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
使用限制

最近更新时间：2020-04-03 16:07:15

TDSQL 分布式实例高度兼容 MySQL 协议和语法，由于分布式数据库和单机数据库存在较大的架构差异，对 MySQL 的部分功能和小语法有使用限制。

TDSQL 大类限制

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

TDSQL 小语法限制

TDSQL 实例暂不支持 DDL、DML、管理 SQL 语言的部分语法，具体限制如下：

DDL

- 暂不支持 CREATE TABLE ... SELECT
- 暂不支持 CREATE TEMPORARY TABLE
- 暂不支持 CREATE/DROP/ALTER SERVER/LOGFILE GROUP/
- 暂不支持 ALTER 对分表键（shardkey）进行改名，但可以修改类型
- 暂不支持 RENAME

DML

- 暂不支持 SELECT INTO OUTFILE/INTO DUMPFILE/INTO var_name
- 暂不支持 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
- 暂不支持非 SELECT 的子查询
- 暂不支持不带列名的 INSERT/REPLACE
- 暂不支持全局的 DELETE/UPDATE 使用 ORDER BY/LIMIT
- 暂不支持不带 WHERE 条件的 UPDATE/DELETE
- 暂不支持 LOAD DATA/XML

- 暂不支持 SQL 中使用 DELAYED 和 LOW_PRIORITY
- 暂不支持 SQL 中对于变量的引用和操作，例如 SET @c=1, @d=@c+1; SELECT @c, @d
- 暂不支持 index_hint
- 暂不支持 HANDLER/DO

管理 SQL 语句

- 暂不支持 ANALYZE/CHECK/CHECKSUM/OPTIMIZE/REPAIR TABLE，需要用透传语法
- 暂不支持 CACHE INDEX
- 暂不支持 FLUSH
- 暂不支持 KILL（非跨城版本数据库支持）
- 暂不支持 LOAD INDEX INTO CACHE
- 暂不支持 RESET
- 暂不支持 SHUTDOWN
- 暂不支持 SHOW BINARY LOGS/BINLOG EVENTS
- 暂不支持 SHOW WARNINGS/ERRORS和LIMIT/COUNT 的组合

TDSQL 与 MongoDB 的 JSON 能力对比

最近更新时间：2019-12-16 17:29:48

TDSQL（MySQL 5.7内核）已支持 json 能力，更多细节请参见 官方的 json 文档。

注意事项

- json 字段不可以作为 shardkey（分表键）。
- json 类型的聚合操作（如 orderby，groupby）不支持混合类型排序，例如，不能将 string 类型和 int 类型做比较或排序，且排序只支持数值类型，string 等类型排序不支持。

TDSQL 与 MongoDB 的 JSON 能力对比

建表语法

TDSQL

```
Create table inventory(id int primary key auto_increment, value json) shardkey=id;
```

MongoDB

```
sh.shardCollection("test.inventory", {"_id":"hashed"})
```

INSERT/UPDATE/DELETE Document

-	MongoDB	TDSQL
插入单个文件	db.inventory.insertOne({ item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } })	Insert into inventory(value) values('{ "item": "canvas", "qty": 100, "tags": ["cotton"], "size": { "h": 28, "w": 35.5, "uom": "cm" } }');
插入多个文件	db.inventory.insertMany([{ item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" }, { item: "mat", qty: 35, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },	insert into inventory(value) values ('{ "item": "journal", "qty": 25, "size": { "h": 14, "w": 21, "uom": "cm" }, "status": "A" }'), ('{ "item": "mat", "qty": 35, "size": { "h": 27.9, "w": 35.5, "uom": "cm" }, "status": "A" }'), ('{ "item": "paper", "qty": 100, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "D" }')

	{ item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" }]);	
更新单个文件	db.inventory.updateOne({ item: "paper" }, { $set: { "size.uom": "cm", status: "P" }, }) 不携带 shardkey 报错，携带 shardkey 可以正确执行	update inventory set value=json_set(value, "$.size.uom", "cm", "$.status", "P") where value->"$.item"="paper" limit 1; 不携带 shardkey 的语句会在多个节点上执行，语法结构可能会修改多条数据，而携带 shardkey 可以确保正确只修改1条数据执行
更新多个文件	db.inventory.updateMany({ "qty": { $lt: 50 } }, { $set: { "size.uom": "in", status: "P" }, })	update inventory set value=json_set(value, "$.size.uom", "in", "$.status", "P") where value->"$.qty"<50 ;
替换文件	db.inventory.replaceOne({ item: "paper" }, { item: "paper", instock: [{ warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 }] }) 不携带 shardkey 报错，携带 shardkey 可以正确执行	update inventory set value= '{ "item": "paper", "instock": [{ "warehouse": "A", "qty": 60 }, { "warehouse": "B", "qty": 40 }]}' where value->"$.item"="paper" limit 1) 不携带 shardkey 的语句会在多个节点上执行，语法结构可能会修改多条数据，而携带 shardkey 可以确保正确只修改1条数据执行
只删除一个符合条件的文件	db.inventory.deleteOne({ status: "A" }) 不携带 shardkey 报错，携带 shardkey 可以正确执行	delete from inventory where value->"$.status"="A" limit 1; 不携带 shardkey 的语句会在多个节点上执行，语法结构可能会修改多条数据，而携带 shardkey 可以确保正确只修改1条数据执行
删除所有符合条件的文档	db.inventory.deleteMany({ status : "A" })	delete from inventory where value->"$.status"="A";

QUERY Document

-	MongoDB	TDSQL
预先插入数据	db.inventory.insertMany([{ item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" , tags: ["blank", "red"], dim_cm: [14, 21] , instock: [{ warehouse: "A", qty:, 5 }, { warehouse: "C", qty: 15 }] } , { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" , tags: ["red", "blank"], dim_cm: [14, 21] , instock: [{ warehouse: "C", qty: 5 }] }, { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "D" , tags: ["red", "blank", "plain"], dim_cm: [14, 21] , instock: [{ warehouse: "A", qty: 60 }, { warehouse: "B", qty: 15 }] }, { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" , tags: ["blank", "red"], dim_cm: [22.85, 30] , instock: [{ warehouse: "A", qty: 40 }, { warehouse: "B", qty: 5 }] }, { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" , tags: ["blue"], dim_cm: [10, 15.25] , instock: [{ warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 }] }]);	insert into inventory(value) values ('{ "item": "canvas", "qty": 100, "size": { "h": 28, "w": 35.5, "uom": "cm" }, "status": "A" , "tags": ["blank", "red"], "dim_cm": [14, 21] , "instock": [{ "warehouse": "A", "qty": 5 }, { "warehouse": "C", "qty": 15 }] }'), ('{ "item": "journal", "qty": 25, "size": { "h": 14, "w": 21, "uom": "cm" }, "status": "A" , "tags": ["red", "blank"], "dim_cm": [14, 21] , "instock": [{ "warehouse": "C", "qty": 5 }] }'), ('{ "item": "mat", "qty": 85, "size": { "h": 27.9, "w": 35.5, "uom": "cm" }, "status": "D" , "tags": ["red", "blank", "plain"], "dim_cm": [14, 21] , "instock": [{ "warehouse": "A", "qty": 60 }, { "warehouse": "B", "qty": 15 }] }'), ('{ "item": "mousepad", "qty": 25, "size": { "h": 19, "w": 22.85, "uom": "cm" }, "status": "P" , "tags": ["blank", "red"], "dim_cm": [22.85, 30] , "instock": [{ "warehouse": "A", "qty": 40 }, { "warehouse": "B", "qty": 5 }] }'), ('{ "item": "notebook", "qty": 50, "size": { "h": 8.5, "w": 11, "uom": "in" }, "status": "P" , "tags": ["blue"], "dim_cm": [10, 15.25] , "instock": [{ "warehouse": "B", "qty": 15 }, { "warehouse": "C", "qty": 35 }] }')
通过路径语法实现对 json 内任意成员的访问	支持	支持
查询文件	db.inventory.find({ status: "D" })	SELECT * FROM inventory WHERE value->"$.status" = "D";
-	db.inventory.find({ status: { $in: ["A", "D"] } })	SELECT * FROM inventory WHERE cast(value->"$.status" as char(4)) in ('"A"', '"D"'); value->"$.status" 是 json 类型，MySQL 对于 json 类型目前不支持 in 比较操作，需要注意类型转换，并且"A"一定要用单引号引起来

-	MongoDB	TDSQL
查询嵌入式/嵌套文档	db.inventory.find({ size: { h: 14, w: 21, uom: "cm" } }) MongoDB 在筛选匹配条件的时候也会考虑字段的顺序，例如 db.inventory.find({ size: { w: 21, h: 14, uom: "cm" } }) 将不会查询到任何结果	SELECT * FROM inventory WHERE value->"$.size" = cast('{"h": 14, "w": 21, "uom": "cm"}' as json) MySQL这种查找方式不会考虑字段顺序 SELECT * FROM inventory WHERE value->"$.size" = cast('{"w": 21, "h": 14, "uom": "in"}' as json) 将会筛选出同样的结果
查询数组	db.inventory.find({ tags: ["red", "blank"] }) 需要考虑数组中的顺序	select * from inventory where value->"$.tags"=cast('["red", "blank"]' as json); 需要考虑数组中的顺序
查找包含“红色”和“空白”元素的数组，而不考虑数组中的顺序或其他元素	db.inventory.find({ tags: { $all: ["red", "blank"] } })	select * from inventory where json_contains(value->"$.tags",cast('["red", "blank"]' as json))=1;
为数组元素指定多个条件	db.inventory.find({ dim_cm: { $gt: 15, $lt: 20 } }) 选出数组中至少有一个元素满足大于15或者小于20或同时满足	不支持
-	db.inventory.find({ dim_cm: { $elemMatch: { $gt: 22, $lt: 30 } } })选出数组中至少有一个元素同时满足大于22和小于30	不支持
按数组索引位置查询元素	db.inventory.find({ "dim_cm.1": { $gt: 25 } })	select * from inventory where value->"$.dim_cm[1]" < 25
按数组长度查询数组	db.inventory.find({ "tags": { $size: 3 } })	select * from inventory where json_length(value->"$.tags") = 3;
查询元素的数组	db.inventory.find({ tags: "red" })	select * from inventory where json_contains(value->"$.tags",cast('"red"' as json))=1;

-	MongoDB	TDSQL
查询嵌入式文档数组	db.inventory.find({ "instock": { warehouse: "A", qty: 5 } }) 需要考虑字段 (warehouse, qty) 的顺序 db.inventory.find({ "instock": { $elemMatch: { qty: 5, warehouse: "A" } } }) 不需要考虑字段 (warehouse, qty) 的顺序	select * from inventory where json_contains(value->"$.instock", cast('{ "warehouse": "A", "qty": 5 }' as json))=1; 不需要考虑字段(warehouse, qty)的顺序
在嵌入文档数组的字段中指定查询条件	db.inventory.find({ 'instock.qty': { $lte: 20 } })	//不支持（qty 是数组 instock内的 field， 只能通过 instock[index].qty 访问，instock.qty 这种访问方式 MySQL 均不支持）

INDEXES

-	MongoDB	TDSQL
单场索引	对 qty 建立索引 db.inventory.createIndex({ qty: 1 })	MySQL 不支持直接针对 json 字段创建 index，需要先创建虚拟列并进行类型转换，如将 value->"$.qty" 作为索引 alter table inventory add value_qty int generated always as (value->"$.qty") virtual; create index idx on inventory(value_qty);
复合索引	db.inventory.createIndex({ "item": 1, "qty": 1 })	alter table inventory add value_item varchar(50) generated always as (value->"$.item") virtual; alter table inventory add value_qty int generated always as (value->"$.qty") virtual; create index idx_1 on inventory(value_item, value_qty);
哈希索引	db.inventory.createIndex({ qty: "hashed" })	Innodb 不支持
多键索引	多键索引要索引包含数组值的字段，MongoDB 会为数组中的每个元素创建一个索引键	不支持

唯一索引	db.inventory.createIndex({ "_id":1, "qty": 1 }, {unique:true}) 需要 shardkey 做前缀	alter table inventory add value_qty int generated always as (value->"$.qty") virtual; create unique index idx on inventory(id, value_qty); 索引中需要包含 shardkey
文本索引	插入数据 sh.shardCollection("test.stores", {"_id":"hashed"}) db.stores.insertMany([{ _id: 1, name: "Java Hut", description: "Coffee and cakes" }, { _id: 2, name: "Burger Buns", description: "Gourmet hamburgers" }, { _id: 3, name: "Coffee Shop", description: "Just coffee" }, { _id: 4, name: "Clothes Clothes Clothes", description: "Discount clothing" }, { _id: 5, name: "Java Shopping", description: "Indonesian goods" }]) 创建索引 db.stores.createIndex({ name: "text", description: "text" }) 根据索引查找 db.stores.find({ $text: { $search: "java coffee shop" } })	TDSQL 暂时不支持，MySQL 5.7 可以按照如下方法进行 插入数据 create table stores(id int primary key auto_increment, value json); insert into stores(value) values('{ "name": "Java Hut", "description": "Coffee and cakes" }'), ('{ "name": "Burger Buns", "description": "Gourmet hamburgers" }'), ('{ "name": "Coffee Shop", "description": "Just coffee" }'), ('{ "name": "Clothes Clothes Clothes", "description": "Discount clothing" }'), ('{ "name": "Java Shopping", "description": "Indonesian goods" }'); 创建 generated column alter table stores add value_name varchar(50) generated always as (value->"$.name") stored; alter table stores add value_description varchar(50) generated always as (value->"$.description") stored; create FULLTEXT index full_idx on stores(value_name, value_description); (generated column 为 stored 会影响 insert，update 性能，详见 此文档)

SHARDING

-	MongoDB	TDSQL
Ranged sharding	支持	不支持

-	MongoDB	TDSQL

Hashed sharding	db.t1.createIndex({"key1":"hashed"})
	sh.shardCollection("test.t1", {"key1":"hashed"})
	db.t1.insertOne({"key1":"value1","key2":"value2"})
		TDSQL 不需要事先创建 hashed index
		
		create table t1(key1 varchar(20), value json) shardkey=key1;
		insert into t1(key1, value) values("value1", '{"key2":"value2"}');
		
		TDSQL 目前不支持按照 json 内的任意字段进行 hashed sharding，如有需要，需要将作为 shardkey 的字段单独提出作为一列。

将含有数据的非 shard 表修改为 shard 表	支持	不支持

MongoDB 和 TDSQL 的shard（分布式）架构相似，因此在水平扩容、容灾等方面各有千秋，此处不做展开。

SHARD INDEX

MongoDB 和 TDSQL 的 index 都是建立在各个 shard 上的，并且只有包含 shardkey 的 index 才可以有全局 unique 的约束条件。无论是包含 shardkey 的 Compound Indexes 还是将对 shardkey 本身建立 index，两者都是先通过 shardkey 确定相关 shard，再在相关的各个 shard 上利用该索引，在没有 shardkey 的情况下，查询会发送到所有的 shard。

JOIN

MongoDB 在非 shard 表下只能支持多表 left join，而在 shard 表下不支持 join，具体实现方法如下所示：

```
插入数据：
db.users.insertMany([{
"email" : "admin@gmail.com",
"userId" : "AD",
"userName" : "admin"
},

{
"email" : "admin1@gmail.com",
"userId" : "AD",
```

```
"userName" : "admin1"
}
]);

db.userinfo.insertMany([{
"userId" : "AD",
"phone" : "0000000000"
},
{
"userId" : "AD",
"phone" : "0000000000"
}
]);

db.userrole.insertMany([{
"userId" : "AD",
"role": "admin"
},
{
"userId" : "AC",
"role" : "admin"
}
]);

左连接操作：
db.users.aggregate([{
// Join with user_info table
lookup:{
from: "userinfo", // other tablename
localField: "userId", // name of users table field
foreignField: "userId", // name of userinfo table field
as: "user_info" // alias for userinfo table
}
},
{ $unwind:"$user_info" },
// $unwind used for getting data in object or for one record only

// Join with user_role table
{
$lookup:{
from: "userrole",
localField: "userId",
foreignField: "userId",
as: "user_role"
```

```
{ $unwind:"$user_role" },

// define some conditions here
{
$match:{$and:[{"userName" : "admin"}]}
},

// define which fields are you want to fetch
{
$project:
_id : 1,
email : 1,
userName : 1,
userPhone : "$user_info.phone",
role : "$user_role.role",
}
}
]);
```

相对 MongoDB，TDSQL 在非 shard 表下可以运用 json 的字段做各种条件 join，当在单个 shard 表下 TDSQL 允许 Join 操作，但是不支持在多个 shard 表。详细操作见下面代码：

```
插入数据
create table users(id int primary key auto_increment, value json);
create table userinfo(id int primary key auto_increment, value json);
create table userrole(id int primary key auto_increment, value json);

insert into users(value) values('{
"email" : "admin@gmail.com",
"userId" : "AD",
"userName" : "admin"}'),
('{
"email" : "admin1@gmail.com",
"userId" : "AD",
"userName" : "admin1"}');

insert into userinfo(value) values('{
"userId" : "AD",
"phone" : "0000000000"}'),
('{
"userId" : "AD",
"phone" : "0000000000"}');

insert into userrole(value) values('{
"userId" : "AD",
```

```
"role" : "admin"}'),
('{
"userId" : "AC",
"role" : "admin"}');

可以按照mysql的join语法根据json字段进行多种join操作
select * from users left join userinfo on users.value
->"$.userId" = userinfo.value
->"$.userId" left join userrole on users.value
->"$.userId" = userrole.value
->"$.userId" where users.value
->"$.userName"="admin";

select * from users left join userinfo on users.value
->"$.userId" = userinfo.value
->"$.userId" right join userrole on users.value
->"$.userId" = userrole.value
->"$.userId" where userrole.value
->"$.role"="admin";
```

对比总结

写入数据

两者都可以以方便的写入 json 串和更新 json 内部的某些字段，但 MongoDB 不支持事务，只有单行操作可保证原子性，多行操作如果需要原子性需要应用层实现两阶段提交。而 TDSQL 的 json 操作可以完整的支持事务特性，sharding 模式下也支持分布式事务。

查询数据

1. Join：TDSQL 支持多表根据 json 字段进行 join 操作，MongoDB 只支持多个 unsharded 表 left join。
2. Index：两者都支持根据 json 的某些（int,string）字段建立索引，MongoDB 还额外支持 multikey index 等索引。
3. 访问 json 内部元素：两者都有各自完善的语法可以访问到 json 内部的各个字段，无需应用层进行 json 解析。
4. 搜索条件：MongoDB 提供的搜索和匹配方面的功能更完善，相比之下，TDSQL 需要时刻注意对选择条件进行类型转换后再进行判断，对开发人员来说不是很友好，并且筛选的功能方面也较 MongoDB 稍弱，适用于对 json 操作相对简单的应用。

综合对比

综合来看，对比于 MongoDB 目前的三大核心功能：json 的灵活性，复制集保证高可用，sharding 保证可扩展，TDSQL 均可以支持。而在 json 细节的支持，MongoDB 提供的功能更加丰富一些。但是，TDSQL 是基于腾讯

TDSQL 金融级分布式架构的，其自身数据强一致、高可用和可扩展也有着完善的解决方案，且能够关系型数据库的事务、join 等功能。

如果您既希望使用 json 类型，又对数据一致性、事务、join 等传统数据库具备的能力也有一定要求的话，TDSQL 将是一个很好的选择。

DDL 语句

最近更新时间：2019-12-16 10:41:13

如您需要阅读或下载全量开发文档，请参见 TDSQL 开发指南。

创建分表

分表：即自动水平拆分的表，水平拆分是基于分表键（shardkey）采用类似于一致性 hash 方式，根据 hash 后的值分配到不同的节点组中的一种技术方案，该能力几乎是所有分布式数据库的核心特性。TDSQL 的设计目标是期望开发者完全无需关注后端分表策略，像使用普通数据库一样使用 TDSQL，因此我们在设计上隐藏了分表的细节方案。

普通的分表创建时必须在最后面指定 shardkey 的值，该值为表中的一个字段名字，会用于后续 sql 的路由选择：

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

由于在 TDSQL 下，shardkey 对应后端数据库的分区字段，因此必须是主键以及所有唯一索引的一部分，否则无法建表；

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

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

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

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

综述，shardkey 的要求如下：

1. shardkey 必须是主键以及所有唯一索引的一部分。
2. shardkey 字段的类型必须是 int,bigint,smallint/char/varchar。
3. shardkey 字段的值不能有中文。
4. 不要 update shardkey 字段的值（如必须，请先 delete 该行，再 insert 新值）。
5. shardkey=a 放在 create 语句的最后。
6. 访问数据尽量都能带上 shardkey 字段（非强制要求，但如果不带 shardkey 将会导致该条 SQL 发送到所有节点，影响效率）。

说明：
某些分表方案可以支持“非主键或唯一索引”成为 shardkey，但此类方案会导致数据不一致，因此 TDSQL 默认禁止“非主键或唯一索引”成为 shardkey。

创建广播表

广播表：又名小表广播功能，创建了广播表后，每个节点都有该表的全量数据，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 set 的 join 操作的性能，常用于配置表等。

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

说明：
广播表采用分布式事务机制，确保数据写入操作的原子性，因此广播表的数据天然情况下是完全一致的。

创建普通表（单表）

普通表（又名单表）：语法和 MySQL 完全一样，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)
```

其他 DDL 操作

alter，drop 等其他 DDL 操作，与 MySQL 语法完全一致。

DML 语句

最近更新时间：2019-12-16 17:32:26

如您需要阅读或下载全量开发文档，请参见 TDSQL 开发指南。

DML 语句语法（部分）

SELECT：建议在条件中带上 shardkey 字段，否则 TDSQL 无法判断 SQL 应该路由至哪些节点，需要进行全表扫描，然后在网关聚合，容易影响执行效率：

```
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，因为 TDSQL 无法判断该条 SQL 应该发往哪个节点，乱发会导致数据异常，因此 TDSQL 将会直接拒绝执行无 shardkey 的 INSERT/REPLACE 语句：

```
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：同上，为安全考虑，在分表和广播表执行该类 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)
```

事务（分布式事务）

最近更新时间：2019-12-16 17:36:13

如您需要阅读或下载全量开发文档，请参见 TDSQL开发指南。

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（5.7或以上版本）默认支持分布式事务，并且对客户端透明，业务像使用单机事务一样使用。例如：

```
begin; //开启事务
delete
update //操作数据，可以跨set
select
insert
commit; //提交事务
```

为便于事务新增 sql 用于查询特定事务的信息：

- select gtid()，获取当前分布式事务的 gtid (事务的全局唯一性标识)，如果该事务不是分布式事务则返回空。
  gtid 的格式：
  '网关id'-'网关随机值'-'序列号'-'时间戳'-'分区号'，例如 c46535fe-b6-dd-595db6b8-25

- select gtid_state("gtid")，获取"gtid"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：
    a)一个小时之后查询，标识事务状态已经清除。
    b)一个小时以内查询，标识事务最终会被回滚。
    当 commit 执行超时或者失败的时候，应该等待几秒之后再调用该接口来查询事务的状态。

- 运维命令：
  xa recover：向后端 set 发送 xa recover 命令，并进行汇总。
  xa lockwait：显示当前分布式事务的等待关系（可以使用 dot 命令将输出转化为等待关系图）。
  xa show：当前网关上正在运行的分布式事务。

补充

TDSQL 分布式事务采用两阶段提交算法（2PC），隔离级别配置为 read committed, repeatable read，或者 serializable。

两级分区

最近更新时间：2019-12-16 17:39:22

如您需要阅读或下载全量开发文档，请参见 TDSQL 开发指南。

TDSQL 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 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。

> 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 会自动增加后续分区，默认往后创建3个分区，以 YEAR 为例，TDSQL 会自动创建2018，2019，2020的分区，后续也会自动增减。

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), and 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), and BIGINT;

CHAR, VARCHAR, BINARY, and 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')
);
```

错误码和错误信息

最近更新时间：2020-02-21 15:26:16

如您需要阅读或下载全量开发文档，请参见 TDSQL 开发指南。

proxy 新增如下错误编码：

```
ER_PROXY_GRAM_ERROR_BEGIN=800,

ER_PROXY_SANITY_ERROR, /*sql类型错误*/
ER_PROXY_BAD_COMMAND, /*sql命令不支持*/
ER_PROXY_SQL_NOT_SUPPORT, /*TDSQL不支持类型*/
ER_PROXY_SQLUSE_NOT_SUPPORT, /*TDSQL不支持该用法*/
ER_PROXY_ERROR_SHARDKEY, /*建表时一级二级分区键选择有问题，如两者要不一样，是表中某一列*/

ER_PROXY_ERROR_SUB_SHARDKEY, /*暂时不用*/
ER_PROXY_PRE_DEAL_ERROR, /*join不符合规则*/
ER_PROXY_SHADKEY_ERROR, /*复杂sql，如derived table需要包含shardkey*/
ER_PROXY_COMBINE_SQL_ERROR, /*重组sql有问题，如没有对应的二级分区表*/
ER_PROXY_SQL_ERROR, /*一般的sql错误*/
ER_PROXY_ONE_SET, /*count（distinct）必须发到一个set上*/
ER_PROXY_STRICT_ERROR, /*暂时不用*/
ER_PROXY_TRANSACTION_ERROR, /*事务相关的错误*/


ER_PROXY_GRAM_ERROR_END, /*语法错误*/

ER_PROXY_SYSTEM_ERROR_BEGIN=900,/*系统错误，出现此类错误时，TDSQL团队会收到告警*/

ER_PROXY_SLICING,
ER_PROXY_NO_DEFAULT_SET, /*set为空*/
ER_PROXY_SOCK_ADDRESS_ERROR, /*set地址为空*/
ER_PROXY_SOCK_ERROR, /*连接后端数据库失败*/
ER_PROXY_STATUS_ERROR, /*group状态出错*/
ER_PROXY_UNKNOWN_ERROR, /*unknown错误*/

ER_PROXY_LOG_GTID_TIMEOUT, /*两阶段提交时，写日志超时*/
ER_PROXY_LOG_GTID_ERROR, /*两阶段提交时，写日志超时*/
ER_PROXY_COMMIT_TEMP_ERROR, /*两阶段提交时，出现临时性错误，agent会最终自动提交事务*/
ER_PROXY_ABORT_TEMP_ERROR, /*事务回滚时，出现临时性错误，agent会最终自动回滚事务*/
ER_PROXY_SQL_RETRY,
ER_PROXY_CONN_BROKEN_ERROR, /*网关与后端DB的连接断开*/
```

JSON 类型

最近更新时间：2019-12-16 17:42:18

如您需要阅读或下载全量开发文档，请参见 TDSQL开发指南。

TDSQL 的 Percona 5.7 内核支持存储 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。如果您既希望使用 JSON 类型，又对数据一致性、事务、join 等传统数据库具备的能力也有一定要求的话，TDSQL 将是一个很好的选择。

TDSQL 的 JSON 是基于 MySQL，与 MongoDB 的使用仍有一些差异，更多对比详情请参见 TDSQL 与 MongoDB 的 JSON 能力对比。

```
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 (0.07 sec)

mysql> INSERT INTO t1 (jdoc,a)VALUES('[1, 2,',5);
ERROR 3140 (22032): Invalid JSON text: "Invalid value." at position 6 in value for column 't1.jdoc'.
mysql> select * from t1;
+--------------------------------------+------+
| jdoc | a |
+--------------------------------------+------+
| {"key1": "value1", "key2": "value2"} | 1 |
+--------------------------------------+------+
1 row in set (0.03 sec)
```

针对 JSON 类型的 orderby 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。

JOIN（分布式 JOIN）

最近更新时间：2019-12-16 17:48:49

如您需要阅读或下载全量开发文档，请参见 TDSQL开发指南。

由于 TDSQL 存在多个物理节点，部分 join 操作可能涉及到多个物理节点的数据，这种跨物理节点数据的 JOIN，一般叫做分布式 JOIN。

- 如果 join 相关的表有 shardkey 相等条件（如下示例），由于分表的一致性原则，会让这部分数据自动存储到同一物理节点，此时相当于单机 JOIN，性能最好。此处涉及到分表 shardkey 的选择，可以参考 常见问题，帮助您更好的判断。
- 如果涉及到跨物理节点数据，此时 proxy 会先从其他节点拉取数据并缓存，由于涉及到网络数据传输，性能会损失。

shardkey 相等条件（性能无损失）

```
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 left join test2 on test1.a=test2.a where test1.a=1;
+---+------+---------+---+------+---------------+
| a | b | c | a | d | e |
+---+------+---------+---+------+---------------+
| 1 | 2 | record1 | 1 | 3 | test2_record1 |
+---+------+---------+---+------+---------------+
1 row in set (0.00 sec)

mysql> select * from test1 join test2 on 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 (0.03 sec)
```

shardkey 不等条件（性能有损失）

```
mysql> select * from test1 join test2;
+---+------+---------+---+------+---------------+
| a | b | c | a | d | e |
+---+------+---------+---+------+---------------+
| 1 | 2 | record1 | 1 | 3 | test2_record1 |
| 2 | 3 | record2 | 1 | 3 | test2_record1 |
| 1 | 2 | record1 | 2 | 3 | test2_record2 |
| 2 | 3 | record2 | 2 | 3 | test2_record2 |
+---+------+---------+---+------+---------------+
4 rows in set (0.06 sec)
```

对于广播表和单表（普通表）相关的 join

如果是单表（普通表）与单表（普通表）JOIN，相当于单机 JOIN，性能无损失。
如果是广播表与分表 JOIN，相当于单机 JOIN，性能无损失。
广播表与广播表 JOIN，相当于单机 JOIN，性能无损失。

> 说明：
> 目前暂不支持"单表（普通表）"和"分表"进行 join 操作。

```
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)
```

如何 KILL 线程

最近更新时间：2019-12-16 17:52:27

使用数据库运行某些语句时，会因数据量太大而导致死锁，没有反应。这个时候，就需要 KILL 掉某个正在消耗资源的 query 语句即可，KILL 命令的语法格式如下：

```
KILL [CONNECTION | QUERY] thread_id
```

请注意，由于 TDSQL 是由多个数据库节点组（SET）组成，您必须使用 /* 透传 自定义注释功能 才能成功杀掉。

```
/*sets:set_1*/kill 890346;
```

由于 TDSQL 是由多个数据库节点组（SET）组成，您可以使用 /*sets:set_1*/SHOW PROCESSLIST 语句查看哪些线程正在运行，并使用 KILL thread_id 语句终止一个线程。

查询 set_ID，可至 控制台，或者使用 /*proxy*/show status; 命令。

```
mysql> /*sets:set_524110864_1*/show processlist;
+---------+--------+---------------------+------+---------+------+-------+-----------------------------
-----------+----------+-----------------+
| Id | User | Host | db | Command | Time | State | Info | Progress | info |
+---------+--------+---------------------+------+---------+------+-------+-----------------------------
-----------+----------+-----------------+
| 1072003 | vitosu | 13.26.12.18:4859 | NULL | Sleep | 189 | | NULL | 0.000 | set_1524110864_1 |
| 1072132 | vitosu | 13.26.12.18:4823 | NULL | Query | 0 | init | /*sets:set_524110864_1*/ show processlist
| 0.000 | set_1524110864_1 |
+---------+--------+---------------------+------+---------+------+-------+-----------------------------
-----------+----------+-----------------+

mysql> /*sets:set_524110864_1*/kill 1072003;
Query OK, 0 rows affected (0.03 sec)
```

如果您的业务有较多线程，无法准确判断哪些事务未提交，可以采用类似 SQL 进行查询线程 ID（举例）：

```
SELECT
it.trx_id AS trx_id,
it.trx_state AS trx_state,
it.trx_started AS trx_started,
it.trx_mysql_thread_id AS trx_mysql_thread_id,
CURRENT_TIMESTAMP - it.trx_started AS RUN_TIME,
pl.user AS USER,
pl.host AS HOST,
pl.db AS db,
```

```
pl.time AS trx_run_time,
pl.INFO as INFO
FROM
information_schema.INNODB_TRX it,
information_schema.processlist pl
WHERE
pl.id=it.trx_mysql_thread_id
ORDER BY RUN_TIME DESC LIMIT 10;
```

如果您的业务有较多线程，无法准确判断哪些事务处于锁等待，可以采用类似 SQL 进行查询线程 ID（举例）：

```
SELECT
r.trx_id waiting_trx_id,
r.trx_mysql_thread_id waiting_thread,
TIMESTAMPDIFF( SECOND, r.trx_wait_started, CURRENT_TIMESTAMP ) wait_time,
r.trx_query waiting_query,
l.lock_table waiting_table_lock,
b.trx_id blocking_trx_id,
b.trx_mysql_thread_id blocking_thread,
SUBSTRING( p. HOST, 1, INSTR(p. HOST, ':') - 1 ) blocking_host,
SUBSTRING(p. HOST, INSTR(p. HOST, ':') + 1) blocking_port,
IF (p.COMMAND = 'Sleep', p.TIME, 0) idel_in_trx,
b.trx_query blocking_query FROM information_schema.INNODB_LOCK_WAITS w INNER JOIN inform
ation_schema.INNODB_TRX b ON b.trx_id = w.blocking_trx_id INNER JOIN information_schema.INNO
DB_TRX r ON r.trx_id = w.requesting_trx_id INNER JOIN information_schema.INNODB_LOCKS l ON w.r
equested_lock_id = l.lock_id LEFT JOIN information_schema. PROCESSLIST p ON p.ID = b.trx_mysql_t
hread_id ORDER BY wait_time DESC;
```

风险提示：大事务 kill 之后，事务需要回滚，数据量较大的情况下也需等待很久，此时可以到控制台单击主从切换，将从机切换为主，以快速恢复业务。**但请务必知悉：使用异步同步、强同步（可退化）复制方案时，由于主从数据同步有延迟，可能丢失/错乱部分数据，请谨慎操作主从切换。**

语言结构

最近更新时间：2019-12-16 17:54:29

如您需要阅读或下载全量开发文档，请参见 TDSQL开发指南。

TDSQL 支持所有 MySQL 使用的文字格式，此处只是列举，并不代表与 MySQL 有差异

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

Date and Time Literals

DATE 支持如下格式：

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

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

DATETIME，TIMESTAMP 支持如下格式：

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

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

Hexadecimal Literals

支持格式如下：

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

Bit-Value Literals

支持格式如下：

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

Boolean Literals

常量 TRUE 和 FALSE 等于 1 和 0，它是大小写不敏感的。

```
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，它是大小写不敏感的，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。

预处理

最近更新时间：2019-12-16 18:46:33

如您需要阅读或下载全量开发文档，请参见 TDSQL开发指南。

TDSQL 支持预处理，使用方式与单机 MySQL 相同，此处只是作为列举，例如：

- 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)
```

子查询

最近更新时间：2019-12-16 18:47:11

如您需要阅读或下载全量开发文档，请参见 TDSQL开发指南。

TDSQL 目前只支持带 shardkey 的 derived table。

```
mysql> select a from (select * from test1) as t;
ERROR 7012 (HY000): Proxy ERROR:sql should has one shardkey
mysql> select a from (select * from test1 where a=1) as t;
+---+
| a |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

字符集与时区

最近更新时间：2019-12-16 18:50:37

如您需要阅读或下载全量开发文档，请参见 TDSQL开发指南。

TDSQL 在后端存储支持 MySQL 的所有字符集和字符序，此处只是列举，并不代表此处有差异

```
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/ |
+--------------------------+----------------------------------------------------+
```

注意：
TDSQL 不支持通过命令行设置参数，请到云控制台进行设置。

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)
```

全局唯一数字序列使用

最近更新时间：2019-12-16 10:52:10

如您需要阅读或下载全量开发文档，请参见 TDSQL 开发指南。

支持全局唯一数字序列（auto_increment）；目前仅保证自增字段全局唯一和递增性，不保证单调递增（即按时间顺序的绝对递增性），这里的 auto_increment 长8字节，最大为18446744073709551616，因此，您无需担心该值溢出。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)) sha
rdkey=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)
```

值得说明的是，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
```

如果用户不指定自增值，可以通过 select last_insert_id() 获取：

```
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)
```

注意：
目前 select last_insert_id() 只能跟 shard 表的自增字段一起使用，不支持单表（普通表）和广播表。

数据库管理语句

最近更新时间：2019-12-16 14:34:45

如您需要阅读或下载全量开发文档，请参见 TDSQL 开发指南。

数据库管理语句

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：

```
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 | xxx.xxx.xxx.xxx:xxxx;xxx.xxx.xxx.xxx:xxxx@1@IDC_4@0,xxx.xxx.xxx.xxx:xxxx@1@IDC_2@0 |
| set_1499859173_1:hash_range | 0---31 |
| set_1499911640_3:ip | xxx.xxx.xxx.xxx:xxxx;xxx.xxx.xxx.xxx:xxxx@1@IDC_4@0,xxx.xxx.xxx.xxx:xxxx@1@IDC_2@0 |
| set_1499911640_3:hash_range | 32---63 |
| set | set_1499859173_1,set_1499911640_3 |
```

同时 proxy 增强了 explain 的返回结果，显示 proxy 修改后的 sql：

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

支持的函数

最近更新时间：2019-12-16 14:46:17

如您需要阅读或下载全量开发文档，请参见 TDSQL 开发指南。

支持的函数

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

Name	Description
FIND_IN_SET()	Return the index position of the first argument within the second argument
FORMAT()	Return a number formatted to specified number of decimal places
FROM_BASE64()	Decode to a base-64 string and return result
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
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()

Name	Description
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
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

Name	Description
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
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

Name	Description
-	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

Name	Description
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()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()

Name	Description
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

Name	Description
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Aggregate (GROUP BY) Functions

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

Name	Description
~	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

自定义注释

最近更新时间：2019-12-16 15:39:00

通过在 sql 语句前增加注释（hint）可以实现特定的功能，TDSQL 提供如下的 hint。

透传到指定物理分片

在分布式中，proxy 会对 sql 进行语法解析，会有比较严格的限制，如果用户想在某个物理分片（set）中执行 sql，可以使用透传 sql 的功能。
透传 sql 的功能：支持透传 sql 到对应的一个或者多个物理分片（set），透传到分表键（shardkey）对应的 set，示例如下：

```
mysql> select * from test1 where a in (select a from test1);
ERROR 7009 (HY000): Proxy ERROR:proxy do not support such use yet
mysql> /*set_1*/select * from test1 where a in (select a from test1);
Empty set (0.00 sec)
```

具体语法：

```
/*sets:set_1*/
/*sets:set_1,set_2*/
/*sets:allsets*/
/*shardkey:10*/
```

强制 sql 走从机实现读写分离

TDSQL 支持多种读写分离方案，通过 /*slave*/ 的方案常用于代码实现时，固化到业务逻辑中，方便开发中使用，示例如下：

```
mysql> /*slave*/select * from test.ff limit 0;
```

> 说明：
> 执行 TDSQL 特定的 sql 请参见 控制指令。