE

- PURGE RECYCLEBIN
- SHOW RECYCLEBIN
- 列存向量化引擎 Join Order Hint

PL 系统包

- DBMS_ADMIN
  - TDSTORE_FORCE_FULL_COMPACTION

系统表和系统视图

- PERFORMANCE_SCHEMA
  - DATA_LOCKS
  - DATA_LOCK_WAITS
  - METADATA_LOCKS
- INFORMATION_SCHEMA
  - DDL_JOBS
  - PERSIST
  - TDSTORE_COLUMNAR_ACTIVE_LOAD_TASK_INFO
  - TDSTORE_COLUMNAR_COMPACTION_INFO
  - TDSTORE_COLUMNAR_FILE_DELETION_INFO
  - TDSTORE_COLUMNAR_FILES_INFO
  - TDSTORE_COLUMNAR_HISTORY_LOAD_TASK_INFO
  - TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO

数据库参数说明

- 计算引擎参数
  - 计算引擎参数概述
  - 计算引擎参数总览
  - auto_increment_increment
  - auto_increment_offset
  - autocommit
  - binlog_dump_cache_max_size
  - character_set_server
  - collation_server
  - connect_timeout
  - ddl_recovery_wait_time
  - long_query_time
  - log_service_dump_batch_size
  - max_allowed_packet
  - max_connect_errors
  - max_connections
  - max_execution_time

max_user_connections
net_read_timeout
net_retry_count
net_write_timeout
optimizer_switch
sql_mode
table_definition_cache
table_open_cache
tdsql_auto_increment_batch_size
tdsql_bulk_commit_records
tdsql_cache_range_max_use_cnt
tdsql_get_index_stats_from_tdstore
tdsql_max_collect_ranges
tdsql_max_retry
tdsql_optimize_pk_point_select
tdsql_optimize_table_rows_count
tdsql_retry_interval_growth
tdsql_retry_interval_msec_max
tdsql_retry_interval_msec_min
tdsql_stale_read_role
tdsql_skip_trans_write_lock
tdsql_split_trans_during_alter_copy
tdsql_transaction_max_participants
tdsql_txn_keep_alive_lease_sec
tdsql_update_local_glsv_interval
tdsql_update_variables_interval
tdsql_use_tdstore_estimated_rows
tdsql_lock_wait_timeout
wait_timeout
tdsql_recycle_bin_enabled
log_service_mysql_client_timeout
log_service_mysql_info_queue_size
audit_log_policy
log_bin_trust_function_creators
max_prepared_stmt_count
tdsql_enable_trigger
tdsql_enable_view
tdsql_use_online_copy_ddl

版权所有：腾讯云计算（北京）有限责任公司

tdstore_bulk_load_txn_rpc_timeout_threshold_ms
存储引擎参数
存储引擎参数概述
存储引擎参数总览
tdstore_bulk_load_total_merge_buffer_size
tdstore_bulk_load_merge_chunk_size
tdstore_deadlock_detect
tdstore_enter_readonly_threshold
tdstore_force_accept_conf_log
tdstore_leave_readonly_threshold
tdstore_long_query_time
tdstore_long_query_with_detail_stage
tdstore_max_db_log_file_num
tdstore_max_db_log_file_size
tdstore_max_txn_size
tdstore_min_free_disk_space
tdstore_raft_data_path
tdstore_txn_max_entry_size
tdstore_user_cf_hard_pending_compaction_bytes_limit
tdstore_user_cf_level0_slowdown_writes_trigger
tdstore_user_cf_level0_stop_writes_trigger
tdstore_user_cf_soft_pending_compaction_bytes_limit
tdstore_enable_node_alive_detection
tdstore_deadlock_victim
tdstore_transfer_leader_job_cost_threshold_ms
tdstore_transfer_leader_raft_cost_threshold_ms
tdstore_data_import_rate_limiter_bytes_per_sec
错误码信息

# 开发指南
# InnoDB 引擎
# 概述

最近更新时间：2022-02-11 10:33:43

在 SQL 使用上，分布式实例高度兼容 MySQL 的协议和语法，但由于架构的差异，对于 SQL 有一定的限制，同时为了更好地发挥分布式的优势，建议业务在使用时尽量参考下文的建议。
分布式实例提供水平扩容能力，适合海量数据的场景。具有如下功能特性：

- 提供了灵活的读写分离模式
- 支持全局的 order by、group by、limit 操作
- 聚合函数支持 sum、count、avg、min、max 等
- 支持跨节点（set）的 join、子查询
- 支持预处理协议
- 支持全局唯一字段，支持 sequence
- 支持分布式事务
- 支持两级分区
- 提供特定的 SQL 查询整个集群的配置和状态

分布式实例支持三种不同类型的表：

- **分表**：即水平拆分表，该表从业务视角是一张完整的逻辑表，但后端根据分表键（shardkey）的 HASH 值将数据分布到不同的节点（set）中。
- **单表**：又名 Noshard 表，无需拆分，且没有做任何特殊处理的表，目前分布式实例将该表默认存放在第一个物理节点组（set）中。
- **广播表**：又名小表广播技术，即设置为广播表后，该表的所有操作都将广播到所有节点（set）中，每个 set 都有该表的全量数据，常用于业务系统的配置表等。

**注意**

- 在分布式实例中，如果两张表**分表键**相等，这意味着，两张表相同的分表键对应的行，一定存储于相同的物理节点组中。这种场景通常被称为组拆分（groupshard），会极大提高业务联合查询等语句的处理效率。
- 由于单表默认放置在第一个 set 上，如果在分布式实例中建立了大量的单表，则会导致第一个 set 的负载太大。
- 除特殊情况外，建议在分布式实例中尽量都使用分表。

# 使用限制

最近更新时间：2024-09-20 11:49:31

## 大特性限制

- 不支持自定义函数、事件、表空间。
- 不支持视图、存储过程、触发器、游标。
- 不支持外键、自建分区。
- 不支持复合语句，如 BEGIN END、LOOP。
- 不支持主备同步相关的 SQL。
- 不支持全文索引。

## 小语法限制

### DDL

- 不支持 CREATE TABLE ... SELECT
- 不支持 CREATE TEMPORARY TABLE
- 不支持 CREATE/DROP/ALTER SERVER/LOGFILE GROUP
- 不支持 ALTER 对分表键（shardkey）进行改名

### DML

- 不支持 SELECT INTO OUTFILE/INTO DUMPFILE/INTO var_name
- 不支持 query_expression_options，如：HIGH_PRIORITY/STRAIGHT_JOIN/SQL_SMALL_RESULT/SQL_BIG_RESULT/SQL_BUFFER_RESULT/SQL_CACHE/SQL_NO_CACHE/SQL_CALC_FOUND_ROWS。
- 不支持不带列名的 INSERT/REPLACE
- 不支持全局的 DELETE/UPDATE 使用 ORDER BY/LIMIT（版本>=1.14.4支持）
- 不支持不带 WHERE 条件的 UPDATE/DELETE
- 不支持 LOAD DATA/XML
- 不支持 SQL 中使用 DELAYED 和 LOW_PRIORITY，没有效果。
- 不支持 INSERT ... SELECT（版本>1.14.4支持）
- 不支持 SQL 中对于变量的引用和操作，如：SET @c=1, @d=@c+1; SELECT @c, @d
- 不支持 index_hint
- 不支持 HANDLER/DO

### 管理 SQL

- 不支持 ANALYZE/CHECK/CHECKSUM/OPTIMIZE/REPAIR TABLE，需要用透传语法。
- 不支持 CACHE INDEX
- 不支持 FLUSH
- 不支持 KILL
- 不支持 LOAD INDEX INTO CACHE
- 不支持 RESET
- 不支持 SHUTDOWN
- 不支持 SHOW BINARY LOGS/BINLOG EVENTS
- 不支持 SHOW WARNINGS/ERRORS和LIMIT/COUNT 的组合
- 不支持 SHOW GRANTS FOR CURRENT_USER 语句，需使用 SHOW GRANTS FOR <用户名称> 进行查询。

## 其他限制

默认支持最大建表数量为5000，如需超越该限制，可 提交工单 申请。

# 兼容性

最近更新时间：2024-04-25 16:13:52

## 语言结构

分布式实例支持所有 MySQL 使用的文字格式，包括：

```
String Literals
Numeric Literals
Date and Time Literals
Hexadecimal Literals
Bit-Value Literals
Boolean Literals
NULL Values
```

## String Literals

String Literals 是一个 bytes 或者 characters 的序列，两端被单引号 `'` 或者双引号 `"` 包围，TDSQL MySQL版 目前不支持 ANSI_QUOTES SQL MODE，双引号 `"` 包围的始终认为是 String Literals，而不是 identifier。
不支持 character set introducer，即 `[_charset_name]'string' [COLLATE collation_name]` 这种格式。
支持的转义字符：

```
\0: ASCII NUL (X'00') 字符
\': 单引号
\": 双引号
\b: 退格符号
\n: 换行符
\r: 回车符
\t: tab 符（制表符）
\z: ASCII 26 (Ctrl + Z)
\\: 反斜杠 \
\%: \%
\_: _
```

## Numeric Literals

数值字面值包括 integer、Decimal 类型、浮点数字面值。
integer 可以包括 `.` 作为小数点分隔，数字前可以有 `-` 或者 `+` 来表示正数或者负数。

精确数值字面值可以表示为如下格式：`1, .2, 3.4, -5, -6.78, +9.10`。
科学记数法，如下格式：`1.2E3, 1.2E-3, -1.2E3, -1.2E-3`。

## Date and Time Literals

DATE 支持如下格式：

```
'YYYY-MM-DD' or 'YY-MM-DD'
'YYYYMMDD' or 'YYMMDD'
YYYYMMDD or YYMMDD

如：'2012-12-31', '2012/12/31', '2012^12^31',  '2012@12@31'  '20070523' ,
'070523'
```

DATETIME，TIMESTAMP 支持如下格式：

```
'YYYY-MM-DD HH:MM:SS' or 'YY-MM-DD HH:MM:SS'
'YYYYMMDDHHMMSS' or 'YYMMDDHHMMSS'
YYYYMMDDHHMMSS or YYMMDDHHMMSS

如'2012-12-31 11:30:45', '2012^12^31 11+30+45', '2012/12/31 11*30*45',
'2012@12@31 11^30^45', 19830905132800
```

## Hexadecimal Literals

支持格式如下：

```
X'01AF'
X'01af'
x'01AF'
x'01af'
0x01AF
0x01af
```

## Bit-Value Literals

支持格式如下：

```
b'01'
B'01'
0b01
```

## Boolean Literals

常量 TRUE 和 FALSE 等于1和0，大小写不敏感。

```
mysql>  SELECT TRUE, true, FALSE, false;
+------+------+-------+-------+
| TRUE | TRUE | FALSE | FALSE |
+------+------+-------+-------+
|    1 |    1 |     0 |     0 |
+------+------+-------+-------+
1 row in set (0.03 sec)
```

## NULL Values

NULL 代表数据为空，大小写不敏感，与 \N（大小写敏感）同义。
需要注意的是 NULL 跟0并不一样，跟空字符串 `''` 也不一样。

## 字符集和时区

支持 MySQL 的所有字符集和字符序：

```
mysql> show character set;
+----------+---------------------------------+---------------------+----
----+
| Charset  | Description                     | Default collation   |
Maxlen |
+----------+---------------------------------+---------------------+----
----+
| big5     | Big5 Traditional Chinese        | big5_chinese_ci     |
2 |
| dec8     | DEC West European               | dec8_swedish_ci     |
1 |
| cp850    | DOS West European               | cp850_general_ci    |
1 |
| hp8      | HP West European                | hp8_english_ci      |
1 |
| koi8r    | KOI8-R Relcom Russian           | koi8r_general_ci    |
1 |
| latin1   | cp1252 West European            | latin1_swedish_ci   |
1 |
| latin2   | ISO 8859-2 Central European     | latin2_general_ci   |
1 |
```

```
| swe7     | 7bit Swedish                    | swe7_swedish_ci     |
1 |
| ascii    | US ASCII                        | ascii_general_ci    |
1 |
| ujis     | EUC-JP Japanese                 | ujis_japanese_ci    |
3 |
| sjis     | Shift-JIS Japanese              | sjis_japanese_ci    |
2 |
| hebrew   | ISO 8859-8 Hebrew               | hebrew_general_ci   |
1 |
| tis620   | TIS620 Thai                     | tis620_thai_ci      |
1 |
| euckr    | EUC-KR Korean                   | euckr_korean_ci     |
2 |
| koi8u    | KOI8-U Ukrainian                | koi8u_general_ci    |
1 |
| gb2312   | GB2312 Simplified Chinese       | gb2312_chinese_ci   |
2 |
| greek    | ISO 8859-7 Greek                | greek_general_ci    |
1 |
| cp1250   | Windows Central European        | cp1250_general_ci   |
1 |
| gbk      | GBK Simplified Chinese          | gbk_chinese_ci      |
2 |
| latin5   | ISO 8859-9 Turkish              | latin5_turkish_ci   |
1 |
| armscii8 | ARMSCII-8 Armenian              | armscii8_general_ci |
1 |
| utf8     | UTF-8 Unicode                   | utf8_general_ci     |
3 |
| ucs2     | UCS-2 Unicode                   | ucs2_general_ci     |
2 |
| cp866    | DOS Russian                     | cp866_general_ci    |
1 |
| keybcs2  | DOS Kamenicky Czech-Slovak      | keybcs2_general_ci  |
1 |
| macce    | Mac Central European            | macce_general_ci    |
1 |
| macroman | Mac West European               | macroman_general_ci |
1 |
| cp852    | DOS Central European            | cp852_general_ci    |
1 |
```

```
| latin7    | ISO 8859-13 Baltic              | latin7_general_ci    |
1 |
| utf8mb4   | UTF-8 Unicode                   | utf8mb4_general_ci   |
4 |
| cp1251    | Windows Cyrillic                | cp1251_general_ci    |
1 |
| utf16     | UTF-16 Unicode                  | utf16_general_ci     |
4 |
| utf16le   | UTF-16LE Unicode                | utf16le_general_ci   |
4 |
| cp1256    | Windows Arabic                  | cp1256_general_ci    |
1 |
| cp1257    | Windows Baltic                  | cp1257_general_ci    |
1 |
| utf32     | UTF-32 Unicode                  | utf32_general_ci     |
4 |
| binary    | Binary pseudo charset           | binary               |
1 |
| geostd8   | GEOSTD8 Georgian                | geostd8_general_ci   |
1 |
| cp932     | SJIS for Windows Japanese       | cp932_japanese_ci    |
2 |
| eucjpms   | UJIS for Windows Japanese       | eucjpms_japanese_ci |
3 |
| gb18030   | China National Standard GB18030 | gb18030_chinese_ci   |
4 |
+----------+---------------------------------+---------------------+----
----+
41 rows in set (0.02 sec)
```

**查看当前连接的字符集：**

```
mysql> show variables like "%char%";
+--------------------------+-------------------------------------------
---------+
| Variable_name            | Value
|
+--------------------------+-------------------------------------------
---------+
| character_set_client     | latin1
|
| character_set_connection | latin1
|
```

```
| character_set_database   | utf8
|
| character_set_filesystem | binary
|
| character_set_results    | latin1
|
| character_set_server     | utf8
|
| character_set_system     | utf8
|
| character_sets_dir       | /data/tdsql_run/8812/percona-
5.7.17/share/charsets/ |
+--------------------------+---------------------------------------------
---------+
```

**设置当前连接相关的字符集：**

```
mysql> set names utf8;
Query OK, 0 rows affected (0.03 sec)

mysql> show variables like "%char%";
+--------------------------+---------------------------------------------
---------+
| Variable_name            | Value
|
+--------------------------+---------------------------------------------
---------+
| character_set_client     | utf8
|
| character_set_connection | utf8
|
| character_set_database   | utf8
|
| character_set_filesystem | binary
|
| character_set_results    | utf8
|
| character_set_server     | utf8
|
| character_set_system     | utf8
|
| character_sets_dir       | /data/tdsql_run/8811/percona-
5.7.17/share/charsets/ |
```

```
+--------------------------+---------------------------------------------
---------+
```

> **说明：**
> 分布式实例不支持设置全局参数，需要调用前台接口设置。

**支持通过设置 time_zone 变量修改时区相关的属性：**

```
mysql> show variables like '%time_zone%';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| system_time_zone | CST    |
| time_zone        | SYSTEM |
+------------------+--------+
2 rows in set (0.00 sec)

mysql> create database test;
Query OK, 0 rows affected (0.37 sec)

mysql> create table test.tt (ts timestamp, dt datetime,c int, PRIMARY
KEY(c)) shardkey=c;
Query OK, 0 rows affected (0.49 sec)

mysql>  insert into test.tt (ts,dt,c)values ('2017-10-01 12:12:12',
'2017-10-01 12:12:12',1);
Query OK, 1 row affected (0.09 sec)

mysql> select * from test.tt;
+---------------------+---------------------+------+
| ts                  | dt                  | c    |
+---------------------+---------------------+------+
| 2017-10-01 12:12:12 | 2017-10-01 12:12:12 |    1 |
+---------------------+---------------------+------+
1 row in set (0.04 sec)

mysql> set time_zone = '+12:00';
Query OK, 0 rows affected (0.00 sec)

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

```
+------------------+--------+
| system_time_zone | CST    |
| time_zone        | +12:00 |
+------------------+--------+
2 rows in set (0.00 sec)

mysql> select * from test.tt;
+---------------------+---------------------+------+
| ts                  | dt                  | c    |
+---------------------+---------------------+------+
| 2017-10-01 16:12:12 | 2017-10-01 12:12:12 |    1 |
+---------------------+---------------------+------+
1 row in set (0.06 sec)
```

## 数据类型

支持 MySQL 的所有数据类型，包括数字，字符，日期，空间类型，JSON。

### 数字

整型支持 INTEGER， INT，SMALLINT，TINYINT，MEDIUMINT，BIGINT。

| 类型 | 字节数 | 最小值（有符号/无符号） | 最大值（有符号/无符号） |
| --- | --- | --- | --- |
| TINYINT | 1 | -128/0 | 127/255 |
| SMALLINT | 2 | -32768/0 | 32767/65535 |
| MEDIUMINT | 3 | -8388608/0 | 8388607/16777215 |
| INT | 4 | -2147483648/0 | 2147483647/4294967295 |
| BIGINT | 8 | -9223372036854775808/0 | 9223372036854775807/18446744073709551615 |

浮点类型支持 FLOAT 和 DOUBLE，格式 FLOAT(M,D) 、REAL(M,D) 或 DOUBLE PRECISION(M,D)
定点类型支持 DECIMAL和NUMERIC，格式 DECIMAL(M,D)

### 字符

支持如下字符类型：

```
CHAR 和 VARCHAR Types
BINARY 和 VARBINARY Types
BLOB 和 TEXT Types
    TINYBLOB, TINYTEXT, MEDIUMBLOB,MEDIUMTEXT, LONGBLOB, LONGTEXT
ENUM Type
SET Type
```

## 日期

支持如下时间类型：

```
DATE, DATETIME,  TIMESTAMP Types
TIME Type
YEAR Type
```

## 空间

支持如下空间类型：

```
GEOMETRY
POINT
LINESTRING
POLYGON

MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION
```

## JSON

支持存储 JSON 格式的数据，使得对 JSON 处理更加有效，同时又能提早检查错误：

> **注意：**
> 对 JSON 类型的字段进行排序时，不支持混合类型排序。如不能将 string 类型和 int 类型做比较，同类型排序只支持数值类型，string 类型，其它类型排序不处理。对下表来说，不支持 `select * from t1 order by t1->"$.key2"` ，因为排序列中包含了数值和字符串类型。

```
mysql>  CREATE TABLE t1 (jdoc JSON,a int,PRIMARY KEY(a)) shardkey=a;
Query OK, 0 rows affected (0.30 sec)
```

```
mysql> INSERT INTO t1 (jdoc,a)VALUES('{"key1": "value1", "key2":
"value2"}',1);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 (jdoc,a)VALUES('{"key1": "value1", "key2": 2}',2);

mysql> INSERT INTO t1 (jdoc,a)VALUES('[1, 2,',5);
ERROR 3140 (22032): Invalid JSON text: "Invalid value." at position 6 in
value for column 't1.jdoc'.

mysql> select * from t1;
+-------------------------------------+---+
| jdoc                                | a |
+-------------------------------------+---+
| {"key1": "value1", "key2": "value2"} | 1 |
| {"key1": "value1", "key2": 2}        | 2 |
+-------------------------------------+---+
2 rows in set (0.00 sec)
```

# 函数支持

## Control Flow Functions

| 名称 | 描述 |
| --- | --- |
| CASE | Case operator |
| IF() | If/else construct |
| IFNULL() | Null if/else construct |
| NULLIF() | Return NULL if expr1 = expr2 |

## String Functions

| 名称 | 描述 |
| --- | --- |
| ASCII() | Return numeric value of left-most character |
| BIN() | Return a string containing binary representation of a number |
| BIT_LENGTH() | Return length of argument in bits |

| | |
|---|---|
| CHAR() | Return the character for each integer passed |
| CHAR_LENGTH() | Return number of characters in argument |
| CHARACTER_LENGTH() | Synonym for CHAR_LENGTH() |
| CONCAT() | Return concatenated string |
| CONCAT_WS() | Return concatenate with separator |
| ELT() | Return string at index number |
| EXPORT_SET() | Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string |
| FIELD() | Return the index (position) of the first argument in the subsequent arguments |
| FIND_IN_SET() | Return the index position of the first argument within the second argument |
| FORMAT() | Return a number formatted to specified number of decimal places |
| FROM_BASE64() | Decode to a base-64 string and return result |
| HEX() | Return a hexadecimal representation of a decimal or string value |
| INSERT() | Insert a substring at the specified position up to the specified number of characters |
| INSTR() | Return the index of the first occurrence of substring |
| LCASE() | Synonym for LOWER() |
| LEFT() | Return the leftmost number of characters as specified |
| LENGTH() | Return the length of a string in bytes |
| LIKE | Simple pattern matching |
| LOAD_FILE() | Load the named file |
| LOCATE() | Return the position of the first occurrence of substring |
| LOWER() | Return the argument in lowercase |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| LPAD() | Return the string argument, left-padded with the specified string |
| LTRIM() | Remove leading spaces |
| MAKE_SET() | Return a set of comma-separated strings that have the corresponding bit in bits set |
| MATCH | Perform full-text search |
| MID() | Return a substring starting from the specified position |
| NOT LIKE | Negation of simple pattern matching |
| NOT REGEXP | Negation of REGEXP |
| OCT() | Return a string containing octal representation of a number |
| OCTET_LENGTH() | Synonym for LENGTH() |
| ORD() | Return character code for leftmost character of the argument |
| POSITION() | Synonym for LOCATE() |
| QUOTE() | Escape the argument for use in an SQL statement |
| REGEXP | Pattern matching using regular expressions |
| REPEAT() | Repeat a string the specified number of times |
| REPLACE() | Replace occurrences of a specified string |
| REVERSE() | Reverse the characters in a string |
| RIGHT() | Return the specified rightmost number of characters |
| RLIKE | Synonym for REGEXP |
| RPAD() | Append string the specified number of times |
| RTRIM() | Remove trailing spaces |
| SOUNDEX() | Return a soundex string |
| SOUNDS LIKE | Compare sounds |
| SPACE() | Return a string of the specified number of spaces |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| STRCMP() | Compare two strings |
| SUBSTR() | Return the substring as specified |
| SUBSTRING() | Return the substring as specified |
| SUBSTRING_INDEX() | Return a substring from a string before the specified number of occurrences of the delimiter |
| TO_BASE64() | Return the argument converted to a base-64 string |
| TRIM() | Remove leading and trailing spaces |
| UCASE() | Synonym for UPPER() |
| UNHEX() | Return a string containing hex representation of a number |
| UPPER() | Convert to uppercase |
| WEIGHT_STRING() | Return the weight string for a string |

Numeric Functions and Operators

| 名称 | 描述 |
|---|---|
| ABS() | Return the absolute value |
| ACOS() | Return the arc cosine |
| ASIN() | Return the arc sine |
| ATAN() | Return the arc tangent |
| ATAN2(), ATAN() | Return the arc tangent of the two arguments |
| CEIL() | Return the smallest integer value not less than the argument |
| CEILING() | Return the smallest integer value not less than the argument |
| CONV() | Convert numbers between different number bases |
| COS() | Return the cosine |
| COT() | Return the cotangent |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| CRC32() | Compute a cyclic redundancy check value |
| DEGREES() | Convert radians to degrees |
| DIV | Integer division |
| / | Division operator |
| EXP() | Raise to the power of |
| FLOOR() | Return the largest integer value not greater than the argument |
| LN() | Return the natural logarithm of the argument |
| LOG() | Return the natural logarithm of the first argument |
| LOG10() | Return the base-10 logarithm of the argument |
| LOG2() | Return the base-2 logarithm of the argument |
| - | Minus operator |
| MOD() | Return the remainder |
| %, MOD | Modulo operator |
| PI() | Return the value of pi |
| + | Addition operator |
| POW() | Return the argument raised to the specified power |
| POWER() | Return the argument raised to the specified power |
| RADIANS() | Return argument converted to radians |
| RAND() | Return a random floating-point value |
| ROUND() | Round the argument |
| SIGN() | Return the sign of the argument |
| SIN() | Return the sine of the argument |
| SQRT() | Return the square root of the argument |
| TAN() | Return the tangent of the argument |
| * | Multiplication operator |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| TRUNCATE() | Truncate to specified number of decimal places |
| - | Change the sign of the argument |

Date and Time Functions

| 名称 | 描述 |
|---|---|
| ADDDATE() | Add time values (intervals) to a date value |
| ADDTIME() | Add time |
| CONVERT_TZ() | Convert from one time zone to another |
| CURDATE() | Return the current date |
| CURRENT_DATE(), CURRENT_DATE | Synonyms for CURDATE() |
| CURRENT_TIME(), CURRENT_TIME | Synonyms for CURTIME() |
| CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP | Synonyms for NOW() |
| CURTIME() | Return the current time |
| DATE() | Extract the date part of a date or datetime expression |
| DATE_ADD() | Add time values (intervals) to a date value |
| DATE_FORMAT() | Format date as specified |
| DATE_SUB() | Subtract a time value (interval) from a date |
| DATEDIFF() | Subtract two dates |
| DAY() | Synonym for DAYOFMONTH() |
| DAYNAME() | Return the name of the weekday |
| DAYOFMONTH() | Return the day of the month (0-31) |
| DAYOFWEEK() | Return the weekday index of the argument |
| DAYOFYEAR() | Return the day of the year (1-366) |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| EXTRACT() | Extract part of a date |
| FROM_DAYS() | Convert a day number to a date |
| FROM_UNIXTIME() | Format Unix timestamp as a date |
| GET_FORMAT() | Return a date format string |
| HOUR() | Extract the hour |
| LAST_DAY | Return the last day of the month for the argument |
| LOCALTIME(), LOCALTIME | Synonym for NOW() |
| LOCALTIMESTAMP, LOCALTIMESTAMP() | Synonym for NOW() |
| MAKEDATE() | Create a date from the year and day of year |
| MAKETIME() | Create time from hour, minute, second |
| MICROSECOND() | Return the microseconds from argument |
| MINUTE() | Return the minute from the argument |
| MONTH() | Return the month from the date passed |
| MONTHNAME() | Return the name of the month |
| NOW() | Return the current date and time |
| PERIOD_ADD() | Add a period to a year-month |
| PERIOD_DIFF() | Return the number of months between periods |
| QUARTER() | Return the quarter from a date argument |
| SEC_TO_TIME() | Converts seconds to 'HH:MM:SS' format |
| SECOND() | Return the second (0-59) |
| STR_TO_DATE() | Convert a string to a date |
| SUBDATE() | Synonym for DATE_SUB() when invoked with three arguments |
| SUBTIME() | Subtract times |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| SYSDATE() | Return the time at which the function executes |
| TIME() | Extract the time portion of the expression passed |
| TIME_FORMAT() | Format as time |
| TIME_TO_SEC() | Return the argument converted to seconds |
| TIMEDIFF() | Subtract time |
| TIMESTAMP() | With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments |
| TIMESTAMPADD() | Add an interval to a datetime expression |
| TIMESTAMPDIFF() | Subtract an interval from a datetime expression |
| TO_DAYS() | Return the date argument converted to days |
| TO_SECONDS() | Return the date or datetime argument converted to seconds since Year 0 |
| UNIX_TIMESTAMP() | Return a Unix timestamp |
| UTC_DATE() | Return the current UTC date |
| UTC_TIME() | Return the current UTC time |
| UTC_TIMESTAMP() | Return the current UTC date and time |
| WEEK() | Return the week number |
| WEEKDAY() | Return the weekday index |
| WEEKOFYEAR() | Return the calendar week of the date (1–53) |
| YEAR() | Return the year |
| YEARWEEK() | Return the year and week |

Aggregate (GROUP BY) Functions

| 名称 | 描述 |
|---|---|
| AVG() | Return the average value of the argument |
| COUNT() | Return a count of the number of rows returned |

| | |
|---|---|
| MAX() | Return the maximum value |
| MIN() | Return the minimum value |
| SUM() | Return the sum |

### Bit Functions and Operators

| 名称 | 描述 |
|---|---|
| BIT_COUNT() | Return the number of bits that are set |
| & | Bitwise AND |
| ~ | Bitwise inversion |
| \| | Bitwise OR |
| ^ | Bitwise XOR |
| << | Left shift |
| >> | Right shift |

### Cast Functions and Operators

| 名称 | 描述 |
|---|---|
| BINARY | Cast a string to a binary string |
| CAST() | Cast a value as a certain type |
| CONVERT() | Cast a value as a certain type |

版权所有：腾讯云计算（北京）有限责任公司

# 连接数据库

最近更新时间：2022-11-24 11:02:29

## 客户端连接

TDSQL MySQL版 提供和 MySQL 兼容的连接方式，用户可通过 IP 地址、端口号以及用户名、密码连接 TDSQL MySQL版：

```
mysql -hxxx.xxx.xxx.xxx -Pxxxx -uxxx -pxxx -c
```

> **注意**
> TDSQL MySQL版 不支持4.0以下的版本以及压缩协议，建议在使用客户端的时候增加 `-c` 选项，以便于使用某些高级功能。

## PHP MySQLli 连接

PHP 需要开启 MySQLli 扩展连接数据库，具体 demo 如下：

```
header("Content-Type:text/html;charset=utf-8");
$host="10.10.10.10";  //实例的 proxy_host_ip
$user="test";  //实例用户
$pwd="test";  //实例用户密码
$db="aaa";  //数据库名
$port="15002";  //proxy_host 端口号
$sqltool=new MySQLli($host,$user,$pwd,$db,$port);
//其他必要代码
$sqltool->close();
echo "ok"."\n";
```

## JDBC 连接

您也可以使用 JDBC 连接 TDSQL MySQL版，例如：

```
private final String USERNAME = "test";
private final String PASSWORD = "123456";
private final String DRIVER = "com.mysql.jdbc.Driver";
private final String URL = "jdbc:mysql://10.10.10.10:3306?
userunicode=true&characterEncoding=utf8mb4";
private Connection connection;
private PreparedStatement pstmt;
```

```
private ResultSet resultSet;
```

## 其他连接方式

您也可以选择其他兼容 MySQL 的连接方式，例如 navicat、odbc 等。

# JOIN 和子查询

最近更新时间：2025-01-26 00:21:42

TDSQL 支持对 SELECT 语句、多表 DELETE 和 UPDATE 操作的 join。

## 分表间 join 示例

如果分表之间带有分表键相等的条件，则相当于单机 join。

```
--构建两张测试表：
DROP TABLE IF EXISTS `test_join_shard_table1`;
CREATE TABLE `test_join_shard_table1` (
  `id` int(10) NOT NULL,
  `b` varchar(10)  NOT NULL DEFAULT '',
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin shardkey=id;
INSERT INTO test_join_shard_table1 (id, b, c) VALUES
    (1,"test1",1), (2,"test2",2), (3,"test3",3),
    (4,"test4",4), (5,"test5",5), (6,"test6",6),
(7,"test7",7), (8,"test8",8), (9,"testX",11);

DROP TABLE IF EXISTS `test_join_shard_table2`;
CREATE TABLE `test_join_shard_table2` (
  `id` int(10) NOT NULL,
  `d` datetime,
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT  CHARSET=utf8  COLLATE=utf8_bin shardkey=id;
INSERT INTO test_join_shard_table2 (id, d, c) VALUES
    (1,NOW(),1), (2,NOW(),2), (3,NOW(),3),
    (4,NOW(),4), (5,NOW(),5), (6,NOW(),6),
(7,NOW(),7), (8,NOW(),8), (9,NOW(),10);

--检查分布式测试表的数据分布情况：
/*sets:allsets*/  select * from test_join_shard_table1;
/*sets:allsets*/  select * from test_join_shard_table2;

--执行带 INNER JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
INNER JOIN test_join_shard_table2 test2
```

```
ON test1.c=test2.c
ORDER BY NAME;

--执行带 LEFT JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
LEFT JOIN test_join_shard_table2 test2
ON test1.c=test2.c
ORDER BY NAME;

--执行带 RIGHT JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
RIGHT JOIN test_join_shard_table2 test2
ON test1.c=test2.c
ORDER BY NAME;

--执行带 CROSS JOIN（笛卡尔积）的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
CROSS JOIN test_join_shard_table2 test2
ORDER BY NAME;
```

## 分表和广播表 join 示例

跨分片的分表与广播表，效果相当于单机 join。

```
--构建两张测试表：
DROP TABLE IF EXISTS `test_join_shard_table1`;
CREATE TABLE `test_join_shard_table1` (
  `id` int(10) NOT NULL,
  `b` varchar(10)  NOT NULL DEFAULT '',
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin shardkey=id;
INSERT INTO test_join_shard_table1 (id, b, c) VALUES
    (1,"test1",1), (2,"test2",2), (3,"test3",3),
    (4,"test4",4), (5,"test5",5), (6,"test6",6),
(7,"test7",7), (8,"test8",8), (9,"testX",11);

DROP TABLE IF EXISTS `test_join_group_table2`;
CREATE TABLE `test_join_group_table2` (
  `id` int(10) NOT NULL,
```

```
  `d` datetime,
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT  CHARSET=utf8  COLLATE=utf8_bin
shardkey=noshardkey_allset;
INSERT INTO test_join_group_table2 (id, d, c) VALUES
    (1,NOW(),1), (2,NOW(),2), (3,NOW(),3),
    (4,NOW(),4), (5,NOW(),5), (6,NOW(),6),
(7,NOW(),7), (8,NOW(),8), (9,NOW(),10);

--检查分布式测试表的数据分布情况:
/*sets:allsets*/  select * from test_join_shard_table1;
/*sets:allsets*/  select * from test_join_group_table2;

--执行带 INNER JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
INNER JOIN test_join_group_table2 test2
ON test1.c=test2.c
ORDER BY NAME;

--执行带 LEFT JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
LEFT JOIN test_join_group_table2 test2
ON test1.c=test2.c
ORDER BY NAME;

--执行带 RIGHT JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
RIGHT JOIN test_join_group_table2 test2
ON test1.c=test2.c
ORDER BY NAME;

--执行带 CROSS JOIN（笛卡尔积）的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
CROSS JOIN test_join_group_table2 test2
ORDER BY NAME;
```

## 分表和单表 join 示例

```
--构建两张测试表:
DROP TABLE IF EXISTS `test_join_shard_table1`;
CREATE TABLE `test_join_shard_table1` (
  `id` int(10) NOT NULL,
  `b` varchar(10)  NOT NULL DEFAULT '',
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin shardkey=id;
INSERT INTO test_join_shard_table1 (id, b, c) VALUES
    (1,"test1",1), (2,"test2",2), (3,"test3",3),
    (4,"test4",4), (5,"test5",5), (6,"test6",6),
(7,"test7",7), (8,"test8",8), (9,"testX",11);

DROP TABLE IF EXISTS `test_join_noshard_table2`;
CREATE TABLE `test_join_noshard_table2` (
  `id` int(10) NOT NULL,
  `d` datetime,
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
INSERT INTO test_join_noshard_table2 (id, d, c) VALUES
    (1,NOW(),1), (2,NOW(),2), (3,NOW(),3),
    (4,NOW(),4), (5,NOW(),5), (6,NOW(),6),
    (7,NOW(),7), (8,NOW(),8), (9,NOW(),10);

--检查分布式测试表的数据分布情况:
/*sets:allsets*/ select * from test_join_shard_table1;
--检查单片表的数据:
select * from test_join_noshard_table2;

--执行带 INNER JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
INNER JOIN test_join_noshard_table2 test2
ON test1.c=test2.c
ORDER BY NAME;

--执行带 LEFT JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
LEFT JOIN test_join_noshard_table2 test2
ON test1.c=test2.c
ORDER BY NAME;
```

```
--执行带 RIGHT JOIN 的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
RIGHT JOIN test_join_noshard_table2 test2
ON test1.c=test2.c
ORDER BY NAME;

--执行带 CROSS JOIN（笛卡尔积）的 SELECT 查询语句
SELECT test1.id, test1.b AS NAME, test2.d AS TIME
FROM test_join_shard_table1 test1
CROSS JOIN test_join_noshard_table2 test2
ORDER BY NAME;
```

## 跨分片 update/delete join 示例

```
--创建测试表：
DROP TABLE IF EXISTS `test_join_shard_table1`;
CREATE TABLE `test_join_shard_table1` (
  `id` int(10) NOT NULL,
  `b` varchar(10)  NOT NULL DEFAULT '',
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin shardkey=id;
INSERT INTO test_join_shard_table1 (id, b, c) VALUES
    (1,"test1",1), (2,"test2",2), (3,"test3",3),
    (4,"test4",4), (5,"test5",5), (6,"test6",6),
(7,"test7",7), (8,"test8",8), (9,"testX",11);

DROP TABLE IF EXISTS `test_join_shard_table2`;
CREATE TABLE `test_join_shard_table2` (
  `id` int(10) NOT NULL,
  `d` datetime,
  `c` int(10) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin shardkey=id;
INSERT INTO test_join_shard_table2 (id, d, c) VALUES
    (1,NOW(),1), (2,NOW(),2), (3,NOW(),3),
    (4,NOW(),4), (5,NOW(),5), (6,NOW(),6),
(7,NOW(),7), (8,NOW(),8), (9,NOW(),10);

--检测分布式测试表的数据分布情况
/*sets:allsets*/  select * from test_join_shard_table1;
```

```
/*sets:allsets*/  select * from test_join_shard_table2;

--UPDATE…JOIN…ON…SET 语句，单字段:
UPDATE test_join_shard_table1 test1
INNER JOIN test_join_shard_table2 test2
ON test1.c=test2.c SET test1.b="TEXTXXXXX"
WHERE test1.id>7;
SELECT * FROM test_join_shard_table1;

--UPDATE…JOIN…ON…SET 语句，同一表多字段
UPDATE test_join_shard_table1 test1
INNER JOIN test_join_shard_table2 test2
ON test1.c=test2.c
SET test1.b="TEXTSSSS", test1.c=88
WHERE test1.id>7;
SELECT * FROM test_join_shard_table1;

--DELETE…FROM…JOIN…ON 语句
DELETE test1 FROM test_join_shard_table1 test1
INNER JOIN test_join_shard_table2 test2
ON test1.c=test2.c
WHERE test1.id>7;
SELECT * FROM test_join_shard_table1;
```

## union 语法

**UNION 将来自多个 SELECT 语句的结果组合到一个结果集中。**
**语法如下:**

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

**示例:**

```
DROP TABLE IF EXISTS t1;
create table t1 (a int primary key, b int) shardkey=a;
DROP TABLE IF EXISTS t2;
create table t2 (a int primary key, b int) shardkey=a;
select * from t1 where t1.a in (select a from t2) union select * from t2
where t2.a>22;
```

各种表的组合场景：

```
分表：
DROP TABLE IF EXISTS s1;
create table s1 (a int primary key, b int) shardkey=a;
DROP TABLE IF EXISTS s2;
create table s2 (a int primary key, b int) shardkey=a;
单表：
DROP TABLE IF EXISTS ns1;
create table ns1 (a int primary key, b int);
DROP TABLE IF EXISTS ns2;
create table ns2 (a int primary key, b int);
广播表：
DROP TABLE IF EXISTS g1;
create table g1 (a int primary key, b int) shardkey=noshardkey_allset;
DROP TABLE IF EXISTS g2;
create table g2 (a int primary key, b int) shardkey=noshardkey_allset;
二级分区表：
DROP TABLE IF EXISTS p1;
create table p1 (a int, b int, PRIMARY KEY(a)) shardkey=a PARTITION BY
range (b) (PARTITION p0 values less than (100), PARTITION p1 values less
than (200));
DROP TABLE IF EXISTS p2;
create table p2 (a int, b int, PRIMARY KEY(a)) shardkey=a PARTITION BY
range (b) (PARTITION p0 values less than (100), PARTITION p1 values less
than (200));

各种类型表之间的union
select * from s1 union select * from s2;
select * from ns1 union select * from ns2;
select * from g1 union select * from g2;
select * from s1 union select * from ns1;
select * from p1 union select * from p2;
select * from s1 where not exists (select * from s2 where s2.a=s1.a
order by s2.a) or b<10 union select * from s2 where s2.a>22;
select a, sum(b) from s1 group by a union select * from s2;
select a, sum(b) from s1 union select * from s2;
select distinct(a) from s1 union select a from s2;
select distinct(a), b from s1 union select a,b from s2;
```

## 子查询带有 shardkey 的 derived table

```
mysql> select a from (select * from test1 where a=1) as t;
+---+
| a |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

**说明：**
子查询时不指定 shardkey，也可查询结果。

## 复杂 SQL

对于不能满足推荐方式的 SQL，由于需要做跨节点的数据交互，所以性能会差一些。
包括：

- 包含子查询的查询。
- 多表的 join 查询，且参与查询的各表的分区字段（shardkey）不相等，或者同时涉及不同类型的表，例如单表和分表。

对于这类复杂查询，通过条件下推，将真正参与查询的数据从后端数据库中抽取出来，存放在本地的临时表中，然后对临时表中的数据进行计算。
因此用户需要明确指定参与查询的表的条件，避免因为抽取大量数据而性能受损。

```
mysql> create table test1 ( a int key, b int, c char(20) ) shardkey=a;
Query OK, 0 rows affected (1.56 sec)

mysql> create table test2 ( a int key, d int, e char(20) ) shardkey=a;
Query OK, 0 rows affected (1.46 sec)

mysql> insert into test1 (a,b,c) values(1,2,"record1"),(2,3,"record2");
Query OK, 2 rows affected (0.02 sec)

mysql> insert into test2 (a,d,e) values(1,3,"test2_record1"),
(2,3,"test2_record2");
Query OK, 2 rows affected (0.02 sec)

mysql> select * from test1 join test2 on test1.b=test2.d;
+---+------+---------+---+------+---------------+
| a | b    | c       | a | d    | e             |
+---+------+---------+---+------+---------------+
| 2 |    3 | record2 | 1 |    3 | test2_record1 |
```

```
| 2 |    3 | record2 | 2 |    3 | test2_record2 |
+---+------+---------+---+------+---------------+
2 rows in set (0.00 sec)

MySQL> select * from test1 where exists (select * from test2 where
test2.a=test1.b);
+---+------+---------+
| a | b    | c       |
+---+------+---------+
| 1 |    2 | record1 |
+---+------+---------+
1 row in set (0.00 sec)
```

**分布式实例还支持丰富的复杂 update/delete/insert 操作。**
**需要注意的是，这类查询是在与之对应的 select 基础上实现的，因此也需要将数据加载至网关临时表，建议用户尽量在查询中指定明确的查询条件，避免大量数据的加载带来性能损耗。**
**另外，网关在加载数据时默认不会对加载的数据进行上锁，这与官方的 MySQL 行为存在略微的差异；如需加锁可以通过修改 proxy 配置来实现。**

```
MySQL [th]> update test1 set test1.c="record" where exists(select 1 from
test2 where test1.b=test2.d);
Query OK, 1 row affected (0.00 sec)

MySQL [th]> update test1, test2 set test1.b=2 where test1.b=test2.d;
Query OK, 1 row affected (0.00 sec)

MySQL [th]> insert into test1 select cast(rand()*1024 as unsigned), d, e
from test2;
Query OK, 2 rows affected (0.00 sec)

MySQL [th]> delete from test1 where b in (select b from test2);
Query OK, 6 rows affected (0.00 sec)

MySQL [th]> delete from test2.* using test1 right join test2 on
test1.a=test2.a where test1.a is null;
Query OK, 2 rows affected (0.00 sec)
```

版权所有：腾讯云计算（北京）有限责任公司

# Sequence

最近更新时间：2024-06-07 11:11:21

关键字 Sequence 语法和 MariaDB/Oracle 兼容，但是保证分布式全局递增且唯一，具体使用如下：

- TDSQL_INCREMENT BY 设置序列步长，正数表示升序序列，负数表示降序序列。设置值必须在序列最大值与最小值之间，缺省则默认1。
- START WITH 设置序列起始值，缺省则默认序列最小值。
- TDSQL_MAXVALUE 和 TDSQL_NOMAXVALUE 设置序列最大值，TDSQL_MAXVALUE 缺省或者设置TDSQL_NOMAXVALUE 时，递增序列最大值为9223372036854775807，递减序列最大值为-1。
- TDSQL_MINVALUE 和 TDSQL_NOMINVALUE 设置序列最小 TDSQL_MINVALUE 缺省或者设置 TDSQL_NOMINVALUE 时，递增序列最小值为1，递减序列最小值为-9223372036854775808。
- TDSQL_CYCLE 和 TDSQL_NOCYCLE 设置序列是否循环，缺省默认不循环。
- TDSQL_CACHE 和 TDSQL_NOCACHE 设置缓存序列数量，缺省默认缓存20。
- TDSQL_ORDER 和 TDSQL_NOORDER。设置序列对于并发请求保证完全有序。TDSQL 目前仅支持无序。

**说明：**

- 在 TDSQL MySQL 版分布式数据库当中使用 Sequence 时，须在该关键字前面加 `tdsql_` 前缀，且要求 SQL 引擎层版本号最低为1.19.5-M-V2.0R745D005；您可在控制台实例详情中的配置信息一栏查看 SQL 引擎层版本号，若 SQL 引擎层版本号较老可以 提交工单 进行升级。
- 目前 Sequence 为保证分布式全局数值唯一，导致性能较差，主要适用于并发不高的场景。

创建序列需要 CREATE SEQUENCE 系统权限。序列的创建语法如下：

```
    CREATE TDSQL_SEQUENCE 序列名
    [START WITH n]
    [{TDSQL_MINVALUE | TDSQL_MAXVALUE} n | TDSQL_NOMINVALUE |
TDSQL_NOMAXVALUE]
    [TDSQL_INCREMENT BY n]
    [{TDSQL_CYCLE|TDSQL_NOCYCLE}]
    [{TDSQL_CACHE|TDSQL_NOCACHE} n]
```

## 创建 Sequence

```
create tdsql_sequence test.s1 start with 12 tdsql_minvalue 10 maxvalue
50000 tdsql_increment by 5 tdsql_nocycle
```

```
create tdsql_sequence test.s2 start with 12 tdsql_minvalue 10 maxvalue
50000 tdsql_increment by 1 tdsql_cycle
```

- 以上 SQL 语句包含开始值、最小值、最大值、步长、缓存大小及是否回绕6个参数。
- 参数默认值，开始值（1）、最小值（1）、最大值（LONGLONG_MAX-1）、步长（1）、是否回绕（0）。

## 删除 Sequence

```
drop tdsql_sequence test.s1
```

## 查询 Sequence

```
show create tdsql_sequence test.s1
```

## 使用 Sequence

### 使用 Sequence 获取下一个数值

```
select tdsql_nextval(test.s2)
select next value for test.s2
```

```
mysql> select tdsql_nextval(test.s1);
+----+
| 12 |
+----+
| 12 |
+----+
1 row in set (0.18 sec)

mysql> select tdsql_nextval(test.s2);
+----+
| 12 |
+----+
| 12 |
+----+
1 row in set (0.13 sec)

mysql> select tdsql_nextval(test.s1);
+----+
| 17 |
```

```
+----+
| 17 |
+----+
1 row in set (0.01 sec)

mysql> select tdsql_nextval(test.s2);
+----+
| 13 |
+----+
| 13 |
+----+
1 row in set (0.00 sec)

mysql> select next value for test.s1;
+----+
| 22 |
+----+
| 22 |
+----+
1 row in set (0.01 sec)
```

## nextval 可以用在 insert 等地方

```
mysql> create database test;
Query OK, 0 rows affected (0.29 sec)

mysql> create table test.t1 (a int, b int, PRIMARY KEY (b));
Query OK, 0 rows affected (0.15 sec)

mysql> insert into test.t1(a,b) values(11,2);
Query OK, 1 row affected (0.03 sec)

mysql> select * from test.t1;
+----+------+
| a  | b    |
+----+------+
| 11 |    2 |
+----+------+
1 row in set (0.00 sec)

mysql> insert into test.t1(a,b) values(tdsql_nextval(test.s2),3);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from test.t1;
+----+------+
| a  | b    |
+----+------+
| 11 |    2 |
| 14 |    3 |
+----+------+
2 rows in set (0.00 sec)
```

如需获取上一次的值以连接相关数据：如果之前没有用 nextval 命令获取过数据，数值将返回为0。

```
select tdsql_lastval(test.s1)
select tdsql_previous value for test.s1;
```

```
mysql> select tdsql_lastval(test.s1);
+----+
| 22 |
+----+
| 22 |
+----+
1 row in set (0.00 sec)

mysql> select tdsql_previous value for test.s1;
+----+
| 22 |
+----+
| 22 |
+----+
1 row in set (0.00 sec)
```

设置下一个序列数值，只能比当前数值大，否则将返回数值为0。

```
select tdsql_setval(test.s2,1000,bool use)  //  use 默认为1，表示1000这个值
用过了，下一次不包含1000，如果为0，则下一个从1000开始。
```

设置下一个序列数值时，如果比当前数值小，则设置不生效。

```
mysql> select tdsql_nextval(test.s2);
+----+
| 15 |
+----+
```

```
| 15 |
+----+
1 row in set (0.01 sec)

mysql> select tdsql_setval(test.s2,10);
+---+
| 0 |
+---+
| 0 |
+---+
1 row in set (0.03 sec)

mysql> select tdsql_nextval(test.s2);
+----+
| 16 |
+----+
| 16 |
+----+
```

**如果比当前数值大，成功返回当前设置的值。**

```
mysql> select tdsql_setval(test.s2,20);
+----+
| 20 |
+----+
| 20 |
+----+
1 row in set (0.02 sec)
mysql> select tdsql_nextval(test.s2);
+----+
| 21 |
+----+
| 21 |
+----+
1 row in set (0.01 sec)
```

**强制设置下一个序列数值，允许设置比当前数值小的值。**

```
select tdsql_resetval(test.s2,1000)
```

**如果强制设置成功，返回当前设置的值，下一个序列数值从该值开始。**

```
mysql> select tdsql_resetval(test.s2,14);
+----+
| 14 |
+----+
| 14 |
+----+
1 row in set (0.00 sec)

mysql> select tdsql_nextval(test.s2);
+----+
| 14 |
+----+
| 14 |
+----+
1 row in set (0.01 sec)
```

**需要注意，Sequence 的部分关键字以 `TDSQL_` 前缀开始：**

> **说明：**
> 若已打开 oldstyle 配置项，proxy 将兼容标准 Sequence 关键字，即关键字前可不添加 `TDSQL_` 前缀。

```
TDSQL_CYCLE
TDSQL_INCREMENT
TDSQL_LASTVAL
TDSQL_MINVALUE
TDSQL_NEXTVAL
TDSQL_NOCACHE
TDSQL_NOCYCLE
TDSQL_NOMAXVALUE
TDSQL_NOMINVALUE
TDSQL_PREVIOUS
TDSQL_RESTART
TDSQL_REUSE
TDSQL_SEQUENCE
TDSQL_SETVAL
```

版权所有：腾讯云计算（北京）有限责任公司

# 常用 DML

最近更新时间：2023-02-21 11:45:07

## select

建议带上 shardkey 字段，proxy 根据该字段的 hash 值直接将 SQL 请求路由至对应的数据库实例进行处理；否则就需要发送给集群中所有的数据库实例执行，然后 proxy 根据数据库返回的结果集进行聚合，影响执行效率：

```
mysql> select * from test1 where a=2;
+------+------+---------+
| a    | b    | c       |
+------+------+---------+
|    2 |    3 | record2 |
|    2 |    4 | record3 |
+------+------+---------+
2 rows in set (0.00 sec)
```

## insert/replace

字段必须包含 shardkey，否则会拒绝执行该 SQL，因为 proxy 不知道将该 SQL 发往哪个后端数据库：

```
mysql> insert into test1 (b,c) values(4,"record3");
ERROR 810 (HY000): Proxy ERROR:sql is too complex,need to send to only
noshard table.
    Shard table insert must has field spec

mysql> insert into test1 (a,c) values(4,"record3");
Query OK, 1 row affected (0.01 sec)
```

## delete/update

使用分表时，为安全考虑，执行该类 SQL 时，必须带有 where 条件，否则拒绝执行该 SQL 命令：

```
mysql> delete from test1;
ERROR 810 (HY000): Proxy ERROR:sql is too complex,need to send to only
noshard table.
    Shard table delete/update must have a where clause

mysql> delete from test1 where a=1;
Query OK, 1 row affected (0.01 sec)
```

# 读写分离

最近更新时间：2023-11-23 10:22:21

TDSQL MySQL版 实例支持下列几种模式的读写分离：

- 通过增加 slave 注释标记，即在 SQL 中添加 `/*slave*/` 这样的标记，并且 mysql 后面增加 -c 参数来解析注释 `mysql -c -e "/*slave*/sql"`，该 SQL 会发送给备机。

> **说明：**
> 支持 `/*slave:slaveonly*/`、`/*slave:20*/`、`/*slave:slaveonly,20*/` 这几种形式，数值表示 slave 应该满足的延迟，slaveonly 表示在没有符合条件的 slave 时，不会将查询发送给主节点。

```
//主机读//
select * from emp order by sal, deptno desc;
//从机读//
/*slave*/ select * from emp order by sal, deptno desc;
```

- 由只读账号发送的请求会根据配置的属性发给备机。

# 分布式事务

最近更新时间：2022-08-22 11:30:44

由于事务操作的数据通常跨多个物理节点，在分布式数据库中，类似方案即称为分布式事务。
TDSQL MySQL版 支持普通分布式事务协议和 XA 分布式事务协议。TDSQL MySQL版（内核5.7或以上版本）默认支持分布式事务，且对客户端透明，像使用单机事务一样方便。
TDSQL MySQL版 分布式事务采用两阶段提交算法（2PC）保证事务的原子性（Atomicity）和一致性（Consistency），隔离级别配置为 Read committed、Repeatable read 或 Serializable。

## 普通分布式事务

```
begin;   # 开启事务
...        # 跨 set 的增删改查等非 DDL 操作
commit;    # 提交事务
```

## XA 分布式事务

XA 分布式事务是指跨实例的事务：

```
xa begin '';          # 开启 XA 事务，事务标识由系统内部生成，因此传入空字符串
...                 # 跨 set 的增删改查等非 DDL 操作
select gtid();        # 获取当前 XA 事务的标识，下面假定为'xid'
xa prepare 'xid';     # 准备事务
xa commit/rollback 'xid'; # 提交或回滚事务
```

## 新增事务接口

- `select gtid()` ：获取当前分布式事务的全局唯一标识。如果为空，则该事务不是分布式事务。
- 普通分布式事务标识的格式为：‘网关id’-‘proxy随机值’-‘序列号’-‘时间戳’-‘分区号’，例如 c46535fe-b6-dd-595db6b8-25。
- XA 分布式事务标识的格式为：‘ex’-‘网关id’-‘proxy随机值’-‘序列号’-‘时间戳’-‘分区号’，例如 ex-c46535fe-b6-dd-595db6b8-25。
- `select gtid_state(“当前分布式事务的全局唯一标识”)` ：在事务提交异常之后（默认3秒后）用来获取事务的状态。可能的结果有：
- COMMIT：标识该事务已经或者最终会被提交。
- ABORT：标识该事务最终会被回滚。
- 空：由于事务的状态会在一个小时之后清除，因此有以下两种可能：
  - 一个小时之后查询，标识事务状态已经清除，

  - 一个小时以内查询，标识事务最终会被回滚。
- `xa boost ‘当前分布式事务的全局唯一标识’`：普通事务提交（commit）发送异常之后，事务在一段时间内（默认30秒）由后台组件自动提交或者回滚掉。如果用户不愿意等待这么长的时间，可以反复调用该接口，促使系统及时地提交或回滚掉事务。该接口会返回事务的状态，即提交或者回滚。
- `xa lockwait`：显示当前分布式事务的等待关系。用户可以通过 dot 工具，将其转化为图片。
- `xa show`：显示当前 proxy 上处于活跃状态的事务。

# 建表

最近更新时间：2024-11-29 11:13:46

## 建分表

分表创建时必须在最后面指定分表键（shardkey）的值，该值为表中的一个字段名字，会用于后续 SQL 的路由选择：

```
mysql> create table test1 ( a int, b int, c char(20),primary key
(a,b),unique key u_1(a,c) ) shardkey=a;
Query OK, 0 rows affected (0.07 sec)
```

在分布式实例中，shardkey 对应后端数据库的分区字段，因此每一个唯一索引和主键都必须要包含这个 shardkey，否则无法创建表。
场景：存在多个唯一索引时报错。

```
mysql> create table test1 ( a int, b int, c char(20),primary key
(a,b),unique key u_1(a,c),unique key u_2(b,c) ) shardkey=a;
```

此时有一个唯一索引 `u_2` 不包含 shardkey，无法创建表，会报如下错误：

```
ERROR 1105 (HY000): A UNIQUE INDEX must include all columns in the
table's partitioning function
```

因为主键索引或者 unique key 索引意味着需要全局唯一，而要实现全局唯一索引，则必须包含 shardkey 字段。
除上面的限制外，shardkey 字段还有如下要求：

- shardkey 字段的类型必须是 int、bigint、smallint、char、varchar、decimal、float、varbinary。
- shardkey 字段类型为 char、varchar 时需定义字段长度。
- shardkey 字段的值不能有中文，proxy 不会转换字符集，因此不同字符集可能会路由到不同的分区。
- 不能 update shardkey 字段的值。
- shardkey=a 放在 SQL 的最后面。
- 访问数据尽量都带上 shardkey 字段，非强制要求，但是不带 shardkey 的 SQL 会路由到所有节点，消耗较多资源。

另外，创建表的时候，可以使用自增序列作为某一列的默认值，并把这一列设置为主键。

```
CREATE TABLE test2(column1 INT,column2 INT,column3 INT
AUTO_INCREMENT,PRIMARY KEY(column3)) AUTO_INCREMENT=5 shardkey=column3;
```

版权所有：腾讯云计算（北京）有限责任公司

插入数据时，设置为自增序列的列插入Null 值。

```
INSERT INTO test2(column1,column2,column3) VALUES(0,0,NULL);
```

**一级分区分表键（shardkey）规则：**

在 TDSQL 中，分表也叫一级分区表。有 hash、range、list 三种规则。一级 hash 分区使用 shardkey 关键字指定拆分键。range 和 list 分区使用tdsql_distributed by 语法指定拆分键。

> **注意：**
> 内核5.7版本不支持 TDSQL_DISTRIBUTED BY range|list 的语法。

**创建一级 HASH 分区表：**

支持类型：

- DATE，DATETIME
- TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT
- CHAR，VARCHAR

> **注意：**
> - Shardkey 字段必须是主键以及所有唯一索引的一部分。
> - Shardkey 字段的值不能为中文，因为 Proxy 不会转换字符集，所以不同字符集可能会路由到不同的分区。
> - Shardkey=a 需放在 SQL 语句的最后。

**使用场景**

对于高并发业务场景的事务表，读写事务频繁，适合单表数据设计容量规模达到千万级别，且数据在可预见期内快速增长的场景。建议单分片容量不超过2TB，单分片对应的分表不超过2千万行，容量10GB-100GB之间。

如业务订单表，数据设计容量规模预计在6千万，需要进行一级分区做水平切分，实例分布4个 set，则分区表平均量级大概在1500万行；如果后续增长规模较快超过初期容量设计，例如达到1亿，则平均单 set 分表可达2500万行，此时就当考虑做节点组 set 水平扩容，以保证单 set 分表数据量在合理范围。

**基础语法**

```
CREATE TABLE [IF NOT EXISTS] tbl_name
[(create_definition)]
[local_table_options]
shardkey=column_name

create_definition: {
col_name column_definition
| {INDEX | KEY} [index_name] [index_type] (key_part,...)
```

```
[index_option] ...
| [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] ...
}

column_definition: {
data_type [NOT NULL | NULL] [DEFAULT]
[AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT 'string']
[COLLATE collation_name]
[COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
[ENGINE_ATTRIBUTE [=] 'string']
| data_type
[UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT 'string']
}

key_part: {col_name [(length)]} [ASC | DESC]

index_type:
USING {BTREE}

index_option: {
index_type | COMMENT 'string'
}
[local_table_options]
Local_table_option: {AUTO_INCREMENT [=] value
| [DEFAULT] CHARACTER SET [=] charset_name
| [DEFAULT] COLLATE [=] collation_name
| COMMENT [=] 'string'
| ENGINE [=] engine_name
| ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT |
COMPACT}
| STATS_AUTO_RECALC [=] {DEFAULT | 0 | 1}
| STATS_PERSISTENT [=] {DEFAULT | 0 | 1}
| STATS_SAMPLE_PAGES [=] value)
}
```

### 示例

```
DROP TABLE IF EXISTS employees_hash;
CREATE TABLE `employees_hash` (
`id`int NOT NULL,
`city` varchar(10),
`fired` DATE NOT NULL DEFAULT '1970.01.01',
PRIMARY KEY(id)
) shardkey=id;

DROP TABLE IF EXISTS s2;
CREATE TABLE s2 (
hiredate date,
data1 int,
data2 char(2),
primary key(hiredate, data1)
) shardkey=hiredate;

DROP TABLE IF EXISTS s3;
CREATE TABLE s3 (
name char(4),
data1 int,
data2 int,
primary key(name, data2)
) shardkey=name;
```

**创建一级 Range|List 分区表：**

**使用场景：**

- 一级分区除了根据 shardkey 进行一致性 hash 分区，也可以根据实际场景进行 Range 或 List 进行一级分区。
- Range 分区是指根据一定字段取值范围进行分区指定，适合例如日志流水表，可以根据一定日期范围进行分区，方便后续数据归档。
- List 分区则是根据字段的散列值进行匹配分区，例如人口统计表按省份进行 List 分区，可以有针对性地对地区属性进行分区管理。

**基础语法**

```
CREATE TABLE [IF NOT EXISTS] tbl_name
[(create_definition)]
[local_table_options]
TDSQL_DISTRIBUTED BY range|list (column_name) [partition_options]
```

```
create_definition: {
col_name column_definition
| {INDEX | KEY} [index_name] [index_type] (key_part,...)
[index_option] ...
| [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] ...
}

column_definition: {
data_type [NOT NULL | NULL] [DEFAULT]
[AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT 'string']
[COLLATE collation_name]
[COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
[ENGINE_ATTRIBUTE [=] 'string']
| data_type
[UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT 'string']
}

key_part: {col_name [(length)]} [ASC | DESC]

index_type:
USING {BTREE}

index_option: {
index_type | COMMENT 'string'
}
[local_table_options]
Local_table_option: {AUTO_INCREMENT [=] value
| [DEFAULT] CHARACTER SET [=] charset_name
| [DEFAULT] COLLATE [=] collation_name
| COMMENT [=] 'string'
| ENGINE [=] engine_name
| ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT |
COMPACT}
| STATS_AUTO_RECALC [=] {DEFAULT | 0 | 1}
```

```
| STATS_PERSISTENT [=] {DEFAULT | 0 | 1}
| STATS_SAMPLE_PAGES [=] value)
}
partition_options:
PARTITION BY
| RANGE{(expr)}
| LIST{(expr)}
[(partition_definition [, partition_definition] ...)]

partition_definition:
PARTITION partition_name
[VALUES
{LESS THAN {(expr | value_list) | MAXVALUE}
|
IN (value_list)}]
[[STORAGE] ENGINE [=] engine_name]
[COMMENT [=] 'string']
```

### 一级 RANGE 分区

**支持类型：**

- DATE，DATETIME，TIMESTAMP
- TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT
- CHAR，VARCHAR

> **说明：**
> 如果分区键是 char 或者 varchar 类型，建议长度不超255。

> **禁止：**
> 避免使用 TIMESTAMP 类型作为分区键，因为 timestamp 受到时区的影响，同时只能使用到2038年。

**示例：**

```
create table t1(a int key, b int) tdsql_distributed by range(a) (s1
values less than(100), s2 values less than(200));
```

### 一级 LIST 分区

**支持类型：**

- DATE，DATETIME，TIMESTAMP
- TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT

- CHAR，VARCHAR

> **注意：**
> - 分区键为字符串时，不要使用中文。
> - tdsql_distributed by ...语法放置于 create table ...的末尾。
> - 创建一级 list 分区表语句中指定的 s1和 s2是每个 set 的别名，基于实现原理，s1、s2不能自定义，只能按照顺序依次命名为 s1、s2…
> - set 的别名可通过/proxy/show status;获取到。
> - 内核5.7版本不支持 TDSQL_DISTRIBUTED BY range|list 的语法。

**示例：**

```
DROP TABLE IF EXISTS employees_list;
CREATE TABLE `employees_list` (
`id`int NOT NULL,
`city` varchar(10),
`fired` DATE NOT NULL DEFAULT '1970.01.01',
PRIMARY KEY(id)
)
TDSQL_DISTRIBUTED BY LIST(id) (
s1 VALUES IN (1,3,5),
s2 VALUES IN (2,4,6)
);
--查看set_1624363222_1和set_1624363251_3的别名分别为s1和s2
```

**二级分区分表键（shardkey）的规则请参考：**
二级分区

## 建单表

支持建立普通的表，语法和 MySQL 完全一致，此时该表的数据全量存在第一个 set 中，所有该类型的表都放在第一个 set 中：

```
mysql> create table noshard_table ( a int, b int key);
Query OK, 0 rows affected (0.02 sec)
```

## 建广播表

支持建小表（广播表），此时该表在所有 set 中都是全量数据，主要方便于跨 set 的 join 操作，同时通过分布式事务保证修改操作的原子性，使得所有 set 的数据完全一致。

```
mysql> create table global_table ( a int, b int key)
shardkey=noshardkey_allset;
Query OK, 0 rows affected (0.06 sec)
```

# 连接保护

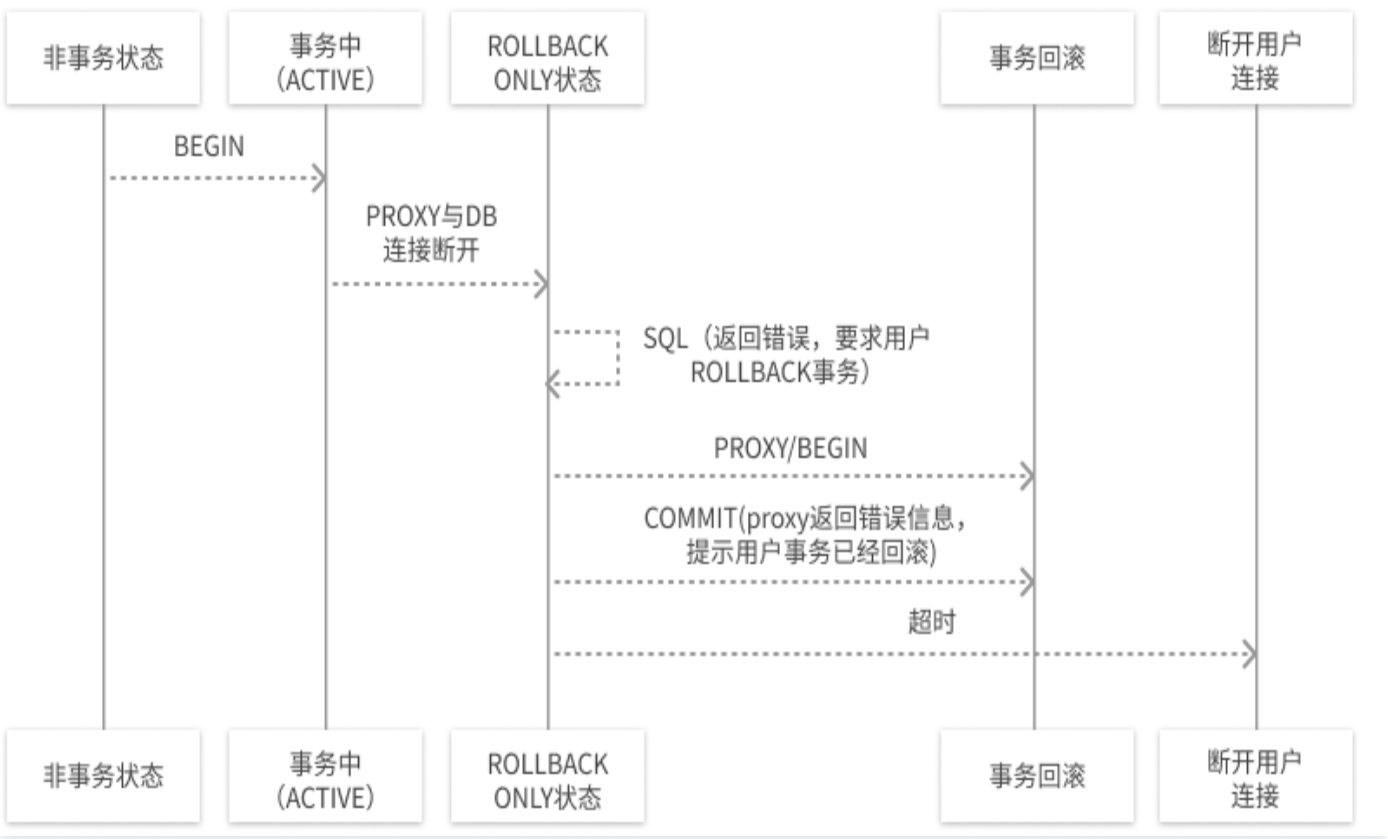
最近更新时间：2022-02-11 10:30:55

连接保护用于 proxy 与后端数据库连接断开时（如后端数据库发生了异常，但异常未影响 proxy），保持客户端和 proxy 之间的连接不断开。
proxy 正在执行 SQL 时，如果与数据库的连接断开，则 proxy 断开与该 SQL 相关的所有连接（除 proxy 与客户端之间的连接），并告知用户错误：

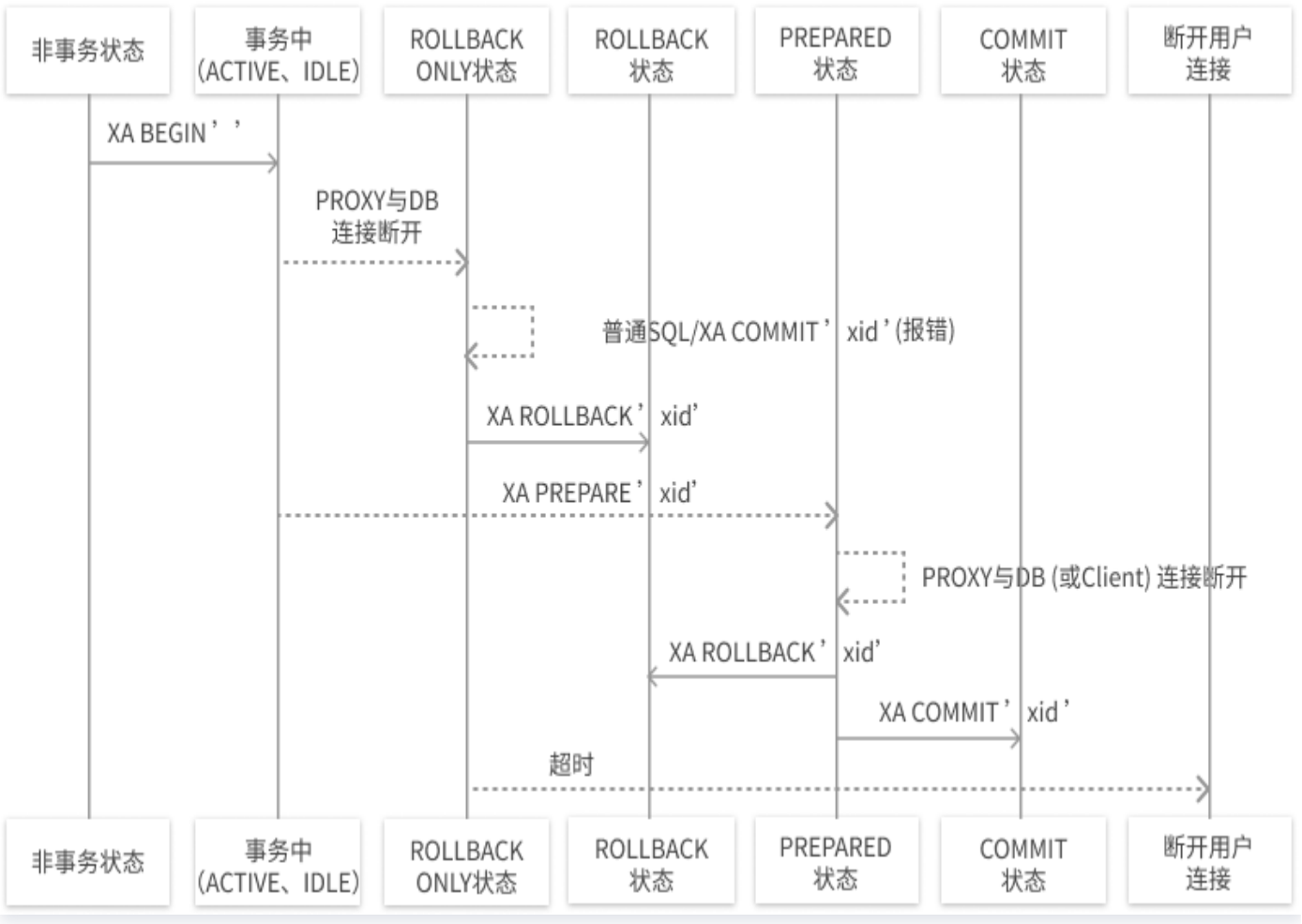
```
ER_PROXY_TRANSACTION_ERROR // 在事务中
ER_PROXY_CONN_BROKEN_ERROR // 非事务中
```

## 处理方式

- 如果 proxy 与后端数据库连接断开时，用户 session 处于 `普通事务` 中，处理如下（图中错误码均为 ER_PROXY_TRANSACTION_ERROR）：




- 如果 proxy 与后端数据库连接断开时，用户正处于 `XA 事务` 中，处理如下（图中错误码均为 ER_PROXY_TRANSACTION_ERROR）：




## 超时配置

> **说明**
> 用户在事务中 proxy 与后端数据库发生连接断开事件，如果用户在超时之前还没有回滚事务，则 proxy 断开与用户的连接。

超时参数在 proxy 配置文件中为：

```
<server_close timeout="60"/>
```

# 保留关键字

最近更新时间：2024-04-30 17:36:23

此文章将列出 TDSQL MySQL 版当前所涉及的保留关键字，在SQL中使用时需添加 `‘’` 才能被 Proxy 解析。

主要保留关键字变更记录

| 关键字 | 变更内容 | 引入版本 |
| --- | --- | --- |
| CYCLE | 允许 CYCLE 关键字做表名字段名 | 22.3.0 |
| SEQUENCE | 允许 SEQUENCE 关键字做表名字段名 | 22.3.0 |
| TDSQL_SUBDISTRIBUTED | proxy 多级分区表 | 22.3.0 |
| WITHOUT | proxy 回收站语法 | 22.3.0 |
| RECYCLE_BIN | proxy 回收站语法 | 22.3.0 |
| RECYCLE_ID | proxy 回收站语法 | 22.3.0 |
| CLEAR | proxy 回收站语法 | 22.3.0 |
| TEMPLATE | set 级全局索引 | 22.2.0 |
| TDSQL_AUTOINCVAL | 通过命令手动获取自增 id | 22.1.4 |
| MANUAL_SWITCH | rebalance 手动切换路由 | 22.1.0 |
| SWITCH | rebalance 手动切换路由 | 22.1.0 |
| SWITCH_DATETIME | rebalance 手动切换路由 | 22.1.0 |
| DDL | 查询 ddl 任务相关信息 | 20.8 |
| OFFLINE | 离线 rebalance | 19.1 |
| REBALANCE_TASK | 查询 rebalance 任务相关信息 | 19.1 |
| TDSQL_RESETVAL | sequence 重置序列号 | 17.2 |
| TDSQL_DISTRIBUTED | range/list 分区表 | 16.3 |
| SKIP | select for update 的 option | 15.1 |
| LOCKED | select for update 的 option | 15.1 |
| NOWAIT | select for update 的 option | 15.1 |

版权所有：腾讯云计算（北京）有限责任公司

| | | |
|---|---|---|
| OF | select for update 的 option | 15.1 |
| CYCLE | proxy 兼容 db 的 sequence 语法 | 15.1 |
| INCREMENT | proxy 兼容 db 的 sequence 语法 | 15.1 |
| LASTVAL | proxy 兼容 db 的 sequence 语法 | 15.1 |
| MAXVALUE | proxy 兼容 db 的 sequence 语法 | 15.1 |
| NEXTVAL | proxy 兼容 db 的 sequence 语法 | 15.1 |
| NOCACHE | proxy 兼容 db 的 sequence 语法 | 15.1 |
| NOCYCLE | proxy 兼容 db 的 sequence 语法 | 15.1 |
| NOMAXVALUE | proxy 兼容 db 的 sequence 语法 | 15.1 |
| NOMINVALUE | proxy 兼容 db 的 sequence 语法 | 15.1 |
| PREVIOUS | proxy 兼容 db 的 sequence 语法 | 15.1 |
| RESTART | proxy 兼容 db 的 sequence 语法 | 15.1 |
| REUSE | proxy兼容 db 的 sequence 语法 | 15.1 |
| SEQUENCE | proxy 兼容 db 的 sequence 语法 | 15.1 |
| SETVAL | proxy 兼容 db 的 sequence 语法 | 15.1 |
| TDSQL_CYCLE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_INCREMENT | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_LASTVAL | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_MINVALUE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_NEXTVAL | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_NOCACHE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_NOCYCLE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_NOMAXVALUE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_NOMINVALUE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_PREVIOUS | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |

版权所有：腾讯云计算（北京）有限责任公司

| | | |
|---|---|---|
| TDSQL_RESTART | prox y支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_REUSE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_SEQUENCE | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |
| TDSQL_SETVAL | proxy 支持带 tdsql_ 前缀的 sequence 语法 | 13.21 |

**更早的保留关键字列表**

```
&&, <, <=, <>, !=, =, >, >=, <<, >>, <=>, ACCESSIBLE, ACTION, ADD,
ADMIN, AFTER, AGAINST, AGGREGATE, ALL, ALGORITHM, ALTER, ALWAYS,
ANALYZE, AND, ANY, AS, ASC, ASCII, ASENSITIVE, AT, AUTHORS,
AUTO_INCREMENT,
AUTOEXTEND_SIZE, AUTO, AVG, AVG_ROW_LENGTH, BACKUP, BEFORE, BEGIN,
BETWEEN, BIGINT, BINARY, BINLOG, BIT, BLOB, BLOCK, BODY, BOOL, BOOLEAN,
BOTH, BTREE, BY, BYTE, CACHE, CALL, CASCADE, CASCADED, CASE,
CATALOG_NAME, CHAIN, CHANGE, CHANGED, CHAR, CHARACTER, CHARSET, CHECK,
CHECKPOINT, CHECKSUM, CIPHER, CLASS_ORIGIN, CLIENT, CLIENT_STATISTICS,
CLOSE, COALESCE, CODE, COLLATE, COLLATION, COLUMN, COLUMN_NAME,
COLUMNS, COLUMN_ADD, COLUMN_CHECK, COLUMN_CREATE, COLUMN_DELETE,
COLUMN_GET, COMMENT, COMMIT, COMMITTED, COMPACT, COMPLETION, COMPRESSED,
CONCURRENT, CONDITION, CONNECTION, CONSISTENT, CONSTRAINT,
CONSTRAINT_CATALOG, CONSTRAINT_NAME, CONSTRAINT_SCHEMA, CONTAINS,
CONTEXT, CONTINUE, CONTRIBUTORS, CONVERT, CPU, CREATE, CROSS, CUBE,
CURRENT, CURRENT_DATE, CURRENT_POS, CURRENT_ROLE, CURRENT_TIME,
CURRENT_TIMESTAMP, CURRENT_USER, CURSOR, CURSOR_NAME, CYCLE, DATA,
DATABASE, DATABASES, DATAFILE, DATE, DATETIME, DAY, DAY_HOUR,
DAY_MICROSECOND, DAY_MINUTE, DAY_SECOND, DEALLOCATE, DEC, DECIMAL,
DECLARE,
DEFAULT, DEFINER, DELAYED, DELAY_KEY_WRITE, DELETE, DESC, DESCRIBE,
DES_KEY_FILE, DETERMINISTIC, DIAGNOSTICS, DIRECTORY, DISABLE, DISCARD,
DISK, DISTINCT, DISTINCTROW, DIV, DO, DOUBLE, DROP, DUAL, DUMPFILE,
DUPLICATE, DYNAMIC, EACH, ELSE, ELSEIF, ENABLE, ENCLOSED, END, ENDS,
ENGINE, ENGINES, ENUM, ERROR, ERRORS, ESCAPE, ESCAPED, EVENT, EVENTS,
EVERY, EXAMINED, EXCHANGE, EXECUTE, EXISTS, EXIT, EXPANSION,
EXPORT, EXPLAIN, EXTENDED, EXTENT_SIZE, FALSE, FAST, FAULTS, FETCH,
FIELDS, FILE, FIRST, FIXED, FLOAT, FLOAT4, FLOAT8, FLUSH, FOR, FORCE,
FOREIGN, FOUND, FROM, FULL, FULLTEXT, FUNCTION, GENERAL, GENERATED,
GEOMETRY, GEOMETRYCOLLECTION, GET_FORMAT, GET, GLOBAL, GRANT, GRANTS,
GROUP, HANDLER, HARD, HASH, HAVING, HELP, HIGH_PRIORITY, HOST, HOSTS,
HOUR, HOUR_MICROSECOND, HOUR_MINUTE, HOUR_SECOND, ID, IDENTIFIED,
IF, IGNORE, IGNORE_SERVER_IDS, IMPORT, IN, INCREMENT, INDEX, INDEXES,
INDEX_STATISTICS, INFILE, INITIAL_SIZE, INNER, INOUT, INSENSITIVE,
```

```
INSERT, INSERT_METHOD, INSTALL, INT, INT1, INT2, INT3, INT4, INT8,
INTEGER, INTERVAL, INTO, IO, IO_THREAD, IPC, IS, ISOLATION, ISSUER,
ITERATE, INVOKER, JOIN, JSON, KEY, KEYS, KEY_BLOCK_SIZE, KILL, LANGUAGE,
LAST, LAST_VALUE, LASTVAL, LEADING, LEAVE, LEAVES, LEFT, LESS,
LEVEL, LIKE, LIMIT, LINEAR, LINES, LINESTRING, LIST, LOAD, LOCAL,
LOCALTIME, LOCALTIMESTAMP, LOCK, LOCKS, LOGFILE, LOGS, LONG, LONGBLOB,
LONGTEXT, LOOP, LOW_PRIORITY, MASTER, MASTER_CONNECT_RETRY,
MASTER_GTID_POS, MASTER_HOST, MASTER_LOG_FILE, MASTER_LOG_POS,
MASTER_PASSWORD, MASTER_PORT, MASTER_SERVER_ID, MASTER_SSL,
MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT, MASTER_SSL_CIPHER,
MASTER_SSL_CRL, MASTER_SSL_CRLPATH, MASTER_SSL_KEY,
MASTER_SSL_VERIFY_SERVER_CERT, MASTER_USER, MASTER_USE_GTID,
MASTER_HEARTBEAT_PERIOD, MATCH, MAX_CONNECTIONS_PER_HOUR,
MAX_QUERIES_PER_HOUR, MAX_ROWS,
MAX_SIZE, MAX_UPDATES_PER_HOUR, MAX_USER_CONNECTIONS, MAXVALUE, MEDIUM,
MEDIUMBLOB, MEDIUMINT, MEDIUMTEXT, MEMORY, MERGE, MESSAGE_TEXT,
MICROSECOND, MIDDLEINT, MIGRATE, MINUTE, MINUTE_MICROSECOND,
MINUTE_SECOND, MINVALUE, MIN_ROWS, MOD, MODE, MODIFIES, MODIFY, MONTH,
MULTILINESTRING, MULTIPOINT, MULTIPOLYGON, MUTEX, MYSQL_ERRNO, NAME,
NAMES, NATIONAL, NATURAL, NDB, NDBCLUSTER, NCHAR, NEW, NEXT,
NEXTVAL, NO, NOMAXVALUE, NOMINVALUE, NOCACHE, NOCYCLE, NO_WAIT,
NODEGROUP, NONE, NOT, NO_WRITE_TO_BINLOG, NULL, NUMBER, NUMERIC,
NVARCHAR, OFFSET, OLD_PASSWORD, ON, ONE, ONLINE, ONLY, OPEN, OPTIMIZE,
OPTIONS, OPTION, OPTIONALLY, OR, ORDER, OUT, OUTER, OUTFILE, OWNER,
PACK_KEYS, PAGE, PAGE_CHECKSUM, PARSER, PARSE_VCOL_EXPR, PARTIAL,
PARTITION, PARTITIONING, PARTITIONS, PASSWORD, PERSISTENT, PHASE,
PLUGIN,
PLUGINS, POINT, POLYGON, PORT, PRECISION, PREPARE, PRESERVE, PREV,
PREVIOUS, PRIMARY, PRIVILEGES, PROCEDURE, PROCESS, PROCESSLIST, PROFILE,
PROFILES, PROXY, PURGE, QUARTER, QUERY, QUICK, RANGE, READ,
READ_ONLY, READ_WRITE, READS, REAL, REBUILD, RECOVER, REDO_BUFFER_SIZE,
REDOFILE, REDUNDANT, REFERENCES, REGEXP, RELAY, RELAYLOG,
RELAY_LOG_FILE, RELAY_LOG_POS, RELAY_THREAD, RELEASE, RELOAD, REMOVE,
RENAME,
REORGANIZE, REPAIR, REPEATABLE, REPLACE, REPLICATION, REPEAT, REQUIRE,
RESET, RESIGNAL, RESTART, RESTORE, RESTRICT, RESUME, RETURNED_SQLSTATE,
RETURN, RETURNING, RETURNS, REUSE, REVERSE, REVOKE, RIGHT, RLIKE,
ROLE, ROLLBACK, ROLLUP, ROUTINE, ROW, ROW_COUNT, ROWS, ROW_FORMAT,
RTREE, SAVEPOINT, SCHEDULE, SCHEMA, SCHEMA_NAME, SCHEMAS,
SECOND,SECOND_MICROSECOND,SECURITY, SELECT, SENSITIVE, SEPARATOR,
SEQUENCE, SERIAL,
SERIALIZABLE, SESSION, SERVER, SET, SETVAL, SHARDKEY, SHARE, SHOW,
SHUTDOWN, SIGNAL, SIGNED, SIMPLE, SLAVE, SLAVES, SLAVE_POS, SLOW,
```

```
SNAPSHOT, SMALLINT, SOCKET, SOFT, SOME, SONAME, SOUNDS, SOURCE, SPATIAL,
SPECIFIC, SQL, SQLEXCEPTION, SQLSTATE, SQLWARNING, SQL_BIG_RESULT,
SQL_BUFFER_RESULT, SQL_CACHE, SQL_CALC_FOUND_ROWS, SQL_NO_CACHE,
SQL_SMALL_RESULT, SQL_THREAD, SQL_TSI_SECOND, SQL_TSI_MINUTE,
SQL_TSI_HOUR,
SQL_TSI_DAY, SQL_TSI_WEEK, SQL_TSI_MONTH, SQL_TSI_QUARTER, SQL_TSI_YEAR,
SSL, START, STARTING, STARTS, STATS_AUTO_RECALC, STATS_PERSISTENT,
STATS_SAMPLE_PAGES, STATUS, STOP, STORAGE, STORED, STRAIGHT_JOIN,
STRING, SUBCLASS_ORIGIN, SUBJECT, SUBPARTITION, SUBPARTITIONS, SUPER,
SUSPEND, SWAPS, SWITCHES, TABLE, TABLE_NAME, TABLES, TABLESPACE,
TABLE_STATISTICS, TABLE_CHECKSUM, TEMPORARY, TEMPTABLE, TERMINATED,
TEXT,
THAN, THEN, TIME, TIMESTAMP, TIMESTAMPADD, TIMESTAMPDIFF, TINYBLOB,
TINYINT, TINYTEXT, TO, TRAILING, TRANSACTION, TRANSACTIONAL, TRIGGER,
TRIGGERS, TRUE, TRUNCATE, TYPE, TYPES, UNCOMMITTED, UNDEFINED,
UNDO_BUFFER_SIZE, UNDOFILE, UNDO, UNICODE, UNION, UNIQUE, UNKNOWN,
UNLOCK, UNINSTALL, UNSIGNED, UNTIL, UPDATE, UPGRADE, USAGE, USE, USER,
USER_RESOURCES, USER_STATISTICS, USE_FRM, USING, UTC_DATE, UTC_TIME,
UTC_TIMESTAMP, VALUE, VALUES, VARBINARY, VARCHAR, VARCHARACTER,
VARIABLES, VARYING, VIA, VIEW, VIRTUAL, WAIT, WARNINGS, WEEK,
WEIGHT_STRING, WHEN, WHERE, WHILE, WITH, WORK, WRAPPER, WRITE, X509,
XOR, XA, XML,
YEAR, YEAR_MONTH, ZEROFILL, ||, BOOST
```

# 全局唯一字段

最近更新时间：2024-11-19 11:51:52

关键字 `auto_increment` ，即支持一个全局的自增字段，auto_increment 可以保证该表某个字段全局唯一，但不保证单调递增，具体使用方法如下：

## 创建

```
mysql> create database shard;
Query OK, 0 rows affected (0.25 sec)

mysql> create table shard.auto_inc (a int,b int,c int auto_increment,d
int,key auto(c),primary key p(a,d)) shardkey=d;
Query OK, 0 rows affected (0.12 sec)
```

## 插入

```
mysql>  insert into shard.auto_inc ( a,b,d,c) values(1,2,3,0),(1,2,4,0);
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from shard.auto_inc;
+---+------+---+---+
| a | b    | c | d |
+---+------+---+---+
| 1 |    2 | 2 | 4 |
| 1 |    2 | 1 | 3 |
+---+------+---+---+
2 rows in set (0.03 sec)
```

如果发生切换、重启等过程，自增长字段中间会有空洞，例如：

```
mysql> insert into shard.auto_inc ( a,b,d,c) values(11,12,13,0),
(21,22,23,0);
Query OK, 2 rows affected (0.03 sec)
mysql> select * from shard.auto_inc;
+------+------+------+------+
| a | b | c | d |
+------+------+------+------+
| 21 | 22 | 2002 | 23 |
```

```
| 1 | 2 | 2 | 4 |
| 1 | 2 | 1 | 3 |
| 11 | 12 | 2001 | 13 |
+------+------+------+------+
4 rows in set (0.01 sec)
```

**更改当前值：**

```
alter table shard.auto_inc auto_increment=100
```

**通过** `select last_insert_id()` **获取最近一个自增值：**

```
mysql> insert into shard.auto_inc ( a,b,d,c) values(1,2,3,0),(1,2,4,0);
Query OK, 2 rows affected (0.73 sec)

mysql> select * from shard.auto_inc;
+---+------+------+---+
| a | b    | c    | d |
+---+------+------+---+
| 1 |    2 | 4001 | 3 |
| 1 |    2 | 4002 | 4 |
+---+------+------+---+
2 rows in set (0.00 sec)

mysql> select last_insert_id();
+------------------+
| last_insert_id() |
+------------------+
| 4001             |
+------------------+
1 row in set (0.00 sec)
```

**目前 select last_insert_id() 只能跟 shard 表和广播表的自增字段一起使用，不支持 noshard 表。**

# 数据导出导入

最近更新时间：2024-04-25 15:36:22

## 导出数据

TDSQL MySQL 版支持通过 mysqldump 导出数据，导出前须设置 net_write_timeout 参数：`set global net_write_timeout = 28800`，命令行有权限限制，请通过 TDSQL MySQL 版控制台 操作。操作步骤如下：

1. 在**实例列表**中，单击**实例 ID** 或操作列的**管理**，进入实例管理页面。
2. 在实例管理页面，选择**参数配置**页，找到 `net_write_timeout` 参数进行配置。
3. 通过 **mysqldump** 导出数据。

```
mysqldump --compact --single-transaction --no-create-info -c db_name
table_name  -utest -h10.xx.xx.34 -P3336  -ptest123
```

**说明：**

- db 和 table 名参数根据实际情况选择，其中 db 和 table 分别代表库名和表名，如果导出的数据要导入到另外一套 TDSQL MySQL 版环境的话，必须加上 -c 选项。
- 导出账号需拥有 `select on *.*` 的权限。
- 高版本 mysqldump 导出低版本数据库时，可能发生报错，可设置参数 `-column-statistics = 0`，用于解决此类问题。

## 导入数据

TDSQL MySQL 版提供专门的导入数据工具，完成 load data outfile 对应数据的导入，该工具的原理是把源文件按照 shardkey 的路由规则，切分成多个文件，然后把每个单独透传到对应的后端数据库。
下载工具

```
[tdengine@TENCENT64 ~/]$./load_data

format:./load_data mode0/mode1 proxy_host proxy_port user password
db_table shardkey_index file field_terminate filed_enclosed

example:./load_data mode1 10.xx.xx.10 3336 test test123 shard.table  1
'/tmp/datafile'  ' ' ''
```

**注意：**

- 源文件必须以 '\n' 作为换行符。
- mode0 只切分源文件，不做数据导入，一般用于调试，正式导入数据使用 mode1。
- shardkey_index 从0开始，如果 shardkey 在第2个字段，则 shardkey_index 为1。

# 数据库管理语句

最近更新时间：2022-02-11 10:32:53

## 状态查询

通过 SQL 可以查看 proxy 的配置以及状态信息，目前支持如下命令：

- `/*proxy*/help;`
- `/*proxy*/show config;`
- `/*proxy*/show status;`

> **说明**
> 如果使用 MySQL 客户端，需要在使用客户端时增加 `-c` 选项，如 mysql -hxxx.xxx.xxx.xxx -Pxxxx -uxxx -pxxx -c。

示例如下：

```
    mysql> /*proxy*/help;
    +-----------------------+-------------------------------------------
------------+
    | command               | description
|
    +-----------------------+-------------------------------------------
------------+
    | show config           | show config from conf
|
    | show status           | show proxy status,like route,shardkey and
so on       |
    | set sys_log_level=N   | change the sys debug level N should be
0,1,2,3        |
    | set inter_log_level=N | change the interface debug level N should
be 0,1      |
    | set inter_time_open=N | change the interface time debug level N
should be 0,1 |
    | set sql_log_level=N   | change the sql debug level N should be 0,1
|
    | set slow_log_level=N  | change the slow debug level N should be
0,1           |
    | set slow_log_ms=N     | change the slow ms
|
```

```
    | set log_clean_time=N  | change the log clean days
|
    | set log_clean_size=N  | change the log clean size in GB
|
    +----------------------+-------------------------------------------
------------+
    10 rows in set (0.00 sec)
```

```
    mysql> /*proxy*/show config;
    +----------------+--------------------+
    | config_name     | value              |
    +----------------+--------------------+
    | version         | V2R120D001         |
    | mode            | group shard        |
    | rootdir         | /shard_922         |
    | sys_log_level   | 0                  |
    | inter_log_level | 0                  |
    | inter_time_open | 0                  |
    | sql_log_level   | 0                  |
    | slow_log_level  | 0                  |
    | slow_log_ms     | 1000               |
    | log_clean_time  | 1                  |
    | log_clean_size  | 1                  |
    | rw_split        | 1                  |
    | ip_pass_through | 0                  |
    +----------------+--------------------+
    14 rows in set (0.00 sec)
```

```
    mysql> /*proxy*/show status;
    +----------------------------+-------------------------------------
----------------------------------------+
    | status_name                | value
|
    +----------------------------+-------------------------------------
----------------------------------------+
    | cluster                    | group_1499858910_79548
|
    | set_1499859173_1:ip        |
10.49.118.165:5025;10.175.98.109:5025@1@IDC_4@0,10.231.23.241:5025@1@IDC
_2@0 |
```

```
    | set_1499859173_1:hash_range | 0---31
|
    | set_1499911640_3:ip         |
10.49.118.165:5026;10.175.98.109:5026@1@IDC_4@0,10.231.23.241:5026@1@IDC
_2@0 |
    | set_1499911640_3:hash_range | 32---63
|
    | set                         | set_1499859173_1,set_1499911640_3
|
```

同时 proxy 增强了 explain 的返回结果，显示 proxy 修改后的 SQL。

```
    mysql> explain select * from test1;
    +------+-------------+-------+------+---------------+------+--------
-+------+------+-------+-----------------------------------------+
    | id   | select_type | table | type | possible_keys | key  | key_len
| ref  | rows | Extra | info                                    |
    +------+-------------+-------+------+---------------+------+--------
-+------+------+-------+-----------------------------------------+
    |    1 | SIMPLE      | test1 | ALL  | NULL          | NULL | NULL
| NULL |   16 |       | set_2,explain select * from shard.test1 |
    |    1 | SIMPLE      | test1 | ALL  | NULL          | NULL | NULL
| NULL |   16 |       | set_1,explain select * from shard.test1 |
    +------+-------------+-------+------+---------------+------+--------
-+------+------+-------+-----------------------------------------+
    2 rows in set (0.03 sec)
```

# 透传 SQL

最近更新时间：2022-11-24 11:02:08

TDSQL MySQL版 实例会对 SQL 进行语法解析，有一定的限制，如果用户想在某个节点（set）中执行 MySQL 支持，但分布式实例不支持的 SQL 时，可以使用透传 SQL 的功能。

**说明**

- 透传 SQL 时，proxy 不会解析 SQL，如果是往两个 set 进行透传写操作，不会使用分布式事务，特殊情况下会发生不一致问题，因此对于写操作建议一次透传一个 set。
- 为保证透传语法生效，连接 MySQL 时请使用 -c 参数。

```
MySQL [test]> repair table test.t1;
ERROR 664 (HY000): Proxy ERROR:SQL is too complex, only applicable to
noshard table: Shard table do not support repair
MySQL [test]> /*sets:allsets*/repair table test.t1;
+---------+--------+----------+----------+------------------+
| Table   | Op     | Msg_type | Msg_text | info             |
+---------+--------+----------+----------+------------------+
| test.t1 | repair | status   | OK       | set_1544429866_3 |
| test.t1 | repair | status   | OK       | set_1544429718_1 |
+---------+--------+----------+----------+------------------+
2 rows in set (0.01 sec)
```

具体语法：

- sets:set_1,set_2：代表指定某几个 set，set 名字可以通过 `/*proxy*/show status` 查询。
- sets:allsets：代表指定全部 set。
- shardkey:10：代表支持透传 SQL 到 shardkey 对应值上的 set。
- shardkey_hash:10：透传到负责 hash 值为10的 set，如果为0，则发送到第一个 set 上。

# 预处理

最近更新时间：2025-06-30 11:21:52

SQL 类型的支持：

- PREPARE Syntax
- EXECUTE Syntax

二进制协议的支持：

- COM_STMT_PREPARE
- COM_STMT_EXECUTE

预处理语句的作用：

- 防止 SQL 注入：由于使用了参数化查询，用户输入的数据被作为参数传递，而不是直接拼接到 SQL 语句中。这有效防止了恶意用户通过输入特殊字符来操控 SQL 语句结构，从而避免了 SQL 注入攻击。
- 提高性能：数据库可以缓存预处理语句的执行计划。当相同的预处理语句多次执行时，数据库无需重新解析和编译 SQL 语句，只需绑定新的参数即可，从而提高了查询性能。

示例：

```
mysql> select * from test1;
+---+------+
| a | b    |
+---+------+
| 5 |    6 |
| 3 |    4 |
| 1 |    2 |
+---+------+
3 rows in set (0.03 sec)
```

```
mysql> prepare ff from "select * from test1 where a=?";
Query OK, 0 rows affected (0.00 sec)
Statement prepared

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

mysql> execute ff using @aa;
+---+------+
| a | b    |
+---+------+
| 3 |    4 |
+---+------+
```

```
1 row in set (0.06 sec)
```

# 二级分区

最近更新时间：2024-07-02 16:58:31

TDSQL MySQL 版目前支持 Range 和 List 两种格式的二级分区，具体建表语法和 MySQL 分区语法类似。

## 二级分区语法

一级 Hash，二级 List 分区示例如下：

```
MySQL [test]> CREATE TABLE customers_1 (
  first_name VARCHAR(25) key,
  last_name VARCHAR(25),
  street_1 VARCHAR(30),
  street_2 VARCHAR(30),
  city VARCHAR(15),
  renewal DATE
) shardkey=first_name

PARTITION BY LIST (city) (
  PARTITION pRegion_1 VALUES IN('Beijing', 'Tianjin', 'Shanghai'),
  PARTITION pRegion_2 VALUES IN('Chongqing', 'Wulumuqi', 'Dalian'),
  PARTITION pRegion_3 VALUES IN('Suzhou', 'Hangzhou', 'Xiamen'),
  PARTITION pRegion_4 VALUES IN('Shenzhen', 'Guangzhou', 'Chengdu')
);
```

一级 Range，二级 List 创建语法如下：

```
MySQL [test]> CREATE TABLE tb_sub_r_l (
   id int(11) NOT NULL,
   order_id bigint NOT NULL,
   PRIMARY KEY (id,order_id))
   PARTITION BY list(order_id)
   (PARTITION p0 VALUES in (2121122),
   PARTITION p1 VALUES in (38937383))
   TDSQL_DISTRIBUTED BY RANGE(id) (s1 values less than (100),s2 values
less than (1000));
Query OK, 0 rows affected, 1 warning (0.35 sec)
```

若使用 month 函数，则查询插入对应的值格式应为20160101，必须满足时间戳的格式要求：

```
CREATE TABLE employees_int (
```

```
id INT key NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired date,
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT)
shardkey=id
PARTITION BY RANGE ( month(hired) ) (
PARTITION p0 VALUES LESS THAN (199102),
PARTITION p1 VALUES LESS THAN (199603),
PARTITION p2 VALUES LESS THAN (200101));
```

### Range 支持类型

- DATE，DATETIME，TIMESTAMP。
- TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT。
- 支持 year、month、day 函数，此时传入的值转换为年月日，然后和分表信息进行对比。

### List 支持类型

- DATE、DATETIME、TIMESTAMP。
- 支持年月日函数。
- TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT、VARCHAR。

**警告：**

- 建议不要使用 TIMESTAMP 类型作为分区键，因为 TIMESTAMP 受到时区的影响，同时只能使用到2038年。
- 如果分区键是 char 或者 varchar 类型，建议长度不超255。
- year、month、day 函数的输入值必须满足时间戳的格式要求。例如： YYYYMMDD, YYMMDDHHMMSS, YYYYMMDDHHMMSS,YY-MM-DD, YYYY-MM-DD, YY-MM-DD HH.MM.SS, YYYYMMDDTHHMMS。

**注意：**

- 分区使用小于符号“<”，如果要存储当年数据（例如，2017），需要创建小于往后一年（<2018）的分区，用户只需创建到当前的时间分区。
- TDSQL 会自动增加后续分区，如果是月则默认往后自动增加2个月范围的分区，如果是年则默认往后自动增加2年范围的分区，如果是天则默认往后自动增加5天范围的分区。
- 用户自行创建的二级分区表命名尽量用 p0，p1，…… pN 这样的命名格式。如需个性化定义需要注意命名规范：注意 p_auto_202407 分区对应 RANGE 范围为202407-202408。否则无法使用

TDSQL 自动创建分区功能。

## 使用场景和方法建议

建议业务尽量都使用一级分区表。

- 使用前根据业务长期场景合理设计表结构，二级分区适用于表结构创建后长期都不需要 DDL 变更、需要定期进行分区数据清理和裁剪的场景，如日志流水表。
- 合理设计二级分区的粒度，二级分区的粒度建议不要划分得太细，避免产生过多的二级子表。如流水表按月进行二级分区，而不是按天/小时进行分区，避免文件系统上数据文件个数过多。
- 在对二级分区表进行 SQL 查询时，查询条件需要尽量带上一级分区和二级分区的键值，避免执行查询时需要打开很多的数据文件进行搜索。
- 在对二级分区表进行 join 查询时，如果查询条件未能带上一级分区和二级分区的键值，操作性能效率较低，建议不要使用。
- 表的主键或唯一索引需要包含分区键，否则无法保证数据唯一性。

# 错误码和错误信息

最近更新时间：2023-08-02 15:47:34

proxy 增加如下错误编码：

```
#define ER_PROXY_GRAM_ERROR_BEGIN 600

#define ER_PROXY_SANITY_ERROR 601           // "Sanity error: %s"
#define ER_PROXY_GS_NOT_SUPPORT 602         // sql 类型不支持
#define ER_PROXY_ORDERBY_INDEX_NEG 603      // order by index is
negative
#define ER_PROXY_ORDERBY_INDEX_TOO_BIG 604  // order by index is too big
#define ER_PROXY_ORDERBY_TYPE_UNSUPPORT 605 // 不支持到 order by 用法
#define ER_PROXY_GROUPBY_INDEX_NEG 606      // group by index is
negative
#define ER_PROXY_GROUPBY_INDEX_TOO_BIG 607  // group by index is too big
#define ER_PROXY_GROUPBY_TYPE_UNSUPPORT 608 // 不支持的 group by 用法
#define ER_PROXY_GET_AUTO_ID_FAILED 609     // 获取自增 id 失败
#define ER_PROXY_TEANS_ROLLED_BACK 610      // 事务已经被回滚
#define ER_PROXY_ONE_SET 611                // 当前 sql 应该被发往一个后端，
但是不是
#define ER_PROXY_CLIENT_HS_ERROR 612        // 解析客户端握手包出错
#define ER_PROXY_ACCESS_DENIED_ERROR 613    // the length of
readu_auth_switch_result is not 20，不应该出现
#define ER_PROXY_TRANS_NOT_ALLOWED 614      // 事务中不允许执行的命令
#define ER_PROXY_TRANS_READ_ONLY 615        // 只读事务中不允许执行的命令
#define ER_PROXY_TRANS_ERROR_DIFFENT_SET 616    // 非 xa 事务中，只读 sql
使用了多个后端
#define ER_PROXY_STRICT_ERROR 617           // strict 模式下，一次仅允许修改
一个 set
#define ER_PROXY_SC_TOO_LONG 618            // 后端断开时间过长，链接断开
#define ER_PROXY_START_TRANS_FAILED 619     // 开启新的 xa 事务失败
#define ER_PROXY_SC_RETRY 620               // server 已经 close，请重试上
一条 sql
#define ER_PROXY_SC_TRANS_IN_ROLLBACK_ONLY 621  // server 已经 close，当前
事务处于 rollback
#define ER_PROXY_SC_COMMIT_LATER 622        // server 已经 close，事务会在
稍后提交
#define ER_PROXY_SC_ROLLBACL_LATER 623      // server 已经 close，事务会在
稍后回滚
#define ER_PROXY_SC_IN_COMMIT_OR_ROLLBACK 624   //  server 在事务提交/回滚
阶段 close
```

```
#define ER_PROXY_SC_NEED_ROLLBACK 625        // server 已经 close，需要回滚
当前事务
#define ER_PROXY_SC_STATE_WILL_ROLLBACK 626 // server 已经 close，将会会滚
#define ER_PROXY_XA_UNSUPPORT 627            // xa 目前不支持的命令
#define ER_PROXY_XA_INVALID_COMMAND 628      // xa 命令不合法
#define ER_PROXY_XA_GTID_INIT_ERROR 629      // gtid log 初始化失败
#define ER_PROXY_XA_GET_SET_IP_PORT_FAILED 630  // 获取 set 地址失败
#define ER_PROXY_XA_UPDATE_GTID_LOG_FAILED 631  // 更新 gtid log 失败
#define ER_PROXY_MYSQL_PARSER_ERROR 632      // 嵌入式库 sql 解析失败
#define ER_PROXY_ILLEGAL_ID 633              // kill id 不合法
#define ER_PROXY_NOT_SUPPORT_CURSOR 634      // CURSOR_TYPE_READ_ONLY 暂不
支持
#define ER_PROXY_UNKNOWN_PREPARE_HANDLER 635    // 执行的 prepare 不明确
#define ER_PROXY_SET_PARA_FAIL 636           // Set parameters failed
#define ER_PROXY_SUBPARTITION_DEAY 637       // 处理二级分区发生错误
#define ER_PROXY_NO_SUBPARTITION_ROUTE 638   // 没有获取到二级分区表的路由信息
#define ER_PROXY_LOCK_MORE_TABLE 639         // 一次只可以锁定一张二级分区表
#define ER_PROXY_GET_ROUTER_LOCK_FAIL 640    // 获取路由锁失败
#define ER_PROXY_PART_NAME_EMPTY 641         // 分区名称为空
#define ER_PROXY_SUB_PART_TABLE_IS_NONE 642 // 没有二级分区
#define ER_PROXY_PART_TYPE_DENY 643          // 二级分区类型不支持
#define ER_PROXY_PART_NAME_ILLEGAL 644       // 分区名不合法
#define ER_PROXY_DROP_ALL_PARTITION_FAIL 645    //  删除所有分区失败，尝试直
接删除表
#define ER_PROXY_GET_OLD_PART_NUM_FAIL 646   // 获取表的分片数失败
#define ER_PROXY_EMPTY_SQL 647               // empty sql，不会返回给客户端
#define ER_PROXY_ERROR_SHARDKEY 648          // sk 必须为某一列
#define ER_PROXY_ERROR_SUB_SHARDKEY 649      // 二级分区键失败
#define ER_PROXY_SQLUSE_NOT_SUPPORT 650      // proxy 不支持这种用法
#define ER_PROXY_DBFW_WHITE_LIST_DENY 651    // 不在白名单，被防火墙拒绝
#define ER_PROXY_DBFW_DENY 652               // 防火墙拒绝
#define ER_PROXY_INCORRECT_ARGS 653          // stmt 参数不正确
#define ER_PROXY_SYSTABLE_UNSUPPORT_NON_READ_SQL 654    // 不支持非只读
sql 访问系统表
#define ER_PROXY_TABLE_NOT_EXIST 655         // 表不存在
#define ER_PROXY_SHARD_JOIN_UNSUPPORT_TYPE 656  // shard join 暂不支持的用
法
#define ER_PROXY_RECURSIVE_JOIN_DENY 657     // 递归 join 不支持
#define ER_PROXY_JOIN_INTERNAL_ERROR 658     // join 异常
#define ER_PROXY_SQL_TOO_COMPLEX 659         // sql 太复杂，groupshard 暂不
支持
#define ER_PROXY_INVALID_ARG_FOR_GTID_STATE 660 // gtid_state() 参数不合法
```

```
#define ER_PROXY_CANT_SET_GLOBAL_AUTOCOMMIT_GS 661  // Global autocommit
cannot be set in groupshard
#define ER_PROXY_INVALID_VALUE_FOR_AUTOCOMMIT 662   // autocommit 值设置不
合法
#define ER_PROXY_XID_ERROR 663              // xid 不合法
#define ER_PROXY_XID_GENERAT_FAILED 664     // xid 不能由用户指定
#define ER_PROXY_CANT_EXEC_IN_INTER_TRANS 665   // "The command cannot
be executed in internal transction"
#define ER_PROXY_XID_TIME_ERROR 666         // "Unexpected time part of
xid"
#define ER_PROXY_XID_TIMEDIFF_TOO_LONG 667  // "timediff > 1800s, it's
not safe to execute boost"
#define ER_PROXY_SAVEPOINT_NOT_EXIST 668    // SAVEPOINT 不存在
#define ER_PROXY_SC_TRANS_IN_ROLLED 669     // 事务已经会滚，由于 serevr 已
经 close
#define ER_PROXY_CANT_BOOST_IN_TRANS 670    // 事务中不允许执行
SQLCOM_BOOST
#define ER_PROXY_TRANS_EXPECTED 671         // "A transaction is
expected, this maybe a bug"
#define ER_PROXY_EXTERNAL_TRANS 672         // 外部 xa 中不允许执行
#define ER_PROXY_AUTO_INC_FAIL 673          // "Deal auto inc failed"
#define ER_PROXY_CHECK_JOIN_FAIL 674        // "Check join failed"
#define ER_PROXY_TABLE_TYPE_NOT_MATCH 675   // "Do not support shard-
table operations in noshard instance"
#define ER_PROXY_UNSUPPORT_NS_IN_INSERT 676 // "Do not support noshard
and noshard_allset in insert sql"
#define ER_PROXY_ALTER_SEQ_ID_FAIL 677      // Alter seq id failed
#define ER_PROXY_ALTER_ID_ILLEGAL 678       // Alter seq id is illegal
#define ER_PROXY_CANT_CHANGE_STEP 679       // "Current table use zk to
get auto inc, do not support to change step: \'%s\'"
#define ER_PROXY_ALTER_STEP_FAIL 680        // Alter step failed
#define ER_PROXY_TOO_MUCH_TABLES 681        // 表的数量超过限制
#define ER_PROXY_TABLE_EXISTED 682          // 表已经存在
#define ER_PROXY_CREATE_STABLE_FAILED 683   // 创建shard表失败，复杂 sql 不
能用来创建 shard 表
#define ER_PROXY_DDL_DENY 684               // ddl 不允许的操作
#define ER_PROXY_SHADKEY_ERROR 685          // SQL should not relate to
subpartition tables
#define ER_PROXY_NO_SK 686                  // reject nosk
#define ER_PROXY_COMBINE_SQL_KEY 687        // Something went wrong
#define ER_PROXY_GET_SK_ERROR 688           // sk 获取失败
#define ER_PROXY_SHOW_FAILED 689            // proxy show 命令错误
#define ER_PROXY_SET_FAILED 690             // proxy set 命令错误
```

```
#define ER_PROXY_UNLOCK_FORMAT_ERROR 691     // sql 格式不正确
#define ER_PROXY_UNLOCK_ROUTER_FAIL 692      // 释放路由锁失败
#define ER_PROXY_LOCK_ROUTER_FAIL 693        // 加路由锁失败
#define ER_PROXY_PROXY_CMD_FAIL 694          // 不支持的/*proxy*/ 命令
#define ER_PROXY_PROCESS_RULE_FILE_FAILED 695   // dump_error
#define ER_PROXY_GET_AUTO_NUM_ERROR 696      // 获取自增值失败
#define ER_PROXY_SEQUENCE_NOT_EXIST 697      // sequence 不存在
#define ER_PROXY_SEQUENCE_ERROR 698          // sequence 不合法
#define ER_PROXY_SEQUENCE_ALREADY_EXIST 699 // sequence 已经存在
#define ER_PROXY_SQL_RETRY 700               // sql 还未提交或回滚
#define ER_PROXY_XA2PC_ABORT 701             // 2pc 失败，事务将会会滚
#define ER_PROXY_XA2PC_COMMIT 702            // 2pc 失败，后续提交
#define ER_PROXY_XA2PC_UNCERTAIN 703         // 2pc 失败，结果未知
#define ER_PROXY_KILL_ERROR 704              // kill 失败
#define ER_PROXY_TRACE_DENY 705        // trace 模式下不允许执行的sql
#define ER_PROXY_SQL_IMCOMPLETE 706       // 事务状态不完整
#define ER_PROXY_SHARDKEY_HASH_ERROR 709     // sk hash错误

#define ER_PROXY_GRAM_ERROR_END 799

// system error ------------------------------------------------------------
---------------
#define ER_PROXY_SYSTEM_ERROR_BEGIN 900

#define ER_PROXY_SLICING 901                 // slice 被修改，可能在扩容阶
段，拒掉当前 sql
#define ER_PROXY_NO_DEFAULT_SET 902          // set 为空
#define ER_PROXY_GET_ADDRESS_FAILED 903      // 还未初始化完成，获取后端地址失
败，稍后重试
#define ER_PROXY_SQL_SIZE_ERROR_IN_GET_CANDIDATE_ADDRESS 904 // 获取后端地
址出错（发往后端个数不正确）
#define ER_PROXY_GET_ADDRESS_ERROR 905       // 获取后端地址出错
#define ER_PROXY_CANDIDATE_ADDRESS_EMPTY 906 // 未获取到后端地址
#define ER_PROXY_CANT_GET_SOCK 907           // socket 获取失败
#define ER_PROXY_GET_SET_SOCK_FAIL 908       // socket 获取失败
#define ER_PROXY_CONNECT_ERROR 909           // 后端连接失败
#define ER_PROXY_NO_SQL_ASSIGN_TO_SET 910    // 分发 sql 异常
#define ER_PROXY_STATUS_ERROR 911            // group 状态异常，断开链接
#define ER_PROXY_CONN_BROKEN_ERROR 912       // server close，sql 状态不正
常
#define ER_PROXY_UNKNOWN_ERROR 913           // proxy 未知错误
#define ER_PROXY_ALL_SLAVES_UNAVAILABLE 914 // 所有备机不可用
#define ER_PROXY_ALL_SLAVES_CHANGE 915       // 备机异常
```

```
#define ER_PROXY_ERROR_END 916
```

**说明**

其中900以上为系统错误，会通过 腾讯云可观测平台 进行告警。

# TDStore 引擎与 MySQL 兼容性使用说明

最近更新时间：2025-04-22 15:40:02

TDSQL MySQL 版（TDStore 引擎）在使用上与 MySQL 8.0 高度一致，用户可以将其视为一个 MySQL 8.0 实例来使用，但请注意以下几点：

## TDStore 引擎暂不支持的语法特性

- 不支持下列对象的 CREATE/ALTER/DROP 语法。
  - event
  - resource group
  - instance
  - server
  - tablespace
  - spatial
- SET GLOBAL log_output = TABLE：不支持 general_log 和 slow_log 输出到表，只能输出到 file（即慢查询日志文件）。
- 暂不支持 LOCK TABLE 语法（默认依赖 LOCK TABLE 语法的数据导入导出工具，建议通过参数设置跳过 LOCK TABLE 的执行避免报错，例如 mysqldump --lock-tables = false）。
- 禁止对 MySQL 库下的任何系统表进行 DDL 操作。
- 暂不支持外键。
- 不允许修改表的存储引擎。
- 暂不支持虚拟列、GEOMETRY 类型、降序索引。
- 不支持 ALTER TABLE .. EXCHANGE PARTITION
- 不支持 SAVEPOINT
- 不支持 Spatial indexes
- 不支持 Transportable tablespace
- 不支持 multi-valued indexes
- 不支持 Fulltext indexes
- 不支持 Group Replication
- 不支持 Partial Update of LOB in InnoDB
- 不支持使用 INPLACE 或者 ONLINE 的方式添加含有 DEFAULT EXPRESS 列；

例如：支持 ALTER TABLE t ADD COLUMN col INT DEFAULT 10; 但不支持 ALTER TABLE t ADD COLUMN col INT DEFAULT (10);

- **实验特性：触发器（trigger）功能在当前版本为实验特性，不建议用户在正式生产环境中使用。**

> **说明:**
> 每个实验特性各自拥有独立开关和默认值。用户如需开启/关闭，请在连接到 TDStore 实例后通过如下 SQL 语句开启（以 tdsql_enable_trigger 为例）：
> `SET PERSIST tdsql_enable_trigger=ON;` 。

| 特性名 | 开关 | 默认值 |
| --- | --- | --- |
| 触发器 | tdsql_enable_trigger | OFF |

## TDStore 引擎与 MySQL 8.0 的使用差异

- TDStore 自增字段暂时只保证全局唯一，不保证全局单调递增。
- add column，add/drop index 支持 online 不阻塞读写；变更 column 类型，变更字符集也支持 online 不锁表。目前会锁表的 DDL 操作主要有：
  - add/drop 主键
  - rename column
  - add/drop/change 自增字段
  - add column 的时候默认值采用了表达式的形式，例如 add column default (a+a)
  - 表是无主键表
  - 表上有关联的触发器

遇到上述不支持 online 的 DDL 系统会有报错提示，用户此时可以设置 set tdsql_use_online_copy_ddl = 0来通过锁表的方式执行这类 DDL。

- set global xxx 只会设置某个节点上的变量，需要用/*# broadcast */ set global xxx 广播到所有节点才能使其在全局生效。
- show variables，show global status 展示的是当前连接的计算节点中的状态信息，连到不同的计算节点展示的信息可能不同。
- show processlist 展示当前连接节点的 processlist，如果需要展示所有节点的，需要用/*# broadcast */show processlist。

## TDStore 实例建表数量的限制

每个 TDStore 实例能创建表的数量有如下限制，此数量限制与构成实例的节点规格及节点数相关，若超出此限制将返回对应错误码（EC_MC_DATA_OBJ_LIMIT_EXCEEDED ）。

| 节点规格（CPU 内 | 基础版实例最大建表数量（单节点） | 集群版实例最大建表数量（3节点为例） |
| --- | --- | --- |

| 存） | | |
| --- | --- | --- |
| 1核2GB | 160 | 480 |
| 2核4GB | 320 | 960 |
| 4核8GB | 640 | 1920 |
| 8核16GB | 1280 | 3840 |
| 16核32GB | 2560 | 7680 |
| 32核64GB | 5120 | 15360 |

# OnlineDDL 说明

最近更新时间：2024-12-24 14:34:12

TDSQL MySQL 版（TDStore 引擎）支持大部分的 Online DDL（包括部分只需要修改元数据的 Inplace DDL，在此也统称为 Online DDL），具体能力说明如下。

## TDStore 引擎版对 Online DDL 的支持

### 二级索引操作

| OP | 语法是否支持 | 是否支持 Online | 不支持 Online 原因 | 是否只修改元数据 | 对用户数据的操作 | 是否支持 DML 并发 |
|---|---|---|---|---|---|---|
| Creating or adding a secondary index | Yes | Yes | / | No | 需要数据回填 | Yes |
| Dropping an index | Yes | Yes | / | No | 需要异步删数据 | Yes |
| Renaming an index | Yes | Yes | / | No | / | Yes |
| Adding a FULLTEXT index | No | No | 引擎暂不支持 | / | / | / |
| Adding a SPATIAL index | No | No | 引擎暂不支持 | / | / | / |
| Changing the index type | No | No | 引擎只支持普通类型，不支持 Btree/Hash 等 | / | / | / |

### 主键操作

涉及主键修改的 DDL 都不支持 Online 方式。

| OP | 语法是否支持 | 是否支持Online | 不支持Online 原因 | 是否只修改元数据 | 对用户数据的操作 | 是否支持DML 并发 |
|---|---|---|---|---|---|---|
| Adding a primary key | Yes | No | 需要重新组织数据 | No | 需要锁表COPY | No |
| Dropping a primary key | Yes | No | 需要重新组织数据 | No | 需要锁表COPY | No |
| Dropping a primary key and adding another | Yes | No | 需要重新组织数据 | No | 需要锁表COPY | No |

## 列操作

| OP | 语法是否支持 | 是否支持Online | 不支持Online 原因 | 是否只修改元数据 | 对用户数据的操作 | 是否支持DML 并发 |
|---|---|---|---|---|---|---|
| Adding a column（尾部追加） | Yes | Yes | / | Yes | | Yes |
| Adding a column(非尾部追加) | Yes | Yes | / | No | 需要Online Copy | Yes |
| Dropping a column | Yes | Yes | / | No | 需要Online Copy | Yes |
| Renaming a column | Yes | Yes | / | Yes | / | Yes |
| Reordering columns | Yes | Yes | / | No | 需要Online Copy | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| Setting a column default value | Yes | Yes | / | Yes | / | Yes |
| Changing the column data type | Yes | Yes | / | No | 需要Online Copy | Yes |
| Extending VARCHAR column size | Yes | Yes | / | Yes | / | Yes |
| Dropping the column default value | Yes | Yes | / | Yes | / | Yes |
| Changing the auto-increment value | Yes | No | / | No | 需要锁表COPY | No |
| Making a column NULL | Yes | Yes | / | No | 需要Online Copy | Yes |
| Making a column NOT NULL | Yes | Yes | / | No | 需要Online Copy | Yes |
| Modifying the definition of an ENUM or SET column | Yes | Yes | / | Yes | / | Yes |

## 表操作

| OP | 语法是否支持 | 是否支持Online | 不支持Online原因 | 是否只修改元数据 | 对用户数据的操作 | 是否支持DML 并发 |
|---|---|---|---|---|---|---|
| Changing the ROW_FORMAT | No | No | 引擎不支持 | / | / | / |

| | | | | | | |
|---|---|---|---|---|---|---|
| Changing the KEY_BLOCK_SIZE | No | No | 引擎不支持 | / | / | / |
| Setting persistent table statistics | No | No | 引擎不支持 | / | / | / |
| Specifying a character set | Yes | Yes | / | No | 需要Online Copy | Yes |
| Converting a character set | Yes | Yes | / | No | 需要Online Copy | Yes |
| Optimizing a table | Yes | / | 使用该语法时直接返回成功，实际未做任何操作，因为TDStore会执行自动compaction，该语法的行为并不适用。 | / | / | / |
| Rebuilding with the FORCE option | Yes | Yes | / | No | 需要Online Copy | Yes |
| Performing a null rebuild | No | No | 引擎不支持 | / | / | / |
| Renaming a table | Yes | Yes | / | Yes | / | Yes |

## 分区操作

| OP | 语法是否支持 | 是否支持Online | 不支持Online 原因 | 是否只修改元数据 | 对用户数据的操作 | 是否支持 DML 并发 |
|---|---|---|---|---|---|---|
| PARTITION BY | Yes | Yes | / | No | 需要Online Copy | Yes |
| ADD PARTITION | Yes | Yes | / | Yes | / | Yes |
| DROP PARTITION | Yes | Yes | / | Yes | / | Yes |
| DISCARD PARTITION | No | No | 引擎不支持 | / | / | / |
| IMPORT PARTITION | No | No | 引擎不支持 | / | / | / |
| TRUNCATE PARTITION | Yes | Yes | / | / | 需要异步删数据 | Yes |
| COALESCE PARTITION | Yes | Yes | / | No | 需要Online Copy | Yes |
| REORGANIZE PARTITION | Yes | Yes | / | No | 需要Online Copy | Yes |
| EXCHANGE PARTITION | No | No | 引擎不支持 | / | / | / |
| ANALYZE PARTITION | No | No | 引擎不支持 | / | / | / |
| CHECK PARTITION | No | No | 引擎不支持 | / | / | / |
| OPTIMIZE PARTITION | Yes | / | 使用该语法时直接返回成功，实际未做任何操 | / | / | / |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | 作，因为 TDStore 会执行自动 compaction，该语法的行为并不适用。 | | | |
| REBUILD PARTITION | Yes | Yes | / | No | 需要 Online Copy | Yes |
| REPAIR PARTITION | Yes | No | 未算作 DDL | / | / | / |
| REMOVE PARTITIONING | Yes | Yes | / | No | 需要 Online Copy | Yes |

### 表空间操作

TDStore 引擎目前暂不支持表空间相关操作。

## Generated Column 操作

TDStore 引擎目前暂不支持 Generated Column 相关操作。

### 外键操作

TDStore 引擎目前暂不支持外键相关操作。

# TDStore 引擎版与社区版 MySQL 支持情况对比

将 TDStore 引擎版的 Inplace 和 Online Copy 以及 社区版 MySQL8.0 的 Instant 和 Inplace 均视为 Online 操作，两者对于常用 DDL 的差异对比说明如下。

| OP | TDStore 引擎版 | MySQL |
|---|---|---|
| Converting a character set | 支持 DML 并发 | 不支持 DML 并发 |
| Changing the column data type | 支持 DML 并发 | 不支持 DML 并发 |
| PARTITION BY | 支持 DML 并发 | 不支持 DML 并发 |

| | | |
|---|---|---|
| Extending VARCHAR column size | 支持 DML 并发 | 部分情况下支持，对 VARCHAR(X) 中的 X 有限制 |
| Adding a primary key | 不支持 DML 并发 | 支持 DML 并发 |
| Dropping a primary key and adding another | 不支持 DML 并发 | 支持 DML 并发 |

**说明：**

- TDStore 引擎版目前尚有少量 DDL 没有支持，因此此处只说明了对比 TDStore 引擎版和社区版 MySQL8.0均支持的部分的差异；
- 对于 Extending VARCHAR column size 类型，社区版 MySQL8.0会对 VARCHAR(X) 指定的字符长度做出限制，主要涉及存储的 pack length。假设以UTF8MB4作为字符集，某字段所占的字节数为 X*4。

进行 DDL 时，假设当前 VARCHAR 字段 X*4<256，且修改后的字段 Y*4<256；则支持 Inplace；
进行 DDL 时，假设当前 VARCHAR 字段 X*4<256，但修改后的字段 Y*4>=256；则不支持 Inplace；
示例：

```
CREATE TABLE t1 ( a INT PRIMARY KEY , b VARCHAR ( 10 ) )
ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci ;
ALTER TABLE t1 CHANGE COLUMN b b VARCHAR ( 20 ) , ALGORITHM = INPLACE ,
LOCK = NONE ; → 成功
ALTER TABLE t1 CHANGE COLUMN b b VARCHAR ( 64 ) , ALGORITHM = INPLACE ,
LOCK = NONE ; → 失败
```

# 列存兼容性说明

# SQL 语法兼容

最近更新时间：2025-06-13 20:31:03

> **说明：**
> TDStore 列存功能在当前版本为实验特性，仅面向受邀用户开放。若您有需要可通过客户经理与我们联系并申请。

## SQL 语法兼容说明

TDStore 列存向量引擎支持大部分标准常用 SQL 查询，支持全部 TPC-H 查询。但是又有自己特性，在某些场景下会由于不支持，导致 Fallback 到 TDStore行存引擎执行。具体限制如下：

### 用户自定义的变量

向量引擎不支持用户定义的变量，当查询 SQL 中出现用户自定义变量时（ E.G: `SELECT @myvar + col FROM table` ）时，该查询不通过向量引擎执行。

```
SET @cnt = 1;

SELECT *
FROM t1 FORCE INDEX (date_col)
WHERE
    col1 + 10000000 > (@cnt := @cnt + 1)
    AND date_col >= DATE('2014-04-01')
    AND date_col < DATE('2014-04-01' + INTERVAL 1 DAY)
ORDER BY date_col DESC;

SELECT @cnt;
```

## SQL_CALC_FOUND_ROWS / FOUND_ROWS()

向量引擎暂时不支持 `SQL_CALC_FOUND_ROWS` 相关功能，带有下属关键字的 SQL 会 fallback 到 TDSQL TP 引擎执行。

## WITH RECURSIVE cte

向量引擎暂时不支持 Recursive Common Table Expressions 功能，会 fallback 到 TDSQL TP 引擎执行。

## UNION/UNION ALL + ORDER BY

对于 UNION/UNION ALL + ORDER BY 查询，向量引擎支持的 `ORDER BY` 字段，需要为 `UNION` 内子查询投影的字段。其他将 fallback 到 TDSQL TP 引擎执行。

```
--Suppoted--
SELECT a n FROM t1 UNION SELECT b n FROM t2 ORDER BY n;
--Unsuppoted--
SELECT a n FROM t1 UNION SELECT b n FROM t2 ORDER BY n * 2;
```

## 字符集和 Collation

向量引擎支持有限字符集和 Collation，其他会 fallback 到 TDSQL TP 引擎执行。

- 支持的字符集："utf8", "utf8mb3", "utf8mb4", "binary"。
- 不支持的 Collation："utf8_german2_ci", "utf8mb4_czech_ci", "latin1_german2_ci", "latin1_bin", "utf8mb4_lithuanian_ci", "utf8_lithuanian_ci", "utf8_czech_ci", "utf8mb4_polish_ci"。

  表级限定，即只要表定义中有字段不属于上述字符集，即 fallback。
- 字符集兼容性：向量引擎对于字符串类型比较时，不区分大小写，区分 accent（即：case insensitive and accent sensitive）。这样对于有语调的字符串会出现与 MySQL 的 TP 结果不一致情形。举例如下：

```
tdsql [db1]> create table t1(c1 varchar(32), c2 varchar(32));
Query OK, 0 rows affected (0.70 sec)

tdsql[db1]> show create table t1;
+-------+-------------------------------------------------------------
-----------------------------------------------------------------+
| Table | Create Table
|
+-------+-------------------------------------------------------------
-----------------------------------------------------------------+
| t1    | CREATE TABLE `t1` (
  `c1` varchar(32) DEFAULT NULL,
  `c2` varchar(32) DEFAULT NULL
) ENGINE=ROCKSDB DEFAULT CHARSET=utf8mb3 |
+-------+-------------------------------------------------------------
-----------------------------------------------------------------+
1 row in set (0.00 sec)

tdsql[db1]> insert into t1 values('hello','hElLÖ');
Query OK, 1 row affected (0.02 sec)
```

```
tdsql[db1]> select * from t1;
+-------+--------+
| c1    | c2     |
+-------+--------+
| hello | hElLÖ  |
+-------+--------+
1 row in set (0.00 sec)

tdsql[db1]> select c1=c2 from t1;
+-------+
| c1=c2 |
+-------+
|     1 |
+-------+
1 row in set (0.00 sec)

tdsql[db1]> set htap_routing_strategy='vector_engine';
Query OK, 0 rows affected (0.00 sec)

tdsql [db1]> set htap_query_passthrough_mode=off;
Query OK, 0 rows affected (0.00 sec)

tdsql[db1]> set htap_print_deparsed_query=on;
Query OK, 0 rows affected (0.00 sec)

tdsql[db1]> select c1=c2 from t1;
+-------+
| c1=c2 |
+-------+
|     0 |
+-------+
1 row in set, 1 warning (0.01 sec)

tdsql[db1]> show warnings;
+-------+------+-------------------------------------------------------
-----------------------------------------------+
| Level | Code | Message
|
+-------+------+-------------------------------------------------------
-----------------------------------------------+
| Note  | 1003 | Deparsed query is : select  ("t1"."c1" = "t1"."c2"
collate nocase ) AS "c1=c2" from "db1"."t1" "t1" |
```

```
+-------+------+-----------------------------------------------------------
-------------------------------------------+
1 row in set (0.00 sec)
```

## 向量引擎相关参数

| 序号 | 变量名称 | 类型 | 说明 |
| --- | --- | --- | --- |
| 1 | htap_avoid_some_mysql_optimizations | 布尔 | 是否进行优化器干预，来优化向量查询。缺省（on） |
| 2 | htap_enable_deparser_warnings | 布尔 | 对于显示针对 VE 解析的报警。缺省（off） |
| 3 | htap_routing_strategy | 字符串 | 查询的策略，有如下可能：<br>'VECTOR_ENGINE'：缺省走向量引擎。<br>'DEFAULT'：缺省都 TP 引擎。 |
| 4 | htap_query_passthrough_mode | 布尔 | 是否进行向量引擎语义解析，为 on 时将 SQL 直接传给 VE 引擎，并取得结果。缺省（off） |
| 5 | htap_print_deparsed_query | 布尔 | 是否将向量解析引擎输出到日志文件。缺省（off） |
| 6 | htap_enable_ve_execution_fallback | 布尔 | 当向量引擎报错时，是否需要 fallback 到 TP 执行或者直接报错。缺省（on） |

# 表达式兼容

最近更新时间：2025-06-13 20:31:03

> **说明：**
> TDStore 列存功能在当前版本为实验特性，仅面向受邀用户开放。若您有需要可通过客户经理与我们联系并申请。

## 数据类型

| 类型 | MySQL 数据库 | TDSQL TDStore 版 | 说明 |
| --- | --- | --- | --- |
| 数字类型 | Tinyint | 支持 | 可以叠加 unsigned 属性 |
| | Smallint | 支持 | |
| | Mediumint | 支持 | |
| | Integer | 支持 | |
| | Bigint | 支持 | |
| | Float | 支持，有差异 | 浮点数，会使用科学计数法表示。 例如 99999999999999999999999999999999999999 这种大数在MySQL 中会用 `1e38` 表示， 经过 `concat` 还是 `1e38` 。<br>向量引擎计算表示方式是 `1e+38` ， 显示增加了 + 符号，格式略微不同。 |
| | Double | 支持，有差异 | TDSQL TP 引擎对于 string -> double 的转换比较灵活，即使string 中有非数字字符，也会进行转换，例如' ' → double，TDSQL TP 引擎将空字符变成0.0，'1x' → double，结果集是1.0。<br>向量引擎无法做这种非法值的转换，如遇到，可以打开 `htap_enable_ve_execution_fallback` 开关，回退到 TP引擎计算，或者修正非法值的转换，继续利用向量引擎高效计算。 |
| | Decimal | 支持，有差异 | 值域不同 |

| | | | |
|---|---|---|---|
| | | | • TDSQL TP 引擎最大支持 `Decimal(65,30)` 。<br>• 向量引擎最大支持 `Decimal(38)` 。 |
| 时间类型 | Year | 支持 | – |
| | Date | 支持，有差异 | • 向量引擎支持 `yyyy-mm-dd` ，不支持 `yyyy.mm.dd` 、 `yyyy#mm#dd` 的表示方法。<br>• 特殊日期 '0000-00-00'。向量引擎对日期类型的表示方式和原始 TDSQL TP 引擎不相同。'0000-00-00' 的这种无法合理表示的特殊时间暂时没有兼容，目前通过0001-00-00 (BC) 代替，和 TDSQL TP 引擎不一致。 |
| | Time | 支持，有差异 | TDSQL TP 引擎的 Time 时间类型区间是 [-838:59:59，-838:59:59]。例如‘-838:59:59'，这种非法时间是允许写入的，向量引擎要求时间在0 – 24之内。如果数据是负数非法时间，会转成合法时间，这种行为和 TDSQL TP 引擎不一致. |
| | Timestamp | 支持，有差异 | |
| | Datetime | 支持 | – |
| 字符类型 | Char | 支持 | • 字符集向量引擎只支持 UTF-8 字符集，不支持 GBK 等其他字符集。<br>• UTF8MB4下的特殊字符兼容字符集下的比较，暂时不支持重音比较，例如'hëllo' != 'hello' |
| | Varchar | 支持 | |
| 布尔类型 | Bool | 支持 | – |

## 表达式函数

| 类型 | MySQL 数据库 | TDStore | 说明 |
|---|---|---|---|
| 比较函数 | 等于(=) | 支持 | – |
| | 大于(>) | 支持 | – |
| | 大于等于(>=) | 支持 | – |

| | | | |
|---|---|---|---|
| | 小于(<) | 支持 | – |
| | 小于等于(<=) | 支持 | – |
| | IN | 支持 | – |
| | BETWEEN | 支持 | – |
| | IS_NULL | 支持 | – |
| | IS_NOTNULL | 支持 | – |
| 逻辑比较 | AND | 支持 | – |
| | OR | 支持 | – |
| | NOT(!) | 支持 | – |
| 分支比较 | IF | 支持 | – |
| | NULLIF | 支持 | – |
| | IFNULL | 支持 | – |
| | CASE WHEN | 支持 | – |
| 字符函数 | COALESCE | 支持 | – |
| | LIKE | 支持 | – |
| | TRUNCATE | 支持 | – |
| | CONCAT | 支持，有差异 | CONCAT(a,b)函数拼接两个字符串，如果任意一方有 NULL，TDSQL TP 引擎的最终结果是 NULL，向量引擎计算结果可以是非 NULL 值。 |
| | SUBSTR | 支持 | – |
| | INSTR | 支持，有差异 | `INSTR(a, b)` 函数用于查找字符串b第一次在a中出现的位置。向量引擎暂时不支持像 TDSQL TP 引擎一样在 Collation 下进行比较。因此无法识别大小写，A 和 a 会被认为不等。 |

| | | | |
|---|---|---|---|
| | TRIM | 支持 | – |
| | LTRIM | 支持 | – |
| | RTRIM | 支持 | – |
| | REGEXP_LIKE | 支持 | – |
| | REGEXP_REPLACE | 支持 | – |
| | CONCAT_WS | 支持 | – |
| | REPLACE | 支持 | – |
| | LOWER | 支持 | – |
| | UPPPER | 支持 | – |
| 转换函数 | CAST | 支持 | – |
| 计算函数 | 加法(+) | 支持 | – |
| | 减法(–) | 支持 | – |
| | 乘法(*) | 支持 | – |
| | 除法(/) | 支持 | – |
| | ROUND | 支持 | – |
| | CEILING | 支持 | – |
| | FLOOR | 支持 | – |
| 时间函数 | EXTRACT | 支持 | – |
| | DATEADD | 支持 | – |
| 聚合函数 | MIN | 支持 | – |
| | MAX | 支持 | – |
| | SUM | 支持 | – |
| | COUNT | 支持 | – |

| | | | |
|---|---|---|---|
| | AVG | 支持 | - |
| | DISTINCT | 支持 | - |

# Stale Read

最近更新时间：2025-06-13 20:31:03

Stale Read（陈旧读）是数据库系统中一种读取历史数据版本的机制，用户可以指定一个具体的历史时间点或者距离当前时间的一个具体时间值来读取历史数据。Stale Read 可提升系统资源利用率，在牺牲一定数据实时性的前提下可降低读取数据的延迟。
TDSQL 提供了语句级别、事务级别、会话级别三种级别的 Stale Read 功能，并且允许用户依据业务情况选择读取数据的副本角色。

## 数据准备

1. 准备表 `t1` ，并完成数据准备。

```
desc t1;
+-------+------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------+------+------+-----+---------+-------+
| id    | int  | NO   | PRI | NULL    |       |
| num   | int  | NO   |     | NULL    |       |
+-------+------+------+-----+---------+-------+

--并且预置下列数据：
select * from t1;
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
+----+-----+
```

2. 将 Stale Read 的读取节点设置成 Follower 节点。
   - `0` ：表示 Leader 节点。
   - `1` ：表示 Follower 节点。

```
set persist tdsql_stale_read_role=1;
```

## 语法级别

提供 `AS OF  TIMESTAMP <datetime>` 语法支持语句级的查询。

```
--查询当前数据
select * from t1;
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
+----+-----+

--查询当前时间
select now();
+---------------------+
| now()               |
+---------------------+
| 2025-02-16 17:44:26 |
+---------------------+

--插入新的数据并查询
insert into t1 values(4,40);
select * from t1;
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
|  4 |  40 |
+----+-----+

--查询历史数据
select * from t1 as of timestamp '2025-02-16 17:44:26';
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
+----+-----+
```

## 会话级别及事务级别

提供 Session 变量 `tdsql_read_staleness_t` ，支持会话级别和事务级别的查询。

```
--查询当前数据
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
|  4 |  40 |
+----+-----+

set tdsql_read_staleness_t='-10'; #读取10秒前的数据

--插入新的数据
insert into t1 values (5,50);

--查看当前时间
select now();
+---------------------+
| now()               |
+---------------------+
| 2025-02-16 17:57:50 |
+---------------------+

--查询当前数据
select * from t1; # 返回历史数据
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
|  4 |  40 |
+----+-----+

--等待一段时间
--查看当前时间
select now();
+---------------------+
| now()               |
+---------------------+
| 2025-02-16 17:58:01 |
```

```
+---------------------+

--再次查询当前数据
select * from t1; # 返回新数据
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
|  4 |  40 |
|  5 |  50 |
+----+-----+
```

设置 `tdsql_read_staleness_t` 后，还可以开始事务查询数据。

```
--查询当前数据
select * from t1;
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
|  4 |  40 |
|  5 |  50 |
+----+-----+

--插入新的数据
insert into t1 values (6,60);

--开启事务
begin;
select * from t1; # 返回历史数据
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
|  4 |  40 |
|  5 |  50 |
+----+-----+
```

```
select sum(num) as total_num from t1;
+-----------+
| total_num |
+-----------+
|       150 |
+-----------+
rollback;

--等待一段时间
--开启事务
begin;
select * from t1; # 返回新数据
+----+-----+
| id | num |
+----+-----+
|  1 |  10 |
|  2 |  20 |
|  3 |  30 |
|  4 |  40 |
|  5 |  50 |
|  6 |  60 |
+----+-----+
select sum(num) as total_num from t1;
+-----------+
| total_num |
+-----------+
|       210 |
+-----------+
rollback;
```

## 注意事项

当前如果查询的历史数据跨越了 DDL，则查询报错。

```
--先为表t1增加一个新列，并查询数据
alter table t1 add column age int not null;
select * from t1;
+----+-----+-----+
| id | num | age |
+----+-----+-----+
|  1 |  10 |   0 |
|  2 |  20 |   0 |
|  3 |  30 |   0 |
```

```
|   4 |   40 |    0 |
|   5 |   50 |    0 |
|   6 |   60 |    0 |
+----+-----+-----+

--查询当前时间
select now();
+--------------------+
| now()              |
+--------------------+
| 2025-02-16 18:10:08 |
+--------------------+

--移除新加的列，并查询数据
alter table t1 drop column age;
select * from t1;
+----+-----+
| id | num |
+----+-----+
|   1 |   10 |
|   2 |   20 |
|   3 |   30 |
|   4 |   40 |
|   5 |   50 |
|   6 |   60 |
+----+-----+

--查询历史数据
select * from t1 as of timestamp '2025-02-16 18:10:08'; # 执行DDL之前的时间

--返回错误信息
Table or index did not exist at the transaction's timestamp. May be
caused by Stale Read or DDL.
```

# SQL 参考
# CREATE DATABASE

最近更新时间：2025-04-27 10:22:53

## 功能描述

该语句用于创建数据库，并可以指定数据库的默认属性（如数据库默认字符集，校验规则等）。

## 权限要求

执行 `CREATE DATABASE` 语句创建数据库需要当前用户拥有全局的 `CREATE` 权限。

## 语法

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] database_name
    [database_option] ...

database_option:
[DEFAULT] {CHARACTER SET | CHARSET} [=] charset_name
| [DEFAULT] COLLATE [=] collate_name
```

## 参数说明

| 参数 | 是否可选 | 描述 |
| --- | --- | --- |
| IF NOT EXISTS | 可选 | 用于指示如果数据库已经存在，则不进行创建。创建数据库时，如果数据库存在且没有指定 IF NOT EXISTS，则会报错。 |
| database_name | 必选 | 指定待创建的数据库名称。 |
| [DEFAULT] {CHARACTER SET \| CHARSET} [=] charset_name | 可选 | 设置数据库的字符集（charset）。 |
| [DEFAULT] COLLATE [=] collate_name | 可选 | 设置数据库的校对规则（collation）。 |

## 示例

```
tdsql [(none)]> CREATE DATABASE IF NOT EXISTS test1 DEFAULT CHARACTER
SET utf8;
Query OK, 1 row affected
```

# CREATE TABLE

最近更新时间：2025-04-27 10:22:53

## 功能描述

创建表。

## 语法

```
CREATE TABLE [IF NOT EXISTS] table_name
      (table_definition_list) [table_option_list] [partition_clause];

table_definition_list:
    table_definition [, table_definition ...]

table_definition:
      column_definition_list
    | [CONSTRAINT [constraint_name]] PRIMARY KEY index_desc
    | [CONSTRAINT [constraint_name]] UNIQUE {INDEX | KEY}
            [index_name] index_desc
    | [CONSTRAINT [constraint_name]] CHECK(expression) constranit_state

column_definition_list:
    column_definition [, column_definition ...]

column_definition:
     column_name data_type
         [DEFAULT const_value] [AUTO_INCREMENT]
         [NULL | NOT NULL] [[PRIMARY] KEY] [UNIQUE [KEY]] [COMMENT
string_value]

index_desc:
   (column_desc_list) [index_option_list]

index_option_list:
    index_option [ index_option ...]

index_option:
      KEY_BLOCK_SIZE [=] INT_VALUE
    | COMMENT string_value

table_option_list:
```

```
    table_option [ table_option ...]

table_option:
      [DEFAULT] {CHARSET | CHARACTER SET} [=] charset_name
    | [DEFAULT] COLLATE [=] collation_name
    | COMMENT string_value
    | ROW_FORMAT [=] REDUNDANT|COMPACT|DYNAMIC|COMPRESSED|DEFAULT

partition_option:
      PARTITION BY HASH(expression)
      [subpartition_option] PARTITIONS partition_count
    | PARTITION BY KEY([column_name_list])
      [subpartition_option] PARTITIONS partition_count
    | PARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
      [subpartition_option] (range_partition_list)
    | PARTITION BY LIST {(expression) | COLUMNS (column_name_list)}
      [subpartition_option] PARTITIONS partition_count
    | PARTITION BY RANGE [COLUMNS]([column_name_list])
(range_partition_list)

subpartition_option:
      SUBPARTITION BY HASH(expression)
      SUBPARTITIONS subpartition_count
    | SUBPARTITION BY KEY(column_name_list)
      SUBPARTITIONS subpartition_count
    | SUBPARTITION BY RANGE {(expression) | COLUMNS (column_name_list)}
      (range_subpartition_list)
    | SUBPARTITION BY LIST(expression)

range_partition_list:
    range_partition [, range_partition ...]

range_partition:
    PARTITION partition_name
    VALUES LESS THAN {(expression_list) | MAXVALUE}

range_subpartition_list:
    range_subpartition [, range_subpartition ...]

range_subpartition:
    SUBPARTITION subpartition_name
    VALUES LESS THAN {(expression_list) | MAXVALUE}
```

```
expression_list:
    expression [, expression ...]

column_name_list:
    column_name [, column_name ...]

partition_name_list:
    partition_name [, partition_name ...]

partition_count | subpartition_count:
    INT_VALUE
```

## 参数说明

| 参数 | 是否可选 | 描述 |
| --- | --- | --- |
| IF NOT EXISTS | 可选 | 如果指定 `IF NOT EXISTS` ，即使待创建的表已存在，也不会报错；如果不指定且待创建的表已存在，则系统会报错。 |
| PRIMARY KEY | 可选 | 为创建的表指定主键。如果不指定，则使用隐藏主键。 |
| KEY \| INDEX | 可选 | 为创建的表指定键或索引。如果不指定索引名，则会使用索引引用的第一列作为索引名，如果命名存在重复，则会使用下划线（_）+ 序号的方式命名。（例如，使用 `c1` 列创建的索引如果命名重复，则会将索引命名为 `c1_2` 。）您可以通过 `SHOW INDEX` 语句查看表上的索引。 |
| ROW_FORMAT | 可选 | 指定表是否开启 Encoding 存储格式。<br>• redundant：不开启 Encoding 存储格式。<br>• compact：不开启 Encoding 存储格式。<br>• dynamic：Encoding 存储格式。<br>• compressed：Encoding 存储格式。<br>• default：等价 dynamic 模式。 |
| KEY_BLOCK_SIZE | 可选 | 指定索引块的大小（以字节为单位）。 |
| CHARSET \| CHARACTER SET | 可选 | 指定表中列的默认字符集。 |
| COLLATE | 可选 | 指定表中列的默认字符序。 |
| COMMENT | 可选 | 注释。不区分大小写。 |

版权所有：腾讯云计算（北京）有限责任公司

| | | |
|---|---|---|
| CHECK | 可选 | 限制列中的值的范围。<br>• 如果对单个列定义 `CHECK` 约束，那么该列级约束可以写到列定义中，并且可以指定名称。<br>• 如果对一个表定义 `CHECK` 约束，那么此约束会应用于表中多个列，且允许出现在列的定义前。在删除表时，表中创建的 `CHECK` 约束也会一起被删除。<br>可以通过如下方式查看约束信息：<br>• 使用 `SHOW CREATE TABLE` 命令。<br>• 查看 `information_schema.TABLE_CONSTRAINTS` 视图。<br>• 查看 `information_schema.CHECK_CONSTRAINTS` 视图。 |
| constraint_name | 可选 | 约束名称，最多包含64个字符。<br>• 约束名称的开头结尾中间都允许有空格，但需要用"`"标识名称的开头和结尾。<br>• 约束名称可以包含特殊字符"$"。<br>• 如果约束名称为保留字，需要用"`"标识，否则会报错。<br>• 在同一 Database 下不允许 `CHECK` 约束的名称重复。 |
| expression | 可选 | 约束表达式。<br>• `expression` 不允许为空。<br>• `expression` 结果不能为非布尔类型。<br>• `expression` 不能包含不存在的列。 |

## 示例

- 创建数据库表。

```
tdsql [test]> CREATE TABLE tbl1 (c1 INT PRIMARY KEY, c2 VARCHAR(50));
Query OK, 0 rows affected
```

- 创建带索引的表。

```
tdsql [test]> CREATE TABLE tbl2 (c1 INT PRIMARY KEY, c2 INT, c3 INT,
INDEX i1 (c2));
Query OK, 0 rows affected
```

- 创建 Hash 分区，分区数为 `8` 的表。

```
tdsql [test]> CREATE TABLE tbl3 (c1 INT PRIMARY KEY, c2 INT) PARTITION
BY HASH(c1) PARTITIONS 8;
Query OK, 0 rows affected
```

- 创建一级分区为 Range 分区，二级分区为 Key 分区的表。

```
tdsql [test]> CREATE TABLE tbl4 (c1 INT, c2 INT, c3 INT) PARTITION BY
RANGE(c1)
            SUBPARTITION BY KEY(c2, c3) SUBPARTITIONS 5
            (PARTITION p0 VALUES LESS THAN(0), PARTITION p1 VALUES
LESS THAN(100));

Query OK, 0 rows affected
```

- 创建一列为 `gbk`，一列为 `utf8` 的表。

```
tdsql [test]> CREATE TABLE tbl5 (
           c1 VARCHAR(10),
           c2 VARCHAR(10) CHARSET GBK COLLATE gbk_bin
       ) DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
Query OK, 0 rows affected
```

- 使用自增列作为分区键。

```
tdsql [test]> CREATE TABLE tbl6(inv_id BIGINT NOT NULL
AUTO_INCREMENT,c1 BIGINT,
       PRIMARY KEY (inv_id) ) PARTITION BY HASH(inv_id) PARTITIONS 8;
Query OK, 0 rows affected
```

- 创建带 `CHECK` 约束的表，并查看约束信息。

```
tdsql [test]> CREATE TABLE tbl7 (col1 INT, col2 INT, col3 INT,
CONSTRAINT equal_check1 CHECK(col1 = col3 * 2));
Query OK, 0 rows affected

tdsql [test]> SHOW CREATE TABLE tbl7;
+-------+-------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
--------------------------------+
```

```
| Table | Create Table
|
+-------+-------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
--------------------------------+
| tbl10 | CREATE TABLE `tbl7` (
  `col1` int DEFAULT NULL,
  `col2` int DEFAULT NULL,
  `col3` int DEFAULT NULL,
  CONSTRAINT `equal_check1` CHECK ((`col1` = (`col3` * 2)))
) ENGINE=ROCKSDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
--------------------------------+
1 row in set
```

# DROP DATABASE

最近更新时间：2025-04-27 10:22:53

## 功能描述

删除数据库。

## 语法

```
drop_database_stmt:
DROP DATABASE [IF EXISTS] database_name;
```

## 参数说明

| 参数 | 是否可选 | 描述 |
| --- | --- | --- |
| IF EXISTS | 可选 | 用于防止当数据库不存在时发生错误。 |
| database_name | 必选 | 指定待删除的数据库名。 |

## 示例

```
tdsql [(none)]> DROP DATABASE test1;
Query OK, 0 rows affected
```

# DROP TABLE

最近更新时间：2025-04-22 15:40:02

## 功能描述

删除表。

## 权限要求

具有该表 `Drop` 权限。

## 语法

```
DROP TABLE table_name [PURGE];
```

## 参数说明

| 参数 | 是否必选 | 说明 |
| --- | --- | --- |
| table_name | 必选 | 表名。 |
| PURGE | 可选 | 直接删除表数据，不再转存入回收站库。 |

## 示例

```
TDSQL [(none)]> drop table test2 purge;
Query OK, 0 rows affected (0.70 sec)
```

# FLASHBACK TABLE

最近更新时间：2025-04-22 15:40:02

## 功能描述

按照指定的方式恢复表。如果有同名表，会报错。

## 权限要求

具有该表 `Drop` 权限。

## 语法

```
FLASHBACK TABLE recycle_table_name TO BEFORE DROP [RENAME TO
new_table_name];
```

## 参数说明

| 参数 | 是否必选 | 说明 |
| --- | --- | --- |
| recycle_table_name | 必选 | 表进入回收站后的别名，可通过 SHOW RECYCLEBIN 的 `recycle_table` 字段获取。 |
| new_table_name | 可选 | 表恢复时的新命名。 |

## 示例

```
TDSQL [(none)]> show recyclebin;
+-----------+---------------+--------------+---------------------------
----+--------------------+--------------------+
| tindex_id | origin_schema | origin_table | recycle_table
| drop_time           | purge_time          |
+-----------+---------------+--------------+---------------------------
----+--------------------+--------------------+
|     10028 | test          | test2        |
bin$/usodaiwngrpdoel2wnlqg==$1 | 2025-01-21 19:12:10 | 2025-01-28
19:12:10 |
+-----------+---------------+--------------+---------------------------
----+--------------------+--------------------+

TDSQL [(none)]> flashback table `bin$/usodaiwngrpdoel2wnlqg==$1` to
before drop rename to test.test3;
```

```
Query OK, 0 rows affected (1.72 sec)

TDSQL [(none)]> show tables;
+----------------+
| Tables_in_test |
+----------------+
| test1          |
| test3          |
+----------------+
2 rows in set (0.01 sec)
```

# OPTIMIZE TABLE

最近更新时间：2025-06-13 20:31:03

## 功能描述

该语句对指定表的数据范围执行 compaction。如果该语句在列存节点上执行，则触发的 compaction 会将指定表的数据转储至列存存储引擎。
该语句的执行不锁表，不阻塞读写请求。

## 权限要求

`SELECT` 和 `INSERT` 权限。

## 语法

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
```

## 参数说明

| 参数 | 是否必选 | 说明 |
| --- | --- | --- |
| NO_WRITE_TO_BINLOG | 可选 | 保留关键字，无实际功能。 |
| LOCAL | 可选 | 保留关键字，无实际功能。 |

## 示例

```
tdsql> optimize table t1;
+---------+----------+----------+----------+
| Table   | Op       | Msg_type | Msg_text |
+---------+----------+----------+----------+
| test.t1 | optimize | status   | OK       |
+---------+----------+----------+----------+
1 row in set (0.01 sec)
```

# PURGE RECYCLEBIN

最近更新时间：2025-04-22 15:40:02

## 功能描述

清理回收站内的某张表，需提供站内表名。
如果不指定站内表名，则表示清空整个回收站。

## 权限要求

具有该表 `Drop` 权限。

## 语法

```
PURGE RECYCLEBIN [recyele_table_name];
```

## 参数说明

| 参数 | 是否必选 | 说明 |
| --- | --- | --- |
| recyele_table_name | 可选 | • 指定要删除的表名。表名为表进入回收站后的别名，可通过 SHOW RECYCLEBIN 的 `recycle_table` 字段获取。<br>• 如果不指定站内表名，则表示清空整个回收站。 |

## 示例

```
TDSQL [(none)]> show recyclebin;
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
| tindex_id | origin_schema | origin_table | recycle_table
| drop_time           | purge_time          |
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
|     10028 | test          | test1        |
bin$tchgyzsnht3y0xtxlbitnq==$1 | 2025-01-21 19:12:10 | 2025-01-28
19:12:10 |
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+

TDSQL[(none)]> purge recyclebin `bin$tchgyzsnht3y0xtxlbitnq==$1`;
```

```
Query OK, 0 rows affected (0.69 sec)
```

# SHOW RECYCLEBIN

最近更新时间：2025-04-22 15:40:02

## 功能描述

`SHOW RECYCLEBIN` 显示回收站内表信息。

## 语法

```
SHOW RECYCLEBIN;
```

## 示例

```
TDSQL [(none)]> show recyclebin;
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
| tindex_id | origin_schema | origin_table | recycle_table
| drop_time           | purge_time          |
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
|     10028 | test          | test2        |
bin$/usodaiwngrpdoel2wnlqg==$1 | 2025-01-21 19:12:10 | 2025-01-28
19:12:10 |
+-----------+---------------+--------------+---------------------------
----+---------------------+---------------------+
```

**以上返回结果中的字段描述如下：**

- `tindex_id`：表唯一标识。
- `origin_schema`：原库名。
- `origin_table`：原表名。
- `recycle_table`：进入回收站后的别名。
- `drop_time`：删除时间。
- `purge_time`：清除时间（从回收站内清除）。

# 列存向量化引擎 Join Order Hint

最近更新时间：2025-06-13 20:31:03

> **说明：**
> TDStore 列存功能在当前版本为实验特性，仅面向受邀用户开放。若您有需要可通过客户经理与我们联系并申请。

## 功能描述

使用 Join Order Hint 指定列存向量化引擎计划片段生成。在选择向量化引擎作为执行器时，在查询的 SQL 语句中添加 MySQL 风格的 Join Order Hint 即可传递给向量化引擎。目前支持三种 Join Order 相关的 Hint：`join_order`，`join_prefix`，`join_suffix`。

| 类型 | 作用范围 | 位置约束 | 适用场景 |
| --- | --- | --- | --- |
| join_order | 全表 join 顺序片段 | 计划树中间位置 | 明确多表关联路径 |
| join_prefix | 前缀表强制左深树顺序 | 计划树最左端 | 优先处理维度表 |
| join_suffix | 后缀表强制右深树顺序 | 计划树最右端 | 延迟处理大表关联 |

## join_order

join_order Hint 类型尝试将 Hint 内指定的几个表 join 顺序固定为一个左深树顺序片段。未指定的表根据默认优化器逻辑和这个片段连接。

- 示例1

```
SELECT /*+join_order(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a;
```

执行计划解析：

```
| physical_plan
 -> Projection: a a (rows=*)
   -> Inner Hash Join (a = a) (rows=*)
     -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
     -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
```

全部表的 join 顺序被指定。Hash Join 情况下，`t2` 是 build side，`t1` 是 probe side，这和 MySQL 计划一致。

- 示例2

```
SELECT /*+join_order(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a JOIN t3
ON t2.a=t3.a;
```

**执行计划解析：**

```
 -> Inner Hash Join (a = a) (rows=*)
   -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
   -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
```

**上面的计划被作为一个计划片段，`t3`表在计划中所处的位置不确定，由优化器决定。**

## join_prefix

**除了具有 join_order 的作用，同时被指定计划片段会出现在整个计划树的最左侧（显示的计划的底部）。**

- **语法示例**

```
SELECT /*+join_prefix(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a JOIN t3
ON t2.a=t3.a;
```

- **执行计划解析：**

```
| physical_plan
 -> Projection: a a a (rows=*)
   -> Inner Hash Join (a = a) (rows=*)
     -> Tdstore Scan Text: test.t3 Projections: a (rows=*)
     -> Inner Hash Join (a = a) (rows=*)
       -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
       -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
```

`t1`和`t2`的 join 顺序被指定，同时被固定在计划最左侧。

## join_suffix

**除了具有 join_order 的作用，同时被指定计划片段会出现在整个计划树的最右侧（显示的计划的顶部）。**

- **语法示例**

```
SELECT /*+join_suffix(t2,t1)*/ * FROM t1 JOIN t2 ON t1.a=t2.a JOIN t3
ON t2.a=t3.a;
```

- **执行计划解析：**

```
| physical_plan
 -> Projection: a a a (rows=*)
   -> Inner Hash Join (a = a) (rows=*)
     -> Tdstore Scan Text: test.t1 Projections: a (rows=*)
     -> Inner Hash Join (a = a) (rows=*)
       -> Tdstore Scan Text: test.t2 Projections: a (rows=*)
       -> Tdstore Scan Text: test.t3 Projections: a (rows=*)
```

`t1` 和 `t2` 的 join 顺序被指定，同时被固定在计划最右侧。

## 注意事项

- Hint 中的表的大小写不区分，且只能指定表名，而非 query block 的名字。
- 在同一个 Hint 中可以有1个或多个表，其中对 join_order 当指定超过2个表时有效。可以同时有多个 Hint。比如下面的所有 Hint 均生效：

```
SELECT /*+join_prefix(t2,t1) join_order(t3,t5,t4)*/ * FROM  t1 JOIN t2
ON t1.a=t2.a
    JOIN t3 ON  t2.a=t3.a
    JOIN t4 ON  t3.a=t4.a
    JOIN t5 ON t4.a=t5.a;
```

- 每个 SQL 查询中至多允许一个 join_prefix 或一个 join_suffix Hint。若存在多个同类型 Hint，仅第一个生效，剩下的会被忽略。
- 当 Hint 之间有冲突时，后指定的 Hint 会被忽略。比如下面第二个 Hint 不会生效：

```
SELECT /*+join_order(t2,t1) join_order(t1,t2)*/ * FROM t1 JOIN t2 ON
t1.a=t2.a;
```

- 如果两表之间没有 join 条件，这两个表若在 Join Order Hint 中毗邻，则该 Hint 会被忽略。比如下面的 Hint 无效：

```
SELECT /*+join_order(t2,t1)*/ * FROM t1,t2;
```

# PL 系统包 DBMS_ADMIN TDSTORE_FORCE_FULL_COMPACTION

最近更新时间：2025-06-13 20:31:03

`DBMS_ADMIN.TDSTORE_FORCE_FULL_COMPACTION()` 用于触发一次全量 compaction，即将所有数据 compact 到 l sm tree 的最下层，在列存节点执行还会将所有符合要求的数据都下沉到列存文件中。

## 语法

```
CALL DBMS_ADMIN.TDSTORE_FORCE_FULL_COMPACTION();
```

## 示例

```
tdsql3_sys_local@localhost [test_base]> CALL
DBMS_ADMIN.TDSTORE_FORCE_FULL_COMPACTION();
Query OK, 0 rows affected (0.48 sec)
```

# 系统表和系统视图 PERFORMANCE_SCHEMA DATA_LOCKS

最近更新时间：2025-04-22 15:40:02

## 功能

`PERFORMANCE_SCHEMA.DATA_LOCKS` 用于展示**集群中所有 hybrid 节点**的 TDStore 上所有**已成功加锁**的悲观锁信息。
在用户没有查询 `DATA_LOCKS` 表的时候，`DATA_LOCKS` 不会带来额外的开销。用户不需要将 `performance_schema` 系统变量设置为 `ON`，即可查询 `DATA_LOCKS` 表。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| ENGINE | varchar(32) | 存储引擎。对于 TDStore 为 `RocksDB`。 |
| ENGINE_LOCK_ID | varchar(128) | 悲观锁的唯一标识符。对于 TDStore，其格式为 `<持有锁的事务 ID>_<锁范围>`。`<锁范围>` 的格式是：<br>• 如果锁的范围是一个单条的 key，则 `<锁范围>` 为 key 的十六进制表示。例如：锁标识符 `28673778183569468_00002797` 表示 ID 为 28673778183569468 的事务在十六进制编码为 `00002797` key 上加了一个悲观锁。<br>• 如果锁的范围是一个 key 区间，则 `<锁范围>` 格式为 `[<hex_start_key>,<hex_end_key>)`（左闭右开）或 `(<hex_start_key>,<hex_end_key>)`（开区间）。例如：锁标识符 `28673778183569468_[00002797,00002798)`，表示 ID 为 `28673778183569468` 的事务在 `[00002797,00002798)` 区间加了一个悲观锁。<br>当 key 的十六进制编码过长的时候，可能无法在128字符内表示完全。这个时候，字段值的后两个字符会被置为 `..`，表示这不是一个完整的锁标识符。这种情况下，用户通过 `ENGINE_LOCK_ID` 查询悲观锁信息时，也需要使用带有 `..` 的字段值做查询。 |

| | | |
|---|---|---|
| ENGINE_TRANSACTION_ID | bigint unsigned | 持有悲观锁的事务的唯一标识符。如果这个悲观锁是读锁，被多个事务共同持有，则只会展示其中一个事务的唯一标识符。 |
| THREAD_ID | bigint unsigned | 对应开启该事务的会话(session) ID。 |
| EVENT_ID | bigint unsigned | 导致加锁的 Performance Schema 事件 ID。对于 TDStore 为 `NULL` 。 |
| OBJECT_SCHEMA | varchar(64) | 悲观锁对应的数据库名。对于 TDStore 为 `NULL` 。 |
| OBJECT_NAME | varchar(64) | 悲观锁对应的表名。对于 TDStore 为 `NULL` 。 |
| PARTITION_NAME | varchar(64) | 悲观锁对应的分区名。对于 TDStore 为 `NULL` 。 |
| SUBPARTITION_NAME | varchar(64) | 悲观锁对应的子分区名。对于 TDStore 为 `NULL` 。 |
| INDEX_NAME | varchar(64) | 悲观锁对应的索引名。对于 TDStore 为 `NULL` 。 |
| OBJECT_INSTANCE_BEGIN | bigint unsigned | 悲观锁在内存中的地址。 |
| LOCK_TYPE | varchar(32) | 悲观锁的类型。对于 TDStore， `KEY` 值代表对单条 key 加锁， `PRE_RANGE` 值代表对 key 区间加锁。 |
| LOCK_MODE | varchar(32) | 悲观锁的读写模式。对于 TDStore， `Write` 值代表加写锁， `Read` 值代表加读锁。 |
| LOCK_STATUS | varchar(32) | 悲观锁的状态。对于 TDStore，值为 `GRANTED` ，即加锁成功。正在加的锁不会展示在 `data_locks` 表中，该信息可以通过查询 `data_lock_waits` 表获取。 |
| LOCK_DATA | varchar(8192) | 额外的信息，值由各个存储引擎自行决定。对于 TDStore，值为 `NULL` 。 |
| START_KEY | varchar(128) | 对于单条 key 上的锁，值为 `NULL` ；对于 key 区间上的锁，值代表 key 区间的左边界的十六进制编码。 |
| END_KEY | varchar(128) | 对于单条 key 上的锁，值代表该 key 的十六进制编码；对于 key 区间上的锁，值代表 key 区间的右边界的十六进制编码。 |

版权所有：腾讯云计算（北京）有限责任公司

| | | |
|---|---|---|
| EXCLUDE_START_KEY | tinyint(1) | 当且仅当悲观锁范围是一个 key 区间，且该区间为开区间时，值为1；否则，值为0。 |
| BLOCKING_TRANSACTION_NUM | bigint unsigned | 在该悲观锁上等待加锁的事务数量。 |
| BLOCKING_CHECK_READ_TRANSACTION_NUM | bigint unsigned | 在该悲观锁上等待快照读的事务数量。当持有悲观锁的事务进入提交流程的时候，悲观锁会阻塞快照读请求。 |
| READ_LOCKED_NUM | bigint unsigned | 该悲观锁上读锁的数量。 |
| TINDEX_ID | int unsigned | 悲观锁对应的 tindex ID。通过这个字段的值，我们可以执行 `SELECT * FROM information_schema.statistics WHERE tindex_id=...` 从系统表中获取 tindex ID 对应的表信息。 |
| DATA_SPACE_TYPE | varchar(32) | 悲观锁对应的 data space。`DATA_SPACE_TYPE_USER` 值代表对用户表的部分数据加锁，`DATA_SPACE_TYPE_SYSTEM` 值代表对系统表的部分数据加锁。 |
| REPLICATION_GROUP_ID | bigint unsigned | 悲观锁对应的 replication group ID。 |
| KEY_RANGE_REGION_ID | bigint unsigned | 悲观锁对应的 key range region ID。 |

**说明：**
TDStore 暂未支持在 `data_locks` 表中直接展示库名、表名等信息。如果需要查看表信息，可以根据 `TINDEX_ID` 字段的值，直接从 `information_schema.statistics` 表查询。

## 示例

- 输出示例：

```
tdsql [(none)]> SELECT * FROM performance_schema.data_locks \G
*************************** 1. row ***************************
                            ENGINE: RocksDB
                     ENGINE_LOCK_ID:
28681376014270466_(0000279780000006,00002798)
               ENGINE_TRANSACTION_ID: 28681376014270466
```

```
                    THREAD_ID: 47601122555917
                     EVENT_ID: NULL
                OBJECT_SCHEMA:
                  OBJECT_NAME:
               PARTITION_NAME: NULL
            SUBPARTITION_NAME: NULL
                   INDEX_NAME: NULL
         OBJECT_INSTANCE_BEGIN: 140502282795216
                    LOCK_TYPE: PRE_RANGE
                    LOCK_MODE: Write
                  LOCK_STATUS: GRANTED
                    LOCK_DATA: NULL
                    START_KEY: 0000279780000006
                      END_KEY: 00002798
            EXCLUDE_START_KEY: 1
      BLOCKING_TRANSACTION_NUM: 0
BLOCKING_CHECK_READ_TRANSACTION_NUM: 0
              READ_LOCKED_NUM: 0
                    TINDEX_ID: 10135
              DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
          REPLICATION_GROUP_ID: 257
           KEY_RANGE_REGION_ID: 7312
*************************** 2. row ***************************
                       ENGINE: RocksDB
               ENGINE_LOCK_ID:
28681376014270466_0000279780000006
         ENGINE_TRANSACTION_ID: 28681376014270466
                    THREAD_ID: NULL
                     EVENT_ID: NULL
                OBJECT_SCHEMA:
                  OBJECT_NAME:
               PARTITION_NAME: NULL
            SUBPARTITION_NAME: NULL
                   INDEX_NAME: NULL
         OBJECT_INSTANCE_BEGIN: 140507882943448
                    LOCK_TYPE: KEY
                    LOCK_MODE: Write
                  LOCK_STATUS: GRANTED
                    LOCK_DATA: NULL
                    START_KEY: NULL
                      END_KEY: 0000279780000006
            EXCLUDE_START_KEY: 0
      BLOCKING_TRANSACTION_NUM: 0
```

```
BLOCKING_CHECK_READ_TRANSACTION_NUM: 0
                     READ_LOCKED_NUM: 0
                           TINDEX_ID: 10135
                     DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
                REPLICATION_GROUP_ID: 257
                 KEY_RANGE_REGION_ID: 7312
```

- **查询示例：**

```
-- 查询当前节点的悲观锁数量
SELECT COUNT(*) FROM performance_schema.data_locks \G

-- 查询当前节点上，replication group 257 的 key range region 7312 的所有的悲观锁信息（仅指定 key_range_region_id 也可查询，但效率较差，无法利用索引加速查询）
SELECT * FROM performance_schema.data_locks WHERE replication_group_id = 257 AND key_range_region_id = 7312 \G

-- 查询当前节点上，和 [0000279780000003, 0000279780000004] 有交集的所有悲观锁的信息
SET @left = "0000279780000003";
SET @right = "0000279780000004";
SELECT * FROM performance_schema.data_locks WHERE
    (lock_type = 'KEY' AND @left <= end_key AND end_key <= @right) OR
-- 如果是单条 key 上的悲观锁，那么对应的 key 存储在 end_key 字段上，只要 end_key 在区间内即可
    (lock_type = 'PRE_RANGE' AND @left < end_key AND (start_key < @right OR (NOT exclude_start_key and start_key = @right))) \G -- 如果是 key 区间上的悲观锁，那么需要根据 exclude_start_key，判断 [start_key, end_key) 或 (start_key, end_key) 和区间是否有交集

-- 查询当前节点上，testdb.test 表所有悲观锁的信息
SELECT data_locks.* FROM performance_schema.data_locks, information_schema.statistics WHERE statistics.tindex_id = data_locks.tindex_id AND statistics.table_schema = 'testdb' AND statistics.table_name = 'test' \G

-- 查询当前节点上，事务 28681535313936395 所加所有悲观锁的信息
SELECT * FROM performance_schema.data_locks WHERE engine_transaction_id = 28681535313936395 \G

-- 查询当前节点上，阻塞了事务的所有悲观锁的信息
```

```
SELECT * FROM performance_schema.data_locks WHERE
blocking_transaction_num > 0 OR blocking_check_read_transaction_num >
0
```

- **展示被阻塞的参与者信息**

  **下面展示了三个并发执行的 session。在这个例子中，session B 和 session C 的查询语句会被 session A 在表 t 上加的范围悲观锁阻塞。**

  - session A：

    ```
    BEGIN;
    SELECT a FROM t FOR UPDATE;
    SELECT SLEEP(100);
    ```

  - session B：

    ```
    SELECT b FROM t FOR UPDATE;
    ```

  - session C：

    ```
    SELECT c FROM t FOR UPDATE;
    ```

**此时，使用以下查询语句去查询哪些事务正在等待，以及它们正在被哪些事务阻塞。**

```
SELECT blocking_engine_transaction_id, blocking_engine_lock_id,
requesting_engine_transaction_id, requesting_engine_lock_id,
tindex_id, replication_group_id, key_range_region_id FROM
performance_schema.data_lock_waits \G
```

**结果如下：**

| blocking_engine_transaction_id | blocking_engine_lock_id | requesting_engine_transaction_id | requesting_engine_lock_id | tindex_id | replication_group_id | key_range_region_id |
|---|---|---|---|---|---|---|
| 28687218495193155 | 2868721849519315 5_[00002819,0000281A) | 28687218897846327 | 28687218897846327_[00002819,0000281A) | 10265 | 257 | 103181 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 28687218495193155 | 286872184951931555_[00002819,0000281A) | 28687218746851349 | 28687218746851349_[00002819,0000281A) | 10265 | 257 | 103181 |

# DATA_LOCK_WAITS

最近更新时间：2025-04-22 15:40:02

## 功能

`PERFORMANCE_SCHEMA.DATA_LOCK_WAITS` 用于展示**集群中所有 hybrid 节点**的 TDStore 上所有悲观锁等待信息。
在用户没有查询 `DATA_LOCK_WAITS` 表的时候，`DATA_LOCK_WAITS` 不会带来额外的开销。用户不需要将 `performance_schema` 系统变量设置为 `ON`，即可查询 `DATA_LOCK_WAITS` 表。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| ENGINE | varchar(32) | 存储引擎。对于 TDStore，值是 `RocksDB`。 |
| REDQUESTING_ENGINE_LOCK_I | varchar(128) | 唯一标识被阻塞的事务正在申请的悲观锁，或者正在读取的 key 范围。对于 TDStore，其格式为 `<持有锁的事务 ID>_<锁范围>`。`<锁范围>` 的格式是：<br>• 如果锁的范围是一个单条的 key，则 `<锁范围>` 为 key 的十六进制表示。例如：锁标识符 `28673778183569468_00002797` 表示 ID 为 28673778183569468的事务在十六进制编码为 `00002797` 的 key 上加了一个悲观锁。<br>• 如果锁的范围是一个 key 区间，则 `<锁范围>` 格式为 `[<hex_start_key>,<hex_end_key>)`（左闭右开）或 `(<hex_start_key>,<hex_end_key>)`（开区间）。例如：锁标识符 `28673778183569468_[00002797,00002798)`，表示 ID 为 28673778183569468的事务在 `[00002797,00002798)` 区间加了一个悲观锁。<br>当 key 的十六进制编码过长的时候，可能无法在128字符内表示完全。这个时候，字段值的后两个字符会被置为 `..`，表示这不是一个完整的锁标识符。这种情况下，用户通过 `ENGINE_LOCK_ID` 查询悲观锁信息时，也需要使用带有 `..` 的字段值做查询。 |

| | | |
|---|---|---|
| REQUESTING_ENGINE_TRANSACTION_ID | bigint unsigned | 被阻塞的事务的唯一标识符。对于 TDStore，如果被阻塞的是只读请求，则值为 `0` 。 |
| REQUESTING_THREAD_ID | bigint unsigned | 执行被阻塞事务的线程 ID。对于 TDStore ，值为 `NULL` 。 |
| REQUESTING_EVENT_ID | bigint unsigned | 导致被阻塞的事务加锁的 Performance Schema 事件 ID。对于 TDStore，值为 `NULL` 。 |
| REQUESTING_OBJECT_INSTANCE_BEGIN | bigint unsigned | 等锁请求在内存中的地址。 |
| BLOCKING_ENGINE_LOCK_ID | varchar(128) | 悲观锁的唯一标识。对于 TDStore ，格式参考 `data_locks` 表的 `ENGINE_LOCK_ID` 字段。 |
| BLOCKING_ENGINE_TRANSACTION_ID | bigint unsigned | 持有悲观锁的事务的唯一标识符。如果这个悲观锁是读锁，被多个事务共同持有，则只会展示其中一个事务的唯一标识符。 |
| BLOCKING_THREAD_ID | bigint unsigned | 执行持有悲观锁的事务的线程 ID。对于 TDStore ，值为 `NULL` 。 |
| BLOCKING_EVENT_ID | bigint unsigned | 导致持有悲观锁的事务加锁的 Performance Schema 事件 ID。对于 TDStore ，值为 `NULL` 。 |
| BLOCKING_OBJECT_INSTANCE_BEGIN | bigint unsigned | 悲观锁在内存中的地址。 |
| TINDEX_ID | int unsigned | 悲观锁对应的 tindex ID。通过这个字段的值，我们可以执行 `SELECT * FROM information_schema.statistics WHERE tindex_id=...` 从系统表中获取 tindex ID 对应的表信息。 |
| DATA_SPACE_TYPE | varchar(32) | 悲观锁对应的 data space。 `DATA_SPACE_TYPE_USER` 代表对用户表的部分数据加锁， `DATA_SPACE_TYPE_SYSTEM` 代表对系统表的部分数据加锁。 |
| REPLICATION_GROUP_ID | bigint unsigned | 悲观锁对应的 replication group ID。 |
| KEY_RANGE_REGION_ID | bigint unsigned | 悲观锁对应的 key range region ID。 |

## 示例

- 输出示例：

```
tdsql [(none)]> SELECT * FROM performance_schema.data_lock_waits \G
*************************** 1. row ***************************
                          ENGINE: RocksDB
       REQUESTING_ENGINE_LOCK_ID:
28681541555060779_[00002797,00002798)
REQUESTING_ENGINE_TRANSACTION_ID: 28681541555060779
            REQUESTING_THREAD_ID: 0
             REQUESTING_EVENT_ID: 0
REQUESTING_OBJECT_INSTANCE_BEGIN: 139759771095568
         BLOCKING_ENGINE_LOCK_ID: 28681535313936395_0000279780000001
  BLOCKING_ENGINE_TRANSACTION_ID: 28681535313936395
              BLOCKING_THREAD_ID: 0
               BLOCKING_EVENT_ID: 0
  BLOCKING_OBJECT_INSTANCE_BEGIN: 139759768476120
                       TINDEX_ID: 10135
                  DATA_SPACE_TYPE: DATA_SPACE_TYPE_USER
            REPLICATION_GROUP_ID: 257
             KEY_RANGE_REGION_ID: 7312
```

- 查询示例：

```
-- 查询当前节点上，等待悲观锁的请求数量
SELECT COUNT(*) FROM performance_schema.data_lock_wais \G

-- 查询当前节点上，replication group 257 的 key range region 7312 的所有的悲观锁等待信息（仅指定 key_range_region_id 也可查询，但效率较差，无法利用索引加速查询）
SELECT * FROM performance_schema.data_lock_waits WHERE
replication_group_id = 257 AND key_range_region_id = 7312 \G

-- 查询当前节点上，事务 28682479434989637 正在等待的悲观锁的信息
SELECT data_locks.* FROM performance_schema.data_locks,
performance_schema.data_lock_waits WHERE
requesting_engine_transaction_id = 28682479434989637 AND
blocking_engine_lock_id = engine_lock_id \G
```

# METADATA_LOCKS

最近更新时间：2025-04-22 15:40:02

## 功能

`METADATA_LOCKS` 记录着当前 SQLEngine 元数据锁的占用情况，包含：

- 已授予的锁：显示哪些会话当前拥有哪些元数据锁。
- 已请求但尚未授予的锁：显示哪些会话正在等待哪些元数据锁。
- 已被死锁检测器终止的锁定请求。
- 已超时并等待请求会话释放锁的锁请求。

方便用户快速定位元数据锁（Metadata Locks，MDL）相关的问题，提高系统的稳定性和性能。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| OBJECT_TYPE | varchar(64) | 锁定的对象类型，例如 TABLE、SCHEMA、FUNCTION、MC 等。<br>其中 MC 是 TDSQL 新增的全局对象锁，在集群层面对数据库对象加锁。 |
| OBJECT_SCHEMA | varchar(64) | 锁定对象所在的数据库名称。 |
| OBJECT_NAME | varchar(64) | 锁定对象的名称。 |
| COLUMN_NAME | varchar(64) | 锁定对象的具体列名称。 |
| OBJECT_INSTANCE_BEGIN | bigint unsigned | 锁定对象在内存中的地址。 |
| LOCK_TYPE | varchar(32) | 锁的类型。<br>• INTENTION_EXCLUSIVE：意图独占锁。这种锁通常用于表示一个事务打算获取一个独占锁，但目前还没有实际获取到。<br>• SHARED：共享锁。允许多个事务同时持有这种锁，但阻止任何事务获取独占锁。<br>• SHARED_HIGH_PRIO：高优先级共享锁。这是一种特殊类型的共享锁，用于高优先级的事务。<br>• SHARED_READ：共享读锁。允许事务读取数据，但阻止其他事务写入或获取独占锁。 |

| | | |
|---|---|---|
| | | • SHARED_UPGRADABLE：可升级的共享锁。这种锁允许事务在持有共享锁的同时尝试获取独占锁，而不会被阻塞。<br>• SHARED_NO_WRITE：不允许写的共享锁。这种锁可能阻止事务写入数据，但允许读取。<br>• SHARED_NO_READ_WRITE：不允许读写的共享锁。这种锁阻止事务读取和写入数据。<br>• EXCLUSIVE：独占锁。只允许一个事务持有这种锁，阻止其他事务获取任何类型的锁。 |
| LOCK DURATION | varchar(32) | 锁的持续时间，例如 TRANSACTION、LONG 等。 |
| LOCK STATUS | varchar(32) | 表示每个锁的状态：<br>• PENDING：当元数据锁被请求但未立即获得时，插入一行状态为 PIDDEN 的记录。<br>• GRANTED：当元数据锁被请求并立即获得时，插入一行状态为GRANTED 的记录。<br>• VICTIM：当死锁检测器取消挂起的锁请求以打破死锁（ER_LOCK_DEADLOCK）时，将其行状态从 PENDING 更新为VICTIM。这表示该锁行即将被删除。<br>• TIMEOUT：当挂起的锁请求超时（ER_LOCK_WAIT_TIMEOUT）时，将其行状态从 PENDING 更新为 TIMEOUT。这表示该锁行即将被删除。<br>• KILLED：当被授予的锁或挂起的锁请求被终止时，将其行状态从GRANTED 或 PENDING 更新为 KILLED。这表示该锁行即将被删除。 |
| SOURCE | varchar(64) | 包含生成事件的已检测代码的源文件的名称以及文件中发生检测的行号。 |
| OWNER THREAD_ID | bigint unsigned | 持有锁的线程 ID。 |
| OWNER_EVENT ID | bigint unsigned | 请求元数据锁的事件ID。 |

## 示例

```
#session1
BEGIN
UPDATE test1 SET k = 0 WHERE id = 999;
```

```
#session2
ALTER TABLE test1 ADD COLUMN new_column VARCHAR(255);

#查看metadata_locks可以看到ddl获取元数据锁被挂起
tdsql [performance_schema]> select * from metadata_locks where
OBJECT_NAME='test1' \G
*************************** 1. row ***************************
          OBJECT_TYPE: TABLE
        OBJECT_SCHEMA: etest
          OBJECT_NAME: test1
          COLUMN_NAME: NULL
OBJECT_INSTANCE_BEGIN: 140400821752448
            LOCK_TYPE: EXCLUSIVE
        LOCK_DURATION: TRANSACTION
          LOCK_STATUS: PENDING
               SOURCE: mdl.cc:3924
      OWNER_THREAD_ID: 5670
       OWNER_EVENT_ID: 1
1 row in set (0.00 sec)
```

# INFORMATION_SCHEMA DDL_JOBS

最近更新时间：2025-06-25 09:51:52

## 功能

`INFORMATION_SCHEMA.DDL_JOBS` 展示每个 DDL 任务的状态信息。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| id | bigint unsigned | 唯一的 DDL 任务 ID。 |
| schema_name | varchar(64) | DDL 任务对应的 schema 名。 |
| table_name | varchar(64) | DDL 任务对应的 table 名。 |
| version | bigint | DDL 任务对应的版本信息。 |
| ddl_status | enum | DDL 任务执行状态。<br>• FAIL：失败。<br>• SUCCESS：成功。<br>• EXECUTING：正在执行。 |
| start_timestamp | timestamp | DDL 任务开启时间。 |
| last_timestamp | timestamp | 最近一次 DDL 任务更新时间。 |
| ddl_sql | longtext | DDL SQL。 |
| info_type | enum | DDL 任务类型。 |
| info | longtext | 记录 DDL 任务执行过程中的必要信息。 |
| is_history | bool | DDL 任务是否已经结束。<br>• 1：已结束。<br>• 0：尚未结束。 |

## 示例

```
tdsql> create table sbtest1 (id int primary key);
tdsql> select * from information_Schema.ddl_jobs order by id desc limit
1\G;
*************************** 1. row ***************************
             ID: 2
    SCHEMA_NAME: tdsql
     TABLE_NAME: sbtest1
        VERSION: 4
     DDL_STATUS: SUCCESS
START_TIMESTAMP: 2025-06-24 15:00:35
 LAST_TIMESTAMP: 2025-06-24 15:00:35
        DDL_SQL: create table sbtest1 (id int primary key)
      INFO_TYPE: CREATE TABLE
           INFO: {"rm_idx":[],"wf_rmed":false,"exec_addr":
{"ip":"10.10.10.10","port":15035},"row_apply_saved":true,"current_schema
_name":"tdsql","ddl_with_complete_info":"","crt_data_obj_task_id":293726
84622102841,"dstr_data_obj_task_id":0,"data_obj_to_be_dstr_arr":
[],"tbl_tidxid":10028,"row_applied":false,"recov_addr":
{"ip":"10.10.10.10","port":15035}}
     IS_HISTORY: 1
1 row in set (0.00 sec)
```

# PERSIST

最近更新时间：2025-04-22 15:40:02

## 功能

`INFORMATION_SCHEMA.PERSIST` 用于展示持久化到 MC 侧的 variables。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| VARIABLE_NAME | varchar(64) | 变量名字。 |
| VARIABLE_VALUE | varchar(1024) | 变量值。 |

## 示例

可以通过 `SELECT` 或者 `SHOW` 查询持久化到 MC 侧的变量。

```
tdsql>SELECT * FROM INFORMATION_SCHEMA.PERSIST;
+-------------------------+----------------+
| VARIABLE_NAME           | VARIABLE_VALUE |
+-------------------------+----------------+
| tdstore_deadlock_detect | ON             |
| tdsql_lock_wait_timeout | 50             |
+-------------------------+----------------+
2 rows in set (0.01 sec)

tdsql>SHOW PERSIST VARIABLES;
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| tdstore_deadlock_detect | ON    |
| tdsql_lock_wait_timeout | 50    |
+-------------------------+-------+
2 rows in set (0.00 sec)
```

# TDSTORE_COLUMNAR_ACTIVE_LOAD_TASK_INFO

最近更新时间：2025-06-13 20:31:03

## 功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_ACTIVE_LOAD_TASK_INFO` 用于展示列存节点当前正在运行中的列存数据转储任务。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| TASK_ID | BIGINT | 列存转储任务的 ID。 |
| START_TIME | VARCHAR | 列存转储任务的开始时间。 |
| RUNNING_STATE | VARCHAR | 列存转储任务的运行状态，包括 `INIT`、`START`、`COMMITING`、`FINISH`、`FAIL`。 |
| PARALLEL_DEGREE | INT | 列存转储任务执行的并行度。 |
| ERROR_MSG | VARCHAR | 如果发生错误，列存转储任务的错误信息。 |
| TABLE_COUNT | INT | 列存转储任务转储的表个数。 |
| TABLE_LIST | VARCHAR | 列存转储任务转储的表列表。 |
| FILE_COUNT | INT | 列存转储任务产生的列存文件数量。 |
| INSERT_ROW_COUNT | INT | 列存转储任务写入的数据行数。 |
| DELETE_ROW_COUNT | INT | 列存转储任务删除的数据行数。 |
| NOT_LOAD_FOR_DECODE_FAILURE_ROW_COUNT | INT | 由于解码失败，列存任务没有转储的数据行数。 |
| NOT_LOAD_FOR_NO_NEED_ROW_COUNT | INT | 由于不需要转储（例如二级索引），列存任务没有转储的数据行数。 |

版权所有：腾讯云计算（北京）有限责任公司

| | | |
|---|---|---|
| NOT_LOAD_FOR_LATER_THAN_GES_ROW_COUNT | INT | 由于数据时间戳大于 GES，列存任务没有转储的数据行数。 |

## 示例

```
SELECT * FROM TDSTORE_COLUMNAR_ACTIVE_LOAD_TASK_INFO;
+---------+--------------------+--------------+-------+---------------
-+-----------+-------------+-----------------------------------------
------+------------+-----------------+-----------------+--------------
-----------------------+-------------------------------+------------
------------------------+
| TASK_ID | START_TIME          | RUNNING_TIME | STATE | PARALLEL_DEGREE
| ERROR_MSG | TABLE_COUNT | TABLE_LIST
| FILE_COUNT | INSERT_ROW_COUNT | DELETE_ROW_COUNT |
NOT_LOAD_FOR_DECODE_FAILURE_ROW_COUNT | NOT_LOAD_FOR_NO_NEED_ROW_COUNT |
NOT_LOAD_FOR_LATER_THAN_GES_ROW_COUNT |
+---------+--------------------+--------------+-------+---------------
-+-----------+-------------+-----------------------------------------
------+------------+-----------------+-----------------+--------------
-----------------------+-------------------------------+------------
------------------------+
|      18 | 2025-04-16 16:08:28 | 21 sec       | START |               4
|           |           3 |
tpch100g.part,tpch100g.orders,tpch100g.lineitem, |          0 |
1344389 |                0 |                                     0 |
487 |                                   0 |
+---------+--------------------+--------------+-------+---------------
-+-----------+-------------+-----------------------------------------
------+------------+-----------------+-----------------+--------------
-----------------------+-------------------------------+------------
------------------------+
```

# TDSTORE_COLUMNAR_COMPACTION_INFO

最近更新时间：2025-06-13 20:31:03

## 功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_COMPACTION_INFO` 用于记录列存 compaction 和 drop table 任务的执行过程与结果。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| node_name | VARCHAR | 执行列存 compaction 节点的节点名。 |
| job_description | VARCHAR | 列存 compaction 任务描述：<br>• 普通 compaction<br>• drop table |
| event_type | VARCHAR | 列存 compaction 任务类型（TD_EVENT_COLUMNAR_COMPACT）。 |
| table_name | VARCHAR | 执行列存 compaction 的表的表名。 |
| tx_id | INT | 执行列存 compaction 的列存事务的事务 ID。 |
| tx_seq_num | INT | 执行列存 compaction 的列存事务提交后获取到的列存元数据库的最老 snapshot 对应的 sequence number。 |
| ret | VARCHAR | 列存 compaction 任务的返回码。 |
| compacted_files | VARCHAR | 列存 compaction 任务回收的文件列表。 |
| new_files | VARCHAR | 列存 compaction 任务生成的文件列表。 |
| begin_time | VARCHAR | 列存 compaction 任务的开始时间。 |
| cost_us | INT | 列存 compaction 任务的总耗时。 |
| detailed_info | VARCHAR | 列存 compaction 任务每一步的耗时。 |

## 示例

```
tdsql3_sys_local@localhost [test_base]> SELECT * FROM
information_schema.TDSTORE_COLUMNAR_COMPACTION_INFO\G
*************************** 1. row ***************************
      node_name: node-run-test_reboot-001
job_description: columnar compaction
     event_type: TD_EVENT_COLUMNAR_COMPACT
     table_name: test_table
          tx_id: 12
     tx_seq_num: 13
            ret: 0(EC_OK)
compacted_files: 10026_2, 10026_1
      new_files: 3
     begin_time: 2025-04-22 18:51:49 215
        cost_us: 4910
  detailed_info:  [COLUMNAR_FILE_DEFRAGMENTATION: <BEGIN: 4695us>,
<REWRITE_FILES: 85us>, <META_MODIFICATION: 123us>, <REMOVE_PKS: 0us>,
<COMMIT: 6us>]
*************************** 2. row ***************************
      node_name: node-run-test_reboot-001
job_description: columnar compaction
     event_type: TD_EVENT_COLUMNAR_COMPACT
     table_name: test_table
          tx_id: 22
     tx_seq_num: 28
            ret: 0(EC_OK)
compacted_files: 10026_3, 10026_5, 10026_4
      new_files: 6
     begin_time: 2025-04-22 18:52:19 235
        cost_us: 6800
  detailed_info:  [COLUMNAR_FILE_DEFRAGMENTATION: <BEGIN: 6563us>,
<REWRITE_FILES: 86us>, <META_MODIFICATION: 141us>, <REMOVE_PKS: 0us>,
<COMMIT: 7us>]
2 rows in set (0.01 sec)
```

# TDSTORE_COLUMNAR_FILE_DELETION_INFO

最近更新时间：2025-06-13 20:31:03

## 功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_FILE_DELETION_INFO` 用于记录列存文件删除的过程与结果。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| node_name | VARCHAR | 执行列存文件删除的节点的节点名。 |
| job_description | VARCHAR | 列存文件删除的任务描述。 |
| event_type | VARCHAR | 列存文件删除的任务类型。 |
| columnar_files_name | VARCHAR | 列存文件删除任务本次删除的文件的列表。 |
| begin_time | VARCHAR | 列存文件删除任务的开始时间。 |
| cost_us | INT | 列存文件删除任务的总耗时。 |
| detailed_info | VARCHAR | 列存文件删除任务各阶段的耗时。 |

## 示例

```
tdsql3_sys_local@localhost [test_base]> SELECT * FROM
INFORMATION_SCHEMA.TDSTORE_COLUMNAR_FILE_DELETION_INFO\G
*************************** 1. row ***************************
          node_name: node-run-test_reboot-001
    job_description: delete files
         event_type: TD_EVENT_COLUMNAR_FILE_DELETE
columnar_files_name: 10026_2, 10026_1
         begin_time: 2025-04-22 18:51:49 859
            cost_us: 365
      detailed_info:  [COLUMNAR_FILE_DELETION: <BEGIN: 364us>,
<DELETE_FILES: 0us>]
*************************** 2. row ***************************
          node_name: node-run-test_reboot-001
    job_description: delete files
```

```
          event_type: TD_EVENT_COLUMNAR_FILE_DELETE
 columnar_files_name: 10026_3, 10026_5, 10026_4
          begin_time: 2025-04-22 18:52:19 859
             cost_us: 336
       detailed_info:  [COLUMNAR_FILE_DELETION: <BEGIN: 335us>,
<DELETE_FILES: 0us>]
2 rows in set (0.01 sec)
```

# TDSTORE_COLUMNAR_FILES_INFO

最近更新时间：2025-06-13 20:31:03

## 功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_FILES_INFO` 用于展示列存节点中列存文件的相关信息。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| TABLE_ID | INT | 文件对应的表的 TIndexID。 |
| DB | VARCHAR | 文件所对应的 DATABASE 名称。 |
| TABLE_NAME | VARCHAR | 文件所对应的 TABLE 名称。 |
| FILE_PATH | VARCHAR | 文件的绝对路径。 |
| FILE_ID | INT | 文件的 ID。 |
| FILE_SIZE_IN_MB | DOUBLE | 文件的大小，单位是 MB。 |
| LAST_WRITE_TIME | VARCHAR | 文件的最后修改时间。 |
| RECORD_COUNT | INT | 文件的总数据行数。 |
| INVALID_RECORD_COUNT | INT | 文件无效的数据行数。 |
| INVALID_RATIO | DOUBLE | 文件中无效数据的占比，即 `INVALID_RECORD_COUNT/RECORD_COUNT` 。 |

## 示例

```
SELECT * FROM TDSTORE_COLUMNAR_FILES_INFO;
+----------+----------+------------+-----------------------------------
-----------------------------------------------------------------------
+---------+-------------------+---------------------+--------------+----
------------------+--------------+
| TABLE_ID | DB        | TABLE_NAME | FILE_PATH
| FILE_ID | FILE_SIZE_IN_MB    | LAST_WRITE_TIME      | RECORD_COUNT |
```

```
INVALID_RECORD_COUNT | INVALID_RATIO |
+----------+----------+------------+-----------------------------------
-----------------------------------------------------------------------
+---------+------------------+--------------------+--------------+----
------------------+---------------+
|    10024 | tpch100g | part       | /data/tdsql/auto-install-clang-
debug-
build/10351/data/10351/tdstore/columnar_data/10024_00000010.parquet |
10 | 25.18278980255127 | 2025-04-16 16:19:25 |       500387 |
0 |             0 |
|    10024 | tpch100g | part       | /data/tdsql/auto-install-clang-
debug-
build/10351/data/10351/tdstore/columnar_data/10024_00000012.parquet |
12 | 25.41611957550049 | 2025-04-16 16:19:32 |       502697 |
0 |             0 |
+----------+----------+------------+-----------------------------------
-----------------------------------------------------------------------
+---------+------------------+--------------------+--------------+----
------------------+---------------+
```

# TDSTORE_COLUMNAR_HISTORY_LOAD_TASK_INFO

最近更新时间：2025-06-13 20:31:03

## 功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_HISTORY_LOAD_TASK_INFO` 用于展示已经结束的列存数据转储任务信息。由于内存限制，最多只会保留过去发生过的2048个任务。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| LOAD_TASK_ID | BIGINT | 列存数据转储任务的 ID。 |
| START_TIME | VARCHAR | 列存数据转储任务的开始时间。 |
| END_TIME | VARCHAR | 列存数据转储任务的结束时间。 |
| EXECUTION_TIME | VARCHAR | 列存数据转储任务的执行时间。 |
| STATE | VARCHAR | 列存数据转储任务的最终状态（可选：`FINISH` / `FAIL` ）。 |
| PARALLEL_DEGREE | INT | 列存数据转储任务的并行度。 |
| ERROR_MSG | VARCHAR | 如果列存转储任务失败，失败的相关信息。 |
| TABLE_COUNT | INT | 列存数据转储任务处理的表数量。 |
| TABLE_LIST | VARCHAR | 列存数据转储任务处理的表列表。 |
| FILE_COUNT | INT | 列存数据转储任务产生的列存文件数量。 |
| INSERT_ROW_COUNT | INT | 列存数据转储任务写入的数据量。 |
| DELETE_ROW_COUNT | INT | 列存转储任务删除的数据行数。 |
| NOT_LOAD_FOR_DECODE_FAIL | INT | 由于解码失败，列存任务没有转储的数据行 |

| | | |
|---|---|---|
| URE_ROW_COUNT | | 数。 |
| NOT_LOAD_FOR_NO_NEED_ROW_COUNT | INT | 由于不需要转储（例如二级索引），列存任务没有转储的数据行数。 |
| NOT_LOAD_FOR_LATER_THAN_GES_ROW_COUNT | INT | 由于数据时间戳大于 GES，列存任务没有转储的数据行数。 |
| GENERATED_FILE_SIZE_IN_MB | VARCHAR | 列存数据转储任务产生的列存文件总大小，单位是MB。 |
| COMMIT_TIME_COST | VARCHAR | 列存数据转储任务在提交元数据 DB 事务时所消耗的时间。 |

## 示例

```
SELECT * FROM TDSTORE_COLUMNAR_HISTORY_LOAD_TASK_INFO;
+-------------+--------------------+--------------------+-----------
----+--------+----------------+----------+------------+-----------+-
----------+-----------------+-----------------+--------------------
----------------+-------------------------------+--------------------
-----------------+--------------------------+-----------------+
| LOAD_TASK_ID | START_TIME          | END_TIME            |
EXECUTION_TIME | STATE  | PARALLEL_DEGREE | ERROR_MSG | TABLE_COUNT |
TABLE_LIST | FILE_COUNT | INSERT_ROW_COUNT | DELETE_ROW_COUNT |
NOT_LOAD_FOR_DECODE_FAILURE_ROW_COUNT | NOT_LOAD_FOR_NO_NEED_ROW_COUNT |
NOT_LOAD_FOR_LATER_THAN_GES_ROW_COUNT | GENERATED_FILE_SIZE_IN_MB |
COMMIT_TIME_COST |
+-------------+--------------------+--------------------+-----------
----+--------+----------------+----------+------------+-----------+-
----------+-----------------+-----------------+--------------------
----------------+-------------------------------+--------------------
-----------------+--------------------------+-----------------+
|           17 | 2025-04-16 16:02:04 | 2025-04-16 16:04:01 |  1 min 56
sec  | FINISH |               4 |          |           0 |           |
0 |           123123 |            12312 |
0 |                        5691930 |
11205443 | 1.35 MB                   |   0 ms          |
+-------------+--------------------+--------------------+-----------
----+--------+----------------+----------+------------+-----------+-
----------+-----------------+-----------------+--------------------
----------------+-------------------------------+--------------------
-----------------+--------------------------+-----------------+
```

# TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO

最近更新时间：2025-06-13 20:31:03

## 功能

`INFORMATION_SCHEMA.TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO` 用于统计一张表的行存数据量和列存数据量以及二者比例。

## 字段说明

| 字段名 | 类型 | 描述 |
| --- | --- | --- |
| node_name | VARCHAR | 被统计表所在的节点。 |
| db_name | VARCHAR | 被统计表所在的数据库名。 |
| table_name | VARCHAR | 被统计表的表名。 |
| table_id | INT | 被统计表的表 id。 |
| column_data_count | INT | 被统计表的列存数据行数。 |
| row_data_count | INT | 被统计表的行存数据行数。 |
| data_ratio | DOUBLE | 被统计表的列存数据行数与行存数据行数的比例（列存数据行数/行存数据行数）。 |

## 示例

```
tdsql3_sys_local@localhost [information_schema]> select * from
TDSTORE_COLUMNAR_NODE_DATA_COUNT_INFO\G
*************************** 1. row ***************************
        node_name: node-run-test_reboot-001
          db_name: test_base
       table_name: test_table
         table_id: 10026
column_data_count: 4
   row_data_count: 2702
       data_ratio: 0.0014803849000740192
```

```
1 row in set (0.00 sec)
```

# 数据库参数说明
# 计算引擎参数
# 计算引擎参数概述

最近更新时间：2025-04-22 15:40:02

通过设置计算引擎参数，您可以灵活地控制数据库系统的各种行为，例如缓存大小、并发连接数、CPU 使用率、内存使用率等。此外，计算引擎参数还可用于配置数据库系统的各种功能。

## 计算引擎参数分类

- **GLOBAL 变量**：表示 GLOBAL 级别的修改，数据库同一实例内的不同用户共享全局变量。全局变量的修改不会随会话的退出而失效。此外，全局变量修改后，对当前已打开的 Session 不生效，需要重新建立 Session 才能生效。
- **SESSION 变量**：表示 SESSION 级别的修改。当客户端连接到数据库后，数据库会复制全局变量来自动生成 Session 变量。Session 变量的修改仅对当前 Session 生效。

## 查询计算引擎参数

- **方式一**：

  1.1 登录 TDSQL MySQL 版控制台 。

  1.2 在**实例列表**页面，选择所需实例 ID。

  1.3 进入**参数配置** > **数据库参数**页面，搜索并查看计算引擎参数。
- **方式二**：使用 `SHOW [GLOBAL] VARIABLES` 语句查询。

```
tdsql> SHOW VARIABLES LIKE 'auto_increment_increment';
tdsql> SHOW GLOBAL VARIABLES LIKE 'auto_increment_increment';
```

## 修改计算引擎参数

- **方式一**：

  1.1 登录 TDSQL MySQL 版控制台 。

  1.2 在**实例列表**页面，选择所需实例 ID。

  1.3 进入**参数配置** > **数据库参数**页面，搜索计算引擎参数，在参数所在行，单击**操作**列**修改**。

  1.4 （可选）若参数需重启生效，请在根据业务需要，重启实例。
- **方式二**：

  1.1 修改计算引擎参数。

```
tdsql> SET GLOBAL auto_increment_increment = 1;
tdsql> SET auto_increment_increment = 1;
```

1.2 （可选）若参数需重启生效，请在根据业务需要，重启实例。

# 计算引擎参数总览

最近更新时间：2025-06-13 20:31:02

| 参数 | 说明 |
| --- | --- |
| auto_increment_increment | 用于设置自增列自增的步长。 |
| auto_increment_offset | 用于设置自增列自增的初始值。 |
| autocommit | 用于设置是否开启单语句自动提交。 |
| character_set_server | 用于设置服务器字符集。 |
| collation_server | 用于设置服务器的默认排序规则。 |
| connect_timeout | 用于设置 SQLEngine 等待连接请求成功的最大时长，单位为秒。 |
| ddl_recovery_wait_time | 用于设置 DDL 语句（如 `ALTER TABLE`、`CREATE TABLE` 等）从失败到后台线程接管恢复的等待时间，以及 DDL 语句（如 `DROP TABLE`、`DROP DATABASE` 等）从 DDL 结束到后台线程清理数据的等待时间，单位为秒。 |
| long_query_time | 用于设置慢查询阈值，单位为秒。 |
| max_allowed_packet | 用于设置 SQLEngine 和客户端在一次传送数据包的过程中所允许最大的数据包大小，单位为 Byte。 |
| max_connect_errors | 用于设置单个主机在被阻止（block）之前允许的最大中断连接数。 |
| max_connections | 用于设置 SQLEngine 允许的最大连接数。 |
| max_execution_time | 用于设置 SELECT 语句的执行超时时间，单位为毫秒。 |
| max_user_connections | 用于设置单个用户最大连接数。 |
| net_read_timeout | 用于设置连接中读数据时最大等待时长，单位为秒。 |
| net_retry_count | 用于设置服务器在尝试重新连接到客户端时的最大重试次数。 |
| net_write_timeout | 用于设置连接中写数据时最大等待时长，单位为秒。 |
| optimizer_switch | 用于设置优化器指示选项开关。 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| sql_mode | 用于设置服务器的 SQL 模式。 |
| table_definition_cache | 用于设置表定义对象的缓存数量。 |
| table_open_cache | 用于设置打开的表对象的缓存数量。 |
| tdsql_auto_increment_batch_size | 用于设置 SQLEngine 单次获取自增值的数量。 |
| tdsql_bulk_commit_records | 用于设置 DDL 过程中拆分出的子事务达到提交条件的数据条数阈值。 |
| tdsql_cache_range_max_use_cnt | 用于设置缓存的 range 信息失效前被使用的最大次数。 |
| tdsql_get_index_stats_from_tdstore | 用于设置是否允许 SQLEngine 从 TDStore 获取统计信息。 |
| tdsql_max_collect_ranges | 表示为获取统计信息所允许涉及的 range 数量。 |
| tdsql_max_retry | 表示允许失败（网络/RPC 等）重试的最大次数。 |
| tdsql_optimize_pk_point_select | 表示优化器是否对主键点查询进行优化。 |
| tdsql_optimize_table_rows_count | 表示是否开启 COUNT 优化。 |
| tdsql_retry_interval_growth | 表示 SQLEngine 与其他节点 RPC 交互的重试间隔的增长倍速。 |
| tdsql_retry_interval_msec_max | 表示两次失败（网络/RPC 等）重试之间最大间隔时间，单位为毫秒。 |
| tdsql_retry_interval_msec_min | 表示两次失败（网络/RPC 等）重试之间最小间隔时间，单位为毫秒。 |
| tdsql_skip_trans_write_lock | 表示在执行写操作的时候，是否跳过获取写锁。 |
| tdsql_split_trans_during_alter_copy | 表示是否在 COPY 算法的 DDL 过程中将大事务拆成小事务。 |
| tdsql_transaction_max_participants | 表示单个事务最大的参与者（ReplicationGroup）数量。该参数为只读参数。 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| tdsql_txn_keep_alive_lease_sec | 表示 TDStore 超过 `tdsql_txn_keep_alive_lease_sec` 秒没有收到某事务的心跳，将此事务清理。 |
| tdsql_update_local_glsv_interval | GLSV 是 Global Largest Schema Version 的缩写，用于在多节点之间同步 DDL 结果，保证分布式系统中的多个节点不会使用过旧的表结构和写入不符合最新定义的数据；DDL 推进 GLSV，并等待 `tdsql_update_local_glsv_interval` 毫秒，在这段时间内，其他节点应该知道了 GLSV 版本变化，并由此更新 SQL 使用的表版本。 |
| tdsql_update_variables_interval | 表示从 MC 获取持久化变量的时间间隔，单位为毫秒。 |
| tdsql_use_tdstore_estimated_rows | 表示是否使用 TDStore 提供的数据行数估算值来优化查询计划。 |
| tdsql_lock_wait_timeout | 表示 TDStore 引擎锁超时时间，单位为毫秒。 |
| wait_timeout | 表示连接中断等待时长，单位为秒。 |
| binlog_dump_cache_max_size | 限制 binlog dump 使用的最大内存。 |
| log_service_dump_batch_size | 落盘 binlog 一次批处理消息的最大长度。 |
| tdsql_recycle_bin_enabled | 是否开启回收站，默认关闭。 |
| log_service_mysql_client_timeout | LogService 控制参数，控制 MySQL 类型的 LogService 同步时内部 Mysql Client 连接的超时时间。 |
| log_service_mysql_info_queue_size | LogService 控制参数，控制 MySQL 类型的 LogService 同步时，内部使用的缓存队列的长度。 |
| audit_log_policy | 控制审计日志的记录策略，决定系统记录哪些操作到审计日志中。 |
| log_bin_trust_function_creators | 控制存储函数/触发器的创建限制，确保启用二进制日志（Binary Log）时，非确定性操作不会导致主从数据不一致或安全风险。 |
| max_prepared_stmt_count | 限制服务器中预处理语句（Prepared Statements）的总数量，防止恶意攻击或异常操作导致内存耗尽。 |
| tdsql_enable_trigger | 是否允许 CREATE/DROP/ALTER 触发器。 |
| tdsql_enable_view | 是否允许 CREATE/DROP/ALTER 视图。 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| tdsql_use_online_copy_ddl | COPY DDL（如 drop column, drop primary key 等）是否使用 Online COPY 方式。 |
| tdstore_bulk_load_txn_rpc_timeout_threshold_ms | Bulkload 事务 commit 的 RPC 超时，单位毫秒 ms。<br>3.0可以采取 Bulkload 的方式批量导入数据，导入事务一般数据量较大，提交该事务可能花费较多时间，需区别于其他 RPC 用法，所以为此 commit RPC 提供单独的参数控制。 |
| tdsql_stale_read_role | 设置 stale read 路由访问的副本。 |

# auto_increment_increment

最近更新时间：2025-04-22 15:40:02

## 功能描述

`auto_increment_increment` 用于设置自增列自增的步长。该变量常与 `auto_increment_offset` 一起使用。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | uint |
| 默认值 | 1 |
| 取值范围 | 1~65535 |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# auto_increment_offset

最近更新时间：2025-04-22 15:40:02

## 功能描述

`auto_increment_offset` 用于设置自增列自增的初始值。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | uint |
| 默认值 | 1 |
| 取值范围 | 1~65535 |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# autocommit

最近更新时间：2025-04-22 15:40:02

## 功能描述

`autocommit` 用于设置是否开启单语句自动提交。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | ON |
| 取值范围 | • ON：自动提交<br>• OFF：不自动提交 |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# binlog_dump_cache_max_size

最近更新时间：2025-04-22 15:40:02

## 功能描述

限制 binlog dump 使用的最大内存，单位为字节。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | uint |
| 默认值 | 3221225472（3GB） |
| 取值范围 | [1073741824（1GB）, 21474836480（20GB）] |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# character_set_server

最近更新时间：2025-04-22 15:40:02

## 功能描述

`character_set_server` 用于设置服务器字符集。

`character_set_server` 参数的主要作用是：

- 设置表的默认字符集。当创建表时，如果没有指定表的字符集，那么将使用 `character_set_server` 参数指定的字符集作为表的默认字符集。
- 设置列的默认字符集。当创建表时，如果没有为某个列指定字符集，那么将使用 `character_set_server` 参数指定的字符集作为该列的默认字符集。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | String |
| 默认值 | utf8 |
| 取值范围 | • utf8<br>• latin1<br>• gbk<br>• utf8mb4<br>• GB18030 |
| 生效范围 | SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# collation_server

最近更新时间：2025-04-22 15:40:03

## 功能描述

`collation_server` 用于设置服务器的默认排序规则。排序规则决定了如何比较和排序字符串。同时，排序规则与字符集相关联，例如 utf8mb4_general_ci、utf8mb4_unicode_ci 等。

`collation_server` 参数的主要作用是：

- 设置服务器的默认字符串比较和排序规则。这对于处理包含特殊字符或重音符号的字符串非常重要，因为不同的排序规则可能导致不同的比较和排序结果。
- 当比较服务器上的字符串时，`collation_server` 参数决定了如何进行比较。例如，在执行 SQL 查询时，如果 `WHERE` 子句中的字符串与服务器上的列值进行比较，那么将使用 `collation_server` 参数指定的排序规则进行比较。

通常，建议使用与 `character_set_server` 参数相匹配的排序规则。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | String |
| 默认值 | utf8_bin |
| 取值范围 | • utf8_general_ci<br>• utf8_bin<br>• utf8_unicode_ci<br>• latin1_general_cs<br>• latin1_general_ci<br>• latin1_swedish_ci<br>• latin1_bin<br>• gbk_chinese_ci<br>• gbk_bin<br>• utf8mb4_general_ci<br>• utf8mb4_bin<br>• utf8mb4_unicode_ci |
| 生效范围 | • SESSION<br>• GLOBAL |
| 是否可修改 | 是 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| 是否需要重启 | 否 |

# connect_timeout

最近更新时间：2025-04-22 15:40:03

## 功能描述

`connect_timeout` 用于设置 SQLEngine 等待连接请求成功的最大时长，单位为秒。当客户端尝试连接到服务器时，如果在 `connect_timeout` 的值内无法建立连接，服务器将拒绝该连接请求。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 10 |
| 取值范围 | 2 ~ 31536000（即2秒~ 1年） |
| 生效范围 | Global |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# ddl_recovery_wait_time

最近更新时间：2025-04-22 15:40:03

## 功能描述

`ddl_recovery_wait_time` 用于设置 DDL 语句（如 `ALTER TABLE` 、 `CREATE TABLE` 等）从失败到后台线程接管恢复的等待时间，以及 DDL 语句（如 `DROP TABLE` 、 `DROP DATABASE` 等）从 DDL 结束到后台线程清理数据的等待时间，单位为秒。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 5 |
| 取值范围 | 0 ~ 3600 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# long_query_time

最近更新时间：2025-04-22 15:40:03

## 功能描述

`long_query_time` 用于设置慢查询阈值，单位为秒。当查询的执行时间超过 `long_query_time` 的值时，服务器将记录慢查询日志。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Numeric |
| 默认值 | 1 |
| 取值范围 | 0 ~ 31536000（即0秒 ~ 1年） |
| 生效范围 | • SESSION<br>• GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# log_service_dump_batch_size

最近更新时间：2025-04-22 15:40:03

## 功能描述

落盘 binlog 一次批处理消息的最大长度，单位为字节。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | uint |
| 默认值 | 10 * 1024 * 1024（10MB） |
| 取值范围 | 1024 ~ 10 * 1024 * 1024 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# max_allowed_packet

最近更新时间：2025-04-22 15:40:03

## 功能描述

`max_allowed_packet` 用于设置 SQLEngine 和客户端在一次传送数据包的过程中所允许最大的数据包大小，单位为 Byte。当客户端发送的数据包大小超过 `max_allowed_packet` 的值时，SQLEngine 将拒绝该数据包。在调整 `max_allowed_packet` 参数时，需要考虑以下因素：

- 系统资源：根据服务器的实际资源情况来设置合适的值。在资源充足的情况下，可以适当增加 `max_allowed_packet` 的值以提高系统的并发处理能力。在资源紧张的情况下，需要减小 `max_allowed_packet` 的值以避免资源耗尽。
- 应用程序需求：根据应用程序的实际需求来设置合适的值。对于对数据包大小要求较高的应用程序，可能需要增加 `max_allowed_packet` 的值以满足并发需求。对于对数据包大小要求较低的应用程序，可以降低 `max_allowed_packet` 的值以节省系统资源。
- 数据库性能：根据数据库的实际性能情况来设置合适的值。对于性能较好的数据库，可能需要增加 `max_allowed_packet` 的值以提高数据库的并发处理能力。对于性能较差的数据库，需要减小 `max_allowed_packet` 的值以避免数据库过载。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 67108864（即64MB） |
| 取值范围 | • 1024 ~ 1073741824<br>• 该变量取值应为1024的整数倍。若取值无法被1024整除，则会提示 warning 并向下取整。例如设置为1025 时，则 TDStore 中的实际取值为 1024。 |
| 生效范围 | • SESSION：在 SESSION 作用域下，该变量为只读变量。<br>• GLOBAL |
| 是否需要重启 | 否 |

# max_connect_errors

最近更新时间：2025-04-22 15:40:03

## 功能描述

`max_connect_errors` 用于设置单个主机在被阻止（block）之前允许的最大中断连接数。这个参数主要用于防止恶意攻击、误操作或者网络故障导致的大量连接请求。当一个主机的连接错误次数超过 `max_connect_errors` 的值时，服务器会将该主机加入黑名单，拒绝其进一步的连接请求，直到达到 `connect_timeout` 设置的时间。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 2000 |
| 取值范围 | 1 ~ 18446744073709551615 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# max_connections

最近更新时间：2025-04-22 15:40:03

## 功能描述

`max_connections` 用于设置 SQLEngine 允许的最大连接数。当服务器的并发连接数超过 `max_connections` 的值时，服务器将拒绝新的连接请求。

在调整 `max_connections` 参数时，需要考虑以下因素：

- 系统资源：根据服务器的实际资源情况来设置合适的值。在资源充足的情况下，可以适当增加 `max_connections` 的值以提高系统的并发处理能力。在资源紧张的情况下，需要减小 `max_connections` 的值以避免资源耗尽。
- 应用程序需求：根据应用程序的实际需求来设置合适的值。对于高并发的应用程序，可能需要增加 `max_connections` 的值以满足并发需求。对于低并发的应用程序，可以降低 `max_connections` 的值以节省系统资源。
- 数据库性能：根据数据库的实际性能情况来设置合适的值。对于性能较好的数据库，可能需要增加 `max_connections` 的值以提高数据库的并发处理能力。对于性能较差的数据库，需要减小 `max_connections` 的值以避免数据库过载。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 2000 |
| 取值范围 | 1 ~ 100000 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# max_execution_time

最近更新时间：2025-04-22 15:40:03

## 功能描述

`max_execution_time` 用于设置 `SELECT` 语句的执行超时时间，单位为毫秒。当 `SELECT` 语句的执行时间超过 `max_execution_time` 设置的值时，查询将被中断，并返回一个错误。

`max_execution_time` 参数的应用场景：

- 对于只读 SELECT 语句， `max_execution_time` 参数适用于所有查询，除非查询中包含 `MAX_EXECUTION_TIME(N)` 优化器提示或 N 为0。
- 对于非只读 SELECT 语句（例如，调用具有副作用的存储函数）， `max_execution_time` 参数不适用。
- 对于存储程序中的 SELECT 语句， `max_execution_time` 参数不适用。

在调整 `max_execution_time` 参数时，需要考虑以下因素：

- 脚本执行时间：根据脚本的实际执行时间来设置合适的值。对于执行时间较长的脚本，可能需要增加 `max_execution_time` 的值以确保脚本能够正常运行。对于执行时间较短的脚本，可以降低 `max_execution_time` 的值以节省系统资源。
- 系统资源：根据服务器的实际资源情况来设置合适的值。在资源充足的情况下，可以适当增加 `max_execution_time` 的值以提高脚本执行效率。在资源紧张的情况下，需要减小 `max_execution_time` 的值以避免资源耗尽。
- 用户体验：根据用户的实际需求和体验来设置合适的值。对于对用户体验要求较高的应用程序，可能需要增加 `max_execution_time` 的值以确保脚本能够在合理的时间内完成执行。对于对用户体验要求较低的应用程序，可以降低 `max_execution_time` 的值以提高系统性能。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 0（表示没有限制） |
| 取值范围 | 0~18446744073709551615 |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# max_user_connections

最近更新时间：2025-04-22 15:40:03

## 功能描述

`max_user_connections` 用于设置单个用户最大连接数。当一个用户的并发连接数超过 `max_user_connections` 的值时，服务器将拒绝该用户的新连接请求。

在调整 `max_user_connections` 参数时，需要考虑以下因素：

- 系统资源：根据服务器的实际资源情况来设置合适的值。在资源充足的情况下，可以适当增加 `max_user_connections` 的值以提高系统的并发处理能力。在资源紧张的情况下，需要减小 `max_user_connections` 的值以避免资源耗尽。
- 应用程序需求：根据应用程序的实际需求来设置合适的值。对于高并发的应用程序，可能需要增加 `max_user_connections` 的值以满足并发需求。对于低并发的应用程序，可以降低 `max_user_connections` 的值以节省系统资源。
- 用户管理：根据用户的实际需求和权限来设置合适的值。对于具有特殊权限的用户，可能需要增加 `max_user_connections` 的值以满足其特殊需求。对于普通用户，可以降低 `max_user_connections` 的值以限制其资源使用。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 0（表示不限制连接数） |
| 取值范围 | 0～4294967295 |
| 生效范围 | • GLOBAL<br>• SESSION：在 SESSION 作用域下，该变量为只读变量。 |
| 是否可修改 | 可以通过 `SET GLOBAL` 修改未来新会话上该参数的值，但不能修改当前会话上该参数的值。 |
| 是否需要重启 | 否 |

# net_read_timeout

最近更新时间：2025-04-22 15:40:03

## 功能描述

`net_read_timeout` 用于设置连接中读数据时最大等待时长，单位为秒。当服务器在指定的时间内没有收到客户端发送的数据时，服务器将关闭与客户端的连接。

在调整 `net_read_timeout` 参数时，需要考虑以下因素：

- 网络环境：根据实际的网络环境和带宽来设置合适的值。在稳定的网络环境下，可以适当增加 `net_read_timeout` 的值以提高服务器的性能。在不稳定的网络环境下，需要减小 `net_read_timeout` 的值以避免过早地关闭与客户端的连接。
- 客户端响应速度：根据客户端的响应速度来设置合适的值。如果客户端响应较慢，可能需要增加 `net_read_timeout` 的值以避免过早地关闭与客户端的连接。如果客户端响应较快，可以降低 `net_read_timeout` 的值以提高服务器的性能。
- 服务器负载：根据服务器的实际负载情况来设置合适的值。在高负载情况下，可能需要增加 `net_read_timeout` 的值以提高服务器的性能。在低负载情况下，可以降低 `net_read_timeout` 的值以节省系统资源。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 30 |
| 取值范围 | 1～31536000（即1秒～1年） |
| 生效范围 | • SESSION<br>• GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# net_retry_count

最近更新时间：2025-04-22 15:40:03

## 功能描述

`net_retry_count` 用于设置服务器在尝试重新连接到客户端时的最大重试次数。当服务器在尝试重新连接到客户端时，如果连续失败达到 `net_retry_count` 的值，那么服务器将放弃重新连接，关闭与客户端的连接。
在调整 `net_retry_count` 参数时，需要考虑以下因素：

- 网络环境：根据实际的网络环境和带宽来设置合适的值。在稳定的网络环境下，可以适当减小 `net_retry_count` 的值以提高服务器的性能。在不稳定的网络环境下，需要增加 `net_retry_count` 的值以提高重新连接的成功率。
- 客户端响应速度：根据客户端的响应速度来设置合适的值。如果客户端响应较慢，可能需要增加 `net_retry_count` 的值以提高重新连接的成功率。如果客户端响应较快，可以降低 `net_retry_count` 的值以提高服务器的性能。
- 服务器负载：根据服务器的实际负载情况来设置合适的值。在高负载情况下，可能需要增加 `net_retry_count` 的值以提高重新连接的成功率。在低负载情况下，可以降低 `net_retry_count` 的值以节省系统资源。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 10 |
| 取值范围 | 0～18446744073709551615 |
| 生效范围 | • SESSION<br>• GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# net_write_timeout

最近更新时间：2025-04-22 15:40:03

## 功能描述

`net_write_timeout` 用于设置连接中写数据时最大等待时长，单位为秒。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 60 |
| 取值范围 | 1~31536000（即1秒~1年） |
| 生效范围 | • SESSION<br>• GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# optimizer_switch

最近更新时间：2025-04-22 15:40:03

## 功能描述

`optimizer_switch` 用于设置优化器指示选项开关。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Set |
| 默认值 | `index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,engine_condition_pushdown=on,index_condition_pushdown=on,mrr=on,mrr_cost_based=on,block_nested_loop=on,batched_key_access=off,materialization=on,semijoin=on,loosescan=on,firstmatch=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,lazy_index_strategy=off,limit_cond_pushdown=on,force_batched_key_access=false` |
| 取值范围 | • `index_merge`：启用或禁用索引合并优化。<br>• `index_merge_union`：启用或禁用索引合并中的 UNION 优化。<br>• `index_merge_sort_union`：启用或禁用索引合并中的排序 UNION 优化。<br>• `index_merge_intersection`：启用或禁用索引合并中的交集优化。<br>• `engine_condition_pushdown`：启用或禁用存储引擎条件下推优化。<br>• `index_condition_pushdown`：启用或禁用索引条件下推优化。<br>• `mrr`：启用或禁用多范围读取优化。<br>• `mrr_cost_based`：启用或禁用基于成本的多范围读取优化。<br>• `block_nested_loop`：启用或禁用块嵌套循环连接优化。<br>• `batched_key_access`：启用或禁用批量键访问优化。<br>• `materialization`：启用或禁用物化子查询优化。<br>• `semijoin`：启用或禁用半连接优化。<br>• `loosescan`：启用或禁用松散扫描优化。<br>• `firstmatch`：启用或禁用第一个匹配优化。<br>• `duplicateweedout`：启用或禁用重复剔除优化。 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| | • `subquery_materialization_cost_based`：启用或禁用基于成本的子查询物化优化。<br>• `use_index_extensions`：启用或禁用索引扩展优化。<br>• `condition_fanout_filter`：启用或禁用条件扇出过滤优化。该优化用于减少在多表连接时的中间结果集大小，从而提高查询性能。<br>• `derived_merge`：启用或禁用派生表合并优化。派生表合并优化会将派生表（子查询）合并到外部查询中，从而避免创建临时表。<br>• `use_invisible_indexes`：启用或禁用使用不可见索引。不可见索引在查询优化器中被忽略，但仍然存在于表中。禁用该选项可以确保查询优化器不会使用这些索引。<br>• `skip_scan`：启用或禁用跳过扫描优化。跳过扫描优化允许在索引扫描时跳过不必要的行，从而提高查询性能。<br>• `lazy_index_strategy`：启用或禁用懒惰索引策略。懒惰索引策略会延迟索引的使用，直到确定索引确实有助于查询性能。<br>• `limit_cond_pushdown`：启用或禁用 LIMIT 条件下推优化。该优化会将 LIMIT 条件下推到子查询或视图中，从而减少处理的数据量，提高查询性能。<br>• `force_batched_key_access`：将带索引的链接都转成 batch rpc 的 bka 连接，减小 RPC 的数据量，提升性能。 |
| 生效范围 | SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# sql_mode

最近更新时间：2025-04-22 15:40:03

## 功能描述

`sql_mode` 用于设置服务器的 SQL 模式。SQL 模式决定了服务器如何处理 SQL 语句的某些特性和行为，例如数据类型检查、空值处理、字符串比较等。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Set |
| 默认值 | `STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION`，多个 modes 间用逗号（`,`）间隔。 |
| 取值范围 | • `ALLOW_INVALID_DATES`：允许插入无效的日期，如 '2022-02-30'。<br>• `ANSI_QUOTES`：使用 ANSI 标准的引号（双引号）来引用标识符，而不是默认的反引号。<br>• `ERROR_FOR_DIVISION_BY_ZERO`：在执行除法操作时，如果除数为零，则抛出错误。<br>• `HIGH_NOT_PRECEDENCE`：使 `NOT` 运算符具有高优先级，与 `AND` 和 `OR` 运算符的优先级相同。<br>• `IGNORE_SPACE`：忽略标识符周围的空格。<br>• `NO_AUTO_VALUE_ON_ZERO`：在插入数据时，如果整数列的值为零，则不自动生成自增值。<br>• `NO_BACKSLASH_ESCAPES`：禁用反斜杠（`\`）作为转义字符。<br>• `NO_DIR_IN_CREATE`：禁止在 `CREATE TABLE` 语句中使用 `DIRECTORY` 选项。<br>• `NO_ENGINE_SUBSTITUTION`：如果指定的存储引擎不可用，则抛出错误，而不是使用默认的存储引擎。<br>• `NO_UNSIGNED_SUBTRACTION`：禁用无符号整数的减法。<br>• `NO_ZERO_DATE`：禁止使用零日期（'0000-00-00'）。<br>• `NO_ZERO_IN_DATE`：禁止在日期中使用零部分，如 '2022-00-01'。<br>• `ONLY_FULL_GROUP_BY`：要求 `GROUP BY` 子句中的所有非聚合列都必须在 `SELECT` 子句中出现。<br>• `PAD_CHAR_TO_FULL_LENGTH`：在插入数据时，将 `CHAR` 类型列的值填充到其定义的长度。 |

| | |
|---|---|
| | • `PIPES_AS_CONCAT`：将管道符（`\|\|`）视为字符串连接运算符，而不是逻辑或运算符。<br>• `REAL_AS_FLOAT`：将 REAL 数据类型视为 FLOAT 类型，而不是 DOUBLE 类型。<br>• `STRICT_ALL_TABLES`：启用所有严格模式选项，包括 `STRICT_TRANS_TABLES`。<br>• `STRICT_TRANS_TABLES`：在事务表中启用严格模式，要求插入、更新和删除操作遵循更严格的规则。 |
| 生效范围 | • SESSION<br>• GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# table_definition_cache

最近更新时间：2025-04-22 15:40:03

## 功能描述

`table_definition_cache` 用于设置表定义对象的缓存数量。表定义缓存存储了数据库中表的元数据（如列名、数据类型等），以便在查询执行过程中快速访问这些信息。当达到此限制时，系统会根据 LRU（最近最少使用）算法关闭最近最少使用的表定义，以便为新的表定义腾出空间。

> **说明：**
> 推荐使用 `table_open_cache` 替代 `table_definition_cache` 参数。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 400 |
| 取值范围 | 400～524288 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# table_open_cache

最近更新时间：2025-04-22 15:40:03

## 功能描述

`table_open_cache` 用于设置打开的表对象的缓存数量。当一个表被打开时，系统会将该表的结构信息存储在表缓存中，以便在后续的查询中重用。 `table_open_cache` 参数的值表示可以同时打开的表的最大数量。当达到此限制时，系统会根据 LRU（最近最少使用）算法关闭最近最少使用的表，以便为新的表腾出空间。

在调整 `table_open_cache` 参数时，需要考虑以下因素：

- 内存限制：增加 `table_open_cache` 参数的值可能会占用更多的内存。确保服务器具有足够的内存来容纳更大的表缓存。
- 表的数量和使用频率：根据数据库中表的数量和使用频率来调整 `table_open_cache` 参数。如果数据库中有大量的表，并且这些表经常被访问，那么可以适当增加 `table_open_cache` 参数的值。
- 监控和调优：监控服务器的性能指标，如 CPU 使用率、内存使用率和磁盘 I/O，以确定是否需要调整 `table_open_cache` 参数。如果发现性能瓶颈，可以尝试调整该参数以优化性能。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 10240 |
| 取值范围 | 1 ~ 524288 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_auto_increment_batch_size

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_auto_increment_batch_size` 用于设置 SQLEngine 单次获取自增值的数量。自增值生成序列存放在存储节点上， 在某些情况下，为了提高插入性能，系统可以一次为多个插入操作分配连续的自增值，从而减少自增值生成器的开销。通过调整 `tdsql_auto_increment_batch_size` 参数，可以控制这种批量分配的大小。在调整 `tdsql_auto_increment_batch_size` 参数时，需要考虑以下因素：

- 插入性能：增加 `tdsql_auto_increment_batch_size` 参数的值可能会提高插入性能，特别是在批量插入数据时。然而，过大的值可能会导致自增值耗尽的风险增加。
- 自增值耗尽： `tdsql_auto_increment_batch_size` 参数的值不应该设置得过大，以避免自增值耗尽的风险。例如，如果自增列的数据类型为 INT，最大值为 $2^{31}-1$，那么 `tdsql_auto_increment_batch_size` 参数的最大值应该小于 $2^{31}$/ 插入操作数。
- 监控和调优：监控服务器的性能指标，如 CPU 使用率、内存使用率和磁盘 I/O，以确定是否需要调整 `tdsql_auto_increment_batch_size` 参数。如果发现性能瓶颈，可以尝试调整该参数以优化性能。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | unsigned int |
| 默认值 | 1 |
| 取值范围 | [1, 10000] |
| 生效范围 | GLOBAL READ_ONLY |
| 是否可修改 | 是 |
| 是否需要重启 | 是 |

# tdsql_bulk_commit_records

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_bulk_commit_records` 用于设置 DDL 过程中拆分出的子事务达到提交条件的数据条数阈值。当达到此限制时，系统会自动提交事务，以便为新的记录腾出空间。

在调整 `tdsql_bulk_commit_records` 参数时，需要考虑以下因素：

- 事务性能：增加 `tdsql_bulk_commit_records` 参数的值可能会提高批量插入、更新和删除操作的性能。然而，过大的值可能会导致事务过大，从而影响系统性能和稳定性。
- 监控和调优：监控服务器的性能指标，如 CPU 使用率、内存使用率和磁盘 I/O，以确定是否需要调整 `tdsql_bulk_commit_records` 参数。如果发现性能瓶颈，可以尝试调整该参数以优化性能。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 3000 |
| 取值范围 | 1 ~ 18446744073709551615 |
| 生效范围 | • SESSION<br>• GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_cache_range_max_use_cnt

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_cache_range_max_use_cnt` 用于设置缓存的 range 信息失效前被使用的最大次数。缓存 range 信息可以减少对磁盘的访问次数，从而提高查询性能。当达到此限制时，系统会根据 LRU（最近最少使用）算法移除最近最少使用的数据块，以便为新的数据块腾出空间。

在调整 `tdsql_cache_range_max_use_cnt` 参数时，需要考虑以下因素：

- 缓存命中率：增加 `tdsql_cache_range_max_use_cnt` 参数的值可能会提高范围查询缓存的命中率，从而提高查询性能。然而，过大的值可能会导致缓存中的数据块过期得不够快，从而降低缓存的效果。
- 监控和调优：监控服务器的性能指标，如 CPU 使用率、内存使用率和磁盘 I/O，以确定是否需要调整 `tdsql_cache_range_max_use_cnt` 参数。如果发现性能瓶颈，可以尝试调整该参数以优化性能。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 100000 |
| 取值范围 | 0 ~ 18446744073709551615 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_get_index_stats_from_tdstore

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_get_index_stats_from_tdstore` 用于设置是否允许 SQLEngine 从 TDStore 获取统计信息。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | ON |
| 取值范围 | • ON<br>• OFF |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_max_collect_ranges

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_max_collect_ranges` 表示为获取统计信息所允许涉及的 range 数量。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 1 |
| 取值范围 | 1 ~ 4294967295 |
| 生效范围 | GLOBAL |
| 是否可修改 | 否 |
| 是否需要重启 | 是 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_max_retry

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_max_retry` 表示允许失败（网络/RPC 等）重试的最大次数。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 40 |
| 取值范围 | 0 ~ 4294967295 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_optimize_pk_point_select

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_optimize_pk_point_select` 表示优化器是否对主键点查询进行优化。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | ON |
| 取值范围 | • ON<br>• OFF |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_optimize_table_rows_count

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_optimize_table_rows_count` 表示是否开启 COUNT 优化。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | ON |
| 取值范围 | • ON<br>• OFF |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_retry_interval_growth

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_retry_interval_growth` 表示 SQLEngine 与其他节点 RPC 交互的重试间隔的增长倍速。默认为2，表示下一次重试间隔是这一次重试间隔的2倍，即1ms、2ms、4ms、8ms、16ms、...、512ms。如果将 `tdsql_retry_interval_growth` 修改为3，那重试间隔3倍增长，即1ms、3ms、9ms、...，直到重试间隔达到 `tdsql_retry_interval_msec_max` 毫秒的上限。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | uint32 |
| 默认值 | 2 |
| 取值范围 | 0～4294967295 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_retry_interval_msec_max

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_retry_interval_msec_max` 表示两次失败（网络/RPC 等）重试之间最大间隔时间，单位为毫秒。当一个 SQL 查询失败时，可能是由于网络延迟、数据库节点故障或其他原因导致的。为了确保查询能够成功执行，系统可以选择重试失败的查询。 `tdsql_retry_interval_msec_max` 参数用于限制重试间隔的最大值，以便在不影响系统性能的情况下，尽可能快地恢复服务。
具体来说， `tdsql_retry_interval_msec_max` 参数定义了重试间隔的最大值。例如，如果设置为1000毫秒（1秒），那么在重试过程中，重试间隔将不会超过1秒。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 1000 |
| 取值范围 | 0 ~ 4294967295 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_retry_interval_msec_min

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_retry_interval_msec_min` 表示两次失败（网络/RPC 等）重试之间最小间隔时间，单位为毫秒。当一个 SQL 查询失败时，可能是由于网络延迟、数据库节点故障或其他原因导致的。为了确保查询能够成功执行，系统可以选择重试失败的查询。 `tdsql_retry_interval_msec_min` 参数用于设置重试间隔的最小值，以便在不影响系统性能的情况下，尽可能快地恢复服务。
具体来说， `tdsql_retry_interval_msec_min` 参数定义了重试间隔的最小值。例如，如果设置为100毫秒，那么在重试过程中，重试间隔将至少为100毫秒。这样可以确保在重试过程中，重试间隔不会过短，从而避免过于频繁的重试导致系统资源耗尽。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 2 |
| 取值范围 | 0 ~ 4294967295 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_stale_read_role

最近更新时间：2025-06-13 20:31:03

## 功能描述

设置 stale read 路由访问的副本。

- 0：leader。
- 1：follower。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | UINT |
| 默认值 | 0 |
| 取值范围 | [0, 1] |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_skip_trans_write_lock

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_skip_trans_write_lock` 表示在执行写操作的时候，是否跳过获取写锁。
为了保证数据的一致性和完整性，通常需要对写操作进行加锁。这样可以确保在多个事务同时访问同一数据时，写操作不会相互干扰，从而避免数据不一致的问题。然而，在某些场景下，为了提高性能，可以选择跳过事务写锁的获取，以减少锁竞争和锁等待的开销。
跳过事务写锁的获取可能会导致数据不一致的问题。因此，在使用 `tdsql_skip_trans_write_lock` 参数时，需要仔细评估系统的需求和场景，确保在跳过锁定的情况下，系统仍能保证数据的一致性和完整性。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | OFF |
| 取值范围 | • ON<br>• OFF |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_split_trans_during_alter_copy

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_split_trans_during_alter_copy` 表示是否在 COPY 算法的 DDL 过程中将大事务拆成小事务。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | ON |
| 取值范围 | • ON<br>• OFF |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_transaction_max_participants

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_transaction_max_participants` 表示单个事务最大的参与者（ReplicationGroup）数量。例如，如果设置为100，那么在一个事务中，最多只能涉及100个参与者。这样可以确保事务的规模在可控范围内，避免因事务过大导致的性能问题和资源消耗。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 10000 |
| 取值范围 | 1 ~ 4294967295 |
| 生效范围 | GLOBAL |
| 是否可修改 | 否 |
| 是否需要重启 | 是 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_txn_keep_alive_lease_sec

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_txn_keep_alive_lease_sec` 表示 TDStore 超过 `tdsql_txn_keep_alive_lease_sec` 秒没有收到某事务的心跳，将此事务清理。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 600 |
| 取值范围 | 1 ~ 4294967295 |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_update_local_glsv_interval

最近更新时间：2025-04-22 15:40:03

## 功能描述

GLSV 是 Global Largest Schema Version 的缩写，用于在多节点之间同步 DDL 结果，保证分布式系统中的多个节点不会使用过旧的表结构和写入不符合最新定义的数据。DDL 推进 GLSV，并等待 `tdsql_update_local_glsv_interval` 毫秒，在这段时间内，其他节点应该知道了 GLSV 版本变化，并由此更新 SQL 使用的表版本。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 100 |
| 取值范围 | 0 ~ 1000 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_update_variables_interval

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_update_variables_interval` 表示从 MC 获取持久化变量的时间间隔，单位为毫秒。例如，如果设置为60000毫秒（1分钟），那么在一个数据库节点上，系统会每隔1分钟更新一次系统变量。这样可以确保系统变量的实时性和一致性，避免因变量不一致而导致的错误和性能下降。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 2000 |
| 取值范围 | 1 ~ 3600000 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_use_tdstore_estimated_rows

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_use_tdstore_estimated_rows` 表示是否使用 TDStore 提供的数据行数估算值来优化查询计划。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | ON |
| 取值范围 | • ON<br>• OFF |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_lock_wait_timeout

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdsql_lock_wait_timeout` 表示 TDStore 引擎锁超时时间，单位为毫秒。
在数据库系统中，为了保证数据的一致性和完整性，系统需要对数据进行协调和管理。锁是一种用于确保数据一致性的机制，它可以防止多个事务同时访问和修改同一份数据，从而导致数据不一致。
例如，如果设置为10000毫秒（10秒），那么在一个事务试图获取锁时，如果锁已经被其他事务持有，系统会等待最多10秒。如果在这段时间内锁没有被释放，系统会抛出一个超时异常，表示等待锁超时。这样可以确保数据的一致性和完整性，避免因锁等待超时而导致的错误和性能下降。
同时该参数受到参数 `tdsql_tdstore_rpc_timeout` 的限制，默认会将 `tdsql_lock_wait_timeout` 限制在小于该参数的值，例如 `tdsql_tdstore_rpc_timeout` 值为100000ms (100s)，`tdsql_lock_wait_timeout` 值为200s，那么默认会把 `tdsql_lock_wait_timeout` 的值降低至小于tdsql_tdstore_rpc_timeout。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 10 |
| 取值范围 | 1 ~ 1073741824 |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# wait_timeout

最近更新时间：2025-04-22 15:40:03

## 功能描述

`wait_timeout` 表示连接中断等待时长，单位为秒。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 28800 |
| 取值范围 | 1 ~ 31536000（即1秒 ~ 1年） |
| 生效范围 | • GLOBAL<br>• SESSION |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# tdsql_recycle_bin_enabled

最近更新时间：2025-04-22 15:40:03

## 功能描述

是否开启回收站，默认关闭。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | bool |
| 默认值 | false |
| 取值范围 | [false, true] |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# log_service_mysql_client_timeout

最近更新时间：2025-04-22 15:40:03

## 功能描述

LogService 控制参数，控制 MySQL 类型的 LogService 同步时内部 Mysql Client 连接的超时时间。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | uint |
| 默认值 | 7200 |
| 取值范围 | [3, 4294967295] |
| 生效范围 | GLOBAL |
| 是否可修改 | 否 |
| 是否需要重启 | 是 |

# log_service_mysql_info_queue_size

最近更新时间：2025-04-22 15:40:03

## 功能描述

LogService 控制参数，控制 MySQL 类型的 LogService 同步时，内部使用的缓存队列的长度。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ulonglong |
| 默认值 | 1000000 |
| 取值范围 | [1, 18446744073709551615] |
| 生效范围 | GLOBAL |
| 是否可修改 | 否 |
| 是否需要重启 | 是 |

# audit_log_policy

最近更新时间：2025-04-22 15:40:03

## 功能描述

控制审计日志的记录策略，决定系统记录哪些操作到审计日志中。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Enum |
| 默认值 | NONE |
| 取值范围 | • ALL：记录所有操作。<br>• NONE：不记录。<br>• LOGINS：仅记录登录事件。<br>• QUERIES：记录所有查询操作。 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# log_bin_trust_function_creators

最近更新时间：2025-04-22 15:40:03

## 功能描述

控制存储函数/触发器的创建限制，确保启用二进制日志（Binary Log）时，非确定性操作不会导致主从数据不一致或安全风险。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | OFF |
| 取值范围 | • ON：信任<br>• OFF：不信任 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# max_prepared_stmt_count

最近更新时间：2025-04-22 15:40:03

## 功能描述

限制服务器中预处理语句（Prepared Statements）的总数量，防止恶意攻击或异常操作导致内存耗尽。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Integer |
| 默认值 | 16382 |
| 取值范围 | 0 ~ 4194304 |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_enable_trigger

最近更新时间：2025-04-22 15:40:03

## 功能描述

是否允许 CREATE/DROP/ALTER 触发器。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Bool |
| 默认值 | false |
| 取值范围 | true,false |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_enable_view

最近更新时间：2025-04-22 15:40:03

## 功能描述

是否允许 CREATE/DROP/ALTER 视图

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Bool |
| 默认值 | true |
| 取值范围 | true,false |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdsql_use_online_copy_ddl

最近更新时间：2025-04-22 15:40:03

## 功能描述

COPY DDL（如 drop column, drop primary key 等）是否使用 Online COPY 方式。

- 此参数为 true 时， COPY DDL 使用 Online 方式，允许 DML 访问正在做 DDL 的表。
- 此参数为 false 时， COPY DDL 锁表，DML 等待 DDL 完成才可访问表。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Bool |
| 默认值 | true |
| 取值范围 | true,false |
| 生效范围 | SESSION,GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

版权所有：腾讯云计算（北京）有限责任公司

# tdstore_bulk_load_txn_rpc_timeout_threshold_ms

最近更新时间：2025-04-22 15:40:03

## 功能描述

Bulkload 事务 commit 的 RPC 超时，单位毫秒 ms。
3.0可以采取 Bulkload 的方式批量导入数据，导入事务一般数据量较大，提交该事务可能花费较多时间，需区别于其他 RPC 用法，所以为此 commit RPC 提供单独的参数控制。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | int32 |
| 默认值 | 20000 |
| 取值范围 | 1 ~ int32_max |
| 生效范围 | GLOBAL |
| 是否可修改 | 是 |
| 是否需要重启 | 否 |

# 存储引擎参数

# 存储引擎参数概述

最近更新时间：2025-04-22 15:40:03

通过存储引擎参数，您可以深入了解存储引擎运行状态，如 TDStore 数据存储的路径、日志路径、Raft 相关数据路径，以及当前实例的写入/读取状态等。

## 存储引擎参数分类

存储引擎参数均为 **GLOBAL 变量**，表示数据库同一实例内的不同用户共享全局变量。全局变量的修改不会随会话的退出而失效。此外，全局变量修改后，对当前已打开的 Session 不生效，需要重新建立 Session 才能生效。

## 查询存储引擎参数

- **方式一：**使用 `SHOW GLOBAL VARIABLES` 语句查询。

```
SHOW GLOBAL VARIABLES LIKE 'tdstore_bulk_load_merge_chunk_size';
```

- **方式二：**使用 `SELECT @@global.xxxxx;` 语句查询。

```
SELECT @@global.tdstore_bulk_load_merge_chunk_size;
```

- **方式三：**通过**参数配置** > **数据库参数**页面查询。

  1.1 登录 TDSQL MySQL 版控制台 。

  1.2 在**实例列表**页面，选择所需实例 ID。

  1.3 进入**参数配置** > **数据库参数**页面，搜索并查看计算引擎参数。

## 修改存储引擎参数

存储引擎参数对用户使用体验和性能有显著影响，目前暂不支持用户自行修改。如有修改需求，请联系腾讯云技术支持工程师。

版权所有：腾讯云计算（北京）有限责任公司

# 存储引擎参数总览

最近更新时间：2025-04-22 15:40:03

| 参数 | 说明 |
| --- | --- |
| tdstore_bulk_load_total_merge_buffer_size | 用于设置在 Bulk Load 事务中进行外部合并排序时，内存缓冲区的总大小，单位为字节。 |
| tdstore_bulk_load_merge_chunk_size | 用于设置在 Bulk Load 事务中，对无序数据进行外部合并排序之前，用于在内存中缓存数据的内存大小。 |
| tdstore_enter_readonly_threshold | 用于设置 TDStore 进入只读模式的阈值。当空间使用率高于该阈值时，TDStore 将自动切换到只读模式。默认值为99%，即当可用空间少于总空间的1%时，TDStore 将进入只读模式。 |
| tdstore_leave_readonly_threshold | 用于设置 TDStore 退出只读模式的阈值。默认值为95%，表示当空间使用率低于 95%时，TDStore 将自动退出只读模式。 |
| tdstore_long_query_time | 用于设置记录长时间运行的 RPC 请求的阈值，单位为秒。执行时间超过 `tdstore_long_query_time` 秒的 RPC 请求将被记录到慢查询日志中。 |
| tdstore_long_query_with_detail_stage | 用于启用或禁用 TDStore 慢查询日志的详细阶段记录。 |
| tdstore_max_db_log_file_num | 用于设置 TDStore 打印日志文件的最大数量。当 TDStore 打印日志数量超过该阈值后，最老的打印日志文件会被自动删除。 |
| tdstore_max_db_log_file_size | 用于设置 TDStore 每个打印日志文件的大小，当一个打印日志文件写满时，会重新开启一个新的打印日志文件继续追加写入。 |
| tdstore_max_txn_size | 用于设置单个事务的内存大小限制。 |
| tdstore_min_free_disk_space | 用于设置启动 TDStore 或从磁盘空间不足错误中恢复所需的最小磁盘空间。 |
| tdstore_raft_data_path | 用于指定 TDStore 中存放 Raft 数据的存储路径。默认为空，表示将 Raft 数据存放在 `tdstore_data_path` 路径的 `raft_data` 子目录下。 |
| tdstore_txn_max_entry_size | 用于设置 TDStore 事务中，每条记录编码为 TDStore 内部存储格式后的最大大小。当 SQL 语句中单条记录在编码后超过该阈值时，语句会执行失败。 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| tdstore_user_cf_hard_pending_compaction_bytes_limit | 用于设置 TDStore 进入停写状态的阈值，当 TDStore 待 compaction 的数据量超过该参数时，TDStore 会进入停写状态，拒绝所有写入。 |
| tdstore_user_cf_level0_slowdown_writes_trigger | 用于设置 TDStore 进入缓写状态的阈值，当位于 LSM-tree level 0层的数据文件数量超过该参数时，TDStore 会进入缓写状态，对所有写入请求进行自适应地限速。 |
| tdstore_user_cf_level0_stop_writes_trigger | 用于设置 TDStore 进入停写状态的阈值，当位于 LSM-tree level 0层的数据文件数量超过该参数时，TDStore 会进入停写状态，拒绝所有写入。 |
| tdstore_user_cf_soft_pending_compaction_bytes_limit | 用于设置 TDStore 进入缓写状态的阈值，当 TDStore 待 compaction 的数据量超过该参数时，TDStore 会进入缓写状态，对所有写入请求进行自适应地限速。 |
| tdstore_deadlock_detect | 是否启用死锁检测。 |
| tdstore_force_accept_conf_log | 是否强制节点在满磁盘状态下接收 Raft Leader 发送的 configuration log。<br>默认为 `true` ，确保多数派进入满磁盘状态后可以快速完成成员变更。 |
| tdstore_enable_node_alive_detection | 控制开启或关闭宕机检测功能。 |
| tdstore_deadlock_victim | 当死锁发生时，决定选取哪个事务回滚。 |
| tdstore_transfer_leader_job_cost_threshold_ms | 该参数用于设定主动切主任务耗时（包含旧主迁移事务以及给新主发请求的耗时）的最大阈值。 |
| tdstore_transfer_leader_raft_cost_threshold_ms | 该参数用于设定 raft 层切主耗时（包含新主选举以及上任的耗时）的最大阈值。 |
| tdstore_data_import_rate_limiter_bytes_per_sec | Fast Online DDL ingest behind 模式的磁盘吞吐量限流参数，用于限制数据回填阶段消耗的读/写吞吐量，即读或写所用的吞吐量都不会超过这个值，该参数会影响 Fast Online DDL 的执行效率。 |

# tdstore_bulk_load_total_merge_buffer_size

最近更新时间：2025-04-22 15:40:03

## 功能描述

Bulk Load 数据导入场景，通常每个 Bulk Load 事务的数据量会比较大，大概在几百 MB 到 GB 的数量级。因此一个 Bulk Load 事务中的数据在提交之前，也会保存在一些临时数据文件中，从而减轻和控制内存开销。对于 Bulk Load 事务中的无序数据（例如二级索引数据），在事务提交阶段会进行外部合并排序。`tdstore_bulk_load_total_merge_buffer_size` 用于设置在 Bulk Load 事务中进行外部合并排序时，内存缓冲区的总大小，单位为字节。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 268435456字节（256 MB） |
| 取值范围 | [4194304, 18446744073709551615] |
| 生效范围 | GLOBAL |

# tdstore_bulk_load_merge_chunk_size

最近更新时间：2025-04-22 15:40:03

## 功能描述

Bulk Load 数据导入场景，通常每个 Bulk Load 事务的数据量会比较大，大概在几百 MB 到 GB 的数量级。因此一个 Bulk Load 事务中的数据在提交之前，也会保存在一些临时数据文件中，从而减轻和控制内存开销。对于 Bulk Load 事务中的无序数据（例如二级索引数据），在事务提交阶段会进行外部合并排序。`tdstore_bulk_load_merge_chunk_size` 用于设置在 Bulk Load 事务中，对无序数据进行外部合并排序之前，用于在内存中缓存数据的内存大小，单位为字节。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 134217728字节（128MB） |
| 取值范围 | [1048576, 18446744073709551615] |
| 生效范围 | GLOBAL |

# tdstore_deadlock_detect

最近更新时间：2025-04-22 15:40:03

## 功能描述

是否启用死锁检测。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | false |
| 取值范围 | • true<br>• false |
| 生效范围 | GLOBAL |

版权所有：腾讯云计算（北京）有限责任公司

# tdstore_enter_readonly_threshold

最近更新时间：2025-04-22 15:40:03

## 功能描述

`tdstore_enter_readonly_threshold` 参数用于设置 TDStore 进入只读模式的阈值。当空间使用率高于该阈值时，TDStore 将自动切换到只读模式。默认值为 99%，即当可用空间少于总空间的 1% 时，TDStore 将进入只读模式。
在只读模式下，TDStore 不允许进行数据写入操作，但仍然支持数据读取。如果需要恢复到可写模式，可以手动操作或等待空间使用率低于 `tdstore_leave_readonly_threshold` 后，TDStore 将自动退出只读模式。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | String |
| 默认值 | 99% |
| 取值范围 | N/A |
| 生效范围 | GLOBAL |

版权所有：腾讯云计算（北京）有限责任公司

# tdstore_force_accept_conf_log

最近更新时间：2025-04-22 15:40:04

## 功能描述

是否强制节点在满磁盘状态下接收 Raft Leader 发送的 configuration log。
默认为 `true` ，确保多数派进入满磁盘状态后可以快速完成成员变更。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | true |
| 取值范围 | • true<br>• false |
| 生效范围 | GLOBAL |

# tdstore_leave_readonly_threshold

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_leave_readonly_threshold` 用于设置 TDStore 退出只读模式的阈值。默认值为95%，表示当空间使用率低于95%时，TDStore 将自动退出只读模式。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | String |
| 默认值 | 95% |
| 取值范围 | N/A |
| 生效范围 | GLOBAL |

# tdstore_long_query_time

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_long_query_time` 参数用于设置记录长时间运行的 RPC 请求的阈值，单位为秒。执行时间超过 `tdstore_long_query_time` 秒的 RPC 请求将被记录到慢查询日志中。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | DOUBLE |
| 默认值 | 0.5 |
| 取值范围 | [0, 1000] |
| 生效范围 | GLOBAL |

版权所有：腾讯云计算（北京）有限责任公司

# tdstore_long_query_with_detail_stage

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_long_query_with_detail_stage` 用于启用或禁用 TDStore 慢查询日志的详细阶段记录。启用此功能后，慢查询日志将包含更精确的统计信息，但可能会带来一定的性能损失。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | Boolean |
| 默认值 | OFF |
| 取值范围 | • ON<br>• OFF |
| 生效范围 | GLOBAL |

# tdstore_max_db_log_file_num

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_max_db_log_file_num` 用于设置 TDStore 打印日志文件的最大数量。当 TDStore 打印日志数量超过该阈值后，最老的打印日志文件会被自动删除。通过合理设置 `tdstore_max_db_log_file_num` ，可以控制 TDStore 打印日志文件的数量，从而避免日志文件过多占用存储空间。根据存放打印日志文件路径（tdstore_log_path）对应的磁盘存储容量，配置合适的数量。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 200 |
| 取值范围 | [1, 18446744073709551615] |
| 生效范围 | GLOBAL |

# tdstore_max_db_log_file_size

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_max_db_log_file_size` 用于设置 TDStore 每个打印日志文件的大小，当一个打印日志文件写满时，会重新开启一个新的打印日志文件继续追加写入。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 134217728字节（128MB） |
| 取值范围 | [8388608, 18446744073709551615] |
| 生效范围 | GLOBAL |

# tdstore_max_txn_size

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_max_txn_size` 用于设置单个事务的内存大小限制。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 1073741824字节（1GB） |
| 取值范围 | [67108864, 18446744073709551615] |
| 生效范围 | GLOBAL |

版权所有：腾讯云计算（北京）有限责任公司

# tdstore_min_free_disk_space

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_min_free_disk_space` 用于设置启动 TDStore 或从磁盘空间不足错误中恢复所需的最小磁盘空间。
通过合理设置 `tdstore_min_free_disk_space` ，可以确保系统在启动或重启时有足够的磁盘空间，从而避免因磁盘空间不足导致的启动失败。根据系统需求和磁盘容量，调整该参数以达到最佳效果。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 4294967296字节（4GB） |
| 取值范围 | [67108864, 18446744073709551104] |
| 生效范围 | GLOBAL |

# tdstore_raft_data_path

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_raft_data_path` 参数用于指定 TDStore 中存放 Raft 数据的存储路径。默认为空，表示将 Raft 数据存放在 `tdstore_data_path` 路径的 `raft_data` 子目录下。
配置时，请确保指定的路径有足够的存储空间和适当的权限，以便 Raft 数据能够正确存储和访问。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | String |
| 默认值 | N/A |
| 取值范围 | N/A |
| 生效范围 | GLOBAL |

# tdstore_txn_max_entry_size

最近更新时间：2025-04-22 15:40:04

## 功能描述

`tdstore_txn_max_entry_size` 用于设置 TDStore 事务中，每条记录编码为 TDStore 内部存储格式后的最大大小。当 SQL 语句中单条记录在编码后超过该阈值时，语句会执行失败。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 67108864字节（64MB） |
| 取值范围 | [1024, 67108864] |
| 生效范围 | GLOBAL |

# tdstore_user_cf_hard_pending_compaction_bytes_limit

最近更新时间：2025-04-22 15:40:04

## 功能描述

TDStore 使用 LSM-tree 结构来组织和存储数据。LSM-tree 将数据按层进行组织，分为 level 0~ level N 层。最新写入的数据，会首先落盘到 level 0层的数据文件。LSM-tree 会不定期地发起 compaction 操作，对存储的数据进行重新组织，将数据逐渐从上层（level i）移动到底层（level i+1），从而维持 LSM-tree 的良好形态。在这个过程中，会清理掉过期版本的数据，从而释放冗余的磁盘空间，以达到最佳的读写性能。当数据库的写入压力过大，compaction 操作来不及将最新写入的数据进行重新组织时，LSM-tree 形态会变差，读写性能下降。达到一定阈值后，TDStore 会自动进入缓写（对写入请求进行限速），甚至停写（拒绝所有写入请求）状态。此时 TDStore 的写入性能达到瓶颈。

`tdstore_user_cf_hard_pending_compaction_bytes_limit` 用于设置 TDStore 进入停写状态的阈值，当 TDStore 待 compaction 的数据量超过该参数时，TDStore 会进入停写状态，拒绝所有写入。通过合理设置 `tdstore_user_cf_hard_pending_compaction_bytes_limit`，可以控制进入停写状态的阈值，从而优化系统性能和资源使用。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 549755813888字节（512GB） |
| 取值范围 | [1, 18446744073709551615] |
| 生效范围 | GLOBAL |

# tdstore_user_cf_level0_slowdown_writes_trigger

最近更新时间：2025-04-22 15:40:04

## 功能描述

TDStore 使用 LSM-tree 结构来组织和存储数据。LSM-tree 将数据按层进行组织，分为 level 0~ level N 层。最新写入的数据，会首先落盘到 level 0层的数据文件。LSM-tree 会不定期地发起 compaction 操作，对存储的数据进行重新组织，将数据逐渐从上层（level i）移动到底层（level i+1），从而维持 LSM-tree 的良好形态。在这个过程中，会清理掉过期版本的数据，从而释放冗余的磁盘空间，以达到最佳的读写性能。当数据库的写入压力过大，compaction 操作来不及将最新写入的数据进行重新组织时，LSM-tree 形态会变差，读写性能下降。达到一定阈值后，TDStore 会自动进入缓写（对写入请求进行限速），甚至停写（拒绝所有写入请求）状态。此时 TDStore 的写入性能达到瓶颈。

`tdstore_user_cf_level0_slowdown_writes_trigger` 用于设置 TDStore 进入缓写状态的阈值，当位于 LSM-tree level 0层的数据文件数量超过该参数时，TDStore 会进入缓写状态，对所有写入请求进行自适应地限速。通过合理设置 `tdstore_user_cf_level0_slowdown_writes_trigger`，可以控制进入缓写状态的阈值，从而优化系统性能和资源使用。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | UINT |
| 默认值 | 64 |
| 取值范围 | [1, 2147483647] |
| 生效范围 | GLOBAL |

# tdstore_user_cf_level0_stop_writes_trigger

最近更新时间：2025-04-22 15:40:04

## 功能描述

TDStore 使用 LSM-tree 结构来组织和存储数据。LSM-tree 将数据按层进行组织，分为 level 0 ~ level N 层。最新写入的数据，会首先落盘到 L0 层。LSM-tree 会不定期地发起 compaction 操作，对存储的数据进行重新组织，将数据逐渐从上层移动到底层，从而维持 LSM-tree 的良好形态。在这个过程中，会清理掉过期版本的数据，从而释放冗余的磁盘空间，以达到最佳的读写性能。当数据库的写入压力过大，compaction 操作来不及将最新写入的数据进行重新组织时，LSM-tree 形态会变差，读写性能下降。达到一定阈值后，TDStore 会自动进入缓写（对写入请求进行限速），甚至停写（拒绝所有写入请求）状态。此时 TDStore 的写入性能达到瓶颈。`tdstore_user_cf_level0_stop_writes_trigger` 用于设置 TDStore 进入停写状态的阈值，当位于 LSM-tree level 0层的数据文件数量超过该参数时，TDStore 会进入停写状态，拒绝所有写入。通过合理设置 `tdstore_user_cf_level0_stop_writes_trigger`，可以控制进入停写状态的阈值，从而优化系统性能和资源使用。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | UINT |
| 默认值 | 128 |
| 取值范围 | [1, 2147483647] |
| 生效范围 | GLOBAL |

# tdstore_user_cf_soft_pending_compaction_bytes_limit

最近更新时间：2025-04-22 15:40:04

## 功能描述

TDStore 使用 LSM-tree 结构来组织和存储数据。LSM-tree 将数据按层进行组织，分为 level 0~ level N 层。最新写入的数据，会首先落盘到 level 0层的数据文件。LSM-tree 会不定期地发起 compaction 操作，对存储的数据进行重新组织，将数据逐渐从上层（level i）移动到底层(level i+1)，从而维持 LSM-tree 的良好形态。在这个过程中，会清理掉过期版本的数据，从而释放冗余的磁盘空间，以达到最佳的读写性能。当数据库的写入压力过大，compaction 操作来不及将最新写入的数据进行重新组织时，LSM-tree 形态会变差，读写性能下降。达到一定阈值后，TDStore 会自动进入缓写（对写入请求进行限速），甚至停写（拒绝所有写入请求）状态。此时 TDStore 的写入性能达到瓶颈。

`tdstore_user_cf_soft_pending_compaction_bytes_limit` 用于设置 TDStore 进入缓写状态的阈值，当 TDStore 待 compaction 的数据量超过该参数时，TDStore 会进入缓写状态，对所有写入请求进行自适应地限速。通过合理设置 `tdstore_user_cf_soft_pending_compaction_bytes_limit` ，可以控制进入缓写状态的阈值，从而优化系统性能和资源使用。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ULONGLONG |
| 默认值 | 137438953472 字节（128 GB） |
| 取值范围 | [1, 18446744073709551615] |
| 生效范围 | GLOBAL |

# tdstore_enable_node_alive_detection

最近更新时间：2025-04-22 15:40:04

## 功能描述

控制开启或关闭宕机检测功能。
启用该功能后，每个 TDStore 节点会和其他所有 TDStore 节点建立长连接。当长连接中断后，如果若干次重试仍失败，则会认为远端 TDStore 节点宕机，此时会触发宕机回调，执行一些清理工作（例如释放残留的参与者上下文）。
启用该功能的负面影响在于，当节点压力非常大的时候，理论上也可能导致连接中断，因此有造成误杀事务的风险。此外，如果节点数量特别多，也会导致连接数量变多。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | BOOL |
| 默认值 | ON |
| 取值范围 | ON/OFF |
| 生效范围 | GLOBAL |

# tdstore_deadlock_victim

最近更新时间：2025-04-22 15:40:04

## 功能描述

当死锁发生时，决定选取哪个事务回滚。

- 当设置为 `"WRITE_LEAST"` 时，优先选取写数据量较少的事务回滚。
- 当设置为 `"START_LATEST"` 时，优先选取较晚开启的事务回滚。

建议使用 `SET PERSIST` 语句以设置该参数，以保证所有节点的配置被同步变更。只有负责检测死锁的节点的该参数值实际影响选取回滚事务的行为，其它节点的该参数值对选取回滚事务的行为不构成影响。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ENUM |
| 默认值 | "WRITE_LEAST" |
| 取值范围 | "WRITE_LEAST" / "START_LATEST" |
| 生效范围 | GLOBAL |

# tdstore_transfer_leader_job_cost_threshold_ms

最近更新时间：2025-04-22 15:40:04

## 功能描述

该参数用于设定主动切主任务耗时（包含旧主迁移事务以及给新主发请求的耗时）的最大阈值。
超过该阈值后，被判定为耗时异常，会对 mysql status 指标 `tdstore_transfer_leader_job_cost_exception_count` 进行递增，该指标会上报到 TDSQL3-运维管理台进行展示。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ulonglong |
| 默认值 | 10000 |
| 取值范围 | 0, UINT64_MAX |
| 生效范围 | GLOBAL |

版权所有：腾讯云计算（北京）有限责任公司

# tdstore_transfer_leader_raft_cost_threshold_ms

最近更新时间：2025-04-22 15:40:04

## 功能描述

该参数用于设定 raft 层切主耗时（包含新主选举以及上任的耗时）的最大阈值。
超过该阈值后，被判定为耗时异常，会对 mysql status 指标 `tdstore_transfer_leader_raft_cost_exception_count` 进行递增，该指标会上报到赤兔进行展示。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ulonglong |
| 默认值 | 5000 |
| 取值范围 | 0, UINT64_MAX |
| 生效范围 | GLOBAL |

# tdstore_data_import_rate_limiter_bytes_per_sec

最近更新时间：2025-04-22 15:40:04

## 功能描述

Fast Online DDL ingest behind 模式的磁盘吞吐量限流参数，用于限制数据回填阶段消耗的读/写吞吐量，即读或写所用的吞吐量都不会超过这个值，该参数会影响 Fast Online DDL 的执行效率。
设置为0代表不限速。

## 属性说明

| 属性 | 说明 |
| --- | --- |
| 参数类型 | ulonglong |
| 默认值 | max(tdstore_data_disk_io_bandwidth*30%,50*1024*1024) |
| 取值范围 | 50*1024*1024, INT64_MAX |
| 生效范围 | GLOBAL |

# 错误码信息

最近更新时间：2023-02-21 11:45:40

TDStore 引擎错误码及说明如下：

> **说明**
> 当您遇到如下错误码需要处理时，您可以通过 在线支持 进行处理。

| 错误码 | 说明 |
| --- | --- |
| 50018 | 通用错误码，无具体含义 |
| 50019 | 存储引擎设置错误，例如 SET default_storage_engine=InnoDB 报此错误 |
| 50020 | 发生在 SQLEngine 进程启动的阶段，SQLEngine 向 MC 注册失败 |
| 50021 | 执行 DML 时，并发的 DDL 修改了表结构，DML 报此错误 |
| 50022 | 执行 DDL 时，DDL 被后台线程中断，DDL 报此错误 |
| 50023 | 执行 DML 时，并发的 DDL 修改了表结构，DML 报此错误 |
| 50024 | 执行 DDL 时，校验表结构失败，DDL 报此错误 |
| 50025 | 同一时间、同一对象上只能有一个 DDL 正在执行，并发的 DDL 报此错误，例如，sqlengine#1 执行 ALTER TABLE db1.tbl1；sqlengine#2 也执行 ALTER TABLE db1.tbl1，sqlengine#2 报错 |

版权所有：腾讯云计算（北京）有限责任公司

| | |
|---|---|
| 50028 | 执行 DDL 时，创建表定义失败 |
| 50030 | 执行 DDL（带有自增值属性的 DDL）时，初始化自增值失败 |
| 50031 | 执行 DDL（如建表、建索引）时，分配 index id 失败 |
| 50032 | SQL Engine 通用错误码，无具体含义 |
| 50033 | 执行 DML，获取自增值 next val 失败 |
| 50034 | 执行 DDL 或 DML 时，TDStore 无法创建迭代器 |
| 50035 | 执行 DML，推进自增值 next val 失败 |
| 50036 | 执行 DDL 时，未定义自增列，但给出了自增值，如 CREATE TABLE t(a INT) AUTO_INCREMENT=10; |
| 50037 | 执行 DDL 或 DML 时，TDStore 上不存在相应事务上下文 |
| 50038 | 执行 DDL 或 DML 时，TDStore 上不存在相应事务上下文，更有可能发生了 "TDStore 切主" |
| 50039 | 发生在进程启动时，初始化路由失败 |
| 50040 | 用户在 SET VARIABLE 时缺少权限 |

| | |
|---|---|
| 50041 | 内置函数 STORAGE_FORMAT 只接受1个 uint32 整数作为入参，输入其他参数会报此错误。如 SELECT STORAGE_FORMAT(10001); |
| 50042 | 执行 DML 提交事务时，乐观事务冲突报错 |
| 50043 | 执行 DML 时，如果有写操作失败，为避免数据不一致，必须手工把事务回滚，然后能执行其他 SQL；在 ROLLBACK 之前执行其他 SQL，报此错误，如 INSERT INTO t – 失败、COMMIT – 报此错误、ROLLBACK – OK、其他 SQL – OK |
| 50044 | 发生在 SQLEngine 进程启动时，集群尚不可用 |
| 50045 | 发生在 SQLEngine 进程启动时，无法建立事务保活机制，会影响到正常事务的执行流程，拒绝启动 |
| 50046 | 发生在 DDL，ALTER TABLE t CHANGE ENGINE 在 RocksDB 和其他引擎之间切换会报此错误 |
| 50047 | 发生在用 DML 直接修改权限相关的表（user/global_grants 等）报错 |
| 50048 | 执行 DML 时，并发的 DDL 修改了表结构，DML 报此错误 |
| 50049/50050 | 发生在 SQLEngine 进程启动时，SQLEngine 向 MC 申请唯一的标识 ID 并持久化在 SQLEngine 的数据目录下，这期间报错误 |
| 50051 | 执行 DDL 时，Online Copy DDL 流程的错误码 |
| 50052 | 执行 DDL 时，Online DDL 流程的错误码 |

| | |
|---|---|
| 50053 | 发生在本地变量版本比 MC 维护的版本低的情况 |
| 50054 | 发生在对只读变量使用 SET PERSIST_ONLY 语法 |
| 50055 | 发生在 TDStore 进行全量备份时报错 |
| 50057 | DDL 会持久化一些任务信息，任务信息有长度上限，超过上限会报此错误 |