TDSQL 分布式开发手册

版本	作者	内容	时间
V1.0			
V1.1			
V1.2			

1 目录

1 概述

1.1 文档说明

本手册涵盖 TDSQL 连接方式，SQL 语句开发编写等内容。目的是指导应用开发。

1.2 范围

本手册适用于使用 TDSQL 分布式实例的应用开发人员、数据库应用设计人员、数据库管理员等。

本手册适用于 TDSQL10.3.16.3.x 版本。

2 产品术语

- 节点：Set 或称为数据节点、分片。基于 MySQL 数据库主从协议联结成若干组。Set 是分布式实例中最小数据单元。每个 set 内部都具有一主 N 备的高可用架构。一个分布式实例是由 N 个 Set 组成，每个 Set 中存有不同范围的数据，所有 set 加到一起是一份全量的数据
- 分片键：根据分片键把一份全量数据进行切分，每份数据称为数据分片
- 分布式实例：Group_Shard，数据分布在 n 个 set 上面。也可以简称为 shard
- 分片表：即水平拆分表（又名 Shard 表）；分表需指定一个字段，使用不同的分片算法（hash、list、range），将数据分布到不同的 set 当中。hash 分片算法使用 shardkey 语法，list 和 range 分片算法采用 tdsql_distributed by 语法
- 单片表：又名 Noshard 表，用于存储一些无需分片的表，该表的数据全量存在第一个物理分片（set）中。所有单片表的数据都放在第一个物理分片（set）中。由于单片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片表，则可能导致第一个 set 的负载太大
- 广播表：又名小表广播，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
- 一级分区表：分片表的同义词

3 TDSQL 支持的数据类型

TDSQL 分布式数据库支持 MySQL 所有数据类型，包括数字类型、字符类型、日期时间类型、Json 数据类型。

数字类型

分布式实例兼容整型、浮点型和定点型三种数字类型，具体兼容类型如下：

➢ 整型支持 INTEGER、 INT、SMALLINT、TINYINT、MEDIUMINT、BIGINT 七种类型，相关信息详见如下表。

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

➢ 浮点型支持 FLOAT 和 DOUBLE，格式支持 FLOAT(M,D) 、REAL(M,D) 、 DOUBLE PRECISION(M,D)。

➢ 定点型支持 DECIMAL 和 NUMERIC，格式 DECIMAL(M,D)。

字符类型

TDSQL 支持的字符类型：CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、TINYBLOB、TINYTEXT，MEDIUMBLOB、MEDIUMTEXT、LONGBLOB、LONGTEXT、ENUM、SET。

其中 CHAR 和 VARCHAR 最为常用，LOB 和 TEXT 类型不建议使用。

CHAR 和 VARCHAR 类型相似，但存储和检索的方式不同。 它们在最大长度和是否保留尾随空格方面也不同。

CHAR 和 VARCHAR 类型声明的长度指示要存储的最大字符数。 例如，CHAR(30) 最多可容纳 30 个字符。CHAR 列的长度固定为您在创建表时声明的长度。长度可以是 0 到 255 之间的任何值。存储 CHAR 值时，它们会用空格右填充到指定的长度。

VARCHAR 列中的值是可变长度的字符串。长度可以指定为 0 到 65,535 之间的值。

日期类型

TDSQL 支持如下时间类型：

类型	日期格式	日期范围
YEAR	YYYY	1901 ~ 2155
TIME	HH:MM:SS	-838:59:59 ~ 838:59:59
DATE	YYYY-MM-DD	1000-01-01 ~ 9999-12-3
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1980-01-01 00:00:01 UTC ~ 2040-01-19 03:14:07 UTC

Json 数据类型

支持存储 Json 格式的数据类型，以便更加有效的对 Json 进行处理，同时又能提早检查错误。

语句如下：

注意事项：对 Json 类型的字段进行排序时，不支持混合类型排序。

例如，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和 String 类型，其它类型排序暂不处理。

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

4 TDSQL 支持的语言结构

分布式实例支持所有 MySQL 使用的文字格式，包括如下：

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

String Literals 格式

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，目前 TDSQL 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 格式，即：[_charset_name]'string' [COLLATE collation_name]格式。

支持如下转义字符：

```
\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 格式

数值字面值包括 Integer 类型 、 Decimal 类型、浮点数字面值。

Integer 可以包括“.”作为小数点分隔，数字前加字符“-”、“+”来表示正数或者负数。

精确数值字面值可以表示多种格式，如格式：1, .2, 3.4, -5, -6.78, +9.10。

科学记数法，如格式：1.2E3, 1.2E-3, -1.2E3, -1.2E-3。

Date and Time Literals 格式

Date 支持如下格式：

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

例如：'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
```

例如：'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 格式

Hexadecimal Literals 支持的格式如下：

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

Bit-Value Literals 格式

Bit-Value Literals 支持的格式如下：

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

Boolean Literals 格式

常量 True=1 和 False =0，其不区分大小写。

```
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，不区分大小写，与命令 \N(不区分大小写) 同义。

注意事项: NULL 跟 0 的意义不一样，跟空字符串 '' 的意义也不一样。

5 TDSQL 的连接

5.1 mysql 命令行方式

TDSQL 通过 Proxy 接口提供和 MySQL 兼容的连接方式，用户可以通过 IP 地址、端口号、用户名以及密码连接 TDSQL 系统，连接语句如下：

语法：

```
mysql -hhost_ip -Pport -uusername -ppassword -c
```

示例：

```
mysql -h10.10.10.10 -P3306 -utest12 -ptest123 -c
```

注意事项： 使用 MySQL 登录命令时，请务必增加-c 参数，这样可以使用注释透传功能。

5.2 JDBC+tomcat 连接配置

在 Tomcat 的 server.xml 中配置数据库连接时，推荐 JDBC 连接串如下：

```
jdbc:mysql://ip:port/db_name?user=your_username&password=your_password&useLocalSessionStates=true&useUnicode=true&characterEncoding=utf-8&serverTimezone=Asia/Shanghai"
```

其他参数说明：

参数	含义	缺省值	推荐值
useLocalSessionState	配置驱动程序是否使用autocommit，read_only和transaction isolation的内部值(jdbc端的本地值),避免JDBC driver每次都去检查target database是否是ReadOnly,autocommit	false	true

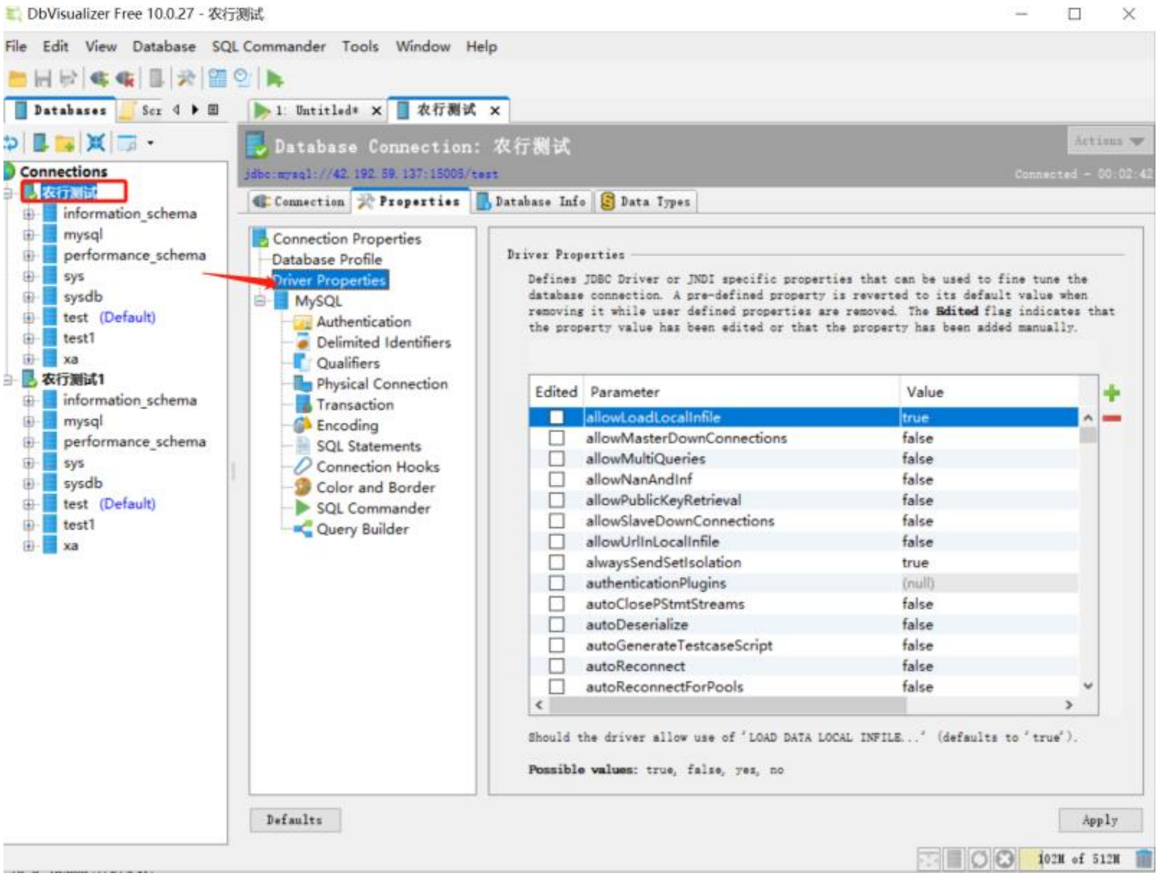
rewriteBatchedStatements	用于保证jdbc driver可以批量执行SQL，按需配置	false	按需配置，建议true
useUnicode	是否使用Unicode字符集	false	按需配置，建议设置true
characterEncoding	字符编码格式	无	按需配置，建议设置utf-8
serverTimezone	时区	local	按需配置，建议中国区部署设置为Asia/Shanghai
netTimeoutForStreamingResults	当使用StremResultSet结果集时，建议配置该参数，保证使用数据库的默认超时时间	600	0（即应用端不配置，直接使用数据库服务器超时时间）
useCursorFetch	是否使用cursor来拉取数据。(分布式不支持游标)	false	false
useSSL	与数据库之间连接是否使用加密连接。建议互联网部署应用开启加密连接。开启后由于数据链路加密传输，	默认开启，即useSSL=true(或sslMode=PREFERRED)	按需配置（关闭方式：useSSL=false(或sslMode=DISABLED)）

	影响部分性能。非互联网应用按需配置。说明：tdsql网关节点进已进行适配，默认开启usessl后，jdbc参数中无需配置allowPublicKeyRetrieval=true		

5.3

5.3 dbvisualizer 连接工具配置

dbvisualizer 是一个 ide 工具，使用 jdbc 连接 mysql，该 ide 默认设置 useCursorFetcht 为 true，需修改该参数为 false 访问 TDSQL。



Driver Properties

Defines JDBC Driver or JNDI specific properties that can be used to fine tune the database connection. A pre-defined property is reverted to its default value when removing it while user defined properties are removed. The **Edited** flag indicates that the property value has been edited or that the property has been added manually.

Edited	Parameter	Value
□	allowLoadLocalInfile	true
□	allowMasterDownConnections	false
□	allowMultiQueries	false
□	allowNanAndInf	false
□	allowPublicKeyRetrieval	false
□	allowSlaveDownConnections	false
□	allowUrlInLocalInfile	false
□	alwaysSendSetIsolation	true
□	authenticationPlugins	(null)
□	autoClosePStmtStreams	false
□	autoDeserialize	false
□	autoGenerateTestcaseScript	false
□	autoReconnect	false
□	autoReconnectForPools	false

Should the driver allow use of 'LOAD DATA LOCAL INFILE...' (defaults to 'true').

Possible values: true, false, yes, no

6 SQL 参考

6.1 TDSQL 使用限制

TDSQL 分布式实例中所编写的 SQL 语句中凡是包含 shardkey、parition、distributed by 等关键字的会交由 proxy 处理，语句的剩余部分会发送到 DB,按照 MYSQL 语法执行。所有 TDSQL 分布式 SQL 不支持使用 DELAYED 和 LOW_PRIORITY，不支持对于变量的引用和操作，比如 SET @c=1, @d=@c+1; SELECT @c, @d 等。具体限制项请参考以下两小节。

6.1.1 TDSQL 大类限制

- 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、临时表
- 不支持复合语句，例如：BEGIN END，LOOP，UNION 的语句
- 不支持主备同步相关的 SQL 语言

6.1.2 TDSQL 小语法限制

TDSQL 分布式实例不支持 DDL、DML、管理 SQL 语言的部分语法，具体限制如下：

- DDL
    - 不支持 CREATE TABLE ... SELECT
    - 不支持 CREATE TEMPORARY TABLE
    - 不支持 CREATE/DROP/ALTER SERVER
    - 不支持 CREATE/DROP/ALTER LOGFILE GROUP
    - 不支持 ALTER 对分表键进行改名，但可以修改类型
    - 不支持 RENAME
- 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
    - 不支持窗口函数
    - 不支持非 SELECT 的子查询
    - 不支持不带列名的 INSERT/REPLACE
    - 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
    - 不支持 SQL 中对于变量的引用和操作，比如 SET @c=1, @d=@c+1; SELECT @c, @d
    - 不支持 INDEX_HINT
    - 不支持 HANDLER/DO
- 管理 SQL 语句

- 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 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

6.2 DDL 语句

本节主要介绍了使用 DDL 语句创建表和常用 DDL 语句说明。

6.2.1 CREATE

6.2.1.1 CREATE DATABASE

本节介绍 CREATE DATABASE 语法。

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_option] ...

create_option: [DEFAULT] {
    CHARACTER SET [=] charset_name
  | COLLATE [=] collation_name
}
```

注意事项：

- CREATE DATABASE 创建具有给定名称的数据库。 要使用此语句，您需要对数据库具有 CREATE 权限。 CREATE SCHEMA 是 CREATE DATABASE 的同义词。

- 如果数据库存在并且您没有指定 IF NOT EXISTS，则会发生错误。

- 在具有活动 LOCK TABLES 语句的会话中不允许 CREATE DATABASE。

- CHARACTER SET 选项指定默认的数据库字符集。 COLLATE 选项指定默认的数据库排序规则。要查看可用的字符集和排序规则，请使用 SHOW CHARACTER SET 和 SHOW COLLATION 语句

示例：

```
create database d2 default charset 'utf8mb4';
```

6.2.1.2 CREATE TABLE

TDSQL 分布式实例支持创建分表、单表和广播表。分表即自动水平拆分的表（Shard 表），水平拆分是基于分表键采用类似于一致性 Hash、Range、List 等方式，根据计算后的值分配到不同的节点组中的一种技术方案。可以将满足对应条件的行将存储在相同的物理节点组中。这种场景称为组拆分（Groupshard）,可以迅速提高应用层联合查询等语句的处理效率。TDSQL 支持 LIST、RANGE、HASH 三种类型的一级分区，同时支持 RANGE、LIST 两种格式的二级分区。

创建一级 hash 分区表语法：

```
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)
}
```

创建一级 range| list 分区表语法：

```
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']
```

6.2.1.2.1 创建分区表

6.2.1.2.1.1 一级分区表

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。

6.2.1.2.1.1.1 一级 HASH 分区

- **一级 hash 分区支持类型**
    - DATE，DATETIME
    - TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT
    - CHAR，VARCHAR

一级 hash 分片注意事项：

- 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

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;
```

6.2.1.2.1.1.2 一级 RANGE 分区

- **一级 range 分区支持类型**
  - DATE，DATETIME，TIMESTAMP
  - TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT
  - CHAR，VARCHAR

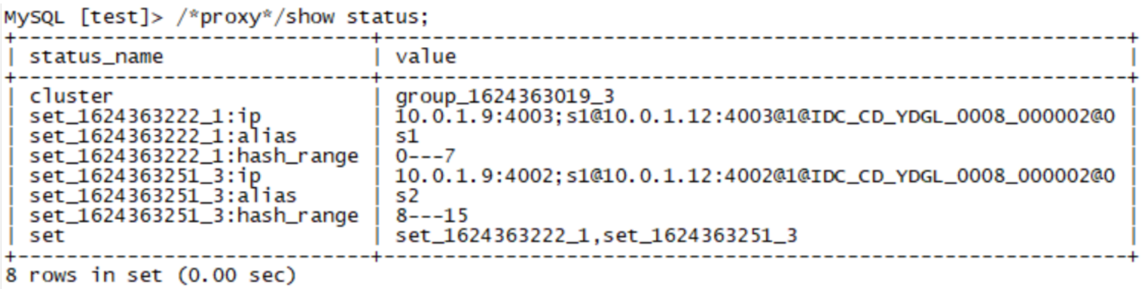
注意事项：

- tdsql_distributed by ...语法放置于 create table ...的末尾
- 创建一级 range 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

示例：

```
--创建分布在 2 个 set 上的分区表：
```

```
DROP TABLE IF EXISTS employees_range;

CREATE TABLE `employees_range` (
  `id`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id)
)
TDSQL_DISTRIBUTED BY RANGE(id) (
  s1 VALUES LESS THAN (6),
  s2 VALUES LESS THAN (11)
);



--查看 set_1624363222_1 和 set_1624363251_3 的别名分别为 s1 和 s3：
```



```
MySQL [test]> /*proxy*/show status;
+-----------------------------+-------------------------------------------------------------+
| status_name                 | value                                                       |
+-----------------------------+-------------------------------------------------------------+
| cluster                     | group_1624363019_3                                          |
| set_1624363222_1:ip         | 10.0.1.9:4003;s1@10.0.1.12:4003@1@IDC_CD_YDGL_0008_000002@0 |
| set_1624363222_1:alias      | s1                                                          |
| set_1624363222_1:hash_range | 0---7                                                       |
| set_1624363251_3:ip         | 10.0.1.9:4002;s1@10.0.1.12:4002@1@IDC_CD_YDGL_0008_000002@0 |
| set_1624363251_3:alias      | s2                                                          |
| set_1624363251_3:hash_range | 8---15                                                      |
| set                         | set_1624363222_1,set_1624363251_3                           |
+-----------------------------+-------------------------------------------------------------+
8 rows in set (0.00 sec)
```

6.2.1.2.1.1.3 一级 LIST 分区

- **一级 list 分区支持类型**

  – DATE，DATETIME，TIMESTAMP

  – TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT

  – CHAR，VARCHAR

注意事项：

– 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

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)
);
```

6.2.1.2.1.2 二级分区表

二级分区是将特定条件的数据进行分区处理，目前 TDSQL 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。

创建二级 range| list 分区表语法：

```
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)}

    [SUBPARTITION BY

        {HASH(expr)

        |(column_list) }

    ]

    [(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' ]

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

```
subpartition_definition:

    SUBPARTITION logical_name

        [[STORAGE] ENGINE [=] engine_name]

        [COMMENT [=] 'string']
```

6.2.1.2.1.2.1 二级 RANGE 分区

- **Range 支持类型**

  - DATE，DATETIME，TIMESTAMP
    —支持 year，month，day 函数，函数为空和 day 函数一样

  - 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

注意事项：

- 使用 tdsql_distributed by ...语法创建分区表时，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...

- 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，默认往后创建 3 个分区，以 Year 为例，TDSQL 会自动往后创建 3 年（2018 年、2019 年、2020 年）的分区，后续也会自动增减。

示例：

```
一级 hash 二级 range 分区：

DROP TABLE IF EXISTS employees_hash_range;

CREATE TABLE `employees_hash_range` (
  `id`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id)
) shardkey=id
PARTITION BY RANGE (month(fired)) (
  PARTITION p0 VALUES LESS THAN (202106),
  PARTITION p1 VALUES LESS THAN (202107)
);
```

一级 list 二级 range 分区:

```
DROP TABLE IF EXISTS employees_list_range;

CREATE TABLE `employees_list_range` (
  `id`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id,fired)
)
PARTITION BY RANGE (month(fired)) (
  PARTITION p0 VALUES LESS THAN (202106),
  PARTITION p1 VALUES LESS THAN (202107)
)
TDSQL_DISTRIBUTED BY LIST(id) (
  s1 VALUES IN (1,3,5),
  s2 VALUES IN (2,4,6)
);
```

一级 range 二级 range 分区:

```
DROP TABLE IF EXISTS employees_range_range;

CREATE TABLE `employees_range_range` (
  `id`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id,fired)
)
PARTITION BY RANGE (month(fired)) (
  PARTITION p0 VALUES LESS THAN (202106),
  PARTITION p1 VALUES LESS THAN (202107)
)
TDSQL_DISTRIBUTED BY RANGE(id) (
  s1 VALUES LESS THAN (6),
  s2 VALUES LESS THAN (11)
);
```

一级 range 二级 range 分区和子分区

```
DROP TABLE if exists tb_sub_ev;

CREATE TABLE tb_sub_ev (

  id int NOT NULL,
  purchased date NOT NULL,
  PRIMARY KEY (id,purchased)
) ENGINE=InnoDB
PARTITION BY RANGE (YEAR(purchased))
    SUBPARTITION BY HASH (TO_DAYS(purchased))
(PARTITION p0 VALUES LESS THAN (1990)
       (SUBPARTITION s0 ENGINE = InnoDB,
       SUBPARTITION s1 ENGINE = InnoDB),
 PARTITION p1 VALUES LESS THAN (2000)
       (SUBPARTITION s2 ENGINE = InnoDB,
       SUBPARTITION s3 ENGINE = InnoDB))
TDSQL_DISTRIBUTED BY RANGE(id) (s1 values less than ('100'),s2 values
less than ('1000'));
```

6.2.1.2.1.2.2 二级 LIST 分区

- **List 支持类型**

  - DATE，DATETIME，TIMESTAMP —支持年月日函数
  - TINYINT, SMALLINT, MEDIUMINT, INT , BIGINT

注意事项：

- 使用 tdsql_distributed by ...语法创建分区表时，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...

示例：

```
一级 hash 二级 list 分区：

DROP TABLE IF EXISTS employees_hash_list;

CREATE TABLE `employees_hash_list` (
  `id`int NOT NULL,
  `region`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id)
) shardkey=id
PARTITION BY LIST (region) (
  PARTITION pRegion_1 VALUES IN (10, 30),
  PARTITION pRegion_2 VALUES IN (20, 40)
);
```

一级 list 二级 list 分区:

```
DROP TABLE IF EXISTS employees_list_list;

CREATE TABLE `employees_list_list` (
  `id`int NOT NULL,
  `region`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id, region)
)
PARTITION BY LIST (region) (
  PARTITION pRegion_1 VALUES IN (10, 30),
  PARTITION pRegion_2 VALUES IN (20, 40)
)
TDSQL_DISTRIBUTED BY LIST(id) (
  s1 VALUES IN (1,3,5),
  s2 VALUES IN (2,4,6)
);
```

一级 range 二级 list 分区:

```
DROP TABLE IF EXISTS employees_range_list;

CREATE TABLE `employees_range_list` (
  `id`int NOT NULL,
  `region`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id,region)
)
PARTITION BY LIST (region) (
  PARTITION pRegion_1 VALUES IN (10, 30),
  PARTITION pRegion_2 VALUES IN (20, 40)
)
TDSQL_DISTRIBUTED BY RANGE(id) (
  s1 VALUES LESS THAN (6),
  s2 VALUES LESS THAN (11)
);
```

6.2.1.2.2 创建广播表

广播表又名小表广播功能，创建时需要指定 noshardkey_allset 关键字。创建广播表后，每个节点都有该表的全量数据，且该表的所有操作都将广播到所有物理分片（set）中。

广播表主要用于提升跨节点组（ Set） 的 Join 操作的性能，常用于配置表等.

示例：

```
DROP TABLE IF EXISTS global_table_a;
CREATE TABLE  global_table_a (a int, b int key) shardkey=noshardkey_allset;
```

6.2.1.2.3 创建单片表

普通表：又名单片表（Noshard 表），创建时无须指定 shardkey 或者 tdsql_distributed by 关键字。单片表无需拆分且没有做任何特殊处理的表。其语法和 MySQL 完全一样，所有该类型表的全量数据默认存放在第一个物理节点组（Set）中。

示例：

```
DROP TABLE IF EXISTS noshard_table;
CREATE TABLE noshard_table (a int, b int key);
```

6.2.1.3 CREATE INDEX

通常，在使用 CREATE TABLE 创建表本身时在表上创建所有索引。该准则对于 InnoDB 表尤其重要，其中主键决定了数据文件中行的物理布局。 CREATE INDEX 使您能够向现有表添加索引。

语法：

```
CREATE [UNIQUE ] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]

    [algorithm_option | lock_option] ...


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


index_option: {

  index_type | COMMENT 'string'

}


index_type:

    USING {BTREE}


algorithm_option:

    ALGORITHM [=] {DEFAULT | INPLACE | COPY}


lock_option:

    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

注意事项：

- CREATE INDEX 不能用于创建 PRIMARY KEY；对于主键，请改用 ALTER TABLE。
- 对于 INNODB 存储引擎，允许的索引类型为 BTREE。

示例：

```
创建测试表：

DROP TABLE IF EXISTS customer;

CREATE TABLE customer(cust_id int key,name varchar(200),job_id
int,job_name varchar(300)) shardkey=cust_id;
```

使用 using 语句指定 index_type，若不指定，默认为 BTREE：

```
CREATE INDEX j_idx ON customer (name) USING BTREE;
```

创建列前缀索引：

```
CREATE INDEX idx_part_name ON customer (name(10));
```

创建降序索引：

```
CREATE INDEX idx_name_desc ON customer (name desc);
```

创建升序索引：

```
CREATE INDEX idx_name_asc ON customer (name asc);
```

创建唯一索引：

```
CREATE UNIQUE INDEX uniq_idx_job_id on customer(cust_id,job_id);
```

创建组合索引：

```
CREATE INDEX idx_cust on customer(name,job_name);
```

使用 COMMENT 语句指定索引页合并阈值：

```
CREATE INDEX j_idx_com ON customer (name) COMMENT 'MERGE_THRESHOLD=40';
```

6.2.2 DROP

6.2.2.1 Drop database

语法如下：

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

注意事项：

- DROP DATABASE 删除数据库中的所有表并删除数据库。 对此语句要非常小心！ 要使用 DROP DATABASE， 您需要 DROP database 的 权限。 DROP SCHEMA 是 DROP DATABASE 的同义词。
- 删除数据库时， 不会自动删除专门为数据库授予的权限，必须手动删除它们。

示例：

```
DROP DATABASE test;
```

6.2.2.2 Drop table

语法如下：

```
DROP TABLE [IF EXISTS]

    tbl_name [, tbl_name] ...

    [RESTRICT | CASCADE]
```

注意事项：

- DROP TABLE 删除一个或多个表。 您必须拥有 DROP 每个表 的 权限。
- 对于每个表，它将删除表定义和所有表数据。 如果表已分区，则该语句将删除表定义，其所有分区，存储在这些分区中的所有数据以及与已删除表关联的所有分区定义。
- 删除表也会删除表的任何触发器。
- DROP TABLE 导致隐式提交。
- 删除表时，不会自动删除专门为该表授予的权限 。 必须手动删除它们。
- 所有 innodb_force_recovery 设置都不支持 DROP TABLE
- RESTRICT 和 CASCADE 关键字什么也不做。 它们被允许使从其他数据库系统移植更容易。

示例：

```
DROP TABLE test;

drop table test RESTRICT;

drop table test5 CASCADE;
```

6.2.2.3 Drop index

语法如下：

```
DROP INDEX index_name ON tbl_name

    [algorithm_option | lock_option] ...


algorithm_option:

    ALGORITHM [=] {DEFAULT | INPLACE | COPY}


lock_option:

    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

注意事项：

- 要删除主键，索引名称始终为 PRIMARY，必须将其指定为带引号的标识符，因为 PRIMARY 是保留字：DROP INDEX `PRIMARY` ON t;

示例：

```

MySQL [test]> show create table customer\G;

*************************** 1. row ***************************

       Table: customer

Create Table: CREATE TABLE `customer` (

  `cust_id` int(11) NOT NULL,

  `name` varchar(200) COLLATE utf8_bin DEFAULT NULL,
  `job_id` int(11) DEFAULT NULL,
  `job_name` varchar(300) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`cust_id`),
  UNIQUE KEY `uniq_idx_job_id` (`cust_id`,`job_id`),
  KEY `idx_cust` (`name`,`job_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin shardkey=cust_id
1 row in set (0.00 sec)

MySQL [test]> drop index uniq_idx_job_id on customer;
Query OK, 0 rows affected (0.04 sec)

MySQL [test]> drop index idx_cust on customer;
Query OK, 0 rows affected (0.08 sec)
```

6.2.3 ALTER

6.2.3.1 ALTER TABLE

本章介绍 ALTER 相关用法。ALTER TABLE 更改表的结构。 例如，您可以添加或删除列、创建或销毁索引、更改现有列的类型或重命名列或表本身。您还可以更改特征，例如用于表或表注释的存储引擎。

但是请注意：线上系统的 DDL 变更请通过赤兔管理控制台的 online-ddl 模块进行。

语法如下：

```
ALTER TABLE tbl_name
    [alter_option [, alter_option] ...]
    [partition_options]

alter_option: {
    table_options
```

```
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX | KEY} [index_name]
        [index_type] (key_part,...) [index_option] ...
  | ALGORITHM [=] {DEFAULT | INSTANT | INPLACE | COPY}
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST | AFTER col_name]
  | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
  | {DISABLE | ENABLE} KEYS
  | DROP [COLUMN] col_name
  | DROP {INDEX | KEY} index_name
  | LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ORDER BY col_name [, col_name] ...
}

partition_options:
    partition_option [partition_option] ...

partition_option: {
    ADD PARTITION (partition_definition)
  | DROP PARTITION partition_names
  | TRUNCATE PARTITION {partition_names | ALL}
}

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

```
index_type:
    USING {BTREE}

index_option: {
index_type | COMMENT 'string'
}

table_options:
    table_option [[,] table_option] ...

table_option: {AUTO_INCREMENT [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | COMPRESSION [=] {'ZLIB' | 'LZ4' | 'NONE'}
  | ENGINE [=] engine_name
  | KEY_BLOCK_SIZE [=] value
  | ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT | 0 | 1}
  | STATS_PERSISTENT [=] {DEFAULT | 0 | 1}
  | STATS_SAMPLE_PAGES [=] value)
}
```

注意事项：

- 要使用 ALTER TABLE，你需要 ALTER， CREATE 和 INSERT 权限。
- 不支持改变 shardkey 类型、删除 shardkey 的操作

- 一级分区，语法和单表一样，只能改变 db 上表结构，不能改变数据分布方式。
- 二级分区，支持添加和删除分区，语法和单表一样，range 分区只能向后追加。

示例：

```
--创建一级 hash 分区表
DROP TABLE IF EXISTS sbtest1;
CREATE TABLE `sbtest1`
(`k` bigint(20) NOT NULL,
`id` bigint(20) NOT NULL,
`c` char(120)  NOT NULL,
`pad` char(60) NOT NULL,
`balance` int(11) NOT NULL,
`lastModifyTime` datetime,
PRIMARY KEY (`k`,`id`),
KEY `k_1` (`k`))
ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_bin shardkey=id;

--添加删除索引
alter table sbtest1 add index idx_blc (balance);
alter table sbtest1 drop index idx_blc;

--修改表字段类型
alter table sbtest1 modify column pad varchar(50);

--增加一个新列为第一列
alter table sbtest1 add column col1 INT NOT NULL first;
```

```
--增加一个到指定列之后
alter table sbtest1 add column col_after_pad INT NOT NULL after pad;

--修改表增加字段
alter table sbtest1 add column mark varchar(50);

--修改表字段名字
alter table sbtest1 change column k k_new1 bigint(20);

--修改表删除字段
alter table sbtest1 drop column mark;

--重组表
ALTER TABLE sbtest1 ENGINE = InnoDB;

--更改 InnoDB 表以使用压缩行存储格式：
ALTER TABLE sbtest1 ROW_FORMAT = COMPRESSED;

--添加（或更改）表注释：
ALTER TABLE sbtest1 COMMENT = 'New table comment';
```

示例：

```
创建二级分区表：
DROP TABLE if exists customers_1;
CREATE TABLE customers_1 (
   first_name VARCHAR(25) primary key,
   last_name VARCHAR(25),

   street_1 VARCHAR(30),

   street_2 VARCHAR(30),

   city_name VARCHAR(15),

   renewal DATE

) shardkey=first_name

PARTITION BY LIST (city_name) (

   PARTITION pRegion_1 VALUES IN('BJ', 'GZ', 'SZ'),

   PARTITION pRegion_2 VALUES IN('SH', 'CD'),

   PARTITION pRegion_3 VALUES IN('GY'),

   PARTITION pRegion_4 VALUES IN('HZ')

);



删除分区：
ALTER TABLE customers_1 drop partition pRegion_4;



增加分区：
ALTER TABLE customers_1 add partition (partition pRegion_4 VALUES IN('TJ'));



截断分区：

ALTER TABLE customers_1 truncate partition pRegion_4;
```

示例：

```
创建二级分区表：

DROP TABLE IF EXISTS employees_list_range;

CREATE TABLE `employees_list_range` (
  `id`int NOT NULL,
  `city` varchar(10),
  `fired` DATE NOT NULL DEFAULT '1970.01.01',
  PRIMARY KEY(id,fired)
)
PARTITION BY RANGE (month(fired)) (
  PARTITION p0 VALUES LESS THAN (202106),
  PARTITION p1 VALUES LESS THAN (202107)
)
TDSQL_DISTRIBUTED BY LIST(id) (
  s1 VALUES IN (1,3,5),
  s2 VALUES IN (2,4,6)
);



删除分区：
ALTER TABLE employees_list_range drop partition p1;


增加分区：

ALTER TABLE employees_list_range add partition(partition p2 values less
than (202108));



截断分区：

ALTER TABLE employees_list_range truncate partition p0;
```

6.2.4 TRUNCATE

语法如下：

```
TRUNCATE [TABLE] tbl_name
```

注意事项：

- 需要有 drop 权限
- 截断操作会导致隐式提交，因此无法回滚
- 第一次执行 truncate 若失败，则进行第二次 truncate

示例：

```
truncate table t1;
```

6.3 DML 语句

本节主要介绍 DML 语句中常用的 Select（查询）、Insert（插入）、Replace（替换）、Update（更新）及 Delete（删除）指令。

6.3.1 SELECT

6.3.1.1 基础查询语法

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    select_expr [, select_expr] ...
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
```

示例：

```
drop table if exists test1;

create table test1 ( a int key, b int, c char(20) ) shardkey=a;

drop table if exists test2;
create table test2 ( a int key, d int, e char(20) ) shardkey=a;


insert into test1 (a,b,c) values(1,2,"record1"),(2,3,"record2");
insert into test2 (a,d,e) values(1,3,"test2_record1"),(2,3,"test2_recor
d2");

select t1.a,t1.b,t1.c,t2.a,t2.d,t2.e  from test1 t1 join test2 t2 on t
1.b=t2.d;


select t1.a,t1.b,t1.c from test1 t1 where t1.a in (select a from test
2);
```

```
select t1.a,t1.b,t1.c from test1 t1  where exists (select t2.a,t2.d,t2.
e  from test2 t2 where t2.a=t1.b);

select t1.a, count(1) from test1 t1 where exists (select t2.a,t2.d,t2.e
 from test2 t2 where t2.a=t1.a) group by t1.a;


select distinct count(1) from test1 t1 where exists (select t2.a,t2.d,t
2.e from test2 t2 where t2.a=t1.a) group by t1.a;

select count(distinct t1.a) from test1 t1 where exists (select t2.a,t2.
d,t2.e from test2 t2 where t2.a=t1.a);
```

6.3.1.2 join

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

6.3.1.2.1 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。

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_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 FULL JOIN 的 SELECT 查询语句，笛卡尔积
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;
```

6.3.1.2.2 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。

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_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=noshar
dkey_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 FULL JOIN 的 SELECT 查询语句，笛卡尔积
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;
```

6.3.1.2.3 分表和单表 join 示例

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 FULL JOIN 的 SELECT 查询语句，笛卡尔积
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;
```

6.3.1.2.4 跨分片 update/delete join 示例

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;
```

6.3.1.3 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 ...]
```

注意事项：

- 参与 UNION 的表所 select 的列的个数需要保持一致。

- UNION 结果集的列名取自第一个 SELECT 语句的列名。

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 t
2 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 les
s 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 les
s 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 ord
er 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;
```

6.3.1.4 子查询

语法如下：

```
SELECT ...

FROM table

WHERE expr operator

(SELECT select_list FROM table)
```

注意事项：

- 一般情况下，由于子查询效率不高，尽量使用 join 的代替子查询

示例：

```
DROP TABLE if exists `test_shard_table1`;
CREATE TABLE `test_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_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_shard_table2`;
CREATE TABLE `test_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_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);

SELECT COUNT(B)
FROM test_shard_table1
WHERE id IN
```

```
(SELECT c FROM test_shard_table2 WHERE id>5);


SELECT MAX(c), MIN(c)

FROM test_shard_table1 WHERE c IN

(SELECT c FROM test_shard_table2 WHERE id<8)

AND id>4 ORDER BY c;
```

6.3.2 INSERT

语法如下：

```
INSERT [IGNORE]

    [INTO] tbl_name

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

    [(col_name [, col_name] ...)]

    {{VALUES | VALUE} (value_list) [, (value_list)] ...

      |

      VALUES row_constructor_list

    }

    [ON DUPLICATE KEY UPDATE assignment_list]


INSERT [IGNORE]

    [INTO] tbl_name

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

    SET assignment_list

    [ON DUPLICATE KEY UPDATE assignment_list]


INSERT [IGNORE]

    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {SELECT ... | TABLE table_name}
    [ON DUPLICATE KEY UPDATE assignment_list]

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

row_constructor_list:
    ROW(value_list)[, ROW(value_list)][, ...]

assignment:
    col_name = [row_alias.]value

assignment_list:
    assignment [, assignment] ...
```

注意事项：

- 对于分片表执行 insert 命令时，字段必须包含 Shardkey，否则系统会拒绝执行 SQL 命令，因为 Proxy 无法判断 SQL 语句发送的后端数据库节点位置

示例：

```
--测试不带 shardkey 字段：
MySQL [test]> DROP TABLE IF EXISTS test1;
```

```
Query OK, 0 rows affected (0.12 sec)


MySQL [test]> create table test1(a int not null primary key,b int,c cha
r(10)) shardkey=a;
Query OK, 0 rows affected (2.64 sec)


MySQL [test]> insert into test1 (b,c) values(10,"record3");
ERROR 683 (HY000): Proxy ERROR: Get shardkeys return error: insert/repl
ace must contain shardkey column


MySQL [test]> insert into test1 (a,c) values(40,"records5");
Query OK, 1 row affected (0.03 sec)




--测试不携带 ignore，会发生主键冲突

MySQL [test]> drop table if exists t1_1_1;

Query OK, 0 rows affected (0.10 sec)

MySQL [test]> create table t1_1_1 (a int primary key, b int) shardkey=
a;

Query OK, 0 rows affected (0.18 sec)

MySQL [test]> drop table if exists t1_1_2;

Query OK, 0 rows affected (0.07 sec)

MySQL [test]> create table t1_1_2 (a int primary key) shardkey=a;

Query OK, 0 rows affected (0.18 sec)


MySQL [test]> insert into t1_1_1 (a,b) values (1,0),(2,0),(3,1);

Query OK, 3 rows affected (0.01 sec)


MySQL [test]> select * from t1_1_1;

+---+------+

| a | b    |
+---+------+
| 1 |    0 |
| 2 |    0 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)

MySQL [test]> insert into t1_1_2 select b from t1_1_1;
ERROR 913 (HY000): Proxy ERROR:Join internal error: Duplicate entry '0' for key 'PRIMARY'

--携带 ignore，会写入部分数据，重复的数据只写一次
MySQL [test]> insert ignore into t1_1_2 select b from t1_1_1;
Query OK, 2 rows affected, 1 warning (0.00 sec)

MySQL [test]> select * from t1_1_2 order by a;
+---+
| a |
+---+
| 0 |
| 1 |
+---+
2 rows in set (0.00 sec)
```

6.3.3 REPLACE

语法如下：

```
REPLACE
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {{VALUES | VALUE} (value_list) [, (value_list)] ...
      |
      VALUES row_constructor_list
    }

REPLACE
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list

REPLACE
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {SELECT ... | TABLE table_name}

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

row_constructor_list:
    ROW(value_list)[, ROW(value_list)][, ...]

assignment:
    col_name = value
```

```
assignment_list:

    assignment [, assignment] ...
```

注意事项：

- 对于分片表执行 Replace 命令时，字段必须包含 Shardkey，否则系统会拒绝执行 SQL 命令，因为 Proxy 无法判断 SQL 语句发送的后端数据库节点位置

示例：

```
--测试不带 shardkey 字段：

MySQL [test]> DROP TABLE IF EXISTS test5;

MySQL [test]> create table test5(a int not null primary key,b int,c cha
r(10)) shardkey=a;

Query OK, 0 rows affected (0.27 sec)



MySQL [test]> replace into test5 (b,c) values(10,"record3");

ERROR 683 (HY000): Proxy ERROR: Get shardkeys return error: insert/repl
ace must contain shardkey column



MySQL [test]> replace into test5(a,b,c) values(3,40,"record1");

Query OK, 1 row affected (0.00 sec)



--测试加载多条数据

MySQL [test]> replace into test5(a,b,c) values(4,50,"record2"),(5,60,"r
ecord3"),(6,70,"record4"),(7,80,"record5"),(8,90,"record6"),(9,100,"rec
ord7");

Query OK, 6 rows affected (0.00 sec)



--测试 replace select 语句
```

```
drop table if exists t1_1_1;

create table t1_1_1 (a int not null primary key, b char(10)) shardkey=
a;

drop table if exists t1_1_2;

create table t1_1_2 (a int not null primary key, b char(10)) shardkey=
a;


insert into t1_1_1 (a,b) values (1,"t1:1"),(3,"t1:3");

insert into t1_1_2 (a,b) values (2,"t2:2"), (3,"t2:3");

replace into t1_1_1 select * from t1_1_2;
```

6.3.4 DELETE

语法如下：

```
DELETE [QUICK] [IGNORE] FROM tbl_name [[AS] tbl_alias]

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

    [WHERE where_condition]

    [ORDER BY ...]

    [LIMIT row_count]
```

注意事项：

- 为了安全考虑，分表和广播表执行 delete 指令的时候必须带“ where ”条件，否则系统拒绝执行该 SQL 命令

示例：

```
--测试不带 shardkey 的 delete

MySQL [test]> DROP TABLE IF EXISTS test3;

MySQL [test]> create table test3(a int not null primary key,b int,c cha
r(10)) shardkey=a;
```

```
MySQL [test]> insert into test3(a,b,c) values (1,2,'A');

Query OK, 1 row affected (0.00 sec)


MySQL [test]> delete from test3;

ERROR 913 (HY000): Proxy ERROR:Join internal error: delete query has no where clause


MySQL [test]> delete from test3 where a=1;

Query OK, 1 rows affected (0.00 sec)


--测试包含子查询的delete

drop table if exists t1_1;

create table t1_1 (a int primary key, b int) shardkey=a;

drop table if exists t1_2;

create table t1_2 (a int primary key, b int) shardkey=a;

insert into t1_1 (a,b) values (20,20);

insert into t1_2 (a,b) values (20,20);

insert into t1_1 (a,b) values (19,19);

insert into t1_2 (a,b) values (19,19);

insert into t1_1 (a,b) values (18,18);

insert into t1_2 (a,b) values (18,18);

insert into t1_1 (a,b) values (17,17);

insert into t1_2 (a,b) values (17,17);

insert into t1_1 (a,b) values (16,16);

insert into t1_2 (a,b) values (16,16);

insert into t1_1 (a,b) values (15,15);

insert into t1_2 (a,b) values (15,15);
insert into t1_1 (a,b) values (14,14);
insert into t1_2 (a,b) values (14,14);
insert into t1_1 (a,b) values (13,13);
insert into t1_2 (a,b) values (13,13);
insert into t1_1 (a,b) values (12,12);
insert into t1_2 (a,b) values (12,12);
insert into t1_1 (a,b) values (11,11);
insert into t1_2 (a,b) values (11,11);
insert into t1_1 (a,b) values (10,10);
insert into t1_2 (a,b) values (10,10);
insert into t1_1 (a,b) values (9,9);
insert into t1_2 (a,b) values (9,9);
insert into t1_1 (a,b) values (8,8);
insert into t1_2 (a,b) values (8,8);
insert into t1_1 (a,b) values (7,7);
insert into t1_2 (a,b) values (7,7);
insert into t1_1 (a,b) values (6,6);
insert into t1_2 (a,b) values (6,6);
insert into t1_1 (a,b) values (5,5);
insert into t1_2 (a,b) values (5,5);
insert into t1_1 (a,b) values (4,4);
insert into t1_2 (a,b) values (4,4);
insert into t1_1 (a,b) values (3,3);
insert into t1_2 (a,b) values (3,3);
insert into t1_1 (a,b) values (2,2);
insert into t1_2 (a,b) values (2,2);
insert into t1_1 (a,b) values (1,1);
insert into t1_2 (a,b) values (1,1);
```

```
delete from t1_1 where a in (select b from t1_2 where a<10);

delete from t1_1 where exists(select 1 from t1_2 where t1_1.a=t1_2.b and t1_2.a>8);


--测试携带和不携带 ignore 的 delete

drop table if exists t8_1;

create table t8_1 (a int NOT NULL, b int, primary key (a));

drop table if exists t8_2;

create table t8_2 (a int NOT NULL, b int, primary key (a));

drop table if exists t8_3;

create table t8_3 (a int NOT NULL, b int, primary key (a));

insert into t8_1 (a,b) values (0, 10),(1, 11),(2, 12);

insert into t8_2 (a,b) values (33, 10),(0, 11),(2, 12);

insert into t8_3 (a,b) values (1, 21),(2, 12),(3, 23);


--不带 ignore 的情况

MySQL [test]> delete t8_1.*, t8_2.* from t8_1,t8_2 where t8_1.a = t8_2.a and t8_1.b <> (select b from t8_3 where t8_1.a < t8_3.a);

ERROR 1242 (21000): Subquery returns more than 1 row


--携带 ignore 的情况

MySQL [test]> delete ignore t8_1.*, t8_2.* from t8_1,t8_2 where t8_1.a = t8_2.a and t8_1.b <> (select b from t8_3 where t8_1.a < t8_3.a);

Query OK, 2 rows affected, 2 warnings (0.01 sec)
```

6.3.5 UPDATE

语法如下：

```
UPDATE [IGNORE] table_reference

    SET assignment_list
    [WHERE where_condition]

    [ORDER BY ...]

    [LIMIT row_count]


value:

    {expr | DEFAULT}


assignment:

    col_name = value


assignment_list:

    assignment [, assignment] ...
```

注意事项：

- 分区表不支持更新 shardkey，需用显示开启事务，再执行 delete 和 insert 替代 update
- 分区表不支持 update set 的值为子查询
- 为了安全考虑，分表和广播表执行 update 指令的时候必须带“ where ”条件，否则系统拒绝执行该 SQL 命令

示例：

```
--测试 update 的累加

DROP TABLE IF EXISTS t1_1;

CREATE TABLE t1_1

(place_id int (10) unsigned NOT NULL,

shows int(10) unsigned DEFAULT '0' NOT NULL,

ishows int(10) unsigned DEFAULT '0' NOT NULL,

ushows int(10) unsigned DEFAULT '0' NOT NULL,
clicks int(10) unsigned DEFAULT '0' NOT NULL,
iclicks int(10) unsigned DEFAULT '0' NOT NULL,
uclicks int(10) unsigned DEFAULT '0' NOT NULL,
ts timestamp,PRIMARY KEY (place_id,ts))
shardkey=place_id;


INSERT INTO t1_1 (place_id,shows,ishows,ushows,clicks,iclicks,uclicks,ts)VALUES (1,0,0,0,0,0,0,20000928174434);


UPDATE t1_1 SET shows=shows+1,ishows=ishows+1,ushows=ushows+1,clicks=clicks+1,iclicks=iclicks+1,uclicks=uclicks+1 WHERE place_id=1 AND ts>="2000-09-28 00:00:00";


--测试带有子查询的update
drop table if exists t1_1;
create table t1_1 (a int primary key, b int) shardkey=a;
drop table if exists t1_2;
create table t1_2 (a int primary key, b int) shardkey=a;
drop table if exists t1_3;
create table t1_3 (a int primary key, b int) shardkey=a;
insert into t1_1(a, b) values (10, 10);
insert into t1_1(a, b) values (9, 9);
insert into t1_1(a, b) values (8, 8);
insert into t1_1(a, b) values (7, 7);
insert into t1_1(a, b) values (6, 6);
insert into t1_1(a, b) values (5, 5);
insert into t1_1(a, b) values (4, 4);
insert into t1_1(a, b) values (3, 3);
insert into t1_1(a, b) values (2, 2);
insert into t1_1(a, b) values (1, 1);

insert into t1_2 select * from t1_1;

insert into t1_3 select * from t1_1;

update t1_1 set b=1 where exists(select * from t1_2 where t1_1.a=t1_2.a order by 1) limit 3;

update t1_1 set b=-1 where a in (select b from t1_2 order by 1) order by a limit 3;



--update 不支持更新主键

MySQL [test]> update t1_1 set a=b where exists(select 1 from t1_2 where a=t1_1.b);

ERROR 658 (HY000): Proxy ERROR: Join internal error: cannot update primary key



--update 不支持更新 shardkey

MySQL [test]> update t1_1 set a=200 where b=1;

ERROR 682 (HY000): Proxy ERROR: Something went wrong: can not update the shardkey



--显示开启事务用 delete/insert 替代 update

begin;

delete from t1_1 where b=1;

insert into t1_1(a,b) values(200,1);

commit;



--不支持 update 列表中带有 sum 的子查询

MySQL [test]> update t1_1 set b=(select max(b) from t1_2 where t1_2.a=t1_1.a) where 1;

ERROR 658 (HY000): Proxy ERROR: Join internal error: do not support subquery/sum in update list
```

```
--多表更新语法，但只更新一个表

MySQL [test]> update t1_1, t1_2 set t1_1.b=-1 where t1_1.a=t1_2.b and t
1_2.a<3;

Query OK, 0 rows affected (0.01 sec)



--不支持 order by 和 limit

MySQL [test]> update t1_1, t1_2 set t1_1.b=-1 where t1_1.a=t1_2.b and t
1_2.a<3 order by t1_1.a limit 3;

ERROR 658 (HY000): Proxy ERROR: Join internal error: Incorrect usage of
 UPDATE and ORDER



--不支持更新多个表

MySQL [test]> update t1_1, t1_2 set t1_1.b=-1, t1_2.b=-1 where t1_1.a=t
1_2.b and t1_2.a<3;

ERROR 658 (HY000): Proxy ERROR: Join internal error: multi update is no
t supported yet.



--更新一个表，但 value 引用了另外一个表

MySQL [test]> update t1_1, t1_2 set t1_1.b= t1_2.b+1 where t1_1.a=t1_2.
b and t1_2.a<3;

Query OK, 2 rows affected (0.01 sec)



--不支持 list 分区表更新分区键

drop table if exists list_user;

CREATE TABLE list_user

(id int, name varchar(255),

city varchar(255), primary key(id))

shardkey=id

PARTITION by list(city)

(PARTITION p0 values in ('Beijin','Shanghai','Shenzhen'),
 PARTITION p1 values in ('Nanjin', 'Chongqing','Wuhan'));

insert into list_user (id, name,city) values (1,'Rain','Beijin'),(22,'Storm','Beijin'),(103,'wind','Nanjin');



MySQL [test]> update list_user set city='Nanjin' where id in (select id from list_user,t1_1 where t1_1.a=list_user.id and t1_1.a <3 );

ERROR 913 (HY000): Proxy ERROR:Join internal error: sub partitioned table do not support such update yet!



MySQL [test]> update list_user set city='Nanjin' where id=1;

ERROR 682 (HY000): Proxy ERROR: Something went wrong: can not update the subshardkey



--不支持范围分区表更新分区键

drop table if exists range_part;

create table range_part

(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));

insert into range_part (a,b) values (1,11),(22,2),(103,1);



MySQL [test]> update range_part set b=11 where a in (select a from range_part,t1_1 where t1_1.a=range_part.id and t1_1.a <3 );

ERROR 913 (HY000): Proxy ERROR:Join internal error: sub partitioned table do not support such update yet!



MySQL [test]> update range_part set b=11 where a=103;

ERROR 682 (HY000): Proxy ERROR: Something went wrong: can not update the subshardkey
```

6.4 Utility 语句

6.4.1 DESCRIBE 语句

DESCRIBE 用于获取表结构信息：

示例：

```
mysql> DESCRIBE City;

+------------+----------+------+-----+---------+----------------+

| Field      | Type     | Null | Key | Default | Extra          |

+------------+----------+------+-----+---------+----------------+

| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |

| Name       | char(35) | NO   |     |         |                |

| Country    | char(3)  | NO   | UNI |         |                |

| District   | char(20) | YES  | MUL |         |                |

| Population | int(11)  | NO   |     | 0       |                |

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

6.4.2 EXPLAIN 语句

6.4.2.1 语法

```
{EXPLAIN | DESCRIBE | DESC}

    tbl_name [col_name | wild]


{EXPLAIN | DESCRIBE | DESC}

    [explain_type]

    {explainable_stmt | FOR CONNECTION connection_id}


{EXPLAIN | DESCRIBE | DESC} ANALYZE [FORMAT = TREE] select_statement
```

```
explain_type: {
    FORMAT = format_name
}

format_name: {
    TRADITIONAL
  | JSON
  | TREE
}

explainable_stmt: {
    SELECT statement
  | TABLE statement
  | DELETE statement
  | INSERT statement
  | REPLACE statement
  | UPDATE statement
}
```

注意事项：

- 查看执行计划，SQL 不会真正执行
- 在只读的 DB 上，无法查看写 SQL 的执行计划

示例：

```
DROP TABLE if exists employees;

CREATE TABLE employees (
    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;


MySQL [test]> explain select id,fname,lname from employees where id=20\
G;

*************************** 1. row ***************************

           id: 1

  select_type: SIMPLE

        table: NULL

   partitions: NULL

         type: NULL

possible_keys: NULL

          key: NULL

      key_len: NULL

          ref: NULL

         rows: NULL

     filtered: NULL

        Extra: no matching row in const table

         info: set_1624363251_3, explain select id,fname,lname from `te
st`.`employees` where (id = 20)

1 row in set (0.00 sec)
```

执行计划中各字段含义：

- id：执行行顺序，按 1,2,3,4…进行排序。在所有组中，id 值越大，优先级越高，越先执行。id 如果相同，可以认为是一组，从上往下顺序执行

- select_type：select 的类型。

- table：输出记录的表，对应行正在访问哪一个表，表名或者别名，可能是临时表或者 union 合并结果集

- partitions：符合的分区

- type：显示的是访问类型，访问类型表示以何种方式去访问数据，例如全表扫描

- possible_keys：优化器可能使用到的索引

- key：优化器实际选择的索引

- key_len：表示索引中使用的字节数，可以通过 key_len 计算查询中使用的索引长度

- ref：显示索引的哪一列被使用了，如果可能的话，是一个常数

- rows：优化器预估的记录数量，根据表的统计信息及索引使用情况，大致估算出找出所需记录需要读取的行数

- filtered：该 filtered 列指示将按表条件过滤的表行的估计百分比。 最大值为 100，这意味着不会对行进行过滤。 值从 100 开始减少表示过滤量增加

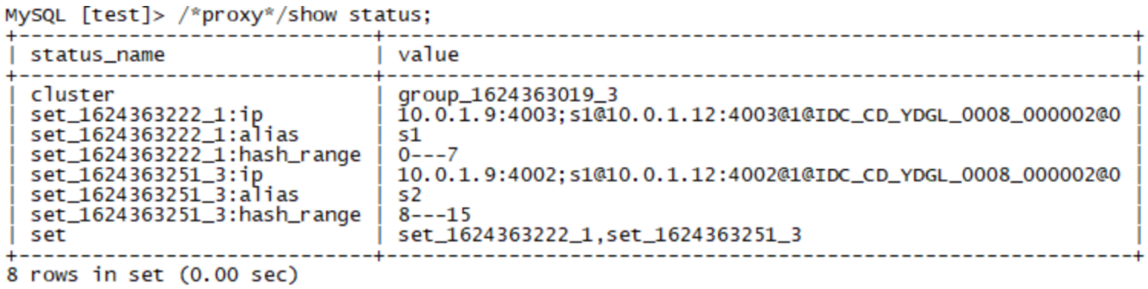
- Extra：额外的显示选项

- info：网关下推，记录了实际发往的 set 名称和 sql 信息，info 这个一列信息是分布式实例执行计划特有的

6.4.2.2 网关下推示例

6.4.2.2.1 测试表准备

```
--创建测试表

drop table if exists t1;

create table t1(a int key, b int) shardkey=a;

drop table if exists t2;

create table t2(a int key, b int) shardkey=a;



--集群的结构，包含 2 个 set
```



```
MySQL [test]> /*proxy*/show status;
+-----------------------------+-------------------------------------------------------------+
| status_name                 | value                                                       |
+-----------------------------+-------------------------------------------------------------+
| cluster                     | group_1624363019_3                                          |
| set_1624363222_1:ip         | 10.0.1.9:4003;s1@10.0.1.12:4003@1@IDC_CD_YDGL_0008_000002@0 |
| set_1624363222_1:alias      | s1                                                          |
| set_1624363222_1:hash_range | 0---7                                                       |
| set_1624363251_3:ip         | 10.0.1.9:4002;s1@10.0.1.12:4002@1@IDC_CD_YDGL_0008_000002@0 |
| set_1624363251_3:alias      | s2                                                          |
| set_1624363251_3:hash_range | 8---15                                                      |
| set                         | set_1624363222_1,set_1624363251_3                           |
+-----------------------------+-------------------------------------------------------------+
8 rows in set (0.00 sec)
```

6.4.2.2.2 select 查询的下推

（1）指定了 shardkey 的单表查询。根据 shardkey 的哈希值计算出目标 set，然后将查询直接下推给目标 set 执行。

```
-- info 字段展示了发送的目标 set，以及下推的查询
MySQL [test]> explain select * From t1 where a=1\G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: no matching row in const table
         info: set_1624363222_1, explain select * from `test`.`t1` wher
e (a = 1)
1 row in set (0.00 sec)
```

（2）未指定 shardkey 的单表查询。将查询广播给所有目标 set 执行

```
-- 广播给两个 set 执行，因此返回了两条记录，其中 info 字段展示了发送的目标 se
t，以及下推的查询
MySQL [test]> explain select * From t1 where b=1\G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
```

```
        table: t1
   partitions: p0,p1,p2,p3,p4,p5,p6,p7
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using where
         info: set_1624363222_1, explain select * from `test`.`t1` wher
e (b = 1)
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: p8,p9,p10,p11,p12,p13,p14,p15
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using where
         info: set_1624363251_3, explain select * from `test`.`t1` wher
e (b = 1)
2 rows in set (0.01 sec)
```

（3）多表连接查询。当 shardkey 相等时，将查询直接下推给 db 执行。

```
-- 广播给两个 set 执行，因此返回了两条记录，其中 info 字段展示了发送的目标 set，以及下推的查询

-- shardkey 相等，但 shardkey 未指定明确的值，因此广播给所有 set 执行。

MySQL [test]> explain select * from t1, t2 where t1.a=t2.a;



-- shardkey 相等，且 shardkey 指定了明确的值，因此 shardkey 的哈希值转给目标 set 执行。

MySQL [test]> explain select * from t1, t2 where t1.a=t2.a and t1.a=1;



-- shardkey 相等，且 shardkey 指定了多个明确的值，因此 shardkey 的哈希值转给多个目标 set 执行。

MySQL [test]> explain select * from t1, t2 where t1.a=t2.a and t1.a in (1,2,3);



-- shardkey 相等，且 shardkey 指定了多个明确的值，但当前网关在计算 shardkey 的值时会忽略'or'谓词，因此将广播给所有 set 执行。

MySQL [test]> explain select * from t1, t2 where t1.a=t2.a and (t1.a=1 or t1.a=2);
```

（4）常用聚合函数，包括 sum、count、avg、max 以及 min 的下推。

```
-- 网关将查询广播给所有 set，并对 set 返回的聚合结果进行累加

MySQL [test]> explain select count(1) from t1;



-- 网关将 avg 转换为 sum、count，并广播给所有 set 执行，再根据 set 返回的 sum、count 值计算出全局的 avg

MySQL [test]> explain select avg(a) from t1;



-- 多表连接时，表的 shardkey 相等，网关将查询广播给所有 set 执行，并对 set 返回的聚合结果进行累加

MySQL [test]> explain select sum(t1.a) from t1, t2 where t1.a=t2.a;
```

```
-- 多表连接时，表的 shardkey 相等，且 shardkey 指定了明确的值，网关将查询转发给目标 set 执行

MySQL [test]> explain select sum(t1.a) from t1, t2 where t1.a=t2.a and t1.a=1;



-- 网关将查询广播给所有 set 执行，再对 set 返回的结果进行归并排序，计算出每个分组的全局 sum 值

MySQL [test]> explain select sum(a) from t1 group by b\G;

*************************** 1. row ***************************

           id: 1

  select_type: SIMPLE

        table: t1

   partitions: p8,p9,p10,p11,p12,p13,p14,p15

         type: ALL

possible_keys: NULL

          key: NULL

      key_len: NULL

          ref: NULL

         rows: 1

     filtered: 100.00

        Extra: Using temporary; Using filesort

         info: set_1624363251_3, explain select sum(a),b, COLLATION(b) from `test`.`t1` group by b order by b

*************************** 2. row ***************************

           id: 1

  select_type: SIMPLE

        table: t1

   partitions: p0,p1,p2,p3,p4,p5,p6,p7

         type: ALL
```

```
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using temporary; Using filesort
         info: set_1624363222_1, explain select sum(a),b, COLLATION(b)
from `test`.`t1` group by b order by b

2 rows in set (0.00 sec)
```

（5）distinct 的下推

```
-- 将 distinct 下推给 set 执行，同时额外追加 order by 操作。网关对 set 返回的有
序元组进行归并排序和去重，从而得到全局去重的结果。

MySQL [test]> explain select distinct b from t1\G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: p8,p9,p10,p11,p12,p13,p14,p15
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using temporary; Using filesort
         info: set_1624363251_3, explain select distinct b from `test`.
```

```
`t1` order by b
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: p0,p1,p2,p3,p4,p5,p6,p7
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using temporary; Using filesort
         info: set_1624363222_1, explain select distinct b from `test`.
`t1` order by b
2 rows in set (0.00 sec)
```

（6）子查询的下推

```
-- 通过等值传递，能够推断出父查询和子查询中表的 shardkey 相等时，则网关将查询
下推给 db 执行。
-- 注意：由于实现方式的不同，部分查询的 explain 的结果为 json 的形式。其中 DBQu
ery 字段描述了下推到 db 执行的查询。
-- IN 子查询
MySQL [test]> explain select * from t1 where t1.a in (select a from t2)
\G;
*************************** 1. row ***************************
trace: [
   {
      "ProxyDeduplicate " : "false",
      "Query" : "set_1624363222_1 set_1624363251_3  , Select `t1`.`a`, `t1`.`b` from `test`.`t1` where (`test`.`t1`.`a`) in (select `test`.`t2`.`a` from `test`.`t2`)",
      "QueryMode" : "Hash"
   }
]


-- EXISTS 子查询
MySQL [test]> explain select * from t1 where exists (select * From t2 where t1.a=t2.a)\G;
*************************** 1. row ***************************
trace: [
   {
      "ProxyDeduplicate " : "false",
      "Query" : "set_1624363222_1 set_1624363251_3  , Select `t1`.`a`, `t1`.`b` from `test`.`t1` where exists(select 1 from `test`.`t2` where (`test`.`t1`.`a` = `test`.`t2`.`a`))",
      "QueryMode" : "Hash"
   }
]


-- 通过等值传递，能够推断出父查询和子查询中表的 shardkey 相等时，则网关将查询下推给 db 执行。
MySQL [test]> explain select * from t1 where t1.a in (select b from t2 where t2.a=t2.b)\G;
*************************** 1. row ***************************
trace: [
   {
      "ProxyDeduplicate " : "false",
      "Query" : "set_1624363222_1 set_1624363251_3  , Select `t1`.`a`, `t1`.`b` from `test`.`t1` where (`test`.`t1`.`a`) in (select `test`.`t2`.`b` from `test`.`t2` where (`test`.`t2`.`a` = `test`.`t2`.`b`))",
      "QueryMode" : "Hash"
   }
]
```

（7）distinct 聚合函数的下推，例如 count(distinct 表达式)、sum(distinct 表达式)等

```
-- 不存在分组(group by)和排序(order by)操作时，网关只下推 distinct 查询给所有 set 执行.

-- 网关对 set 返回的结果再次去重，从而计算 count(distinct b)的值。

MySQL [test]> explain select count(distinct b) from t1 \G

*************************** 1. row ***************************

trace: [
   {
      "AggFunc " : "count(distinct `test`.`t1`.`b`)",
      "ProxyDeduplicate " : "false",
      "Query" : "set_1624363222_1 set_1624363251_3  , Select DISTINCT `t1`.`b` from `test`.`t1` where 1",
      "QueryMode" : "Hash"
   }
]


-- 当存在分组(group by)操作时，网关下推 distinct 操作，并在下推的查询中额外添加 order by 语句。

-- 网关对 set 返回的有序元组按照'分组列'进行归并排序，并计算每个分组的聚合函数 count(distinct b)的值。

MySQL [test]> explain select count(distinct b) from t1 group by a\G

*************************** 1. row ***************************

trace: [
   {
      "AggFunc " : "count(distinct `test`.`t1`.`b`)",
      "DBGroupColumns " : "`test`.`t1`.`a`",

      "DBSortedColumns " : "`test`.`t1`.`a`",

      "ProxyDeduplicate " : "false",

      "Query" : "set_1624363222_1 set_1624363251_3  , Select DISTINCT `
t1`.`a`, `t1`.`b`, `test`.`t1`.`a` from `test`.`t1` where 1 order by 3
",

      "QueryMode" : "Hash"

   }

]



-- 当同时存在分组(group by)以及排序(order by)操作时，网关按照前面的例子先计算出分组聚合操作的结果，再利用临时表对分组聚合的结果进行排序。

-- 其中 PorxyTmptable 字段展示了创建的临时表；ProxyQuery 展示了需要在临时表上执行的查询。

MySQL [test]> explain select a, count(distinct b) as cnt from t1 group
by a order by cnt \G

*************************** 1. row ***************************

trace: [

   {

      "AggFunc " : "count(distinct `test`.`t1`.`b`)",

      "DBGroupColumns " : "`test`.`t1`.`a`",

      "DBSortedColumns " : "`test`.`t1`.`a`",

      "ProxyDeduplicate " : "false",

      "ProxyQuery" : "SELECT f0 ,f1  FROM proxy_tmpdb.tmptbl  ORDER BY
f1  ",

      "ProxySortedColumns " : "count(distinct `test`.`t1`.`b`)",

      "ProxyTmptable" : "CREATE TEMPORARY TABLE proxy_tmpdb.tmptbl (f0
int(11),f1  bigint)",

      "Query" : "set_1624363222_1 set_1624363251_3  , Select DISTINCT `
t1`.`a`, `t1`.`b`, `test`.`t1`.`a` from `test`.`t1` where 1 order by 3
",
      "QueryMode" : "Hash"
   }
]
```

6.4.2.2.3 Delete/update 的下推

（1）指定了 shardkey 值的单表查询

```
-- 网关根据 shardkey 的值计算出目标 set，并将查询直接下推给目标 set。
-- 注意：info 字段展示了目标 set 以及下推的查询语句
MySQL [test]> explain delete from t1 where a=1\G
*************************** 1. row ***************************
           id: 1
  select_type: DELETE
        table: t1
   partitions: p1
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: Using where
         info: set_1624363222_1, explain delete  from `test`.`t1` where
 (a = 1)

MySQL [test]> explain update t1 set b=1 where a=1\G
*************************** 1. row ***************************
           id: 1
  select_type: UPDATE
        table: t1
   partitions: p1
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: Using where
         info: set_1624363222_1, explain update `test`.`t1` SET b=1 whe
re (a = 1)
```

（2）没有指定 shardkey 值的单表查询

```
-- 将查询广播给所有 set。
-- 注意：info 字段展示了目标 set 以及下推的查询语句
MySQL [test]> explain delete from t1 where 1\G
*************************** 1. row ***************************
           id: 1
  select_type: DELETE
        table: t1
   partitions: p0,p1,p2,p3,p4,p5,p6,p7
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
```

```
     filtered: 100.00
        Extra: NULL
         info: set_1624363222_1, explain delete  from `test`.`t1` where
 1
*************************** 2. row ***************************
           id: 1
  select_type: DELETE
        table: t1
   partitions: p8,p9,p10,p11,p12,p13,p14,p15
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: NULL
         info: set_1624363251_3, explain delete  from `test`.`t1` where
 1


MySQL [test]> explain update t1 set b=1 where 1\G
*************************** 1. row ***************************
           id: 1
  select_type: UPDATE
        table: t1
   partitions: p0,p1,p2,p3,p4,p5,p6,p7
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: NULL
         info: set_1624363222_1, explain update `test`.`t1` SET b=1 whe
re 1
*************************** 2. row ***************************
           id: 1
  select_type: UPDATE
        table: t1
   partitions: p8,p9,p10,p11,p12,p13,p14,p15
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: NULL
         info: set_1624363251_3, explain update `test`.`t1` SET b=1 whe
re 1
```

（3）多表更新操作，且表的 shardkey 相等

```
-- 多表更新操作，且 shardkey 相等时，网关将查询直接下推给后端 set 执行。
-- 如果 shardkey 为一个明确的值，则根据 shardkey 的值计算出目标 set；否则，将
查询广播给所有 set 执行。
MySQL [test]> explain update t1, t2 set t1.b=t2.b where t1.a=t2.a and t
1.a=202\G
*************************** 1. row ***************************
```

```
           id: 1

  select_type: UPDATE

        table: NULL

   partitions: NULL

         type: NULL

possible_keys: NULL

          key: NULL

      key_len: NULL

          ref: NULL

         rows: NULL

     filtered: NULL

        Extra: no matching row in const table

         info: set_1624363222_1, explain update `test`.`t1` join `test
`.`t2` SET t1.b=t2.b where ((t1.a = t2.a) and (t1.a = 202))
```

（4）多表更新操作，且表的 shardkey 不相等，或者包含子查询

网关将构建与更新操作对应的 select 查询，计算出被更新行的主键、被更新列的新值，再构建相应的 update 语句发送给 set 执行。因此，其下推策略同 select 查询。

```
-- 对于如下查询，网关将构建与之对应的 select 查询：

--      select * from t1, t2 where t1.a=1 and t2.a=21 and t1.b != t2.b
group by t1.a;

-- 然后再根据执行的结果，为 t1 中需要被更新的每个元组构建如下查询给 set 执行：

--      update t1 set t1.b=... where t1.a= ...

MySQL [test]> explain update t1, t2 set t1.b=t2.b where t1.a=1 and t2.a
=21\G

*************************** 1. row ***************************

trace: [

   {

      "optype" : "TableRename",
      "t1" : "T2e(a,b)",

      "t2" : "T6(a,b)",

      "timecost" : "0.031000"

   },

   {

      "0.OpType         " : "Load table",

      "1.TableName      " : "T6",

      "2.PushedDownCond " : "( /*filter*/((`test`.`t2`.`a`=21)))",

      "3.NumOfRows      " : "0",

      "4.AddedCond      " : "`test`.`t2`.`a` is null",

      "Query" : "set_1624363251_3  , select `a`,`b` from `test`.`t2` t2
 where ( /*filter*/((`test`.`t2`.`a`=21))) limit 1000",

      "QueryMode" : "Hash",

      "timecost" : "0.640000"

   },

   {

      "0.OpType         " : "Load table",

      "1.TableName      " : "T2e",

      "2.PushedDownCond " : "(0)",

      "3.NumOfRows      " : "0",

      "4.AddedCond      " : "`test`.`t1`.`a` is null",

      "Query" : "AllSets , select `a`,`b` from `test`.`t1` t1 where (0)
 limit 1000",

      "QueryMode" : "All",

      "timecost" : "0.544000"

   },

   {

      "Query" : " select `test`.`t1`.`a`,`test`.`t1`.`b`,`test`.`t2`.`b
` from `test`.`T2e` `t1` join `test`.`T6` `t2` where 0 group by `test`.
`t1`.`a` for update of `test`.`t1`  ",
       "timecost" : "0.001000"

    }

]
```

6.4.3 USE 语句

语法如下：

```
use db_name
```

示例：

```
MySQL [test]> USE db1;

MySQL [test]> SELECT COUNT(*) FROM mytable;

MySQL [test]> USE db2; SELECT COUNT(*) FROM mytable;
```

6.5 注释透传功能

注释透传指支持透传 SQL 语句到对应的一个或者多个物理分片（Set），并透传到分表键（Shardkey）对应的分片（Set）中的操作方式。

具体语法如下：

```
/*sets:set_1*/
/*sets:set_1,set_2*/ （set 名字可以通过/*proxy*/show status 查询）
/*sets:allsets */
```

注意事项：

- 对于分布式实例，Proxy 会对 SQL 进行语法解析，但有比较严格的限制，如果用户想在某个物理分片（set）中执行 SQL 语句，可以使用该功能。

示例：

```
MySQL [test]> DROP TABLE IF EXISTS test1;

Query OK, 0 rows affected (0.08 sec)
```

```
MySQL [test]> create table test1 (a int key, b int, c char(20)) shardkey=a;
Query OK, 0 rows affected (1.71 sec)


--加载300行数据到test1表之后：
MySQL [test]> select count(*) from test1;
+----------+
| count(*) |
+----------+
|      300 |
+----------+
1 row in set (0.12 sec)

MySQL [test]> select count(*) from test1;
+----------+
| count(*) |
+----------+
|      300 |
+----------+
1 row in set (0.11 sec)

MySQL [test]> /*sets:allsets */ select count(*) from test1;
+----------+------------------+
| count(*) | info             |
+----------+------------------+
|      150 | set_1619374020_1 |
|      150 | set_1619508344_3 |
+----------+------------------+
2 rows in set (0.02 sec)

MySQL [(none)]> /*proxy*/ show status;
+-----------------------------+-------------------------------------------+
| status_name                 | value                                     |
+-----------------------------+-------------------------------------------+
| cluster                     | group_1619373877_13                       |
| set_1619374020_1:ip         | 10.0.0.17:4007;s1@10.0.0.16:4007@1@IDC1@0 |
| set_1619374020_1:alias      | s1                                        |
| set_1619374020_1:hash_range | 0---31                                    |
| set_1619508344_3:ip         | 10.0.0.17:4008;s1@10.0.0.16:4008@1@IDC1@0 |
| set_1619508344_3:alias      | s2                                        |
| set_1619508344_3:hash_range | 32---62                                   |
| set                         | set_1619374020_1,set_1619508344_3         |
+-----------------------------+-------------------------------------------+
8 rows in set (0.00 sec)

MySQL [test]> /*sets:set_1619374020_1*/ select count(*) from test1;
+----------+------------------+
| count(*) | info             |
+-----------+------------------+
|       150 | set_1619374020_1 |
+-----------+------------------+
1 row in set (0.04 sec)

MySQL [test]> /*set_1619508344_3*/ select count(*) from test1;
+-----------+
| count(*) |
+-----------+
|       150 |
+-----------+
1 row in set (0.11 sec)


MySQL [test]> delete from test1;
ERROR 913 (HY000): Proxy ERROR:Join internal error: delete query has no where
clause

MySQL [test]> /*sets:allsets*/delete from test1;
Query OK, 300 rows affected (0.04 sec)
```

6.6 预处理语句

TDSQL 支持预处理协议，使用方式与单机 MySQL 相同，例如：

- PREPARE Syntax
- EXECUTE Syntax

二进制协议的支持：

- COM_STMT_PREPARE
- COM_STMT_EXECUTE

注意事项：

- 目前 TDSQL 只对 Prepare/Execute 命令做语法兼容，从性能角度的话，在分布式下建议用户尽量不要使用该种方式，直接使用文本协议。

示例：

```
MySQL [test]> DROP TABLE IF EXISTS test1;

Query OK, 0 rows affected (0.08 sec)



MySQL [test]> create table test1(a int not null primary key,b int) shardkey=a;
Query OK, 0 rows affected (1.71 sec)

MySQL [test]> insert into test1(a,b) values(5,6),(3,4),(1,2);
Query OK, 3 rows affected (0.06 sec)
Records: 3  Duplicates: 0  Warnings: 0

MySQL [test]> select a,b from test1;
+---+------+
| a | b    |
+---+------+
| 1 |    2 |
| 3 |    4 |
| 5 |    6 |
+---+------+
3 rows in set (0.02 sec)

mysql> prepare ff from "select a,b 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)
```

7 全局唯一数字序列的使用

TDSQL 支持全局唯一数字序列（auto_increment）的使用；暂时仅保证自增字段全局唯一和递增性，但是不保证单调递增（即按时间顺序的绝对递增性）。

全局唯一数字序列（auto_increment）长8字节，最大为18446744073709551616，因此，您无需担心该值溢出。

注意事项：

- select last_insert_id()命令只能与Shard表和广播表的自增字段一起使用，不支持与Noshard表的使用。

示例：

创建自增字段的表：

```
mysql> DROP TABLE IF EXISTS auto_inc;


mysql> create table 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)
```

插入自增字段的分表：

```
mysql> insert into 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 [test]> select * from auto_inc;

+---+------+------+---+

| a | b    | c    | d |

+---+------+------+---+

| 1 |    2 | 1008 | 4 |

| 1 |    2 | 1007 | 3 |

+---+------+------+---+

2 rows in set (0.00 sec)
```

自增字段的空洞处理：

由于 auto_increment 仅保证自增字段全局唯一和递增性，如果在节点调度切换、重启等过程中，自增长字段中间会出现空洞，例如：

```
MySQL [test]>insert into auto_inc (a,b,d,c)
values(11,12,13,0),(21,22,23,0);

Query OK, 2 rows affected (0.00 sec)



MySQL [test]> select * from auto_inc;

+----+------+-----+----+

| a  | b    | c    | d  |

+----+------+-----+----+

| 11 |   12 | 1009 | 13 |

| 21 |   22 | 1010 | 23 |

|  1 |    2 | 1008 |  4 |

|  1 |    2 | 1007 |  3 |

+----+------+-----+----+

4 rows in set (0.00 sec)
```

可更改当前值，命令如下：

```
MySQL [test]> alter table auto_inc auto_increment=100;

Query OK, 0 rows affected (0.03 sec)
```

目前不支持通过 `insert into auto_inc set c=100` 语法插入数据，如果用户要指定自增的值，需要使用以下语法：

```
insert into auto_inc (a,b,d,c) values(300,400,100,500);
```

通过 select last_insert_id()命令获取自增值，如果用户不指定自增值，可以通过 select last_insert_id()命令获取，暂不支持直接从 Insert 返回包获取，详见如下：

```
MySQL [test]> insert into auto_inc (a,b,d,c)
```

```
values(5,6,7,8),(11,12,14,19);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

MySQL [test]> select * from auto_inc;
+-----+------+------+-----+
| a   | b    | c    | d   |
+-----+------+------+-----+
|  11 |   12 | 1009 |  13 |
|   5 |    6 |    8 |   7 |
|  11 |   12 |   19 |  14 |
| 300 |  400 |  500 | 100 |
|  21 |   22 | 1010 |  23 |
|   1 |    2 | 1008 |   4 |
|   1 |    2 | 1007 |   3 |
+-----+------+------+-----+
7 rows in set (0.00 sec)

MySQL [test]> select last_insert_id();
+------------------+
| last_insert_id() |
+------------------+
| 1009             |
+------------------+
1 row in set (0.00 sec)
```

8 SEQUENCE

本节主要介绍创建、删除、查询和使用 Sequence，以及获取显示 Sequence 的值。Sequence 语法和 MariaDB 兼容，但是需保证分布式全局递增且数值唯一。

注意事项：

- 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。

示例：

创建 Sequence：

```
create tdsql_sequence test.seq1 start with 12 tdsql_minvalue 10 maxvalue
50000  tdsql_increment by 5  tdsql_nocycle;

create tdsql_sequence test.seq2 start with 12 tdsql_minvalue 10 maxvalue
50000  tdsql_increment by 1  tdsql_cycle;
```

查询 Sequence：

```
show create tdsql_sequence test.seq2;
```

使用 Sequence 获取下一个数值，语句如下：

```
select tdsql_nextval(test.seq2);

select next value for test.seq2;
```

删除 Sequence：

```
drop tdsql_sequence test.seq1;

drop tdsql_sequence test.seq2;
```

nextval 命令可以用在 insert 语句中。使用如下：

```
MySQL [test]> DROP TABLE IF EXISTS test3;

MySQL [test]> create table test3(a int not null primary key,b int,c char
(10)) shardkey=a;

MySQL [test]> insert into test3(a,c) values(1,'A');

Query OK, 1 row affected (0.00 sec)

MySQL [test]> insert into test3(a,c) values(40,'records5');

Query OK, 1 row affected (0.00 sec)


MySQL [test]> select a,c from test3;

+----+----------+

| a  | c        |

+----+----------+

|  1 | A        |

| 40 | records5 |

+----+----------+

2 rows in set (0.00 sec)


MySQL [test]> insert into test3(a,c) values(tdsql_nextval(test.seq2),3);

Query OK, 1 row affected (0.01 sec)
```

Seq2 的初始值为 12，此次 insert 的值为 12

```
MySQL [test]> select a,c from test3;

+----+----------+

| a  | c        |

+----+----------+

| 40 | records5 |

|  1 | A        |

| 12 | 3        |

+----+----------+

3 rows in set (0.00 sec)
```

如需获取上一次的值：

```
MySQL [test]> select tdsql_lastval(test.seq2);
+----+
| 12 |
+----+
| 12 |
+----+
1 row in set (0.00 sec)

MySQL [test]> select tdsql_previous value for test.seq2;
+----+
| 12 |
+----+
| 12 |
+----+
1 row in set (0.00 sec)
```

设置下一个序列数值为 2000，tdsql_setval 内的第三个参数默认为 1，表示 2000 这个值用过了，下一次不包含 2000，如果为 0，则下一个从 2000 开始。

```
MySQL [test]> select tdsql_setval(test.seq2,2000,1)
    -> ;
+------+
| 2000 |
+------+
| 2000 |
+------+
1 row in set (0.01 sec)
```

设置的值只能比当前数值大，否则将返回数值为 0。设置下一个序列数值时，如果比当前数值小，则系统将没有反应，例如：

```
MySQL [test]> select tdsql_nextval(test.seq2);

+------+

| 2001 |

+------+

| 2001 |

+------+

1 row in set (0.01 sec)
```

seq2 设置为 10，系统返回 0

```
MySQL [test]> select tdsql_setval(test.seq2,10);

+---+

| 0 |

+---+

| 0 |

+---+

1 row in set (0.03 sec)
```

如果设置的比当前数值大，成功返回当前设置的值。

```
MySQL [test]> select tdsql_setval(test.seq2,2010);

+------+

| 2010 |

+------+

| 2010 |

+------+

1 row in set (0.02 sec)
```

```
MySQL [test]> select tdsql_nextval(test.seq2);

+------+

| 2011 |

+------+

| 2011 |

+------+

1 row in set (0.01 sec)
```

9 支持的字符集和时区

TDSQL 在后端存储支持 MySQL 的所有字符集和字符序。具体显示如下：

```
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)
```

查看当前连接的相关字符集：

```
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/sh
are/charsets/ |
+--------------------------+---------------------------------------
--------------+
```

设置当前连接的相关字符集：

```
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/sh
are/charsets/ |
+--------------------------+---------------------------------------
--------------+
```

注意事项：TDSQL 不支持通过命令行设置参数，需要通过赤兔管理台进行设置。

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 table test.tt (ts timestamp, dt datetime,c int key) s
hardkey=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)
```

10 支持的函数

分布式实例支持以下 7 种类型的函数：

- 流程控制函数（Control Flow Functions）
- 字符串函数（String Functions）
- 数字函数（Numeric Functions and Operators）
- 日期时间函数（Date and Time Functions）
- 聚合函数（Aggregate (GROUP BY) Functions）
- 位函数（Bit Functions and Operators）
- 转换函数（Cast Functions and Operators）

以上类型的各函数具体描述如下：

流程控制函数（Control Flow Functions）

函数名	描述

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

字符串函数（String Functions）

函数名	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

函数名	描述
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

函数名	描述
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
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

函数名	描述
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
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

函数名	描述
	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

数字函数（Numeric Functions and Operators）

函数名	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

函数名	描述
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
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

函数名	描述
-	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

函数名	描述
TRUNCATE()	Truncate to specified number of decimal places
-	Change the sign of the argument

日期时间函数（Date and Time Functions）

函数名	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

函数名	描述
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)
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()

函数名	描述
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 SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date

函数名	描述
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
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 datetim 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 sinc Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date

函数名	描述
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

聚合函数（Aggregate (GROUP BY) Functions）

函数名	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

位函数（Bit Functions and Operators）

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

转换函数（Cast Functions and Operators）

函数名	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