阿里云
HybridDB for MySQL

事务引擎手册

文档版本：20181213

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 禁止： 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 警告： 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 说明： 您也可以通过按**Ctrl** + **A**选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 DDL语法说明

1.1 创建表

标准语法

```
CREATE TABLE [ IF NOT EXISTS ] table_name
( { column_name column_definition | table_constraints } [, ... ]  )
DISTRIBUTE_KEY (column_name)
[table_attribute]
```

```
DROP TABLE table_name
```

说明：

在HybridDB for MySQL中创建表，必须指定分区键。

表定义示例：

```
CREATE TABLE mytable (
    id bigint(20) NOT NULL AUTO_INCREMENT,
    name varchar(128) NOT NULL,
    ts timestamp DEFAULT CURRENT_TIMESTAMP,
    title varchar(256) DEFAULT NULL,
    content text DEFAULT NULL,
    view_count int DEFAULT 0,
    PRIMARY KEY (id),
    KEY idx_name(name)
) DEFAULT CHARSET=utf8
  DISTRIBUTE_KEY (name);
```

表定义说明：

- tablename表名不得超过32字节，只能使用英文字母、阿拉伯数字和下划线'_'；
- columnname列名不得超过32字节，只能使用英文字母、阿拉伯数字和下划线'_'；
- 暂不支持更新UPDATE主键（PRIMARY KEY）和分区键（DISTRIBUTE_KEY）的值；若需要变更主键和分区键的值，需要先 DELETE 后重新 INSERT；
- 最多支持512个列；
- 一行数据的大小不得超过16MB；
- 表名和列名不得选用关键字，如果必须使用，则在使用时必须为表名和列名加入反引号（`）（注意不是单引号'）。

支持的数据类型data_type:

```
TINYINT[(length)] [UNSIGNED] [ZEROFILL]
```

```
SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
INT[(length)] [UNSIGNED] [ZEROFILL]
INTEGER[(length)] [UNSIGNED] [ZEROFILL]
BIGINT[(length)] [UNSIGNED] [ZEROFILL]
DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
CHAR[(length)] [BINARY]
VARCHAR(length) [BINARY]
DATE
TIME
DATETIME
TIMESTAMP
TINYTEXT [BINARY]
TEXT [BINARY]
MEDIUMTEXT [BINARY]
LONGTEXT [BINARY]
BOOL
BOOLEAN
```

列定义**column_definition**

```
column_definition:
    data_type [ { NOT NULL | NULL } ][ DEFAULT default_expr ]
    [ AUTO_INCREMENT ][ { [ UNIQUE | PRIMARY ] KEY } ]
    [ COMMENT column_comment ]
```

说明：

- 列注释column_comment不得使用任何英文字符外的注释。
- AUTO_INCREMENT自增字段指明该列为唯一值，该列必须为BIGINT类型，分布式数据库会自动为该列生成一个64位的唯一值，且该字段将自动成为主键，对于表的字段，若用户插入该字段，那么数据库将存储用户提供的值，用户需要自行保证该值的唯一性，否则可能在多个分区上出现相同的值，引发冲突；若用户插入NULL，或者不插入该字段，那么数据库将为用户生成该字段的值，数据库将自行保证该值的唯一性。

表约束**table_constraints**：

```
[ KEY index_name( index_column_name [, ... ] ) ]
[ UNIQUE KEY index_name( index_column_name [, ... ] ) ]
[ PRIMARY KEY ( column_name [, ... ] )  ]
index_column_name:
    column_name [(length)] [ASC | DESC]
```

说明：

- 用户的表必须指定一个主键，否则在迁入数据时有可能出现重复数据；
- 创建的索引必须要有一个索引名字；

- 主键（PRIMARY KEY）暂不支持更新；若需要变更主键数据，需要先删除后，再重新插入；
- 唯一键（UNIQUE KEY）仅允许事先创建，通过UNIQUE INDEX index_name (col1, col2, col3, …)方式指定。主键和唯一键仅支持分区内的唯一性，不支持全局唯一性。若要保证全局唯一性，请将主键或唯一键的一个字段指定为分区键。

表分区

```
DISTRIBUTE_KEY (column_name)
```

 说明：

1. 当前必须在DDL内指定分区键，分区键仅支持指定一个列，该列的数据类型只能为整数（TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT）或字符型（CHAR, VARCHAR）中的一种，数据按照分区键维度分散到各个分区中；
2. 分区键（DISTRIBUTE_KEY）暂不支持更新UPDATE；若需要变更分区键数据，需要先DELETE 后重新 INSERT。

表属性**table_attribute**：

```
[ DEFAULT CHARSET = table_charset ]
[ COMMENT table_comment ]
```

 说明：

- 表注释table_comment不得使用任何英文字符外的注释；
- 字符集仅支持utf8。

不支持的约束检查：

- 唯一性约束：允许用户建立主键和唯一性索引，但是不保证它们的全局唯一性，仅保证自增主键的全局唯一性；
- 外键约束；
- CHECK约束。

系统库和系统表：

目前并未开放任何系统库和系统表，包括information_schema等。

查看表定义

标准语法：

```
SHOW CREATE TABLE table_name
DESC table_name
DESC table_name DISTRIBUTE INFO
```

参数说明：

- SHOW CREATE TABLE table_name 用于展示表的基础表定义，此表定义当前不含分区键定义；
- DESC table_name 用于展示表的所有列名，列定义中不含分区键定义；
- DESC table_name DISTRIBUTE INFO 用于展示表的分区键。

1.2 修改表定义

标准语法

```
ALTER TABLE table_name [alter_specification [, alter_specification
] ...]
alter_specification:
        | ADD [COLUMN] column_name column_definition
        | ADD [COLUMN] (column_name column_definition [, column_name
column_definition] ...)
        | CHANGE [COLUMN] old_column_name new_column_name column_def
inition
        | MODIFY [COLUMN] column_name column_definition
        | DROP [COLUMN] column_name
column_definition:
 data_type [ { NOT NULL | NULL } ][ DEFAULT default_expr ] [
AUTO_INCREMENT ]
```

参数说明

- ADD [COLUMN]用于为表增加新列；
- CHANGE [COLUMN]用于修改表的旧列，允许修改列名；
- MODIFY [COLUMN]用于修改表的旧列，不允许修改列名；
- DROP [COLUMN]用于删除表的列；
- 支持增加和修改列定义，与CREATE TABLE语法相同；
- 支持在一个表定义变更语句中，指明对多个列的变更。

1.3 重命名表

标准语法

```
RENAME TABLE table_name TO new_table_name
```

注意事项

- 目前不支持通过 `ALTER TABLE table_name RENAME [TO|AS] new_tbl_name` 语法变更表名；
- 表名变更过程中不得访问该表。

1.4 创建索引

标准语法

```
CREATE INDEX index_name ON table_name (index_col_name,...)
index_column_name:
column_name [(length)] [ASC | DESC]
DROP INDEX index_name ON table_name
```

注意事项

- CREATE INDEX 语法与 ALTER TABLE table_name ADD INDEX 语法都能为表新增索引；
- DROP INDEX 语法与 ALTER TABLE DROP INDEX 语法都能为表删除索引；
- 索引可以对其引用的列指明按照升序或降序排序；
- 不支持动态添加UNIQUE INDEX，因为此操作会造成副本间数据不一致。

2 DML语法说明

2.1 SELECT语句

标准语法：

SELECT语法的总体结构：

```
[ WITH with_subquery_table_name AS ( query ) ]
SELECT
[DISTINCT] select_expr [, select_expr ...]
[FROM  table_reference [, ...] ]
[WHERE filter_condition]
[GROUP BY { expr | ROLLUP ( expr_list ) | CUBE ( expr_list ) |
GROUPING SETS ( expr_list )} , ...]
[HAVING having_condition]
[ORDER BY {col_name | expr }
[ASC | DESC], ...]
[{ UNION [ ALL ] | INTERSECT | EXCEPT } (SELECT select_expr..)]
[LIMIT {row_count}]
```

WITH子句

WITH语句用于定义一个或者多个子查询，每个子查询定义一个临时表，类似于视图的定义； 在WITH中定义的临时表可以在当前查询的其他子句中引用；所有的WITH语句定义的临时表，都可以通过SELECT子句中的子查询定义来完成类似的效果，但是当这些子查询或者临时表被后面的字句多次引用时，WITH语句只需要计算一次临时表结果，然后多次复用，从而达到减少公共表达式计算的次数。

语法：

```
[ WITH with_subquery [, ...] ]
```

而 with_subquery的语法为：

```
with_subquery_table_name AS ( query )
```

参数：

- with_subquery_table_name：当前查询中一个唯一的临时表名称
- query：所有可以支持的SELECT查询

例如：

```
with t as (select x,y from A) select t.y from t order by t.x limit 10
```

SELECT列表

SELECT语句中的列投影的基本结构为：

```
SELECT [ ALL | DISTINCT ] * | expression [ AS column_alias ] [, ...]
```

参数：

- `ALL`：当不需要定义`DISTINCT`时的一个可选冗余字段。
- `DISTINCT`：用于消除重复的行。
- `*`：返回所有的列。
- `expression`：一个或者多个列引用，也可以是带函数的列表达式。
- `AS column_alias`：用于定义select列的别名，AS关键字可选。AS后面接的**alias**如果是一个带空格的字符串，可以使用 ` 符号括起来。

FROM子句

FROM子句的语法为：

```
FROM table_reference [, ...]
```

其中 table_reference支持的格式如下：

```
with_subquery_table_name [ [ AS ] alias ]
table_name [ * ] [ [ AS ] alias ]
( subquery ) [ AS ] alias
table_reference join_type table_reference
[ ON join_condition ]
```

参数：

- `with_subquery_table_name`：WITH语句定义的子查询名字。
- `table_name`：表名或者视图名称。
- `alias`：表或者视图的别名。

- join_type：连接类型，包括[INNER] JOIN、 LEFT [OUTER] JOIN、RIGHT [OUTER] JOIN和CROSS JOIN。
- join_condition：用于join的on条件，on后面的条件只能是等值关系，非等值关系需在where子句中定义。

WHERE子句

WHERE子句语法为：

```
[WHERE filter_condition]
```

其中filter_condition可以是用AND，OR等二元逻辑操作符连接的多个表达式，也可以是IN、 NOT IN、EXIST、NOT EXIST构成的相关或非相关子查询（或Semi Join）。

GROUP BY子句

GROUP BY 用于做分组操作，语法为：

```
GROUP BY [ROLLUP | CUBE] expression [, ...]
```

GROUP BY 后的表达式必须是select list中的非聚合表达式。

 说明：

group by不支持别名，例如：`select x+1 as t from table group by t`。

HAVING子句

Having子句用于做分组后面的过滤，语法为：

```
[ HAVING condition ]
```

备注：

- HAVING 条件引用的表达式必须出现在group by的列中，或者引用聚合列表达式。
- HAVING 条件不支持select list中列的别名，必须要重写列表达式。
- HAVING 条件不支持select list中列的下标引用。

ORDER BY子句

ORDER BY子句用于做排序，语法为：

```
[ ORDER BY expression
[ ASC | DESC ]
```

```
[ LIMIT { count | ALL } ]
```

参数：

- `expression`：定义如何排序，可以是select list中的列或者别名，也可以是没有出现在select list中的列。
- `ASC | DESC`：定义排序的方式，升序（ASC）或者降序（DESC），NULL值默认排在前面。
- `LIMIT number | ALL`：控制返回结果集的行数，number指定行数，ALL等同于返回所有行。

备注：

- 不支持offset。

UNION / INTERSECT / EXCEPT / MINUS子句

UNION / INTERSECT / EXCEPT / MINUS用于进行集合求并、交、差操作，语法为：

```
query
{ UNION [ ALL ] | INTERSECT | EXCEPT | MINUS }
query
```

参数：

- `query`：操作符前后的query输出的列数目和类型都必须完全一致。
- `UNION [ALL]`：集合求并操作，输出合并后的结果，ALL表示无需去重。
- `INTERSECT`：集合求交操作，输出各个query的交集。
- `EXCEPT`：集合求差操作，返回query的差集结果。
- `MINUS`：集合求差操作，同`EXCEPT`。

集合操作的排序：

`UNION`和`EXCEPT` 操作符都是左结合（left-associative），从左至右进行运算，例如：

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

相当于先做完t1 union t2 except t3, 最后才是对前面的结果按照c1进行全局排序。

Intersect操作符的优先级是高于 UNION和EXCEPT的, 例如：

```
select * from t1
union
select * from t2
intersect
select * from t3
order by c1;
```

等同于：

```
select * from t1
union
(select * from t2
intersect
select * from t3)
order by c1;
```

备注：

- 集合操作符前的query是不可以带order by语句的，如果要带，需要用括号括起来。

层次查询 **Connect by** 子句

- 支持start with ... connect by prior的层次查询，涉及的字段需要出现在select投影中。
- start with允许比较条件、in表达式，不允许子查询、不允许与where连用。
- connect by prior允许一个等值join条件，不支持多个或者非等值条件，不允许子查询。
- 不支持prior之外的connect by。
- 不支持level、sys_connect_by_path等其他隐藏字段或层次函数。

```
-- 一般的使用方式
select id, long_test from test start with id < 100 connect by prior id =
 long_test

-- 整体做子查询
select * from (select id, long_test from test start with id in (1,2,3)
 connect by prior id = long_test) as hier order by 1,2
```

SELECT语句支持的MySQL函数

HybridDB for MySQL目前支持在SELECT查询语句中使用如下的SQL函数：

表 2-1: 操作符

名称	说明	别名	是否支持
AND, &&			Y
=			Y
BETWEEN AND			Y
COALESCE	return the first non-null arg		Y
&			Y
~			Y
^			Y
CASE			Y
DIV			Y
/			Y
=			Y
<=>	NULL-safe equal to		Y
>			Y
>=			Y
GREATEST			Y
LEAST			Y
IN			Y
NOT IN			Y
INVERVAL			Y
IS			Y
IS NOT			Y
IS NOT NULL			Y
IS NULL			Y
<<			Y

名称	说明	别名	是否支持
<			Y
<=			Y
LIKE			Y
-			Y
%, MOD			Y
NOT, !			Y
NOT BETWEEN AND			Y
!=, <>			Y
NOT LIKE			Y
NOT REGEXP		NOT SIMILAR TO	Y
OR			Y
+			Y
REGEXP		SIMILAR TO	Y
>>			Y
RLIKE		REGEXP	Y
NOT RLIKE		NOT REGEXP	Y
*			Y
-			Y
XOR			Y

表 2-2: 控制流函数 Control Flow Functions

函数名	说明	别名	是否支持
CASE WHEN[test1] THEN [result1]…ELSE [default] END	如果testN是真，则返回resultN，否则返回default		Y
CASE [test] WHEN[val1] THEN [result]…ELSE [default] END	如果test和valN相等，则返回resultN，否则返回default		Y
IF(test,t,f)	如果test是真，返回t；否则返回 f		Y

函数名	说明	别名	是否支持
IFNULL(arg1,arg2)	如果arg1不是空，返回arg1，否则返回arg2		Y
NULLIF(arg1,arg2)	如果arg1=arg2返回NULL；否则返回arg1		Y

表 **2-3:** 字符函数 **String Functions**

函数名	说明	别名	是否支持
ASCII(char)	返回字符的ASCII码值	ORD(char)	Y
BIN(n)	返回字符的二进制值	CONV(N, 10, 2)	Y
BIT_LENGTH(str)	返回字符串的比特长度		Y
CHAR(N, …)	将传入的整数转换成字符		Y
CHAR_LENGTH(str)	返回字符串的长度，按字符个数计	CHARACTER_LENGTH(str)	Y
CONCAT(s1,s2…,sn)	将s1,s2…,sn连接成字符串，任何sn为NULL则返回NULL		Y
CONCAT_WS(sep,s1,s2…,sn)	将s1,s2…,sn连接成字符串，并用sep字符间隔		Y
ELT(N, s1, s2, …)	返回第N个sn字符串		Y
EXPORT_SET(bits, on, off [,seperator [, number_of_bits]])			Y
FIELD(target, s1, s2 ,…)	返回target在s1,s2…里的位置		Y
FIND_IN_SET(str, strlist)	分析逗号分隔的list列表，如果发现str，返回str在strlist中的位置		Y
FORMAT(X, D)			Y
FROM_BASE64(str)			Y
HEX(str), HEX(N)			Y

函数名	说明	别名	是否支持
INSERT(str,x,y,instr)	将字符串str从第x位置开始，y个字符长的子串替换为字符串instr，返回结果		Y
INSTR(str, substr)			Y
LCASE(str)	将字符串str中所有字符变成小写，并返回结果	LOWER(str)	Y
LOWER(str)			Y
LEFT(str,x)	返回字符串str中最左边的x个字符		Y
LENGTH(s)	返回字符串str中的字符数，按字节计		Y
LOCATE(substr, str), LOCATE(substr, str, pos)			Y
LPAD(str, len, padstr)			Y
LTRIM(str)	去掉字符串str中开头的空格		Y
MAKE_SET(bits, str1, str2, …)			Y
MID(str, pos, len)		SUBSTRING(str, pos, len)	Y
OCT(N)		CONV(N, 10, 8)	Y
OCTET_LENGTH(str)		LENGTH(str)	Y
ORD(str)			Y
POSITION(substr IN str)	返回子串substr在字符串str中第一次出现的位置	LOCATE(substre, str)	Y
QUOTE(str)	用反斜杠转义str中的单引号		Y
REPEAT(str, count)	返回字符串str重复count次的结果		Y

函数名	说明	别名	是否支持
REPLACE(str, from_str, tp_str)			Y
REVERSE(str)	颠倒字符串str，并返回结果		Y
RIGHT(str, len)	返回字符串str中最右边的len个字符		Y
RPAD(str, len, padstr)			Y
RTRIM(str)	删除字符串str尾部的空格，并返回结果		Y
SPACE(N)	返回重复N个空格的字符串		Y
SUBSTR(str, pos), SUBSTR(str FROM pos), SUBSTR(str, pos, len), SUBSTR(str FROM pos FOR len)		SUBST(str, pos[, length])	Y
SUBSTRING(str, pos), SUBSTRING (str FROM pos), SUBSTRING(str, pos, len), SUBSTRING(str FROM pos FOR len)		SUBSTRING(str, pos[, length])	Y
SUBSTRING_INDEX(str, delim, count)			Y
TO_BASE64(str)			Y
TRIM([{BOTH / LEADING / TRAILING } [remstr] FROM] str), TRIM([remstr FROM] str)	去除字符串首部和尾部的所有空格/声明char	TRIM_STR([remchar,] str)	Y
UCASE(str)	返回将字符串str中所有字符转变为大写后的结果	UPPER(str)	Y
UNHEX(str)			Y
UPPER(str)			Y

函数名	说明	别名	是否支持
expr LIKE pat [ESCAPE ‘escape_char’]			Y
expr NOT LIKE pat [ESCAPE ‘escape_char’]			Y
STRCMP(expr1, expr2)			Y
expr NOT REGEXP pat, expr NOT RLIKE pat		NOT SIMILAR TO	Y
expr REGEXP pat, expr RLIKE pat		SIMILAR TO	Y

表 **2-4:** 数字函数 **Numeric Functions**

函数名	说明	别名	是否支持
DIV, /, -, %, MOD, +, *, -	算术操作符		Y
ABS(x)	返回x的绝对值		Y
ACOS(x)			Y
ASIN(x)			Y
ATAN(x)			Y
ATAN(Y, X), ATAN2(Y, X)			Y
CEIL(X)		CEILING(x)	Y
CEILING(x)	返回大于或等于x的最小整数值	CEIL(x)	Y
CONV(N, from_base, to_base)			Y
COS(x)			Y
COT(x)			Y
CRC32(expr)			Y

函数名	说明	别名	是否支持
DEGREES(x)			Y
EXP(x)	返回值 e（自然对数的底）的x次方		Y
FLOOR(x)	返回小于或等于x的最大整数值		Y
FORMAT(X, D)			Y
HEX(N_or_S)			Y
LN(x)	返回x的自然对数		Y
LOG(b,x), LOG(x)	返回x的以b为底的对数，b默认约是2.718		Y
LOG2(x)	返回x的以2为底的对数		Y
LOG10(x)	返回x的以10为底的对数		Y
MOD(N, M), N % M, N MOD M	返回N/M的模（余数）		Y
PI()	返回pi的值（圆周率）		Y
POW(x, y)		POWER(x, y)	Y
POWER(x, y)		POW	Y
RADIANS(x)			Y
RAND([N])	返回 0 到 1 内的随机值,可以通过提供一个参数（种子）使RAND()随机数生成器生成一个指定的值		Y
ROUND(x,[y])	返回参数x的四舍五入的带y位小数的值, y默认0，即x取整		Y
SIGN(x)	返回代表数字x的符号的值，返回-1, 0, 1		Y
SIN(x)			Y
SQRT(x)	返回一个数的平方根		Y

函数名	说明	别名	是否支持
TAN(x)			Y
TRUNCATE(x,y)	返回数字x截短为y位小数的结果		Y

表 **2-5:** 日期及时间函数 **Date and Time Functions**

函数名	说明	别名	是否支持
ADDDATE(date, INTERVAL expr unit), ADDDATE(expr, days)	只支持ADDDATE(expr, days)这种格式		Y
ADDTIME(expr1, expr2)	目前’01:00:00.999999’和’25:00:01’ 这两种时间格式在精度上是有区别的		Y
CONVERT_TZ(dt, from_tz, to_tz)			Y
CURDATE()	返回当前的日期	CURRENT_DATE()	Y
CURRENT_DATE, CURRENT_DATE()		CURDATE()	Y
CURRENT_TIME([fsp])	返回当前的时间	CURTIME()	Y
CURRENT_TIMESTAMP([fsp])			Y
CURTIME([fsp])			Y
DATE(expr)			Y
DATEDIFF(expr1, expr2)			Y
DATE_ADD(date, INTERVAL int unit)		ADDDATE	Y
DATE_FORMAT(date, format)			Y
DATE_SUB(date, INTERVAL expr, unit)		SUBDATE	Y
DAY(date)		DAYOFMONTH	Y

函数名	说明	别名	是否支持
DAYOFWEEK(date)	返回date所代表的一星期中的第几天(1~7)		Y
DAYOFMONTH(date)	返回date是一个月的第几天(1~31)	DAY(date)	Y
DAYOFYEAR(date)	返回date是一年的第几天(1~366)		Y
DAYNAME(date)	返回date的星期名		Y
EXTRACT(unit FROM date)	在这里并不兼容mysql所提供的所有unit		Y
FROM_DAYS(N)			Y
FROM_UNIXTIME(unix_timestamp), FROM_UNIXTIME(unix_timestamp, format)	支持FROM_UNIXTIME(unix_timestamp)		Y
GET_FORMAT({DATE / TIME / DATETIME}, {‘EUR’/‘USA’/‘JIS’/‘ISO’/‘INTERNAL’})			Y
HOUR(time)	返回time的小时值(0~23)，注意，在MySQL中可以返回大于24的值		Y
LAST_DAY(date)	不支持不合理的日期，如’2003-03-32‘，会返回null		Y
LOCALTIME([fps])		NOW()	Y
LOCALTIMESTAMP([fsp])		NOW()	Y
MAKEDATE(year, dayofyear)			Y
MAKETIME(hour, minute, second)			Y
MICROSECOND(expr)			Y

函数名	说明	别名	是否支持
MINUTE(time)	返回time的分钟值(0~59)		Y
MONTH(date)	返回date的月份值(1~12)		Y
MONTHNAME(date)	返回date的月份名		Y
NOW([fsp])	返回当前的日期和时间	CURRENT_TIMESTAMP(), LOCALTIME(), LOCALTIMESTAMP(), SYSDATE()	Y
PERIOD_ADD(P, N)			Y
PERIOD_DIFF(P1, P2)			Y
QUARTER(date)	返回date在一年中的季度(1~4)		Y
SECOND(time)			Y
SEC_TO_TIME(seconds)			Y
STR_TO_DATE(str, format)			Y
SUBDATE(date, INTERVAL expr unit), SUBDATE(expr, days)			Y
SUBTIME(expr1, expr2)			Y
SYSDATE([fsp])			Y
TIME(expr)			Y
TIMEDIFF(expr1, expr2)			Y
TIMESTAMP(expr), TIMESTAMP(expr1, expr2)			Y

函数名	说明	别名	是否支持
TIMESTAMPADD(unit, internal, datetime_expr)			Y
TIMESTAMPDIFF(unit, datetime_expr1, datetime_expr2)			Y
TIME_FORMAT(time, format)			Y
TIME_TO_SEC(time)			Y
TO_DAYS(date)	只接收String		Y
TO_SECONDS(expr)			Y
UNIX_TIMESTAMP([date])			Y
UTC_DATE()			Y
UTC_TIME()			Y
UTC_TIMESTAMP([fsp])			Y
WEEK(date[,mode])	返回日期date为一年中第几周(0~53)		Y
WEEKDAY(date)			Y
WEEKOFYEAR(date)			Y
YEAR(date)	返回日期date的年份(1000~9999)		Y
YEARWEEK(date[, mode])			Y

表 2-6: 转换函数 Cast Functions

函数名	说明	别名	是否支持
CAST(expr AS type)	仅仅支持部分类型转换，是mysql CAST的子集		Y
CONVERT(expr, type)		CAST(expr AS type)	Y

函数名	说明	别名	是否支持
CONVERT(expr USING transcoding_name)			Y

表 2-7: 位函数和操作符 **Bit Functions and Operators**

函数名	说明	别名	是否支持
BIT_COUNT()			Y
&, ~, , ^, <<, >>			Y

表 2-8: 信息函数 **Information Functions**

函数名	说明	别名	是否支持
DATABASE()	返回当前数据库名	SCHEMA()	Y
BENCHMARK(count, expr)			N
CHARSET(str)			Y
COERCIBILITY(str)			Y
COLLATION(str)			Y
CONNECTION_ID()			Y
SCHEMA()			Y
FOUND_ROWS()	返回最后一个SELECT查询进行检索的总行数		N
LAST_INSERT_ID([expr])			Y
ROW_COUNT()			N
USER()	返回当前登录的用户名	SYSTEM_USER(), SESSION_USER()	Y
VERSION()	返回MySQL服务器的版本		Y

表 2-9: 聚合函数 Aggregate Functions

函数名	说明	别名	是否支持
AVG([DISTINCT] expr)	返回指定列的平均值		Y
BIT_AND(expr)			Y
BIT_OR(expr)			Y
BIT_XOR(expr)			Y
COUNT(expr)			Y
COUNT(DISTINCT expr,[expr…])	返回指定列中非NULL值的个数		Y
GROUP_CONCAT(expr)	返回由属于一组的列值连接组合而成的结果		Y
MAX([DISTINCT] expr)	返回指定列的最大值		Y
MIN([DISTINCT] expr)			Y
SUM([DISTINCT] expr)	返回指定列的所有值之和		Y
STD(expr)			Y
STDDEV(expr)			Y
STDDEV_POP(expr)			Y
STDDEV_SAMP(expr)			Y
VAR_POP(expr)			Y
VAR_SAMP(expr)			Y
VARIANCE(expr)			Y
WITH ROLLUP	区别于MySQL，使用方式为`group by rollup(c1, c2, ..., cn)`		Y

兼容性说明

- 不支持从用户定义函数（UDF）中传入的超出范围的数字，这种情况下会抛出“`out of range`”错误。

SELECT语句尚未支持的MySQL函数

与*MySQL V5.6*的函数相比较，HybridDB for MySQL尚未支持以下的MySQL函数：

表 **2-10:** 操作符

名称	说明	别名	是否支持
:=			N
BINARY		CAST(expr AS BINARY), CONVERT(expr USING BINARY)	N
ANY, SOME			N

表 **2-11:** 字符函数 **String Functions**

函数名	说明	别名	是否支持
LOAD_FILE(file_name)			N
SOUNDEX(str)			N
SOUNDS LIKE			N
WEIGHT_STRING(str [AS {CHAR / BINARY }(N)] LEVEL levels flags)			N

表 **2-12:** 转换函数 **Cast Functions**

函数名	说明	别名	是否支持
BINARY			N

表 **2-13:** 信息函数 **Information Functions**

函数名	说明	别名	是否支持
FOUND_ROWS()	返回最后一个SELECT查询进行检索的总行数		N
ROW_COUNT()			N

表 2-14: 其他函数 **Miscellaneous Functions**

函数名	说明	别名	是否支持
DEFAULT(col_name)			N
FORMAT(X,D)			N
GET_LOCK(str, timeout)			N
INET_ATON(expr)			N
INET_NTOA(expr)			N
INET6_ATON(expr)			N
INET6_NTOA(expr)			N
IS_FREE_LOCK(str)			N
IS_IPV4(expr)			N
IS_IPV4_COMPAT(expr)			N
IS_IPV4_MAPPED(expr)			N
IS_IPV6(expr)			N
IS_USERD_LOCK(str)			N
MASTER_POS_WAIT(log_name, log_pos[, timeout])			N
NAME_CONST(name,value)			N
RELEASE_LOCK(str)			N
SLEEP(duration)			N
UUID()			N
UUID_SHORT()			N
VALUES(col_name)			N

SELECT语句支持的Oracle函数

HybridDB for MySQL目前支持在SELECT查询语句中使用如下的Oracle函数：

表 2-15: 操作符

函数名	说明	别名	是否支持
ROLLUP	用在GROUP BY子句里，如`group by rollup(c1, c2, ..., cn)`		Y
CUBE	用在GROUP BY子句里，如`group by cube(c1, c2, ..., cn)`		Y
GROUPING			Y
OVER	开窗函数窗口声明		Y
RANK	排名函数，可同开窗函数一同使用		Y
DENSE_RANK	排名函数，可同开窗函数一同使用		Y
ROW_NUMBER	排名函数，可同开窗函数一同使用		Y

- 若无特殊说明，以下函数均为*MySQL V5.6*中的函数定义。
- 如下函数目前仅可以在**SELECT查询语句中使用**，尚不支持在其他SQL语句中使用（如UPDATE、DELETE、INSERT、REPLACE等）。

2.2 DELETE 语句

标准语法

```
DELETE FROM table_name WHERE filter_condition
```

限制说明

- 当前不支持分布式事务，如果一次 delete 多个行，且这些行不在同一个分区，那么数据库会开启一个不完整的分布式事务，在部分分区提交成功部分分区提交失败时，可能导致回滚不一致，用户应慎用或不用；
- 允许 delete 整张表，但不建议使用；
- 不支持 delete limit 从句。

2.3 INSERT 语句

标准语法

```
INSERT [IGNORE] [INTO] table_name (column_name [, ...]) VALUES (
insert_expr_list) [, insert_expr_list [, ...]] [on duplicate key
update  column_name = expr [, ...]]
insert_expr_list:
    insert_expr [, ...]
```

限制说明

- 允许 insert 多个值，且这些值可以在任意多个分区。
- 当前不支持分布式事务，如果一次 insert 多个行，且这些行不在同一个分区，那么数据库会开启一个不完整的分布式事务，在部分分区提交成功部分分区提交失败时，可能导致回滚不一致，用户应慎用或不用。
- 若用户的表包含自增主键，则 insert 时该列的生成规则需遵守自增主键的使用方法，自增主键会产生唯一值，但该值不一定单调递增，也不一定连续。
- insert_expr 列内容前不得附带字符集等前缀描述，如：_utf8’a’，是不支持的。
- 对于分区键，用户必须自行保证分区键所定义的列与用户 insert 时填入的列值的类型是匹配的。若用户建表时列定义为整型，而 insert 时为浮点型或字符串型，则会因数据类型截断，导致数据分布紊乱。
- on duplicate key update 从句的特性类似于 replace，无法保证非分区字段的唯一性，虽然语法上没有禁止，但是不建议用户使用，使用者请自行保证唯一性。

INSERT SELECT子句

INSERT SELECT子句的语法如下：

```
INSERT [IGNORE] INTO tbl_name (col_name [, col_name] ...) SELECT ...
REPLACE INTO tbl_name (col_name [, col_name] ...) SELECT ...
```

该子句的限制条件如下：

- SELECT子句必须包含INSERT/REPLACE目标表的分区键。
- INSERT/REPLACE和SELECT子句必须包含具体的列名。
- 如果INSERT/REPLACE目标表包含自增ID主键，SELECT子句需要对自增ID主键显式赋值。

2.4 UPDATE 语句

标准语法

```
UPDATE table_name SET column_name = update_expr [, ...] WHERE
filter_condition
```

限制说明

- 当前不支持分布式事务，如果一次 update 多个行，且这些行不在同一个分区，那么数据库会开启一个不完整的分布式事务，在部分分区提交成功部分分区提交失败时，可能导致回滚不一致，用户应慎用或不用；
- 不允许更新主键和分区键，若需要变更主键和分区键，需要先 delete 后重新 insert；
- update_expr 列内容前不得附带字符集等前缀描述，如：_utf8'a'，是不支持的。
- 不支持 update limit 从句。

2.5 REPLACE 语句

标准语法

```
REPLACE [INTO] table_name (column_name [, ...]) VALUES (replace_ex
pr_list) [, (replace_expr_list) [, ...]]
```

限制说明

- 当前不支持分布式事务，如果一次 replace 多个行，且这些行不在同一个分区，那么数据库会开启一个不完整的分布式事务，在部分分区提交成功部分分区提交失败时，可能导致回滚不一致，用户应慎用或不用；
- 若用户的表包含自增主键，则 replace 时该列的生成规则需遵守自增主键的使用方法，自增主键会产生唯一值，但该值不一定单调递增，也不一定连续；
- replace_expr 列内容前不得附带字符集等前缀描述，如：_utf8'a'，是不支持的；
- 对于分区键，用户必须自行保证分区键的列定义，与用户 replace 时填入的列值的个数是匹配的，若用户建表时列定义为整型，而 replace 时为浮点型或字符串型，则会因数据类型截断，导致数据分布紊乱；
- 不支持 replace set 从句。

3 事务性说明

3.1 基本SET语句

- **set autocommit = 0/1语句**：`set autocommit = 0`语句用于设置连接长期开启事务，若用户不显式进行任何 commit，则该连接之前的更新均不会提交。
- **autocommit 由0转1**：若用户通过`set autocommit = 0`开启一个事务，稍后未进行提交而通过`set autocommit = 1`提交一个事务，那么分布式数据库将隐式地帮助用户向所有分区发送一个`set autocommit = 1`，这条语句的特性与普通 commit 相同。由于HybridDB for MySQL的commit必然会退出事务，此处的`set autocommit = 1`也将必然成功并返回ok结果，后续的操作将不再自动进入事务。
- **set transaction isolation level {read uncommitted|read committed|repeatable read| serializable}** 语句：HybridDB for MySQL单分区事务的事务隔离级别兼容MySQL的事务隔离级别，多分区事务的事务隔离级别总是为 read_committed。`set transaction isolation level XXXXXX`与`set tx_isolation = XXXXXX`仅影响单分区事务的事务隔离级别。
- **set transaction read only|write语句**：该语句用于设置连接是否为只读或读写的。
- **set names XXXXXX语句**：该语句用来设置连接的字符集。
- 事务中，环境变量设置将向每一个参与事务的分区立即发送一条环境变量设置语句，除了 set autocommit 0转1之外，其它的 set 动作若失败，不会影响事务状态。
- 连接内新建立的后端分区连接，均会进行一轮全量环境变量设置，保证新连接具有最新的环境变量内容。
- 不支持具有历史依赖关系的环境变量，如 set a = 1, set b = a + 1, set a = b + 1，所有环境变量的设置必须是自包含且幂等的单变量设置语句。

3.2 其他SET语句

- 其它的环境变量均被解析器拦截，报告给用户一个 warning 消息，即设置环境变量的动作未执行。
- 非事务中，环境变量设置过程与 begin 语句开启的过程相同，均为延迟开启。即用户的环境变量语句或 begin 语句到达后，立即向用户报告 ok 结果。待用户真正的请求到达时，再根据请求涉及的后端分区，进行一轮独立的环境变量设置或 begin 的过程。完成后，再进行实际的请求投递。若这轮延迟的环境变量设置或 begin 失败，则向用户报错，用户下次请求到达时，仍然会进行一轮新的环境变量设置或 begin 的过程。

3.3 BEGIN（开启事务）

begin 语句用于开启单次事务。

对于连续 begin，若用户通过`begin/start transaction/set autocommit = 0`开启第一个事务，稍后未进行提交而进行第二个begin/start transaction，那么分布式数据库将隐式地帮助用户 commit 上一个事务，这个 commit 的特性与普通 commit 相同，提交完成后，将自动进入下一个事务，HybridDB for MySQL这个工作流程与MySQL一致。

一个单行事务，若不在显式事务语句的保护下：若它是单分区的事务，那么该单行事务将具有与MySQL相同的事务特性；若它是跨分区的事务，那么该单行事务将自动使用一阶段提交分布式事务，在部分分区提交成功部分分区提交失败时，可能导致回滚不一致。

3.4 COMMIT（提交事务）

由于当前分布式数据库仅使用了一阶段提交事务，因此提交时，若一部分分区成功，而另一部分分区失败或异常关闭连接，那么将造成分区数据不一致。HybridDB for MySQL的commit无论提交成功或失败，都将退出事务。commit成功，则所有更新将可见；commit失败，则所有更新将自动rollback。

3.5 ROLLBACK（回滚）

无论分区回滚成功或失败，或分区异常关闭连接，分布式数据库保证始终向客户端返回 ok 结果。

3.6 DEADLOCK（死锁）

- HybridDB for MySQL在发现事务中的普通语句存在死锁后，将仅保留一个事务并允许其操作，同时清除其它事务的死锁，并回滚这些事务。
- 若事务中的更新语句一次仅涉及一个分区，死锁的行存在于两个分区，那么死锁过程不会立即被检测出来。多个事务的死锁更新会请求锁，直到锁超时，然后由HybridDB for MySQL通知更新error。这个 error 结果不会令分区退出事务状态，后续的操作与普通事务相同，分布式数据库将向用户返回锁超时错误。
- 若事务中的更新语句一次仅涉及一个分区，死锁的行存在于一个分区，那么死锁过程会立即被检测出来。多个事务的死锁更新，仅有一个被保留，其它事务将被立即回滚。由于事务更新历史中存在跨分区的可能，因此分布式数据库将强行锁定所有未通过HybridDB for MySQL死锁检测且被清除的事务，强制用户只能进行 rollback 而不得进行其它任何操作。对于那个通过 HybridDB

for MySQL死锁检测的事务，后续的操作与普通事务相同，分布式数据库将向用户返回死锁错误，后续非 rollback 语句将向用户返回仅支持 rollback 错误。

- 若事务中的更新语句一次涉及多个分区，死锁的行存在于两个分区，那么死锁过程不会立即被检测出来。多个事务的死锁更新会请求锁，直到锁超时，然后由HybridDB for MySQL通知更新error。这个 error 结果不会令分区退出事务状态，后续的操作与普通事务相同，分布式数据库将向用户返回数据不一致错误。
- 若事务中的更新语句一次涉及多个分区，死锁的行存在于一个分区，那么死锁过程会立即被检测出来。多个事务的死锁更新，仅有一个被保留，其它事务将被立即回滚。由于事务更新历史中存在跨分区的可能，因此分布式数据库将强行锁定所有未通过HybridDB for MySQL死锁检测且被清除的事务，强制用户只能进行 rollback 而不得进行其它任何操作。对于那个通过 HybridDB for MySQL死锁检测的事务，后续的操作与普通事务相同，分布式数据库将向用户返回数据不一致错误，后续非 rollback 语句将向用户返回仅支持 rollback 错误。

3.7 连接异常关闭

若当前 session 处于事务中，而任意一个参与该事务的分区异常关闭了连接，则该 session 也将关闭与客户端的连接，并回滚事务。

3.8 不一致情况

当前仅支持单分区的完整事务。如果一个请求更新了多个分区，则没有分布式事务的保证，如果部分分区更新成功但部分分区更新失败，则可能出现数据不一致。

事务状态不一致

事务中，遭遇单分区死锁，该分区将退出事务状态，其它分区可能仍然在事务状态，此时分区事务状态不一致，分布式数据库要求 client 仅能发送 rollback，而不会为 client 自动进行其它分区的 rollback 调用。

事务开启时，begin/start transaction 部分成功部分失败，则成功的部分分区连接将被关闭（减少参与失败事务的分区数，减小事务失败的影响），同时向 client 返回 error 结果。

事务开启时，set autocommit = 0部分成功部分失败，则失败的部分分区连接将被关闭（set autocommit = 0语句被延迟执行，此前已经向 client 保证了成功，此处仅能关闭执行失败的部分分区，成功的分区将进入事务），同时向 client 返回 error 结果。

已提交数据不一致

单行事务跨分区语句部分成功部分失败，此时将向 client 报出严重错误的异常，此时数据将不一致，并没有任何 rollback 的机会。

多行事务跨分区语句，commit 或隐式 commit 部分成功部分失败，此时将向 client 报出严重错误的异常，此时数据将不一致，并没有任何 rollback 的机会。

未提交数据不一致

多行事务跨分区语句，普通语句部分成功部分失败，此时将报出数据不一致的异常， client 仍然有 rollback 的机会。

3.9 其他语句说明

HybridDB for MySQL对其他SQL语句的使用限制:

- 不支持grants语句；
- 对于desc，当前也支持desc XXXXXX表定义；
- show processlist返回当前所有连接的信息，但是不支持show full processlist；
- show slave hosts|status不支持；
- show binary|master logs不支持；
- show open tables不支持；
- show master status不支持；
- show binlog events不支持；
- show engine XXXXXX status不支持；
- show grants for XXXXXX不支持；
- kill XXX 或者kill connection XXX将强行关闭掉指定的连接，连接的id可以利用select connection_id()语句获取，也可以从show processlist语句中获取；

3.10 客户端及连接池

客户端连接

- 使用JDBC等连接器，必须指明连接的库，否则无法执行；
- 不支持游标访问；
- 跨分区语句将会在每个分区上建立一条连接，可能导致分区的连接数成比例增加；

- COM_QUERY语句长度：当前限制用户SQL语句不得超过16 MB，间接要求用户的单行不得超过16 MB大小；
- 客户端可使用的命令类型：COM_PING | COM_QUIT | COM_QUERY | COM_KILL | COM_INIT_DB | COM_PROCESS_INFO | COM_PROCESS_KILL；其它命令均不支持（包括prepare命令 COM_STMT_XXXXXX、binglog命令COM_BINLOG_XXXXXX等）。

服务器端连接池

- HybridDB for MySQL可以提供服务器端连接池；默认情况下此连接池未开启，用户若有需要请通过工单系统申请；
- 连接池功能开启后，将有效减少服务器到后端分区的连接数，最大化的复用连接，但其实现与HybridDB for MySQL的支持紧密相关；
- 启用连接池后，单行事务的每条语句都将附带当前库名和当前全量环境变量到后端HybridDB for MySQL，单行事务语句执行完毕后，立即将后端分区归还到全局连接池；
- 启用连接池后，以begin/start transaction/set autocommit = 0 开启的多行事务，仅在第一条环境变量设置语句中附带当前库名和当前全量环境变量到后端HybridDB for MySQL，直到事务提交、回滚、隐式提交，才将后端分区归还到全局连接池；
- 启用连接池后，将对性能产生一些影响。

4 性能优化建议

4.1 分区设计

用户存有海量数据的表应该按照数据规模进行拆解，表的数据将拆解成多个数据分区独立存储，通常的设计原则是：

主键（**Primary Key**）

单实例数据库不要求表一定要有主键，但是对于分布式数据库，主键则是必须的，以保证一行数据是全局唯一的，避免迁移过程出现问题。如果用户没有特殊的性能需求或业务耦合问题，则应将主键设计为无意义的整数值或者自增的数值。

分区键（**Partition Key**）

用户的分区表必须按照一种维度进行数据划分，用户在按照分区键维度进行查询时，就能做到线性性能增长，分区键通常有如下选择方法：

- 按业务ID切分，如用户ID、商品ID等，适合每个业务ID的数据较均匀且查询简单的场景；
- 按多个业务维度切分，用户建立多张表存入相同数据，但是每张表按照不同业务维度切分，查询时根据过滤条件选择不同的表，以提升访问性能，适合查询复杂且单一切分方式不能满足需求的场景；
- 按自增主键切分，若表分区方式为分区表，主键为自增，且该字段同时为分区键，则此时写入为随机分区，按照非主键查询时需要读取所有分区，适合有数据偏斜且写多读少的场景；
- 至少保证在分区键上有一个索引。

业务列

其它的列统归为此类，仅用于存储数据，通常要考虑：

- 列不宜过长，够用即可，临时加载的长列会消耗额外内存，影响查询性能；
- 准确定义列的类型，避免查询时使用的类型与表定义的类型不匹配，从而进行类型转换，以致不能正确使用索引；
- 优先使用timestamp类型代替其它时间日期类型，且使用时严格遵守MySQL的时间日期格式。

主键索引

若主键是自增类型，则主键索引不会对整体性能优化有改善，若主键与业务相关，则可以对最频繁使用的SQL语句的查询条件建立主键索引，这样可以提升性能。

辅助索引

其他情况下，如果不能通过将SQL语句拆解成单分区的，且不能利用主键进行索引优化时，需要对全部分区进行扫描，此时可以对这部分全分区扫描的语句的查询条件建立索引，使得在每个分区上进行访问时，仍然能取得较高的性能

说明：

HybridDB for MySQL目前暂不支持广播表（广播表的数据在每个数据分区均有相同副本，用于与分区表进行join）。

4.2 优化建议

常用优化包括

- 合理的选择拆分字段。选择拆分字段时需要综合考虑查询性能、分布式事务、热点、数据迁移等多个因素；
- 掌握SQL的执行计划，尤其是核心SQL。对于不确定的SQL应在分布式数据库执行‘explain sql’命令，确定SQL有没有跨分区、有没有改写以及底层有没有合适的索引，合并时是否进行了排序和分组动作；
- 对底层MySQL建立合适的索引，这一点看似与分布式数据库无关，但却是最重要的。分布式数据库的高性能依赖于底层数据库的高性能，而对底层数据库性能来说，建立需要的索引是重要的环节；
- 确保语句能正确使用到索引，例如查询条件能被索引完全覆盖到，保证分区键上有索引等；
- 查询尽量在单机完成，最为简单的方式就是在分区字段上指定等值条件，使操作只发送到一个后台数据库节点。若不指定，则操作需要发送到每个后台节点，可能导致性能大幅下降；
- 尽量避免分布式事务和分布式查询；
- 同时使用其他适用于MySQL的优化手段。

设计表结构的过程

- 预估数据量、访问规模，准备测试数据，测试原始数据库的性能基线；
- 分析和设计表结构、约束、索引；
- 分析和设计分区方式和分区字段；
- 分析和设计常用sql语句访问的频度和分区数量
- 分析和设计需要聚合、排序、分组、条件过滤的字段；

- 重新在分布式数据测试性能基线；
- 调整分布式数据库的配置参数，重新观察系统；
- 调整读写比例、并发活跃连接数，重新观察系统；
- 比较结果，找出正确的优化方式。

5 限制说明

- DDL 当前仅支持表和索引方面的操作；
- DDL 执行期间，事务隔离级别为读未提交；
- 不支持`add column before|after XXXXXX`；
- 不应使用`alter table add|drop index`语法变更索引，这个语法为阻塞式的，会影响正常的DML语句执行，应该使用`create|drop index on`语法；
- 目前并未开放任何系统库和系统表，包括information_schema等；
- 不支持以下高级特性：
  - 不支持表空间；
  - 不支持 MySQL 中关于分区表的语法；
  - 不支持存储过程和用户自定义函数；
  - 不支持触发器；
  - 不支持 event；
  - 不支持游标。
- 不支持的约束：
  - 不支持唯一性约束，允许用户建立主键和唯一性索引，但是仅起提示作用，仅仅能保证自增主键的唯一性；
  - 不支持外键约束；
  - 不支持CHECK约束。