分布式数据库 TDSQL

开发指南

产品文档

腾讯云

文档目录

开发指南

概述

最近更新时间：2020-08-18 15:57:44

在 SQL 使用上，分布式实例高度兼容 MySQL 的协议和语法，但由于架构的差异，对于 SQL 有一定的限制，同时为了更好地发挥分布式的优势，建议业务在使用时尽量参考下文的建议。

分布式实例提供水平扩容能力，适合海量数据的场景。具有如下功能特性：

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

分布式实例支持三种不同类型的表：

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

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

使用限制

最近更新时间：2020-08-27 18:54:15

大特性限制

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

小语法限制

DDL

- 不支持 CREATE TABLE ... LIKE
- 不支持 CREATE TABLE ... SELECT
- 不支持 CREATE TEMPORARY TABLE
- 不支持 CREATE/DROP/ALTER SERVER/LOGFILE GROUP
- 不支持 ALTER 对分表键（shardkey）进行改名，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
- 不支持不带列名的 INSERT/REPLACE
- 不支持全局的 DELETE/UPDATE 使用 ORDER BY/LIMIT（版本>=14.4支持）
- 不支持不带 WHERE 条件的 UPDATE/DELETE
- 不支持 LOAD DATA/XML
- 不支持 SQL 中使用 DELAYED 和 LOW_PRIORITY，没有效果
- 不支持 INSERT ... SELECT（版本>14.4支持）
- 不支持 SQL 中对于变量的引用和操作，如：SET @c=1, @d=@c+1; SELECT @c, @d
- 不支持 index_hint
- 不支持 HANDLER/DO

管理 SQL

- 不支持 ANALYZE/CHECK/CHECKSUM/OPTIMIZE/REPAIR TABLE，需要用透传语法
- 不支持 CACHE INDEX
- 不支持 FLUSH
- 不支持 KILL
- 不支持 LOAD INDEX INTO CACHE

- 不支持 RESET
- 不支持 SHUTDOWN
- 不支持 SHOW BINARY LOGS/BINLOG EVENTS
- 不支持 SHOW WARNINGS/ERRORS和LIMIT/COUNT 的组合

其他限制

默认支持最大建表数量为1000，如需超越该限制，可 提交工单 申请。

兼容性

最近更新时间：2020-12-24 10:48:23

语言结构

分布式实例支持所有 MySQL 使用的文字格式，包括：

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

String Literals

String Literals 是一个 bytes 或者 characters 的序列，两端被单引号 ' 或者双引号 " 包围，TDSQL MySQL版 目前不支持 ANSI_QUOTES SQL MODE，双引号 " 包围的始终认为是 String Literals，而不是 identifier。

不支持 character set introducer，即 `[_charset_name]'string' [COLLATE collation_name]` 这种格式。

支持的转义字符：

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

Numeric Literals

数值字面值包括 integer、Decimal 类型、浮点数字面值。
integer 可以包括 . 作为小数点分隔，数字前可以有 - 或者 + 来表示正数或者负数。
精确数值字面值可以表示为如下格式：`1, .2, 3.4, -5, -6.78, +9.10` 。
科学记数法，如下格式：`1.2E3, 1.2E-3, -1.2E3, -1.2E-3` 。

Date and Time Literals

DATE 支持如下格式：

```
'YYYY-MM-DD' or 'YY-MM-DD'
'YYYYMMDD' or 'YYMMDD'
YYYYMMDD or YYMMDD
如：'2012-12-31', '2012/12/31', '2012^12^31', '2012@12@31' '20070523' , '070523'
```

DATETIME，TIMESTAMP 支持如下格式：

```
'YYYY-MM-DD HH:MM:SS' or 'YY-MM-DD HH:MM:SS'
'YYYYMMDDHHMMSS' or 'YYMMDDHHMMSS'
YYYYMMDDHHMMSS or YYMMDDHHMMSS
如'2012-12-31 11:30:45', '2012^12^31 11+30+45', '2012/12/31 11*30*45', '2012@12@31 11^30^45', 198
30905132800
```

Hexadecimal Literals

支持格式如下：

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

Bit-Value Literals

支持格式如下：

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

Boolean Literals

常量 TRUE 和 FALSE 等于1和0，大小写不敏感。

```
mysql> SELECT TRUE, true, FALSE, false;
+------+------+-------+-------+
| TRUE | TRUE | FALSE | FALSE |
+------+------+-------+-------+
| 1 | 1 | 0 | 0 |
```

```
+------+------+-------+-------+
1 row in set (0.03 sec)
```

NULL Values

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

字符集和时区

支持 MySQL 的所有字符集和字符序：

```
mysql> show character set;
+----------+---------------------------------+---------------------+--------+
| Charset | Description | Default collation | Maxlen |
+----------+---------------------------------+---------------------+--------+
| big5 | Big5 Traditional Chinese | big5_chinese_ci | 2 |
| dec8 | DEC West European | dec8_swedish_ci | 1 |
| cp850 | DOS West European | cp850_general_ci | 1 |
| hp8 | HP West European | hp8_english_ci | 1 |
| koi8r | KOI8-R Relcom Russian | koi8r_general_ci | 1 |
| latin1 | cp1252 West European | latin1_swedish_ci | 1 |
| latin2 | ISO 8859-2 Central European | latin2_general_ci | 1 |
| swe7 | 7bit Swedish | swe7_swedish_ci | 1 |
| ascii | US ASCII | ascii_general_ci | 1 |
| ujis | EUC-JP Japanese | ujis_japanese_ci | 3 |
| sjis | Shift-JIS Japanese | sjis_japanese_ci | 2 |
| hebrew | ISO 8859-8 Hebrew | hebrew_general_ci | 1 |
| tis620 | TIS620 Thai | tis620_thai_ci | 1 |
| euckr | EUC-KR Korean | euckr_korean_ci | 2 |
| koi8u | KOI8-U Ukrainian | koi8u_general_ci | 1 |
| gb2312 | GB2312 Simplified Chinese | gb2312_chinese_ci | 2 |
| greek | ISO 8859-7 Greek | greek_general_ci | 1 |
| cp1250 | Windows Central European | cp1250_general_ci | 1 |
| gbk | GBK Simplified Chinese | gbk_chinese_ci | 2 |
| latin5 | ISO 8859-9 Turkish | latin5_turkish_ci | 1 |
| armscii8 | ARMSCII-8 Armenian | armscii8_general_ci | 1 |
| utf8 | UTF-8 Unicode | utf8_general_ci | 3 |
| ucs2 | UCS-2 Unicode | ucs2_general_ci | 2 |
| cp866 | DOS Russian | cp866_general_ci | 1 |
| keybcs2 | DOS Kamenicky Czech-Slovak | keybcs2_general_ci | 1 |
| macce | Mac Central European | macce_general_ci | 1 |
| macroman | Mac West European | macroman_general_ci | 1 |
| cp852 | DOS Central European | cp852_general_ci | 1 |
```

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

查看当前连接的字符集：

```
mysql> show variables like "%char%";
+--------------------------+----------------------------------------------------------+
| Variable_name | Value |
+--------------------------+----------------------------------------------------------+
| character_set_client | latin1 |
| character_set_connection | latin1 |
| character_set_database | utf8 |
| character_set_filesystem | binary |
| character_set_results | latin1 |
| character_set_server | utf8 |
| character_set_system | utf8 |
| character_sets_dir | /data/tdsql_run/8812/percona-5.7.17/share/charsets/ |
+--------------------------+----------------------------------------------------------+
```

设置当前连接相关的字符集：

```
mysql> set names utf8;
Query OK, 0 rows affected (0.03 sec)
mysql> show variables like "%char%";
+--------------------------+----------------------------------------------------------+
| Variable_name | Value |
+--------------------------+----------------------------------------------------------+
| character_set_client | utf8 |
| character_set_connection | utf8 |
```

```
| character_set_database | utf8 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | utf8 |
| character_set_system | utf8 |
| character_sets_dir | /data/tdsql_run/8811/percona-5.7.17/share/charsets/ |
+--------------------------+----------------------------------------------------------+
```

说明：

分布式实例不支持设置全局参数，需要调用前台接口设置。

支持通过设置 time_zone 变量修改时区相关的属性：

```
mysql> show variables like '%time_zone%';
+------------------+--------+
| Variable_name | Value |
+------------------+--------+
| system_time_zone | CST |
| time_zone | SYSTEM |
+------------------+--------+
2 rows in set (0.00 sec)
mysql> create table test.tt (ts timestamp, dt datetime,c int) shardkey=c;
Query OK, 0 rows affected (0.49 sec)
mysql> insert into test.tt (ts,dt,c)values ('2017-10-01 12:12:12', '2017-10-01 12:12:12',1);
Query OK, 1 row affected (0.09 sec)
mysql> select * from test.tt;
+---------------------+---------------------+------+
| ts | dt | c |
+---------------------+---------------------+------+
| 2017-10-01 12:12:12 | 2017-10-01 12:12:12 | 1 |
+---------------------+---------------------+------+
1 row in set (0.04 sec)
mysql> set time_zone = '+12:00';
Query OK, 0 rows affected (0.00 sec)
mysql> show variables like '%time_zone%';
+------------------+--------+
| Variable_name | Value |
+------------------+--------+
| system_time_zone | CST |
| time_zone | +12:00 |
+------------------+--------+
2 rows in set (0.00 sec)
```

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

数据类型

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

数字

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

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

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

定点类型支持 DECIMAL和NUMERIC，格式 DECIMAL(M,D)

字符

支持如下字符类型：

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

日期

支持如下时间类型：

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

空间

支持如下空间类型：

```
GEOMETRY
POINT
LINESTRING
POLYGON
MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION
```

JSON

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

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

```
mysql> CREATE TABLE t1 (jdoc JSON,a int) shardkey=a;
Query OK, 0 rows affected (0.30 sec)
mysql> INSERT INTO t1 (jdoc,a)VALUES('{"key1": "value1", "key2": "value2"}',1);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO t1 (jdoc,a)VALUES('{"key1": "value1", "key2": 2}',2);
mysql> INSERT INTO t1 (jdoc,a)VALUES('[1, 2,',5);
ERROR 3140 (22032): Invalid JSON text: "Invalid value." at position 6 in value for column 't1.jdo
c'.
mysql> select * from t1;
+------------------------------------+---+
| jdoc | a |
+------------------------------------+---+
| {"key1": "value1", "key2": "value2"} | 1 |
| {"key1": "value1", "key2": 2} | 2 |
+------------------------------------+---+
2 rows in set (0.00 sec)
```

函数支持

Control Flow Functions

Name	Description
CASE	Case operator
IF()	If/else construct
IFNULL()	Null if/else construct
NULLIF()	Return NULL if expr1 = expr2

String Functions

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Return the index position of the first argument within the second argument
FORMAT()	Return a number formatted to specified number of decimal places
FROM_BASE64()	Decode to a base-64 string and return result
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters

Name	Description
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH	Perform full-text search
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
RPAD()	Append string the specified number of times

Name	Description
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 occurrences of the delimiter
TO_BASE64()	Return the argument converted to a base-64 string
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for a string

Numeric Functions and Operators

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

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

Date and Time Functions

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

Name	Description
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0

Name	Description
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1–53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Aggregate (GROUP BY) Functions

Name	Description
AVG()	Return the average value of the argument
COUNT()	Return a count of the number of rows returned
MAX()	Return the maximum value
MIN()	Return the minimum value
SUM()	Return the sum

Bit Functions and Operators

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

Cast Functions and Operators

Name	Description
BINARY	Cast a string to a binary string
CAST()	Cast a value as a certain type
CONVERT()	Cast a value as a certain type

连接数据库

最近更新时间：2020-12-23 19:02:25

客户端连接

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

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

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

PHP MySQLi 连接

PHP 需要开启 MySQLi 扩展连接数据库，具体 demo 如下：

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

JDBC 连接

您也可以使用 JDBC 连接 TDSQL MySQL版，例如：

```
private final String USERNAME = "test";
private final String PASSWORD = "123456";
private final String DRIVER = "com.mysql.jdbc.Driver";
private final String URL = "jdbc:mysql://10.10.10.10:3306?userunicode=true&characterEncoding=utf8
mb4";
private Connection connection;
private PreparedStatement pstmt;
private ResultSet resultSet;
```

其他连接方式

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

JOIN 和子查询

最近更新时间：2020-10-26 14:47:10

对于分布式实例，数据水平拆分在各个节点，为提高性能，建议优先优化表结构和 SQL，尽量使用不跨节点的方式。

推荐方式

多个分表，带有分表键相等的条件

```
MySQL > select * from test1 join test2 where test1.a=test2.a;
+---+------+---------+---+------+---------------+
| a | b | c | a | d | e |
+---+------+---------+---+------+---------------+
| 1 | 2 | record1 | 1 | 3 | test2_record1 |
| 2 | 3 | record2 | 2 | 3 | test2_record2 |
+---+------+---------+---+------+---------------+
2 rows in set (0.00 sec)

MySQL > select * from test1 left join test2 on test1.a<test2.a where test1.a=1;
+---+------+---------+------+------+---------------+
| a | b | c | a | d | e |
+---+------+---------+------+------+---------------+
| 1 | 2 | record1 | 2 | 3 | test2_record2 |
+---+------+---------+------+------+---------------+
1 row in set (0.00 sec)

MySQL> select * from test1 where test1.a in (select a from test2);
+---+------+---------+
| a | b | c |
+---+------+---------+
| 1 | 2 | record1 |
| 2 | 3 | record2 |
+---+------+---------+
2 rows in set (0.00 sec)

MySQL> select a, count(1) from test1 where exists (select * from test2 where test2.a=test1.a) gro
up by a;
+---+----------+
| a | count(1) |
+---+----------+
| 1 | 1 |
| 2 | 1 |
+---+----------+
```

```
2 rows in set (0.00 sec)

MySQL> select distinct count(1) from test1 where exists (select * from test2 where test2.a=test1.
a) group by a;
+----------+
| count(1) |
+----------+
| 1 |
+----------+
1 row in set (0.00 sec)

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

均为单表

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

mysql> create table noshard_table_2 ( a int, b int key);
Query OK, 0 rows affected (0.00 sec)

mysql> select * from noshard_table,noshard_table_2;
Empty set (0.00 sec)

mysql> insert into noshard_table (a,b) values(1,2),(3,4);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> insert into noshard_table_2 (a,b) values(10,20),(30,40);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> select * from noshard_table,noshard_table_2;
+------+---+------+----+
| a | b | a | b |
+------+---+------+----+
```

```
| 1 | 2 | 10 | 20 |
| 3 | 4 | 10 | 20 |
| 1 | 2 | 30 | 40 |
| 3 | 4 | 30 | 40 |
+------+---+------+----+
4 rows in set (0.00 sec)
```

广播表

```
MySQL> create table global_test(a int key, b int)shardkey=noshardkey_allset;
Query OK, 0 rows affected (0.00 sec)

MySQL> insert into global_test(a, b) values(1,1),(2,2);
Query OK, 2 rows affected (0.00 sec)

MySQL> select * from test1, global_test;
+---+------+---------+---+------+
| a | b | c | a | b |
+---+------+---------+---+------+
| 1 | 2 | record1 | 1 | 1 |
| 2 | 3 | record2 | 1 | 1 |
| 1 | 2 | record1 | 2 | 2 |
| 2 | 3 | record2 | 2 | 2 |
+---+------+---------+---+------+
4 rows in set (0.00 sec)
```

子查询带有 shardkey 的 derived table

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

> **说明：**
> 子查询时不指定 shardkey，即可查询结果。

复杂 SQL

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

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

对于这类复杂查询，通过条件下推，将真正参与查询的数据从后端数据库中抽取出来，存放在本地的临时表中，然后对临时表中的数据进行计算。

因此用户需要明确指定参与查询的表的条件，避免因为抽取大量数据而性能受损。

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

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

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

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

mysql> select * from test1 join test2 on test1.b=test2.d;
+---+------+---------+---+------+--------------+
| a | b | c | a | d | e |
+---+------+---------+---+------+--------------+
| 2 | 3 | record2 | 1 | 3 | test2_record1 |
| 2 | 3 | record2 | 2 | 3 | test2_record2 |
+---+------+---------+---+------+--------------+
2 rows in set (0.00 sec)

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

分布式实例还支持丰富的复杂 update/delete/insert 操作。

需要注意的是，这类查询是在与之对应的 select 基础上实现的，因此也需要将数据加载至网关临时表，建议用户尽量在查询中指定明确的查询条件，避免大量数据的加载带来性能损耗。

另外，网关在加载数据时默认不会对加载的数据进行上锁，这与官方的 MySQL 行为存在略微的差异；如需加锁可以通过修改 proxy 配置来实现。

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

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

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

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

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

sequence

最近更新时间：2020-08-18 16:04:01

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

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

```
CREATE SEQUENCE 序列名
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE/ MINVALUE n| NOMAXVALUE}]
[{CYCLE|NOCYCLE}]
[{CACHE n| NOCACHE}];
```

> 说明：
> 目前 sequence 为保证分布式全局唯一，性能较差，适用于并发不高的场景。

创建

示例如下：

```
create sequence test.s1 start with 12 minvalue 10 maxvalue 50000 increment by 5 nocycle
create sequence test.s2 start with 12 minvalue 10 maxvalue 50000 increment by 1 cycle
```

参数有开始值，最小值，最大值，步长，是否回绕。

删除

示例如下：

```
drop sequence test.s1
```

当前限制条件：参数都为正整数。

查看

示例如下：

```
show create sequence test.s1
```

使用

操作表的序列

```
select nextval(test.s1)
select next value for test.s1
```

示例：

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

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

mysql> select nextval(test.s1);
+----+
| 17 |
+----+
| 17 |
+----+
1 row in set (0.01 sec)

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

mysql> select next value for test.s1;
+----+
| 22 |
```

```
+----+
| 22 |
+----+
1 row in set (0.01 sec)
```

nextval 可以用在 insert 等地方：

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

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

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

获取上次的值

如果之前没有用 nextval 获取过，则返回0：

```
select lastval(test.s1)
select previous value for test.s1;
```

示例：

```
mysql> select lastval(test.s1);
+----+
| 22 |
+----+
| 22 |
+----+
```

```
1 row in set (0.00 sec)

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

设置下一个值

只能变大，否则返回0：

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

变小没反应：

```
mysql> select nextval(test.s2);
+----+
| 15 |
+----+
| 15 |
+----+
1 row in set (0.01 sec)

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

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

变大，成功返回当前设置的值：

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

需要注意，sequence 的部分关键字以 TDSQL_ 前缀开始：

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

常用 DML

最近更新时间：2020-08-18 16:04:08

select

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

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

insert/replace

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

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

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

delete/update

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

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

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

读写分离

最近更新时间：2020-12-23 19:02:09

TDSQL MySQL版 实例支持下列几种模式的读写分离：

- 全局读写分离：proxy 开启语法解析的配置，通过语法解析过滤出用户的 select 读请求，默认把读请求直接发给备机。
- 通过增加 slave 注释标记，将指定的 SQL 发往备机，即在 SQL 中添加 `/*slave*/` 这样的标记，该 SQL 会发送给备机。

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

- 由只读帐号发送的请求会根据配置的属性发给备机。

分布式事务

最近更新时间：2020-12-23 19:01:50

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

普通分布式事务

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

XA 分布式事务

XA 分布式事务是指跨实例的事务：

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

新增事务接口

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

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

建表

最近更新时间：2020-12-16 16:11:14

建分表

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

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

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

- 场景1：存在多个唯一索引时报错。

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

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

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

  因为主键索引或者 unique key 索引意味着需要全局唯一，而要实现全局唯一索引，则必须包含 shardkey 字段。

- 场景2：存在选择自增列为主键时报错。
  建表时，当主键设置自增列且计划将主键设置为 shardkey 时，例如：

  ```
  create table test1 ( a int UNSIGNED AUTO_INCREMENT, b int, c char(20),primary key (a),unique k
  ey u_2(b,c) ) shardkey=a;
  ```

  此时无法基于主键建表成功，会报如下错误：

  ```
  1503, “A UNIQUE INDEX must include all colums in the table‘s patitioning function”
  ```

  因为，自增列会自行保证唯一性，无需主键，会与 innodb 冲突，此时将自增列去掉主键限制，选择唯一索引的一部分即可建表成功。

除上面的限制外，shardkey 字段还有如下要求：

- shardkey 字段的类型必须是 int、bigint、smallint、char、varchar。

- shardkey 字段的值不能有中文，proxy 不会转换字符集，因此不同字符集可能会路由到不同的分区。
- 不能 update shardkey 字段的值。
- shardkey=a 放在 SQL 的最后面。
- 访问数据尽量都带上 shardkey 字段，非强制要求，但是不带 shardkey 的 SQL 会路由到所有节点，消耗较多资源。

建广播表

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

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

建单表

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

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

连接保护

最近更新时间：2020-08-25 11:15:30

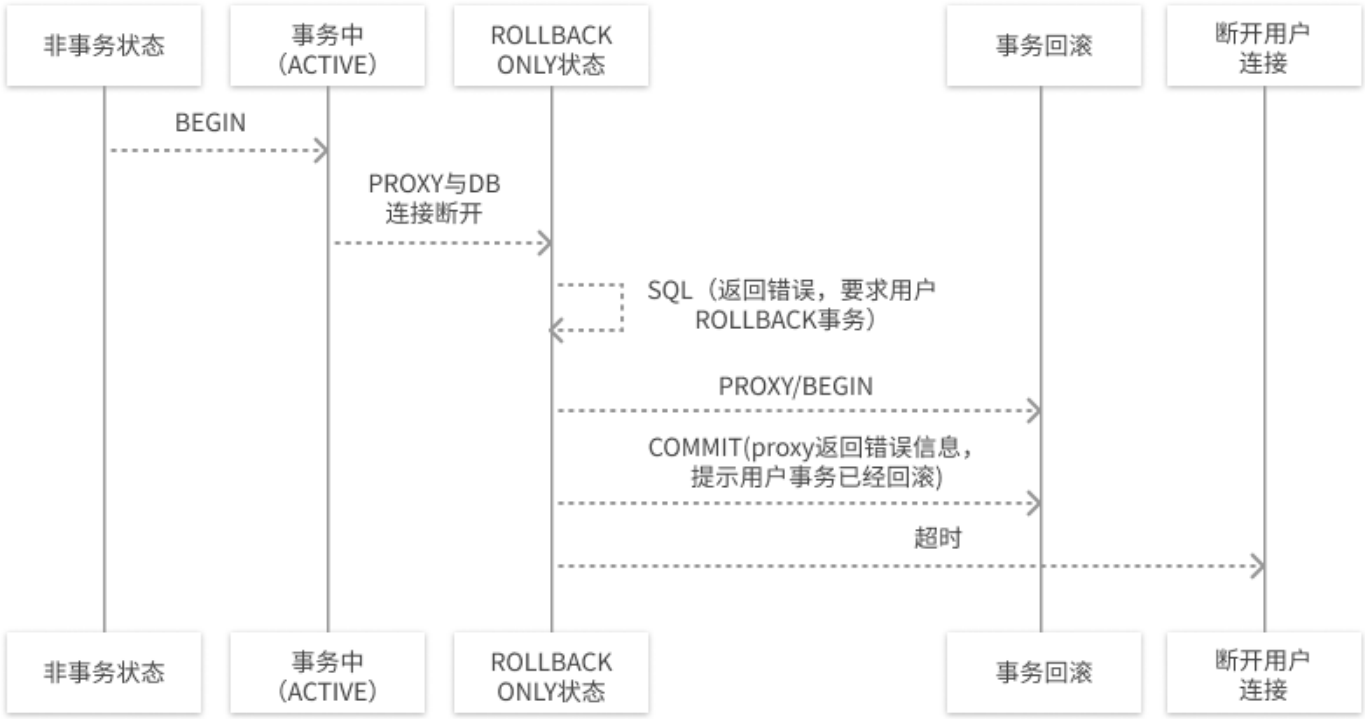
连接保护用于 proxy 与后端数据库连接断开时（如后端数据库发生了异常，但异常未影响 proxy），保持客户端和 proxy 之间的连接不断开。

proxy 正在执行 SQL 时，如果与数据库的连接断开，则 proxy 断开与该 SQL 相关的所有连接（除 proxy 与客户端之间的连接），并告知用户错误：

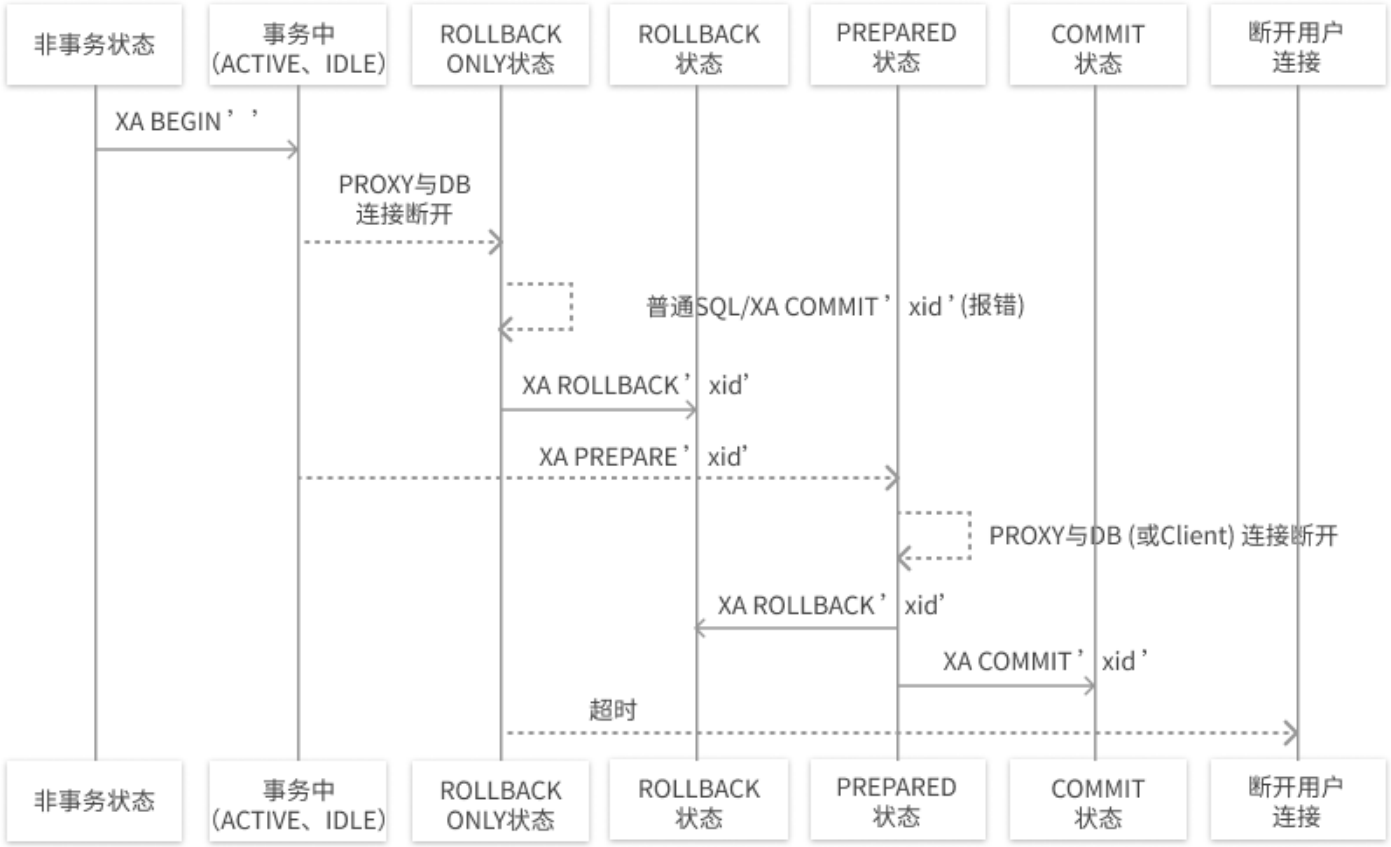
```
ER_PROXY_TRANSACTION_ERROR // 在事务中
ER_PROXY_CONN_BROKEN_ERROR // 非事务中
```

处理方式

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



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



超时配置

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

超时参数在 proxy 配置文件中为：

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

两级分区

最近更新时间：2020-12-23 19:01:41

TDSQL MySQL版 只用 HASH 方式进行数据拆分，不利于删除特定条件的数据，如流水类型，删除旧的数据，为解决这个问题，可以使用两级分区。

TDSQL MySQL版 支持 range 和 list 格式的两级分区，具体建表语法和 MySQL 分区语法类似。

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 (INTEGER)、BIGINT：支持 year、month、day 函数，此时传入的值转换为年月日，然后和分表信息对比，函数为空则直接使用该 int 值和分表信息对比。

示例：
如果 hired 是 date 类型，则查询插入对应的值格式为20160101 10:20:20、20160101。

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

如果 hired 是 int 类型，则查询插入对应的值格式为1474961034，proxy 首先会转换成对应的 date 格式20160927，然后和分表信息对比。

```
CREATE TABLE employees_int (
id INT key NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired int,
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
```

```
store_id INT
)
shardkey=id
PARTITION BY RANGE ( month(hired) ) (
PARTITION p0 VALUES LESS THAN (199102),
PARTITION p1 VALUES LESS THAN (199603),
PARTITION p2 VALUES LESS THAN (200101)
);
```

list 支持类型

- DATE、DATETIME、TIMESTAMP：支持年月日函数。
- TINYINT、SMALLINT、MEDIUMINT、INT (INTEGER)、BIGINT、CHAR、VARCHAR、BINARY、VARBINARY。

```
CREATE TABLE customers_1 (
first_name VARCHAR(25),
last_name VARCHAR(25),
street_1 VARCHAR(30),
street_2 VARCHAR(30),
city VARCHAR(15),
renewal DATE
) shardkey=first_name
PARTITION BY LIST (city) (
PARTITION pRegion_1 VALUES IN('1', '2', '3'),
PARTITION pRegion_2 VALUES IN('4', '5', '6'),
PARTITION pRegion_3 VALUES IN('7', '8', '9'),
PARTITION pRegion_4 VALUES IN('10', '11', '12')
);
```

> **注意：**
> 分区使用的是小于 < 符号，因此如果要存储当年的数据的话（2017），需要创建 <2018 的分区，用户只需创建到当前的时间分区，TDSQL 会自动增加（立即生效）后续分区，默认往后创建3个分区，二级分区只支持年/月/日的分片自动创建，以 YEAR 为例，TDSQL 会自动创建2018，2019，2020的分区，后续会自动增加。

全局唯一字段

最近更新时间：2020-08-18 16:04:49

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

创建

```
mysql> create 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

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

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

如果发生切换、重启等过程，自增长字段中间会有空洞，例如：

```
mysql> insert into shard.auto_inc ( a,b,d,c) values(11,12,13,0),(21,22,23,0);
Query OK, 2 rows affected (0.03 sec)
mysql> select * from shard.auto_inc;
+------+------+------+------+
| a | b | c | d |
+------+------+------+------+
| 21 | 22 | 2002 | 23 |
| 1 | 2 | 2 | 4 |
| 1 | 2 | 1 | 3 |
| 11 | 12 | 2001 | 13 |
```

```
+------+------+------+------+
4 rows in set (0.01 sec)
```

更改当前值：

```
alter table auto_inc auto_increment=100
```

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

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

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

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

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

数据导出导入

最近更新时间：2020-12-23 19:02:15

导出数据

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

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

> 说明：
> db 和 table名参数根据实际情况选择，如果导出的数据要导入到另外一套 TDSQL MySQL版 环境的话，必须加上-c选项。

导入数据

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

下载工具

```
[tdengine@TENCENT64 ~/]$./load_data
format:./load_data mode0/mode1 proxy_host proxy_port user password db_table shardkey_index file field_terminate filed_enclosed
example:./load_data mode1 10.xx.xx.10 3336 test test123 shard.table 1 '/tmp/datafile' ' ' ''
```

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

数据库管理语句

最近更新时间：2020-08-18 16:05:00

状态查询

通过 SQL 可以查看 proxy 的配置以及状态信息，目前支持如下命令：

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

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

示例如下：

```
mysql> /*proxy*/help;
+----------------------+-------------------------------------------------------------+
| command | description |
+----------------------+-------------------------------------------------------------+
| show config | show config from conf |
| show status | show proxy status,like route,shardkey and so on |
| set sys_log_level=N | change the sys debug level N should be 0,1,2,3 |
| set inter_log_level=N | change the interface debug level N should be 0,1 |
| set inter_time_open=N | change the interface time debug level N should be 0,1 |
| set sql_log_level=N | change the sql debug level N should be 0,1 |
| set slow_log_level=N | change the slow debug level N should be 0,1 |
| set slow_log_ms=N | change the slow ms |
| set log_clean_time=N | change the log clean days |
| set log_clean_size=N | change the log clean size in GB |
+----------------------+-------------------------------------------------------------+
10 rows in set (0.00 sec)
```

```
mysql> /*proxy*/show config;
+----------------+-------------------+
| config_name | value |
+----------------+-------------------+
| version | V2R120D001 |
| mode | group shard |
| rootdir | /shard_922 |
```

```
| sys_log_level | 0 |
| inter_log_level | 0 |
| inter_time_open | 0 |
| sql_log_level | 0 |
| slow_log_level | 0 |
| slow_log_ms | 1000 |
| log_clean_time | 1 |
| log_clean_size | 1 |
| rw_split | 1 |
| ip_pass_through | 0 |
+-----------------+-------------------+
14 rows in set (0.00 sec)
```

```
mysql> /*proxy*/show status;
+-----------------------------+-------------------------------------------------------------------------------
------------+
| status_name | value |
+-----------------------------+-------------------------------------------------------------------------------
------------+
| cluster | group_1499858910_79548 |
| set_1499859173_1:ip | 10.49.118.165:5025;10.175.98.109:5025@1@IDC_4@0,10.231.23.241:5025@1@IDC_
2@0 |
| set_1499859173_1:hash_range | 0---31 |
| set_1499911640_3:ip | 10.49.118.165:5026;10.175.98.109:5026@1@IDC_4@0,10.231.23.241:5026@1@IDC_
2@0 |
| set_1499911640_3:hash_range | 32---63 |
| set | set_1499859173_1,set_1499911640_3 |
```

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

```
mysql> explain select * from test1;
+------+-------------+-------+------+---------------+------+---------+------+------+-------+-----
-----------------------------------+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra | info |
+------+-------------+-------+------+---------------+------+---------+------+------+-------+-----
-----------------------------------+
| 1 | SIMPLE | test1 | ALL | NULL | NULL | NULL | NULL | 16 | | set_2,explain select * from shar
d.test1 |
| 1 | SIMPLE | test1 | ALL | NULL | NULL | NULL | NULL | 16 | | set_1,explain select * from shar
d.test1 |
+------+-------------+-------+------+---------------+------+---------+------+------+-------+-----
```

```
-----------------------------------+
2 rows in set (0.03 sec)
```

透传 SQL

最近更新时间：2020-12-23 19:02:35

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

> 说明：
> 透传 SQL 时，proxy 不会解析 SQL，如果是往两个 set 进行透传写操作，不会使用分布式事务，特殊情况下会发生不一致问题，因此对于写操作建议一次透传一个 set。

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

具体语法：

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

预处理

最近更新时间：2020-08-18 16:05:13

SQL 类型的支持：

- PREPARE Syntax
- EXECUTE Syntax

二进制协议的支持：

- COM_STMT_PREPARE
- COM_STMT_EXECUTE

示例：

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

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

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

mysql> execute ff using @aa;
+---+------+
| a | b |
+---+------+
| 3 | 4 |
+---+------+
1 row in set (0.06 sec)
```

错误码和错误信息

最近更新时间：2020-08-28 14:53:38

proxy 增加如下错误编码：

```
#define ER_PROXY_GRAM_ERROR_BEGIN 600

#define ER_PROXY_SANITY_ERROR 601 // "Sanity error: %s"
#define ER_PROXY_GS_NOT_SUPPORT 602 // sql 类型不支持
#define ER_PROXY_ORDERBY_INDEX_NEG 603 // order by index is negative
#define ER_PROXY_ORDERBY_INDEX_TOO_BIG 604 // order by index is too big
#define ER_PROXY_ORDERBY_TYPE_UNSUPPORT 605 // 不支持到 order by 用法
#define ER_PROXY_GROUPBY_INDEX_NEG 606 // group by index is negative
#define ER_PROXY_GROUPBY_INDEX_TOO_BIG 607 // group by index is too big
#define ER_PROXY_GROUPBY_TYPE_UNSUPPORT 608 // 不支持的 group by 用法
#define ER_PROXY_GET_AUTO_ID_FAILED 609 // 获取自增 id 失败
#define ER_PROXY_TEANS_ROLLED_BACK 610 // 事务已经被回滚
#define ER_PROXY_ONE_SET 611 // 当前 sql 应该被发往一个后端，但是不是
#define ER_PROXY_CLIENT_HS_ERROR 612 // 解析客户端握手包出错
#define ER_PROXY_ACCESS_DENIED_ERROR 613 // the length of readu_auth_switch_result is not 20, 不
应该出现
#define ER_PROXY_TRANS_NOT_ALLOWED 614 // 事务中不允许执行的命令
#define ER_PROXY_TRANS_READ_ONLY 615 // 只读事务中不允许执行的命令
#define ER_PROXY_TRANS_ERROR_DIFFENT_SET 616 // 非 xa 事务中，只读 sql 使用了多个后端
#define ER_PROXY_STRICT_ERROR 617 // strict 模式下，一次仅允许修改一个 set
#define ER_PROXY_SC_TOO_LONG 618 // 后端断开时间过长，链接断开
#define ER_PROXY_START_TRANS_FAILED 619 // 开启新的 xa 事务失败
#define ER_PROXY_SC_RETRY 620 // server 已经 close，请重试上一条 sql
#define ER_PROXY_SC_TRANS_IN_ROLLBACK_ONLY 621 // server 已经 close，当前事务处于 rollback
#define ER_PROXY_SC_COMMIT_LATER 622 // server 已经 close，事务会在稍后提交
#define ER_PROXY_SC_ROLLBACL_LATER 623 // server 已经 close，事务会在稍后回滚
#define ER_PROXY_SC_IN_COMMIT_OR_ROLLBACK 624 // server 在事务提交/回滚阶段 close
#define ER_PROXY_SC_NEED_ROLLBACK 625 // server 已经 close，需要回滚当前事务
#define ER_PROXY_SC_STATE_WILL_ROLLBACK 626 // server 已经 close，将会会滚
#define ER_PROXY_XA_UNSUPPORT 627 // xa 目前不支持的命令
#define ER_PROXY_XA_INVALID_COMMAND 628 // xa 命令不合法
#define ER_PROXY_XA_GTID_INIT_ERROR 629 // gtid log 初始化失败
#define ER_PROXY_XA_GET_SET_IP_PORT_FAILED 630 // 获取 set 地址失败
#define ER_PROXY_XA_UPDATE_GTID_LOG_FAILED 631 // 更新 gtid log 失败
#define ER_PROXY_MYSQL_PARSER_ERROR 632 // 嵌入式库 sql 解析失败
#define ER_PROXY_ILLEGAL_ID 633 // kill id 不合法
#define ER_PROXY_NOT_SUPPORT_CURSOR 634 // CURSOR_TYPE_READ_ONLY 暂不支持
#define ER_PROXY_UNKNOWN_PREPARE_HANDLER 635 // 执行的 prepare 不明确
```

```
#define ER_PROXY_SET_PARA_FAIL 636 // Set parameters failed
#define ER_PROXY_SUBPARTITION_DEAY 637 // 处理二级分区发生错误
#define ER_PROXY_NO_SUBPARTITION_ROUTE 638 // 没有获取到二级分区表的路由信息
#define ER_PROXY_LOCK_MORE_TABLE 639 // 一次只可以锁定一张二级分区表
#define ER_PROXY_GET_ROUTER_LOCK_FAIL 640 // 获取路由锁失败
#define ER_PROXY_PART_NAME_EMPTY 641 // 分区名称为空
#define ER_PROXY_SUB_PART_TABLE_IS_NONE 642 // 没有二级分区
#define ER_PROXY_PART_TYPE_DENY 643 // 二级分区类型不支持
#define ER_PROXY_PART_NAME_ILLEGAL 644 // 分区名不合法
#define ER_PROXY_DROP_ALL_PARTITION_FAIL 645 // 删除所有分区失败，尝试直接删除表
#define ER_PROXY_GET_OLD_PART_NUM_FAIL 646 // 获取表的分片数失败
#define ER_PROXY_EMPTY_SQL 647 // empty sql，不会返回给客户端
#define ER_PROXY_ERROR_SHARDKEY 648 // sk 必须为某一列
#define ER_PROXY_ERROR_SUB_SHARDKEY 649 // 二级分区键失败
#define ER_PROXY_SQLUSE_NOT_SUPPORT 650 // proxy 不支持这种用法
#define ER_PROXY_DBFW_WHITE_LIST_DENY 651 // 不在白名单，被防火墙拒绝
#define ER_PROXY_DBFW_DENY 652 // 防火墙拒绝
#define ER_PROXY_INCORRECT_ARGS 653 // stmt 参数不正确
#define ER_PROXY_SYSTABLE_UNSUPPORT_NON_READ_SQL 654 // 不支持非只读 sql 访问系统表
#define ER_PROXY_TABLE_NOT_EXIST 655 // 表不存在
#define ER_PROXY_SHARD_JOIN_UNSUPPORT_TYPE 656 // shard join 暂不支持的用法
#define ER_PROXY_RECURSIVE_JOIN_DENY 657 // 递归 join 不支持
#define ER_PROXY_JOIN_INTERNAL_ERROR 658 // join 异常
#define ER_PROXY_SQL_TOO_COMPLEX 659 // sql 太复杂，groupshard 暂不支持
#define ER_PROXY_INVALID_ARG_FOR_GTID_STATE 660 // gtid_state() 参数不合法
#define ER_PROXY_CANT_SET_GLOBAL_AUTOCOMMIT_GS 661 // Global autocommit cannot be set in groupsha
rd
#define ER_PROXY_INVALID_VALUE_FOR_AUTOCOMMIT 662 // autocommit 值设置不合法
#define ER_PROXY_XID_ERROR 663 // xid 不合法
#define ER_PROXY_XID_GENERAT_FAILED 664 // xid 不能由用户指定
#define ER_PROXY_CANT_EXEC_IN_INTER_TRANS 665 // "The command cannot be executed in internal tran
sction"
#define ER_PROXY_XID_TIME_ERROR 666 // "Unexpected time part of xid"
#define ER_PROXY_XID_TIMEDIFF_TOO_LONG 667 // "timediff > 1800s, it's not safe to execute boost"
#define ER_PROXY_SAVEPOINT_NOT_EXIST 668 // SAVEPOINT 不存在
#define ER_PROXY_SC_TRANS_IN_ROLLED 669 // 事务已经会滚，由于 serevr 已经 close
#define ER_PROXY_CANT_BOOST_IN_TRANS 670 // 事务中不允许执行 SQLCOM_BOOST
#define ER_PROXY_TRANS_EXPECTED 671 // "A transaction is expected, this maybe a bug"
#define ER_PROXY_EXTERNAL_TRANS 672 // 外部 xa 中不允许执行
#define ER_PROXY_AUTO_INC_FAIL 673 // "Deal auto inc failed"
#define ER_PROXY_CHECK_JOIN_FAIL 674 // "Check join failed"
#define ER_PROXY_TABLE_TYPE_NOT_MATCH 675 // "Do not support shard-table operations in noshard in
stance"
#define ER_PROXY_UNSUPPORT_NS_IN_INSERT 676 // "Do not support noshard and noshard_allset in inse
```

```
rt sql"
#define ER_PROXY_ALTER_SEQ_ID_FAIL 677 // Alter seq id failed
#define ER_PROXY_ALTER_ID_ILLEGAL 678 // Alter seq id is illegal
#define ER_PROXY_CANT_CHANGE_STEP 679 // "Current table use zk to get auto inc, do not support to
change step: \'%s\'"
#define ER_PROXY_ALTER_STEP_FAIL 680 // Alter step failed
#define ER_PROXY_TOO_MUCH_TABLES 681 // 表的数量超过限制
#define ER_PROXY_TABLE_EXISTED 682 // 表已经存在
#define ER_PROXY_CREATE_STABLE_FAILED 683 // 创建shard表失败，复杂 sql 不能用来创建 shard 表
#define ER_PROXY_DDL_DENY 684 // ddl 不允许的操作
#define ER_PROXY_SHADKEY_ERROR 685 // SQL should not relate to subpartition tables
#define ER_PROXY_NO_SK 686 // reject nosk
#define ER_PROXY_COMBINE_SQL_KEY 687 // Something went wrong
#define ER_PROXY_GET_SK_ERROR 688 // sk 获取失败
#define ER_PROXY_SHOW_FAILED 689 // proxy show 命令错误
#define ER_PROXY_SET_FAILED 690 // proxy set 命令错误
#define ER_PROXY_UNLOCK_FORMAT_ERROR 691 // sql 格式不正确
#define ER_PROXY_UNLOCK_ROUTER_FAIL 692 // 释放路由锁失败
#define ER_PROXY_LOCK_ROUTER_FAIL 693 // 加路由锁失败
#define ER_PROXY_PROXY_CMD_FAIL 694 // 不支持的/*proxy*/ 命令
#define ER_PROXY_PROCESS_RULE_FILE_FAILED 695 // dump_error
#define ER_PROXY_GET_AUTO_NUM_ERROR 696 // 获取自增值失败
#define ER_PROXY_SEQUENCE_NOT_EXIST 697 // sequence 不存在
#define ER_PROXY_SEQUENCE_ERROR 698 // sequence 不合法
#define ER_PROXY_SEQUENCE_ALREADY_EXIST 699 // sequence 已经存在
#define ER_PROXY_SQL_RETRY 700 // sql 还未提交或回滚
#define ER_PROXY_XA2PC_ABORT 701 // 2pc 失败，事务将会会滚
#define ER_PROXY_XA2PC_COMMIT 702 // 2pc 失败，后续提交
#define ER_PROXY_XA2PC_UNCERTAIN 703 // 2pc 失败，结果未知
#define ER_PROXY_KILL_ERROR 704 // kill 失败
#define ER_PROXY_TRACE_DENY 705 // trace 模式下不允许执行的sql
#define ER_PROXY_SQL_IMCOMPLETE 706 // 事务状态不完整
#define ER_PROXY_SHARDKEY_HASH_ERROR 709 // sk hash错误

#define ER_PROXY_GRAM_ERROR_END 799

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

#define ER_PROXY_SLICING 901 // slice 被修改，可能在扩容阶段，拒掉当前 sql
#define ER_PROXY_NO_DEFAULT_SET 902 // set 为空
#define ER_PROXY_GET_ADDRESS_FAILED 903 // 还未初始化完成，获取后端地址失败，稍后重试
#define ER_PROXY_SQL_SIZE_ERROR_IN_GET_CANDIDATE_ADDRESS 904 // 获取后端地址出错（发往后端个数不正
确）
```

```
#define ER_PROXY_GET_ADDRESS_ERROR 905 // 获取后端地址出错
#define ER_PROXY_CANDIDATE_ADDRESS_EMPTY 906 // 未获取到后端地址
#define ER_PROXY_CANT_GET_SOCK 907 // socket 获取失败
#define ER_PROXY_GET_SET_SOCK_FAIL 908 // socket 获取失败
#define ER_PROXY_CONNECT_ERROR 909 // 后端连接失败
#define ER_PROXY_NO_SQL_ASSIGN_TO_SET 910 // 分发 sql 异常
#define ER_PROXY_STATUS_ERROR 911 // group 状态异常，断开链接
#define ER_PROXY_CONN_BROKEN_ERROR 912 // server close，sql 状态不正常
#define ER_PROXY_UNKNOWN_ERROR 913 // proxy 未知错误
#define ER_PROXY_ALL_SLAVES_UNAVAILABLE 914 // 所有备机不可用
#define ER_PROXY_ALL_SLAVES_CHANGE 915 // 备机异常

#define ER_PROXY_ERROR_END 916
```

说明：

其中900以上为系统错误，会通过 云监控平台 进行告警。