OBASE 分布式数据库

SQL 使用手册

上海丛云信息科技有限公司

2017 年 8 月

目录

1. 编写目的

本文列出了 OBASE 分布式数据库所支持的 SQL 语句、语法规则；同时给出了针对 OBASE 分布式数据库编写高性能 SQL 语句的方法，为基于 OBASE 的应用开发提供指导性意见。

2. 读者对象

- 数据库管理员
- 应用开发工程师

读者应具有使用 SQL 语言的经验，通过本文可以了解 OBASE 数据库所支持的数据类型和语法与其它数据库系统在细节上的区别，以便顺利地使用 OBASE 分布式数据库。

3. 基本的数据库对象

与 DB2 等数据库系统不同，OBASE 数据库中没有 SUBSYSTEM、TABLESPACE、PLAN、TRIGGER、CURSOR、PACKAGE 和存储过程等数据库对象。OBASE 目前支持 DATABASE、TABLE、VIEW、INDEX 和 SEQUENCE 四种数据库对象。

OBASE 数据库目前不支持临时表、衍生表、全局临时表、局部临时表等类型的关系表，也不支持外键和约束。OBASE 数据库中表的主键由能够唯一确定表中记录一个或多个列组成，但不允许将表中所有属性定义为主键，即表中至少要有一个非主键属性。

4. SQL 基础

OBASE 数据库完全兼容 MySQL 的通讯协议，所以用户可以直接使用 MySQL 客户端、MySQL JDBC Driver 连接 OBASE 数据库。

OBASE 数据库的 SQL 语法遵循 SQL92 标准，使用单引号标注字符串，使用双引号标注表名、列名或函数名，双引号内可以出现 SQL 保留的关键字。OBASE 数据库 SQL 语句中的关键字、表名、列名、函数名等均大小写不敏感。表名和列名都被转换为小写之后存入系统表中，所以即使建表时列名是大写的，查询时获得的列名也是小写。如果用户需要保存大写字母，请使用双引号标注表名和列名，例如："Info"。

4.1. 数据类型

OBASE 数据库所支持的每种数据类型都有有效值范围。

表 4-1：OBASE 数据库支持的数据类型

数据类型	说明
bigint	按 8 字节有符号整型格式存储，显示为 int；
int/integer/mediumint /smallint/tinyint	int、integer、mediumint 无论语义还是实现都是等价的，按 4 字节有符号整型格式存储，显示为 int32；
binary/char /varbinary/varchar	字符串，使用单引号进行标注。在 OBASE 数据库中，这四种类型均存储为 varchar 类型，显示为 varchar；
bool	布尔类型，值为 1 或者 0。 插入 bool 型数据 true 为 1，false 为 0； 插入数值型数据，非 0 值为 1，0 为 0； 插入 varchar 型数据，‘true’、‘t’、‘yes’、‘y’为 1，其它值为 0；
createtime	特殊数据类型，用于记录本条数据第一次插入时的时间，由系统自动维护，用户不能直接修改； 该类型的列不能作为主键的组成部分。
modifytime	特殊数据类型，用于记录本条数据最近一次被修改的时间，由系统自动维护，用户不能直接修改； 该类型的列不能作为主键的组成部分。
datetime/timestamp	时间戳类型，支持的格式有 YYYY-MM-DD、YYYY-MM-DD HH:MI:SS、YYYY-MM-DD HH:MI:SS. SSSSSS； **不支持 time with time zone, timestamp with time zone；**
date	日期类型，格式 YYYY-MM-DD；
time	时间类型，格式 HH:MI:SS；
double/real	8 字节浮点数，double/real 均被存储为 double 类型；
float	4 字节浮点数；

数据类型	说明
decimal(p,s)	decimal 类型，p 表示总位数，不能超过 38，s 表示小数部分的精度，最大不超过 37，即整数位最少 1 位；小数部分超过 s 的部分将被截取(非四舍五入)，不足时，将会补 0；

4.2. 函数

4.2.1. 系统函数

- **CAST(expr AS type)**

将表达式 expr 的值转换为 type 数据类型。

```
mysql> select CAST(123 as int),CAST(123 as varchar),CAST(123 as bool);
+------------------+----------------------+-------------------+
| CAST(123 as int) | CAST(123 as varchar) | CAST(123 as bool) |
+------------------+----------------------+-------------------+
|              123 | 123                  |                 1 |
+------------------+----------------------+-------------------+
1 row in set (0.00 sec)
```

- **COALESCE(expr, expr, expr, …)**

依次检查各参数表达式，遇到非 NULL 值即停止并返回该值，如果所有表达式都是 NULL 值，最终将返回一个 NULL 值。

```
mysql> select COALESCE(null,null,3,4,5),COALESCE(null,null,null);
+---------------------------+-------------------------+
| COALESCE(null,null,3,4,5) | COALESCE(null,null,null) |
+---------------------------+-------------------------+
|                         3 |                    NULL |
+---------------------------+-------------------------+
1 row in set (0.00 sec)
```

- **DECODE(expr, if1, then1, [if2, then2, … ,] else)**

计算参数表达式 expr 的值，依次进行比较，如果 expr=if1 则返回 then1 值，如果 expr=if2 则返回 then2 值，… ；如果 expr 的值不匹配任何 if 条件，则返回 else 值。所有的参数都可以是表达式，最多支持 255 个参数。

```
mysql> select decode(year(now()),2016,'Last Year',2017,'This Year','Long long ago') as now,
    -> decode(year(now())-1,2016,'Last Year',2017,'This Year','Long long ago') as last,
    -> decode(year(now())-10,2016,'Last Year',2017,'This Year','Long long ago') as ago
    -> ;
+-----------+-----------+---------------+
| now       | last      | ago           |
+-----------+-----------+---------------+
| This Year | Last Year | Long long ago |
```

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

- **FLOOR(expr)**

对参数表达式 expr 的值向下取整。

```
mysql> select floor(-1.5),floor(3.14),floor(5/3),floor('0.618');
+-------------+-------------+------------+----------------+
| floor(-1.5) | floor(3.14) | floor(5/3) | floor('0.618') |
+-------------+-------------+------------+----------------+
|          -2 |           3 |          1 |              0 |
+-------------+-------------+------------+----------------+
1 row in set (0.00 sec)
```

- **CEIL(expr)**

对参数表达式 expr 的值向上取整。

```
mysql> select ceil(-1.5),ceil(321),ceil(3.14),ceil(5/3),ceil('0.618');
+------------+-----------+------------+-----------+---------------+
| ceil(-1.5) | ceil(321) | ceil(3.14) | ceil(5/3) | ceil('0.618') |
+------------+-----------+------------+-----------+---------------+
|         -1 |       321 |          4 |         1 |             1 |
+------------+-----------+------------+-----------+---------------+
1 row in set (0.00 sec)
```

- **DATABASE()**

返回当前连接的数据库名称，无参数。

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

- **ROUND(expr, digits)**

对参数表达式 expr 的值进行四舍五入操作，保留 digits 位小数。参数 digits 必须为整数，可以大于 0、等于 0 或者小于 0。

```
mysql> select round(3.1415,2),round(123.5,0),round(-123.4,-2),round(5.0/3,3);
+-----------------+----------------+-----------------+----------------+
| round(3.1415,2) | round(123.5,0) | round(-123.4,-2) | round(5.0/3,3) |
+-----------------+----------------+-----------------+----------------+
|            3.14 |            124 |            -100 |          1.667 |
+-----------------+----------------+-----------------+----------------+
```

- **HEX(str)**

将字符串转换为十六进制数。参数 str 必须是整数或字符串，当输入是整数(不限进制)时，输出该整数的十六进制表示；当输入是字符串时，输出是该字符

串字节流的十六进制表示。

注：OBASE 数据库中两个十六进制数位表示一个字符，因此需要用偶数位数来表示，例如 0x123 转换会失败，0x0123 可以转换成功。

```
mysql> select hex('OBASE'),hex(123),hex(null),hex(0x0123);
+--------------+----------+-----------+-------------+
| hex('OBASE') | hex(123) | hex(null) | hex(0x0123) |
+--------------+----------+-----------+-------------+
| 4f42415345   | 7b       | NULL      | 0123        |
+--------------+----------+-----------+-------------+
1 row in set (0.00 sec)
```

- **UNHEX(str)**

HEX(str)的反向操作，即将十六进制数转换为字符串。参数 str 必须是字符串或 NULL 值。当字符串 str 是合法的十六进制数时，将其转换为对应的字符串，当 str 不是合法的十六进制数时返回 NULL 值，str 为 NULL 时返回 NULL。

```
mysql> select unhex('4f42415345'),unhex('7b'),unhex('123g'),unhex(null),unhex('0123');
+---------------------+-------------+---------------+-------------+---------------+
| unhex('4f42415345') | unhex('7b') | unhex('123g') | unhex(null) | unhex('0123') |
+---------------------+-------------+---------------+-------------+---------------+
| OBASE               | {           | NULL          | NULL        | #             |
+---------------------+-------------+---------------+-------------+---------------+
1 row in set (0.00 sec)
```

- **INT2IP(int_value)**

将一个整数转换成 IP 地址。输入整数值，按数值转换；输入非整数返回 NULL 值；目前只支持 IPV4 地址转换。

```
mysql>select int2ip(16777216),int2ip(1),int2ip(4294967295),int2ip(true),int2ip(now()),
int2ip(10.1);
+------------------+-----------+--------------------+--------------+---------------+-----------
---+
| int2ip(16777216) | int2ip(1) | int2ip(4294967295) | int2ip(true) | int2ip(now()) | int2ip(10.1)
|
+------------------+-----------+--------------------+--------------+---------------+-----------
---+
| 0.0.0.1          | 1.0.0.0   | 255.255.255.255    | NULL         | NULL          | NULL
|
+------------------+-----------+--------------------+--------------+---------------+-----------
---+
```

- **IP2INT(str)**

将字符串表示的 IP 地址转换为整数，当输入的参数不是字符串或者字符串表示的不是合法 IPV4 地址时，返回 NULL 值。

```
mysql>select ip2int('0.0.0.1'),ip2int('192.168.0.1'),ip2int('255.255.255.255'),
```

```
ip2int('256.0.0.0');
+------------------+----------------------+--------------------------+--------------------+
| ip2int('0.0.0.1') | ip2int('192.168.0.1') | ip2int('255.255.255.255') | ip2int('256.0.0.0') |
+------------------+----------------------+--------------------------+--------------------+
|          16777216 |              16820416 |               4294967295 |               NULL |
+------------------+----------------------+--------------------------+--------------------+
```

- **LENGTH(str)**

返回字符串 str 的长度，int 型整数，单位为字节。如果参数 str 不是字符串，则返回 NULL 值。

```
mysql> select length('OBASE'),length('中国'),length(123),length(10.1),length(null);
+-----------------+-----------------+-------------+--------------+--------------+
| length('OBASE') | length('中国') | length(123) | length(10.1) | length(null) |
+-----------------+-----------------+-------------+--------------+--------------+
|               5 |               6 |        NULL |         NULL |         NULL |
+-----------------+-----------------+-------------+--------------+--------------+
```

- **CHAR_LENGTH(str)**

类似与 length()返回字符串 str 的长度，以字符为单位。多字节字符算单个字符。这意味着对于包含 5 个 2 字节字符的字符串，LENGTH()返回 10，而 CHAR_LENGTH()返回 5。

```
mysql>select char_length('OBASE'),char_length('中国'),char_length(123),char_length(10.1),
char_length(null);
+----------------------+----------------------+------------------+-------------------+---------
-----------+
|  char_length('OBASE')  |  char_length(' 中 国 ')  |  char_length(123)  |  char_length(10.1)  |
char_length(null) |
+----------------------+----------------------+------------------+-------------------+---------
-----------+
|                  5 |                  2 |           NULL |            NULL |            NULL |
+----------------------+----------------------+------------------+-------------------+---------
-----------+
1 row in set (0.00 sec)
```

- **LOWER(str)|LCASE(str)**

lcase(str)等同于 lower(str)，将字符串 str 转换为小写字符，兼容中文字符。

```
mysql> select lower('欢迎使用 OBASE'),lcase('欢迎使用 OBASE'),lower(null),lower(123.1);
+----------------------------+----------------------------+-------------+--------------+
|     lower('欢迎使用 OBASE') |     lcase('欢迎使用 OBASE') | lower(null) | lower(123.1) |
+----------------------------+----------------------------+-------------+--------------+
| 欢迎使用 obase               | 欢迎使用 obase               | NULL        | 123.1        |
+----------------------------+----------------------------+-------------+--------------+
1 row in set (0.00 sec)
```

- **UPPER(str)|UCASE(str)**

将字符串 str 转换为大写字符，兼容中文字符。

```
mysql> select upper('欢迎使用 obase'),ucase('欢迎使用 obase'),upper(null),upper(123.1);
```

```
+---------------------------+---------------------------+-------------+--------------+
|     upper('欢迎使用 obase') |     ucase('欢迎使用 obase') | upper(null) | upper(123.1) |
+---------------------------+---------------------------+-------------+--------------+
| 欢迎使用 OBASE               | 欢迎使用 OBASE               | NULL        | 123.1        |
+---------------------------+---------------------------+-------------+--------------+
1 row in set (0.00 sec)
```

- **SUBSTR(str, pos, len)**

返回字符串 str 中起始于位置 pos，长度为 len 的子字符串，也支持 SUBSTR(str, pos)和 SUBSTR(str FROM pos)格式。不带 len 参数时，返回从 pos 位置到字符串结尾的子字符串。pos 值为负数时，位置从字符串的结尾字符数起。len 小于等于 0 时，返回结果为空字符串。如果 pos 指示的位置在字符串中不存在时，返回为空字符串。

```
mysql>select substr('0123456789',2,3) as c1,substr('0123456789',-5,3) as c2,
substr('0123456789',-5) as c3,substr('0123456789',22) as c4,substr('0123456789',2,-10) as c5;
+------+------+-------+------+------+
| c1   | c2   | c3    | c4   | c5   |
+------+------+-------+------+------+
| 123  | 567  | 56789 |      |      |
+------+------+-------+------+------+
1 row in set (0.00 sec)
```

- **TRIM([[{both | leading | trailing}] [remstr] FROM] str)**

删除字符串 str 的前缀和（或）后缀，默认为 both，即删除前缀和后缀。参数 remstr 为可选项，指定前缀和后缀的内容，默认为空格。

```
mysql> select trim(' x  test  x ') c1 ,trim(leading 'ab' from 'ab123456789ab') c2 ,trim(trailing
'ab' from 'ab123456789ab') c3;
+------------+-------------+-------------+
| c1         | c2          | c3          |
+------------+-------------+-------------+
| x  test  x | 123456789ab | ab123456789 |
+------------+-------------+-------------+
1 row in set (0.00 sec)
```

- **LISTAGG(col_name, split_char)**

列转行函数，参数 col_name 为列名，参数 split_char 为输出时的分隔符。可选参数[within group(order by col_name[desc|asc][, col_name[desc|asc]]) [desc]]用于指定排序列。

```
mysql> select * from tbl01;
+------+------+------+------+
| pk   | col1 | col2 | col3 |
+------+------+------+------+
|    1 |    1 |    1 |    1 |
|    2 |    2 |    2 |    2 |
|    3 |    3 |    3 |    3 |
|    4 |    4 |    4 |    4 |
+------+------+------+------+
4 rows in set (0.00 sec)

mysql> select listagg(pk,'---') from tbl01;
+-------------------+
```

```
| listagg(pk,'---') |
+-------------------+
| 1---2---3---4     |
+-------------------+
1 row in set (0.00 sec)

mysql> select listagg(pk,',') within group(order by col3) from tbl01;
+-------------------------------------------+
| listagg(pk,',') within group(order by col3) |
+-------------------------------------------+
| 1,2,3,4                                   |
+-------------------------------------------+
1 row in set (0.00 sec)
```

- **MOD(N, M)**

返回 N 除以 M 后的余数。

```
mysql> select mod(10,3);
+-----------+
| mod(10,3) |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)
```

- **NVL(expr1, expr2)| VALUE(expr1, expr2)**

如果 expr1 的结果为 null 值，则返回 expr2。如果 expr1 的结果不是 null 值，则返回 expr1。expr1 和 expr2 可以是任意一种数据类型。如果 expr1 与 expr2 的结果皆为 null 值，则返回 NULL.。

```
mysql> select nvl(null,123);
+---------------+
| nvl(null,123) |
+---------------+
|           123 |
+---------------+
1 row in set (0.00 sec)

mysql> select nvl('',123);
+-------------+
| nvl('',123) |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)
```

- **NULLIF(expr1, expr2)**

如果第一个参数等于第二个参数，则返回 NULL，否则返回第一个参数。

```
mysql> select nullif(1,1),nullif(1,2);
+-------------+-------------+
| nullif(1,1) | nullif(1,2) |
+-------------+-------------+
|        NULL |           1 |
+-------------+-------------+
1 row in set (0.00 sec)
```

- **POSITION(SUBSTR in STR)**

在字符串 STR 里面,字符串 SUBSTR 出现的第一个位置(INDEX)，INDEX 是从 1 开

始计算，如果没有找到就直接返回 0，如果 substr 或 str 为 NULL，则返回 NULL。

```
mysql> select position('ab' in 'abcde'),position('ab' in ''),position('ab' in null),position(null
in 'abcde');
+--------------------------+---------------------+-----------------------+-----------------
---------+
| position('ab' in 'abcde') | position('ab' in '') | position('ab' in null) | position(null in
 'abcde') |
+--------------------------+---------------------+-----------------------+-----------------
---------+
| 1                        | 0                   | NULL                  | NULL
|
+--------------------------+---------------------+-----------------------+-----------------
---------+
```

- **INSTR(STR, SUBSTR)**

在字符串 STR 里面,字符串 SUBSTR 出现的第一个位置(INDEX)，INDEX 是从 1 开始计算，如果没有找到就直接返回 0，如果 substr 或 str 为 NULL，则返回 NULL。

```
mysql> select instr('abcde' , 'de'),instr('ab', ''), instr('ab', null),instr(null , 'abcde');
+----------------------+-----------------+-------------------+----------------------+
| instr('abcde' , 'de') | instr('ab', '') | instr('ab', null) | instr(null , 'abcde') |
+----------------------+-----------------+-------------------+----------------------+
| 4                    | 1               | NULL              | NULL                 |
+----------------------+-----------------+-------------------+----------------------+
1 row in set (0.00 sec)
```

- **DEC(expr, int, int) | DECIMAL(expr, int, int)**

将 expr 转成 decimal 类型，第 2 个参数和第 3 个参数分别表示 decimal 类型的位数和精度，等价于 cast(expr as decimal(p,s))。

```
mysql> select dec('123.45',8,4),dec('123.45',5,1);
+-------------------+-------------------+
| dec('123.45',8,4) | dec('123.45',5,1) |
+-------------------+-------------------+
|         123.4500 |             123.4 |
+-------------------+-------------------+
1 row in set (0.00 sec)
```

- **CURRENT_TIMESTAMP() | NOW()**

获取系统当前时间戳，精确到毫秒，格式为”YYYY-MM-DD HH:MM:SS.SSSSSS”。

```
mysql> select current_time(),current_timestamp(),now();
+----------------+----------------------------+----------------------------+
| current_time() | current_timestamp()        | now()                      |
```

```
+----------------+---------------------------+---------------------------+
| 13:46:36       | 2017-08-07 10:46:36.867302 | 2017-08-07 10:46:36.867302 |
+----------------+---------------------------+---------------------------+
1 row in set (0.00 sec)
```

- **STRICT_CURRENT_TIMESTAMP()**

当系统中运行多个 CG 时，由于不同 CG 可能存在微小的系统时钟误差，将导致 NOW()的运算结果不一致。STRICT_CURRENT_TIMESTAMP()是从主 TG 上获取时间，从而保证所有客户端获取的时间是一致的。

```
mysql> select strict_current_timestamp();
+---------------------------+
| strict_current_timestamp() |
+---------------------------+
| 2017-08-07 10:47:21.339091 |
+---------------------------+
1 row in set (0.01 sec)
```

- **CURRENT_DATE()**

以'YYYY-MM-DD'格式返回当前日期值。

```
mysql> select current_date();
+----------------+
| current_date() |
+----------------+
| 2017-08-07     |
+----------------+
1 row in set (0.00 sec)
```

- **CURRENT_TIME()**

以'HH:MM:SS'格式返回当前时间值。

```
mysql> select current_time();
+----------------+
| current_time() |
+----------------+
| 10:47:54       |
+----------------+
1 row in set (0.00 sec)
```

- **YEAR(datetime_value)**

返回时间参数的年份信息。输入参数类型可以为 varchar、date 和 timestamp。

```
mysql> select year('2015-10-10 12:00:00.1'),year(current_date()),year(current_timestamp());
+------------------------------+---------------------+--------------------------+
| year('2015-10-10 12:00:00.1') | year(current_date()) | year(current_timestamp()) |
+------------------------------+---------------------+--------------------------+
|                         2015 |                2017 |                     2017 |
+------------------------------+---------------------+--------------------------+
1 row in set (0.00 sec)
```

- **MONTH(datetime_value)**

返回时间参数的月份信息，输入参数类型可以为 varchar、date 和 timestamp。

```
mysql> select month('2015-10-10 12:00:00.1'),month(current_date()),month(current_timestamp());
+-------------------------------+----------------------+---------------------------+
| month('2015-10-10 12:00:00.1') | month(current_date()) | month(current_timestamp()) |
+-------------------------------+----------------------+---------------------------+
| 10                            | 8                    | 8                         |
+-------------------------------+----------------------+---------------------------+
1 row in set (0.00 sec)
```

- **DAY(datetime_value)**

返回时间参数的日信息，输入参数类型可以为日期格式的 varchar、date 和 timestamp。

```
mysql> select day('2015-10-10 12:00:00.1'),day(date '2016-07-05'),day(current_timestamp());
+-----------------------------+-----------------------+--------------------------+
| day('2015-10-10 12:00:00.1') | day(date '2016-07-05') | day(current_timestamp()) |
+-----------------------------+-----------------------+--------------------------+
|                          10 |                     5 |                        7 |
+-----------------------------+-----------------------+--------------------------+
1 row in set (0.00 sec)
```

- **HOUR(datetime_value)**

返回时间参数的小时信息（0~23），输入参数类型可以为 varchar、time 和 timestamp。

```
mysql> select hour('2015-10-10 12:00:00.1'),
hour(time '12:01:02'),hour(current_time());
+-----------------------------+----------------------+----------------------+
| hour('2015-10-10 12:00:00.1') | hour(time '12:01:02') | hour(current_time()) |
+-----------------------------+----------------------+----------------------+
|                          12 |                   12 |                   10 |
+-----------------------------+----------------------+----------------------+
```

- **MINUTE(datetime_value)**

返回时间参数的分钟信息，输入参数类型可以为 varchar、time 和 timestamp。

```
mysql> select minute('2015-10-10 12:00:00.1'),minute(time '12:01:02'),minute(current_time());
+-------------------------------+------------------------+------------------------+
| minute('2015-10-10 12:00:00.1') | minute(time '12:01:02') | minute(current_time()) |
+-------------------------------+------------------------+------------------------+
|                             0 |                      1 |                     57 |
+-------------------------------+------------------------+------------------------+
1 row in set (0.00 sec)
```

- **SECOND(param)**

返回时间参数的秒信息，输入参数类型可以为 varchar、time 和 timestamp。

```
mysql> select second('2015-10-10 12:00:00.1'),second(time '12:01:02'),second(current_time());
+---------------------------------+-------------------------+-------------------------+
| second('2015-10-10 12:00:00.1') | second(time '12:01:02') | second(current_time()) |
+---------------------------------+-------------------------+-------------------------+
|                               0 |                       2 |                       2 |
+---------------------------------+-------------------------+-------------------------+
1 row in set (0.00 sec)
```

- **DAYS(param)**

返回时间参数从'0001-01-01'起的天数，输入参数类型可以为 varchar、date 和 timestamp。

```
mysql> select days('2015-10-10 12:00:00.1'),days(date '2015-10-11'),days(current_timestamp());
+-------------------------------+------------------------+--------------------------+
| days('2015-10-10 12:00:00.1') | days(date '2015-10-11') | days(current_timestamp()) |
+-------------------------------+------------------------+--------------------------+
|                        735881 |                 735882 |                    736548|
+-------------------------------+------------------------+--------------------------+
1 row in set (0.00 sec)
```

- **DATE_ADD(date, INTERVAL expr type) | ADDDATE(date, INTERVAL expr type)**

ADDDATE()与 DATE_ADD()相同,日期加法运算,返回参数 date 与表达式 expr 相加后的结果,结果的类型格式与参数 date 一致。参数 date 可以为 DATE、TIME、TIMESTAMP 类型的数值或数据表的列；参数 expr 是结果为整数类型的表达式；参数 type 为关键字 YEARS、MONTHS、DAYS、HOURS、MINUTES、SECONDS、MICROSECONDS 中的任意一个。

参数 date 类型为 DATE 时，只能与 type 参数为 YEARS、MONTHS、DAYS 进行运算；date 类型为 TIME 时，只能与 type 参数为 HOURS、MINUTES、SECONDS 进行运算；date 类型为 TIMESTAMP 时，能与 type 参数为任意类型进行运算。

```
mysql> select date_add(date '2015-10-11',interval 1 days), date_add(time '11:10:10',interval 1
hours);
+-------------------------------------------+-----------------------------------------+
| date_add(date '2015-10-11',interval 1 days) | date_add(time '11:10:10',interval 1 hours) |
+-------------------------------------------+-----------------------------------------+
| 2015-10-12                                | 12:10:10                                |
+-------------------------------------------+---------------------------- -------------+
1 row in set (0.00 sec)
```

- **DATE_SUB(date,INTERVAL expr type) | SUBDATE(date, INTERVAL expr type)**

SUBDATE()与 DATE_SUB()相同,日期减法运算，返回参数 date 与表达式 expr 相减后的结果,结果的类型格式与参数 date 一致。参数 date 可以为 DATE、TIME、TIMESTAMP 类型的数值或数据表的列；参数 expr 是结果为整数类型的表达式；参数 type 为关键字 YEARS、MONTHS、DAYS、HOURS、MINUTES、SECONDS、MICROSECONDS 中的任意一个。

参数 date 类型为 DATE 时，只能与 type 参数为 YEARS、MONTHS、DAYS 进行运算；date 类型为 TIME 时，只能与 type 参数为 HOURS、MINUTES、SECONDS 进行运算；date 类型为 TIMESTAMP 时，能与 type 参数为任意类型进行运算。

```
mysql> select date_sub(date '2015-10-11',interval 1 days), date_sub(time '11:10:10',interval 1
hours);
+-------------------------------------------+-------------------------------------------+
| date_sub(date '2015-10-11',interval 1 days) | date_sub(time '11:10:10',interval 1 hours) |
+-------------------------------------------+-------------------------------------------+
| 2015-10-10                                | 10:10:10                                  |
+-------------------------------------------+-------------------------------------------+
1 row in set (0.00 sec)
```

4.2.2. 聚集函数

使用 OBASE 数据库的聚集函数时，仅支持一个参数表达式。例如，不支持 COUNT(c1, c2)，仅支持 COUNT(c1)。数据表 test 中的数据如下：

```
mysql> select * from testbl;
+------+------+
| id   | num  |
+------+------+
|    1 |   10 |
|    2 |   20 |
|    3 |   30 |
|    4 | NULL |
+------+------+
4 rows in set (0.00 sec)
```

- **AVG(expr)**

返回组中表达式 expr 的平均值，空值将被忽略。

```
mysql> select avg(num) from testbl;
+----------+
| avg(num) |
+----------+
|      20 |
+----------+
1 row in set (0.00 sec)
```

- **COUNT (expr)**

返回组中表达式 expr 值的计数，空值将被忽略，但 COUNT(*)不会忽略空值。

```
mysql> select count(*),count(num) from testbl;
+----------+------------+
| count(*) | count(num) |
+----------+------------+
|       4 |         3 |
+----------+------------+
1 row in set (0.00 sec)
```

- **MAX (expr)**

返回组中表达式 expr 的最大值，空值将被忽略。

```
mysql> select max(num) from testbl;
+----------+
| max(num) |
+----------+
|      30 |
```

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

- **MIN (expr)**

返回组中表达式 expr 的最小值，空值将被忽略。

```
mysql> select min(num) from testbl;
+----------+
| min(num) |
+----------+
|       10 |
+----------+
1 row in set (0.01 sec)
```

- **SUM(expr)**

返回组中表达式 expr 的和，空值将被忽略。

```
mysql> select sum(num) from testbl;
+----------+
| sum(num) |
+----------+
|       60 |
+----------+
1 row in set (0.00 sec)
```

- **ROW_NUMBER()**

对查询语句结果集中的每一条记录按分组排序指定一个行号，行号从 1 开始，常与子查询结合使用以实现分页功能，语法格式为：

```
ROW_NUMBER() OVER ([partition_clause] [order_by_clause])
```

```
mysql> select ename,deptno,row_number() over () from emp;
+--------+--------+----------------------+
| ename  | deptno | row_number() over () |
+--------+--------+----------------------+
| SMITH  |     20 |                    1 |
| ALLEN  |     30 |                    2 |
| WARD   |     30 |                    3 |
| JONES  |     20 |                    4 |
| MARTIN |     30 |                    5 |
| BLAKE  |     30 |                    6 |
| CLARK  |     10 |                    7 |
| SCOTT  |     20 |                    8 |
| KING   |     10 |                    9 |
| TURNER |     30 |                   10 |
| ADAMS  |     20 |                   11 |
| JAMES  |     30 |                   12 |
| FORD   |     20 |                   13 |
| MILLER |     10 |                   14 |
+--------+--------+----------------------+
14 rows in set (0.00 sec)
```

```
mysql> select ename,deptno,row_number() over (partition by deptno order by sal desc) from emp;
+--------+--------+-------------------------------------------------------------+
| ename  | deptno | row_number() over (partition by deptno order by sal desc) |
+--------+--------+-------------------------------------------------------------+
| KING   |     10 |                                                           1 |
| CLARK  |     10 |                                                           2 |
| MILLER |     10 |                                                           3 |
| SCOTT  |     20 |                                                           1 |
| FORD   |     20 |                                                           2 |
| JONES  |     20 |                                                           3 |
| ADAMS  |     20 |                                                           4 |
| SMITH  |     20 |                                                           5 |
| BLAKE  |     30 |                                                           1 |
| ALLEN  |     30 |                                                           2 |
| TURNER |     30 |                                                           3 |
| WARD   |     30 |                                                           4 |
| MARTIN |     30 |                                                           5 |
| JAMES  |     30 |                                                           6 |
+--------+--------+-------------------------------------------------------------+
14 rows in set (0.01 sec)
```

4.3. 运算符

OBASE 数据库支持四种类型的运算符，包括算数运算符、比较运算符、逻辑运算符和拼接运算符。

表 4-2：OBASE 数据库中的运算符

类型	运算符	含义
算数运算符	+, -, *	加减乘;
	/	除法，返回商，如果除数为 0 则报错;
	%或 mod	除法，返回余数，如果除数为 0 则报错;
比较运算符	=，>=，>，<=，<	等于，大于等于，大于，小于等于，小于;
	!=或<>	不等于;
	between … and …	存在于指定范围;
	in	存在于指定集合，最多允许指定 5120 个值;
	is null	为 null;
	is not null	不为 null;
	is true/false/unknown	为真/为假/为未知;

类型	运算符	含义
	is not true/false/unknown	不为真/不为假/不为未知;
	like	字符串通配符匹配，通配符“%”表示匹配任意长度字符串，包括空字符串； 通配符“_”表示匹配单个字符，不包括空字符;
逻辑运算符	not	逻辑非;
	and	逻辑与;
	or	逻辑或;
拼接运算符	||	将值拼接在一起构成单个值，可实现字符串拼接;

当进行混合运算时，需要了解运算符的优先级。表 4-3 从高到低列出了各种运算符的优先级。在实际运用时，建议用“( )”将需要优先计算的操作括起，这样既起到优先计算的作用，又便于理解。

表 4-3：OBASE 数据库中运算符的优先级

优先级	运算符
1	*，/，%，mod
2	+，-
3	=，>，>=，<，<=，<>，!=，IS，LIKE，IN
4	between … and
5	not
6	and
7	or

4.4. 转义字符

转义字符是在特定字符前加反斜线'\'，用于表示特殊的字符。OBASE 数据库中常用的转义字符如表 4-4 所示

表 4-4：OBASE 数据库中的转义字符

转义字符	含义
\b	退格符
\f	换页符
\n	换行符
\r	回车符
\t	Tab 字符
\\	反斜线字符
\'	单引号
\"	双引号
_	_字符
\%	%字符
\0	空字符(NULL)

5. SQL 语法

5.1. 数据定义语句

5.1.1. CREATE DATABASE

创建新的数据库，语法格式为：

```
CREATE DATABASE db_name;
```

说明：

- 数据库名长度不能超过 15 个字符，包括字母、数字、下划线，首字符可以是字母或下划线；

```
mysql> create database sales;
Query OK, 0 rows affected (0.12 sec)

mysql> show databases;
+----------+
| Database |
+----------+
| TANG     |
| sales    |
+----------+
2 rows in set (0.00 sec)
```

5.1.2. DROP DATABASE

删除数据库，语法格式为：

DROP DATABASE **db_name**;

说明：

- 当 db_name 所指定的数据库不存在时，语句会报错；
- 删除数据库之前需要先删除库中全部的数据表，否则无法删除数据库；
- 删除数据库可能会导致会话的默认数据库为空值，此时需要为会话重新指定默认数据库，否则该会话无法在默认数据库下创建表；

```
mysql> use sales;
Database changed
mysql> show tables;
Empty set (0.00 sec)

mysql> drop database sales;
Query OK, 1 row affected (0.09 sec)

mysql> select database();
+------------+
| database() |
+------------+
|            |
+------------+
1 row in set (0.00 sec)

mysql> use test;
Database changed
mysql> select database();
+------------+
| database() |
+------------+
| test       |
+------------+
1 row in set (0.00 sec)
```

5.1.3. CREATE TABLE

创建新的数据表，语法格式为：

CREATE TABLE [IF NOT EXISTS] **tbl_name** (

col_name data_type [NOT NULL | NULL] [DEFAULT **default_value**] [AUTO_INCREMENT],

…… ,
PRIMARY KEY (**col_name1, col_name2**…)) [**table_options_list**];

说明:

- 建表时必须指定主键，但不支持全主键的表;
- 表名长度限 255 个字符，包括字母、数字、下划线，首字符必须是字母;
- 列名长度限 127 个字符，包括字母、数字、下划线，首字符可以是字母或下划线;
- 每张表的列数限制为最大 496 列;
- 如果指定 IF NOT EXISTS，即使创建的表已经存在，也不报错;不指定时，若创建的表已经存在则报错;
- OBASE 数据库在执行合并操作时不允许创建表;

table_options_list 用于指定创建表时的一些可选参数，详见表 5-1。

表 5-1：OBASE 数据库的建表参数

参数	含义	示例
EXPIRE_INFO	数据合并时删除满足条件的数据，常用于自动删除过期数据;数据表创建索引后，不能再修改过期删除条件;	EXPIRE_INFO = '$SYS_DATE' > c1 + 24 * 60 * 60 * 100000 自动删除 c1 值比当前时间小 24 小时的数据，EXPIRE_INFO 取值单位:微秒。
TABLET_MAX_SIZE	设置 tablet 大小的上限，单位为字节，默认为 64M;	TABLET_MAX_SIZE=268435456 设置 tablet 大小为 256M
REPLICA_NUM	设置 tablet 的副本数，默认为 3，最大不超过 6;	REPLICA_NUM=3
COMPRESS_METHOD	存储数据时使用的压缩方法 1. none(默认值，不压缩) 2. lzo_1.0 3. snappy_1.0	COMPRESS_METHOD = 'none'

```
mysql> create table customer(
    -> id int not null,
    -> name varchar(10) not null ,
    -> address varchar(50),
    -> last_login datetime,
    -> remmark varchar(1024) default 'N/A',
    -> primary key(id)
    -> )compress_method='snappy_1.0',
    -> replica_num=4,
    -> expire_info='$SYS_DATE > last_login+38*24*60*60*1000000',
```

```
    -> tablet_max_size=134217728;
Query OK, 0 rows affected (0.93 sec)

mysql> desc customer;
+------------+---------------+-----------+------+------+---------+-------+
| Field      | Type          | Collation | Null | Key  | Default | Extra |
+------------+---------------+-----------+------+------+---------+-------+
| id         | int32         | NULL      |    0 |    1 | NULL    |       |
| name       | varchar(10)   | NULL      |    0 |    0 | NULL    |       |
| address    | varchar(50)   | NULL      |    1 |    0 | NULL    |       |
| last_login | timestamp     | NULL      |    1 |    0 | NULL    |       |
| remmark    | varchar(1024) | NULL      |    1 |    0 | N/A     |       |
+------------+---------------+-----------+------+------+---------+-------+
5 rows in set (0.00 sec)
```

5.1.4. ALTER TABLE

修改已存在表的设计，支持增加列、删除列、表重命名和列重命名。

增加列： ALTER TABLE **tbl_name** ADD [COLUMN] **col_name data_type**;

```
mysql> alter table customer add phone varchar(11);
Query OK, 0 rows affected (0.28 sec)

mysql> desc customer;
+------------+---------------+-----------+------+------+---------+-------+
| Field      | Type          | Collation | Null | Key  | Default | Extra |
+------------+---------------+-----------+------+------+---------+-------+
| id         | int32         | NULL      |    0 |    1 | NULL    |       |
| name       | varchar(10)   | NULL      |    0 |    0 | NULL    |       |
| address    | varchar(50)   | NULL      |    1 |    0 | NULL    |       |
| last_login | timestamp     | NULL      |    1 |    0 | NULL    |       |
| remmark    | varchar(1024) | NULL      |    1 |    0 | N/A     |       |
| phone      | varchar(11)   | NULL      |    1 |    0 | NULL    |       |
+------------+---------------+-----------+------+------+---------+-------+

6 rows in set (0.00 sec)
```

删除列： ALTER TABLE **tbl_name** DROP [COLUMN] **col_name**;

```
mysql> alter table customer drop phone;
Query OK, 0 rows affected (0.33 sec)

mysql> desc customer;
+------------+---------------+-----------+------+------+---------+-------+
| Field      | Type          | Collation | Null | Key  | Default | Extra |
+------------+---------------+-----------+------+------+---------+-------+
| id         | int32         | NULL      |    0 |    1 | NULL    |       |
| name       | varchar(10)   | NULL      |    0 |    0 | NULL    |       |
| address    | varchar(50)   | NULL      |    1 |    0 | NULL    |       |
| last_login | timestamp     | NULL      |    1 |    0 | NULL    |       |
| remmark    | varchar(1024) | NULL      |    1 |    0 | N/A     |       |
+------------+---------------+-----------+------+------+---------+-------+
5 rows in set (0.00 sec)
```

表重命名：

ALTER TABLE **tbl_name** RENAME TO **new_tbl_name**;

RENAME TABLE **tbl_name** TO **new_tbl_name**;

```
mysql> rename table customer to customer_bak;

Query OK, 0 rows affected (0.27 sec)
```

```
mysql> alter table customer_bak rename to customer;

Query OK, 0 rows affected (0.25 sec)
```

列重命名：

ALTER TABLE **tbl_name** RENAME [COLUMN] **col_name** TO **new_col_name**;

```
mysql> alter table customer RENAME last_login TO login;
Query OK, 0 rows affected (0.27 sec)
```

说明：

- 若在列上已经创建索引，则该列不能被删除；
- 数据表的列名修改后，索引中相应的列名不会做修改；
- OBASE 数据库执行合并操作时不允许修改表结构；

5.1.5. DROP TABLE

删除 OBASE 数据库中的表及其索引，语法格式为：

DROP TABLE [IF EXISTS] **tbl_name1** [, **tbl_name2**];

说明：

- 指定 IF EXISTS 选项时，即使要删除的表不存在也不报错；不指定时，若要删除的表不存在则报错；
- 需要同时删除多个表时，多个表名用逗号分隔；
- OBASE 数据库执行合并操作时不允许删除表；

```
mysql> drop table customer,testbl;
Query OK, 0 rows affected (1.68 sec)
```

5.1.6. CREATE VIEW

创建视图，语法格式为：

CREATE [OR REPLACE] **VIEW** relation_factor [opt_view_expr_list] AS select_stmt

注：创建 view 时，以上选项均为可选项，对于用户未输入的参数均采取默认值选项，具体的默认选项见选项含义。

OPT_VIEW_EXPR_LIST

此选项无默认值，该选项指定视图列名，其中列名个数应与 `select_stmt` 中查询

结果集的列数相同。若该选项未输入参数，视图列名同 `select_stmt` 中查询结果集的列名。

- 不支持视图的更新功能

5.1.7. CREATE INDEX

为数据表创建索引，语法格式为：

CREATE INDEX **idx_name** FROM **tbl_name**(**col_list1**) [STORING (**col_list2**)];

说明：

- idx_name 为索引名，系统会在用户指定的索引名前添加前缀‘__xxxx__idx__’，其中 xxxx 为索引的表 ID；
- col_list1 为索引列，其列数与源表主键列数之和不能超过 16，col_list2 为冗余列；
- 一张数据表最多可以创建 10 个索引；
- 一张数据表所有索引的列数之和不能超过 100；
- OBASE 数据库执行合并操作时不允许创建索引；
- 新创建的索引需要等待一次数据合并后才可以生效使用；
- 索引的状态
  - ✧ NOT_AVALIBALE：索引初始化完成，正在做备份迁移；
  - ✧ AVALIBALE：索引状态正常，可以正常使用；
  - ✧ ERROR：索引状态不正常，不能使用；
  - ✧ WRITE_ONLY：索引正在被删除；
  - ✧ INDEX_INIT：索引创建成功，等待数据合并，不能使用；

```
mysql> create index last_login on customer(last_login) storing (name);
Query OK, 0 rows affected (0.41 sec)

mysql> show index from customer;
+----------+------------+-------------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table    | Non_unique | Key_name                | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+----------+------------+-------------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| customer |          0 | PRIMARY                 |            1 | id          | A         |           0 |     NULL | NULL   |      | BTREE      |         |               |
| customer |          1 | __3045__idx__last_login |            1 | last_login  | A         |
```

```
0 |     NULL | NULL   |      | BTREE      |         | INDEX_INIT |
| customer |          1 | __3045__idx__last_login |            2 | id           | A          |
0 |     NULL | NULL   |      | BTREE      |         | INDEX_INIT |
+----------+------------+-------------------------+--------------+------------+-----------+---
----------+----------+--------+------+------------+---------+---------------+
3 rows in set (0.00 sec)
```

5.1.8. DROP INDEX

删除数据表的索引，语法格式为：

DROP INDEX **idx1** [, **idx2**] ON **tbl_name**;

说明：

- OBASE 数据库执行合并操作时不允许删除索引；

```
mysql> drop index last_login on customer;
Query OK, 0 rows affected (1.10 sec)

mysql> show index from customer;
+----------+------------+----------+--------------+-------------+-----------+-------------+----
------+--------+------+------------+---------+---------------+
| Table    | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality |
Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+----------+------------+----------+--------------+-------------+-----------+-------------+----
------+--------+------+------------+---------+---------------+
| customer |          0 | PRIMARY  |            1 | id          | A         |           0 |     NULL
| NULL   |      | BTREE      |         |               |
+----------+------------+----------+--------------+-------------+-----------+-------------+----
------+--------+------+------------+---------+---------------+
1 row in set (0.00 sec)
```

5.1.9. TRUNCATE TABLE

清空数据表中的数据，语法格式为：

TRUNCATE [TABLE] [IF EXISTS] **tbl_name1** [, **tbl_name2**, ...];

说明：

- 允许同时 truncate 多张表，执行成功返回“0 rows affected”，执行失败返回错误码；
- OBASE 数据库执行合并操作时不允许 truncate 数据表；

```
mysql> select * from customer;
+------+------+---------+------------+---------+
| id   | name | address | last_login | remmark |
+------+------+---------+------------+---------+
|   10 | tom  | NULL    |       NULL | N/A     |
|   11 | tom  | NULL    |       NULL | N/A     |
+------+------+---------+------------+---------+
2 rows in set (0.00 sec)
```

```
mysql> truncate table customer;
Query OK, 0 rows affected (0.02 sec)

mysql> select * from customer;
Empty set (0.00 sec)
```

5.1.10. CREATE SEQUENCE

创建序列，语法格式为：

```
CREATE [OR REPLACE] SEQUENCE seq_name
    [AS data_type / AS INTEGER]
    [START WITH num_constant / START WITH 1]
    [INCREMENT BY num_constant / INCREMENT BY 1]
    [MINVALUE num_constant / NO MINVALUE]
    [MAXVALUE num_constant / NO MAXVALUE]
    [CYCLE / NO CYCLE]
    [QUICK / NO QUICK]
```

选项含义

- OR REPLACE

若新建序列的 seq_name 已经存在，则对该序列进行重新定义；

- AS data_type

指定序列的数据类型，可以为 INTEGER 或 DECIMAL。其中 INTEGER 为 64 位整型，DECIMAL 的范围（precision）为 1~31，精度（scale）必须为 0，默认为 INTEGER；

- START WITH

设定序列的起始值，正数、负数和零均可，但必须为整数。设定的起始值可以不在最大值和最小值所限定的范围之内。若创建序列时没有指定最大值或最小值，则对于递增序列来说，其最小值等于起始值；对于递减序列来说，其最大值等于起始值；默认值为 1；

- INCREMENT BY

设定序列每次自增或自减的步长。步长为正数时是递增序列，步长为负数时是递减序列，步长为 0 时是常数序列，默认值为 1；

- MINVALUE 或 NO MINVALUE

设定序列的最小值。若创建序列时未指定 MINVALUE 或设定为 NO

MINVALUE，则采用默认最小值，递增序列的默认最小值为 START WITH 选项所设定的值，递减序列的默认最小值为数据类型所能表示的最小值；

- MAXVALUE 或 NO MAXVALUE

设定序列的最大值。若创建序列时未指定 MAXVALUE，或设定为 NO MAXVALUE，则采用默认最大值，递增序列的默认最大值为数据类型所能表示的最大值，递减序列的默认最大值为 START WITH 选项所设定的值；

- CYCLE 或 NO CYCLE

设定序列到达边界（即 MINVALUE 与 MAXVALUE 所确定的范围）后是否重新开始循环，默认为 NO CYCLE，表示当序列到达边界后无法再继续生成新的值；

- QUICK 或 NO QUICK

设定会话使用序列的方式，独占（NO QUICK）或非独占（QUICK），默认为 NO QUICK。在 NO QUICK 方式下，同一时刻只能有一个会话独占使用该序列，其它会话处于等待状态；在 QUICK 方式下，所有会话可以同时使用该序列，但不同会话获取的序列值可能出现重复情况；QUICK 方式能够提供更好的性能；

说明：

- 仅支持 NEXTVAL FOR seq_name 和 PREVVAL FOR seq_name 方式使用；
- 在对用户授予 SEQUENCE 功能权限时，该用户即可对所有序列进行修改与删除等操作；
- 如果创建序列时未指定 QUICK 选项，调用 NEXTVAL 后就代表序列生成的值已经被消费，不会再次生成该值（CYCLE 与 RESTART 场景除外）；如果创建序列时指定 QUICK 选项，在 insert 或 select 语句中调用 NEXTVAL 所获取的值，在该语句执行失败的场景下不会被计入序列，即再次调用 NEXTVAL 时仍然会生成该值；
  - ✧ 例如：未指定 QUICK 选项时，使用序列的 NEXTVAL 作为主键值向数据表中插入记录，如果该值已经作为主键存在，则插入语句失败，但序列认为该值已经被使用，再次执行这条插入语句，序列会生成新的 NEXTVAL 值；在指定 QUICK 选项时，只有插入语句成功后，序列才认为 NEXTVAL 的值被使用。
- 目前 OBASE 数据库的序列不支持 ORDER 选项和 CACHE 选项；
- 目前 OBASE 数据库的序列是面向所有会话有效，不支持面向单个会话的序列；

- 不支持在 CASE 表达式中使用序列；
- 不支持在聚集函数的参数列表中使用序列；
- 不支持在 JOIN 操作的连接条件中使用序列；
- 不支持在 SELECT 语句的 ORDER BY 子句中使用序列；
- 使用序列时，查询 NEXTVAL 的操作必须在查询 PREVVAL 之前；

```
mysql> create or replace sequence seq1 as integer start with 1 increment by 1 minvalue -1 maxvalue
2 cycle;

Query OK, 1 row affected (0.01 sec)


mysql> select prevval for seq1, nextval for seq1;

ERROR 65535 (HY000): OB--1: THE PREVVAL expression of  sequence [seq1] can't be used before using
NEXTVAL

mysql> select  nextval for seq1;

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

| nextval for seq1 |

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

| 2              |

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

1 row in set (0.03 sec)
```

5.1.11. ALTER SEQUENCE

修改序列，语法格式为：

```
ALTER SEQUENCE seq_name
    [RESTART / RESTART WITH num_constant]    [INCREMENT BY num_constant]
    [MINVALUE num_constant / NO MINVALUE]
    [MAXVALUE num_constant / NO MAXVALUE]
    [CYCLE / NO CYCLE]    [QUICK / NO QUICK]
```

选项含义：

- RESTART

重新设定序列生成新值的开始位置，若指定 RESTART 选项，则从创建序列时 START WITH 选项设置的值开始；若指定 RESTART WITH num_constant 选项，则从 num_constant 开始；

- 其它选项

其它选项的含义与创建序列语句的选项含义保持一致。在修改序列时，未指定的选项保持不变，与创建时的设定一致。

说明：

- 在指定 RESTART WITH num_constant 选项时，将会重置序列的当前值，而不会修改创建序列时指定的 START WITH 的值；如需改变 START WITH 的设定值，只能通过 CREATE OR REPLACE SEQUENCE 命令进行修改。

5.1.12. DROP SEQUENCE

删除序列，语法格式为：

```
DROP SEQUENCE seq_name RESTRICT
```

选项含义：

- RESTRICT

检查是否有其它数据库对象依赖于待删除的序列，若有则禁止删除。未指定该选项则可以直接删除序列。

```
mysql> drop sequence seq1 ;
Query OK, 1 row affected (0.02 sec)

mysql> select  nextval for seq1;
ERROR 65535 (HY000): OB--1: the sequence::[seq1] you input is not exist,please check!
```

5.2. 数据操作语句

5.2.1. INSERT

向数据表中添加一条或多条数据记录，新插入的数据记录必须包含所有主键列的值。语法格式为：

- 插入一条或多条记录

```
INSERT INTO tbl_name[(col_name, ...)] VALUES (col_value, ...)[,( col_value, ...), ···] ;
```

- 插入子查询的结果集（子查询结果集小于 2M）

```
INSERT INTO tbl_a[(a1, a2, ..., am)] SELECT b1, b2, ..., bm FROM tbl_b WHERE ···;
```

- 分批次插入子查询的结果集（不能保证语句执行的原子性）

```
INSERT INTO tbl_a[(a1, a2, ..., am)]
    SELECT /*+I_MULTI_BATCH*/ b1, b2,···, bm FROM table_b WHERE ··· ;
```

- 在插入语句中引用序列的值

```
INSERT INTO tbl_name(col1, ...) VALUES(NEXTVAL/PREVVAL FOR seq_name, ...)
```

说明：

- 在插入语句中使用子查询时，子查询的输出列数应当等于插入语句所指定的待插入列数，并且对应列的数据类型必须兼容，对应于主键列的值不能为空；
- 目前 OBASE 数据库存在数据包大小限制，在 CG 和 TG 之间传输的单个数据包大小上限是 2M，每个事务所产生的事务日志大小上限也是 2M。这个限制决定了执行批量插入时，如果插入数据量超过 2M，则需要使用多批次插入策略，每个批次作为一个事务执行。这种策略会把一条插入语句拆分成多个事务执行，不保证插入语句执行的原子性；因此 OBASE 数据库提供了两种 INSERT … SELECT … 语句模式，一种模式可以保证语句执行的原子性，但只允许插入 2M 以内的数据量，如果超出则报错；另一种模式通过 hint /*+I_MULTI_BATCH*/指定采用多批次插入策略，允许插入超过 2M 的数据量，但不保证语句执行的原子性；如果某一批次插入不成功，则这一批次前面的数据都会插入成功，并会返回成功插入了几批数据以及一共插入多少条数据；
- 在插入语句中可以使用序列产生的值。需要注意的是，在一条插入语句中同一个序列的所有的 PREVVAL 返回的都是同一个值。

```
mysql> create or replace sequence cust_id as integer start with 1 increment by 1;
Query OK, 1 row affected (0.02 sec)
mysql> insert into customer values(nextval for cust_id,'Jerry','Disney Avennue',now());
Query OK, 1 row affected (0.08 sec)
mysql> insert into customer values(nextval for cust_id,'Jerry','Disney Avennue',now()),
    -> (nextval for cust_id,'Alice','Disney Avennue',now()),
    -> (nextval for cust_id,'Bob','Disney Avennue',now());
Query OK, 3 rows affected (0.09 sec)
mysql> select * from customer;
+------+-------+----------------+----------------------------+
| id   | name  | address        | last_login                 |
+------+-------+----------------+----------------------------+
|    1 | Tom   | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    2 | Jerry | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    3 | Alice | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    4 | Bob   | Disney Avennue | 2018-05-13 13:29:42.200672 |
+------+-------+----------------+----------------------------+
4 rows in set (0.01 sec)
```

5.2.2. REPLACE

向数据表中添加一条数据记录，与 INSERT 语句的区别在于，如果数据记录的主键已经存在于数据表中，则修改对应列的值为新值，如不存在则插入该条数据记录。语法格式为：

REPLACE INTO **tbl_name** [(**col_name**, …)] VALUES (**col_value,** …);

说明：

- 目前 REPLACE 语句中不支持使用序列；
- 与 INSERT 语句不同，REPLACE 语句在执行时直接将数据从 CG 发送给

TG，不会向 DG 请求数据来判断是否存在主键冲突，因此 REPLACE 语句的性能要优于 INSERT 语句；

```
mysql> select * from customer;
+------+-------+----------------+----------------------------+
| id   | name  | address        | last_login                 |
+------+-------+----------------+----------------------------+
|    1 | Tom   | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    2 | Jerry | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    3 | Alice | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    4 | Bob   | Disney Avennue | 2018-05-13 13:29:42.200672 |
+------+-------+----------------+----------------------------+
4 rows in set (0.01 sec)

mysql> select * from customer;
+------+-------+----------------+----------------------------+
| id   | name  | address        | last_login                 |
+------+-------+----------------+----------------------------+
|    1 | Micky | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    2 | Jerry | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    3 | Alice | Disney Avennue | 2018-05-13 13:29:42.200672 |
|    4 | Bob   | Disney Avennue | 2018-05-13 13:29:42.200672 |
+------+-------+----------------+----------------------------+
4 rows in set (0.00 sec)
```

5.2.3. UPDATE

修改数据表中符合条件数据记录的列值，语法格式为：

UPDATE [/*+UD_MULTI_BATCH*/] **tbl_name**
SET **col1** = **col_value1** [**col2**=**col_value2**] WHERE **where_condition**;

说明：

- 通过 hint /*+UD_MULTI_BATCH*/可以指定采用多批次更新策略，允许更新超过 2M 的数据量，但不保证语句执行的原子性；更新大量数据时，需要根据机器性能调整超时时间，避免语句因执行超时而失败；
- OBASE 数据库只支持对单表进行更新操作，在 where_condition 中可以使用序列的 PREVVAL 值，但不能使用序列的 NEXTVAL 值；

```
mysql> update customer set address='New York',last_login=now() where id=2;
Query OK, 1 row affected (0.07 sec)
```

5.2.4. DELETE

删除数据表中符合条件的数据记录，语法格式为：

DELETE [/*+UD_MULTI_BATCH*/] FROM **tbl_name** [WHERE **where_condition**];

说明：

- 通过 hint /*+UD_MULTI_BATCH*/可以指定采用多批次删除策略，允许删除超过 2M 的数据量，但不保证语句执行的原子性；删除大量数据时，需

要根据机器性能调整超时时间，避免语句因执行超时而失败；

- OBASE 数据库只支持对单表进行删除操作，在 where_condition 中可以使用序列的 PREVVAL 值，但不能使用序列的 NEXTVAL 值；

5.2.5. SELECT

查询数据表中的内容，语法格式为：

```
SELECT  [ ALL | DISTINCT ]  select_list [AS other_name]
FROM  table_name
[WHERE where_conditions ]
[GROUP BY group_by_list ]
[HAVING search_confitions ]
[ORDER  BY  order_list [ASC | DESC ] ]
[LIMIT {[offset,] row_count | row_count OFFSET offset}];
```

说明：

- 目前 OBase 不支持在 SELECT 查询语句的 where 子句中使用列别名，但是支持在 GROUP BY ,ORDER BY，HAVING 子句中使用列别名
- SELECT 查询语句的 from 子句中出现多张表或子查询结果集时，每张表（或子查询结果集）都至少根据一个等值连接条件与其他表（或子查询结果集）连接。
- LIMIT 子句强制 SELECT 语句返回指定的记录数，参数 row_count 和 offset 必须是整数常量，row_count 指定返回的记录数，offset 指定返回记录的偏移量，初始记录的偏移量是 0（不是 1）；

```
mysql> select id,name from customer order by id desc;
+------+-------+
| id   | name  |
+------+-------+
|    4 | Bob   |
|    3 | Alice |
|    2 | Jerry |
|    1 | Micky |
+------+-------+
4 rows in set (0.01 sec)

mysql> select id,name from customer limit 2;
+------+-------+
| id   | name  |
+------+-------+
|    1 | Micky |
|    2 | Jerry |
+------+-------+
2 rows in set (0.01 sec)
```

```
mysql> select id,name from customer limit 1,2;
+------+-------+
| id   | name  |
+------+-------+
|    2 | Jerry |
|    3 | Alice |
+------+-------+
2 rows in set (0.00 sec)
```

5.2.6. JOIN

OBASE 数据库支持在查询语句中显式指定关系表连接操作的方式，包括内连接、左外连接、右外连接和全外连接。语法格式为：

SELECT **expr_list** FROM **tbl1** INNER JOIN **tbl2** ON **join_condition**;

```
mysql> select * from customer c inner join emp e on c.id = e.empno;
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
| id   | name   | address  | last_login | empno | ename  | job      | mgr  | hiredate   | sal
| comm    | deptno |
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
| 7369 | SMITH  | New York |       NULL |  7369 | SMITH  | CLERK    | 7902 | 1980-12-17 |  800.00
|    NULL |     20 |
| 7654 | MARTIN | NULL     |       NULL |  7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00
| 1400.00 |     30 |
| 7788 | SCOTT  | New York |       NULL |  7788 | SCOTT  | ANALYST  | 7566 | 1982-12-09 | 3000.00
|    NULL |     20 |
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
3 rows in set (0.00 sec)

SELECT expr_list FROM tbl1 LEFT OUTER JOIN tbl2 ON join_condition;
mysql> select * from customer c left outer join emp e on c.id = e.empno;
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
| id   | name   | address  | last_login | empno | ename  | job      | mgr  | hiredate   | sal
| comm    | deptno |
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
| 7369 | SMITH  | New York |       NULL |  7369 | SMITH  | CLERK    | 7902 | 1980-12-17 |  800.00
|    NULL |     20 |
| 7654 | MARTIN | NULL     |       NULL |  7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00
| 1400.00 |     30 |
| 7788 | SCOTT  | New York |       NULL |  7788 | SCOTT  | ANALYST  | 7566 | 1982-12-09 | 3000.00
|    NULL |     20 |
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
3 rows in set (0.00 sec)

SELECT expr_list FROM tbl1 RIGHT OUTER JOIN tbl2 ON join_condition;
mysql> select * from customer c left outer join emp e on c.id = e.empno;
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
| id   | name   | address  | last_login | empno | ename  | job      | mgr  | hiredate   | sal
| comm    | deptno |
+------+--------+----------+------------+-------+--------+----------+------+------------+------
---+---------+--------+
| 7369 | SMITH  | New York |       NULL |  7369 | SMITH  | CLERK    | 7902 | 1980-12-17 |  800.00
```

```
|    NULL |     20 |

| 7654 | MARTIN | NULL     |       NULL |  7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00
| 1400.00 |     30 |

| 7788 | SCOTT  | New York |       NULL |  7788 | SCOTT  | ANALYST  | 7566 | 1982-12-09 | 3000.00
|    NULL |     20 |

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

3 rows in set (0.00 sec)


SELECT expr_list FROM tbl1 FULL OUTER JOIN tbl2 ON join_condition;


mysql> select * from customer c left outer join emp e on c.id = e.empno;

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

| id   | name   | address  | last_login | empno | ename  | job      | mgr  | hiredate   | sal
| comm    | deptno |

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

| 7369 | SMITH  | New York |       NULL |  7369 | SMITH  | CLERK    | 7902 | 1980-12-17 |  800.00
|    NULL |     20 |

| 7654 | MARTIN | NULL     |       NULL |  7654 | MARTIN | SALESMAN | 7698 | 1981-09-28 | 1250.00
| 1400.00 |     30 |

| 7788 | SCOTT  | New York |       NULL |  7788 | SCOTT  | ANALYST  | 7566 | 1982-12-09 | 3000.00
|    NULL |     20 |

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

3 rows in set (0.00 sec)
```

说明:

- 在连接条件 join_condition 中，必须包含一个涉及两表的等值连接条件，并且左表应出现在等值表达式的左侧，如 tbl1.col1=tbl2.col2;

5.2.7. UNION

对两条 SELECT 语句的执行结果进行集合并运算。语法格式为:

select_statement1 UNION [ALL] **select_statement2**;

说明:

- 两条 SELECT 语句的执行结果应当具有相同的列数，并且对应列的数据类型应当相互兼容;
- UNION 操作缺省将会去除结果集中重复记录，通过使用 UNION ALL 可以保留重复记录;

5.2.8. INTERSECT

对两条 SELECT 语句的执行结果进行集合交运算。语法格式为:

select_statement1 INTERSECT [ALL] **select_statement2**;

说明：

- 两条 SELECT 语句的执行结果应当具有相同的列数，并且对应列的数据类型应当相互兼容；
- INTERSECT 操作缺省将会去除结果集中重复记录，通过使用 INTERSECT ALL 可以保留重复记录；

5.2.9. EXCEPT

对两条 SELECT 语句的执行结果进行集合差运算。语法格式为：

select_statement1 EXCEPT [ALL] **select_statement2**;

说明：

- 两条 SELECT 语句的执行结果应当具有相同的列数，并且对应列的数据类型应当相互兼容；
- EXCEPT 操作缺省将会去除结果集中重复记录，通过使用 EXCEPT ALL 可以保留重复记录；

5.2.10. SELECT … FOR UPDATE

加排它锁读取数据表中的数据记录，语法格式为：

select_statement1 FOR UPDATE;

说明：

- 在事务中使用加锁读时，所读取的数据记录不能被其它事务修改，直到本事务结束；
- 加锁读只能针对单表查询，并且查询语句的结果集只能包含一条记录；

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customer where id = 7369 for update;
+------+-------+----------+------------+
| id   | name  | address  | last_login |
+------+-------+----------+------------+
| 7369 | SMITH | New York |       NULL |
+------+-------+----------+------------+
1 row in set (0.00 sec)
```

5.2.11. PREPARE

预备执行语句，语法格式为：

PREPARE **stmt_name** FROM **prepare_statement**;

说明:

- prepare_statement 是预备执行的数据操作语句，在整个会话期间都可以 stmt_name 来执行预备好的语句;
- 在预备执行的数据操作语句中可以使用占位符'?'来标识执行时需要绑定的参数;

5.2.12. EXECUTE

执行已经预备好的数据操作语句，语法格式为:

EXECUTE **stmt_name** [USING **@var_name1** [,**@var_name2**,…]];

说明:

- 如果预备好的语句中有占位符标识的参数，需要使用 USING 子句提供相同个数的执行时绑定值，所绑定的值必须为 SET 语句定义的用户变量;

5.2.13. DEALLOCATE

删除预备好的数据操作语句，语法格式为:

{DEALLOCATE | DROP} PREPARE **stmt_name**;

```
mysql> prepare stmt1 from select id,name from customer where id=?;
Query OK, 0 rows affected (0.00 sec)

mysql> set @id=7369;
Query OK, 0 rows affected (0.00 sec)

mysql> execute stmt1 using @id;
+------+-------+
| id   | name  |
+------+-------+
| 7369 | SMITH |
+------+-------+
1 row in set (0.00 sec)

mysql> deallocate prepare stmt1;
Query OK, 0 rows affected (0.00 sec)
```

6. 事务处理

OBASE 数据库系统变量 autocommit 的值决定了每个会话中事务提交的方式，默认值为 1，表示每条语句执行完自动提交；当将 autocommit 的值设为 0 时，则需要对事务进行显式提交。

OBASE 数据库通过系统变量控制查询和事务的超时时间，事务在超时时间范围内没有提交，则会被系统自动回滚。如果需要执行一个长时间的事务，可以通过修改当前会话系统变量 ob_tx_timeout 的值来延长超时时间范围。修改系统变

量的方法参见 7.3 和 7.4 节内容。

6.1. START TRANSACTION

开启事务，别名为 BEGIN，语法格式为：

```
START TRANSACTION;
```

```
BEGIN;
```

说明：

- 通过 START TRANSACTION 或 BEGIN 语句开启的事务，必须进行显式提交；

6.2. COMMIT

显式提交当前事务，语法格式为：

```
COMMIT;
```

6.3. ROLLBACK

显式回滚当前事务，语法格式为：`ROLLBACK;`

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into customer(id,name) values(12,'tom');
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.02 sec)
```

7. 数据库管理

OBASE 通过数据库(database)来管理系统中的所有用户数据表，类似于 MySQL 中的 database 或者 DB2 中的 schema 对象。不同数据库下管理的数据表相对独立，允许不同数据库下的数据表重名。在 OBASE 数据库系统初次启动并初始化时会创建一个名为 TANG 的默认数据库。用户在连接 OBASE 系统时如果没有指定所使用的数据库名，则系统默认用户使用 TANG 数据库。

在 OBASE 数据库系统中使用数据表时，如果不指定数据表所属的数据库，则认为使用的是当前会话所连接数据库中的数据表。可以通过在数据表名前添加数据库名前缀的方式显式指定所使用的数据库，如“db_name.tbl_name”。

OBASE 将系统中的库表定义、配置参数和环境变量等系统元数据以数据表的形式进行存储，这些数据表被称为系统表。与用户创建的数据表不同，系统表不属于任何一个数据库(包括默认库 TANG)，所有的系统表名都以双下划线开头，用户可以通过查询系统表获取系统的各种元数据信息。

7.1. 查询元数据

7.1.1. 查看数据库

查看系统中所有的数据库，语法格式为：

SHOW DATABASES;

```
mysql> show databases;
+----------+
| Database |
+----------+
| TANG     |
| test     |
| testdb   |
+----------+
3 rows in set (0.00 sec)
```

7.1.2. 查看当前会话默认数据库

查看当前会话的默认数据库，语法格式为：SHOW CURRENT DATABASE;

```
mysql> SHOW CURRENT DATABASE;
+----------+
| Database |
+----------+
| test     |
+----------+
1 row in set (0.00 sec)
```

7.1.3. 设置当前会话默认数据库

指定当前会话所使用的默认数据库，语法格式为：

USE DATABASE **db_name**;

说明：

- 指定了当前会话所使用的默认数据库后，在使用数据表时如果不加库名前缀，则认为是使用当前默认库下的数据表；
- 可以通过连续使用 USING DATABASE 语句来切换当前会话所使用的默认数据库；
- 在使用 MySQL 客户端连接 OBASE 数据库时，可以通过-D 参数指定当前会话所使用的默认数据库，如果不加-D 参数，则当前会话的默认数据库为系统的默认库 TANG；

7.1.4. 查看系统表

查看系统表，语法格式为：

SHOW SYSTEM TABLES ;

7.1.5. 查看当前会话默认库中用户数据表

查看当前会话默认数据库中的所有数据表，语法格式为：

SHOW TABLES [LIKE '**pattern**' | WHERE **expr**];

```
mysql> show tables like '%cus%';
+----------+
| Tables   |
+----------+
| customer |
+----------+
1 row in set (0.00 sec)
```

7.1.6. 查看数据表定义

查看数据表的定义，语法格式为：

DESCRIBE **tbl_name**;

```
mysql> describe customer;
+------------+---------------+-----------+------+------+---------+-------+
| Field      | Type          | Collation | Null | Key  | Default | Extra |
+------------+---------------+-----------+------+------+---------+-------+
| id         | int32         | NULL      |    0 |    1 | NULL    |       |
| name       | varchar(10)   | NULL      |    0 |    0 | NULL    |       |
| address    | varchar(50)   | NULL      |    1 |    0 | NULL    |       |
| last_login | timestamp     | NULL      |    1 |    0 | NULL    |       |
| remmark    | varchar(1024) | NULL      |    1 |    0 | N/A     |       |
+------------+---------------+-----------+------+------+---------+-------+
5 rows in set (0.00 sec)
```

7.1.7. 查看建表语句

查看指定表的建表语句，语法格式为：

SHOW CREATE TABLE **tbl_name**;

```
mysql> show create table customer;

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

| Table         | Create Table                        |

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

| test.customer | CREATE TABLE test.customer (

id INT32

, name VARCHAR

, address VARCHAR

, last_login DATETIME

, remmark VARCHAR
```

```
, PRIMARY KEY(id)

)  TABLET_MAX_SIZE  =  134217728,  TABLET_BLOCK_SIZE  =  16384,  EXPIRE_INFO  =  '$SYS_DATE  >
last_login+38*24*60*60*1000000', USE_BLOOM_FILTER = FALSE |

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

1 row in set (0.00 sec)
```

7.1.8. 查看列定义

查看指定表中指定列的定义，语法格式为：

SHOW COLUMNS {FROM | IN} **tbl_name** [LIKE '**pattern**' | WHERE **expr**];

```
mysql> show columns from customer like '%i%';
+------------+-----------+-----------+------+------+---------+-------+
| Field      | Type      | Collation | Null | Key  | Default | Extra |
+------------+-----------+-----------+------+------+---------+-------+
| id         | int32     | NULL      |    0 |    1 | NULL    |       |
| last_login | timestamp | NULL      |    1 |    0 | NULL    |       |
+------------+-----------+-----------+------+------+---------+-------+
2 rows in set (0.00 sec)
```

7.1.9. 查看索引

查看数据表上定义的索引，语法格式为：

SHOW INDEX {FROM|IN} **tbl_name**;

```
mysql> show index from customer;

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

| Table    | Non_unique | Key_name                | Seq_in_index | Column_name | Collation |
Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |

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

| customer |          0 | PRIMARY                 |            1 | id          | A         |
0 |     NULL | NULL   |      | BTREE      |         |               |

| customer |          1 | __3045__idx__last_login |            1 | last_login  | A         |
0 |     NULL | NULL   |      | BTREE      |         |               |

| customer |          1 | __3045__idx__last_login |            2 | id          | A         |
0 |     NULL | NULL   |      | BTREE      |         |               |

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

3 rows in set (0.00 sec)
```

7.2. 用户及权限管理

7.2.1. 新建用户

创建一个或多个用户，语法格式为：

CREATE USER '**user1**' IDENTIFIED BY '**password**' [,'**user2**' IDENTIFIED BY '**password**'];

说明：

- 用户名和密码不能为空字符串，限制最大长度均为 31 个字符；
- 新建用户仅拥有系统表“__all_server”和“__all_cluster”的 SELECT 权限，以

及“__all_client”的 REPLACE 和 SELECT 权限；

- 必须拥有全局的 CREATE USER 权限或对“__all_user”表的 INSERT 权限才可以创建用户；
- 新建用户后，“__all_user”表中会新增一行对应该用户的表项，如果同名用户已经存在则会报错；
- 新建用户时指定的密码会以密文方式存入“__all_user”表；
- 同时创建多个用户时用“,”分隔；

```
mysql> select user_id,user_name,pass_word from __all_user;
+---------+-----------+-------------------------------------------+
| user_id | user_name | pass_word                                 |
+---------+-----------+-------------------------------------------+
|       1 | dsadmin   | f314cc4760637e4c425528d7af105a6bb07b6654 |
|       2 | test_user   | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |
+---------+-----------+-------------------------------------------+
2 rows in set (0.00 sec)
```

7.2.2. 删除用户

删除一个或多个用户，语法格式为：

DROP USER '**user_name1**' [,'**user_name2**',…];

说明：

- 必须拥有全局的 CREATE USER 权限或对“__all_user”表的 DELETE 权限才可以删除用户；
- 成功删除用户后，该用户的所有权限也会被一同删除；
- 同时删除多个用户时用“,”分隔；

```
mysql> drop user 'test_user';
Query OK, 0 rows affected (0.07 sec)

mysql> select user_id,user_name,pass_word from __all_user;
+---------+-----------+-------------------------------------------+
| user_id | user_name | pass_word                                 |
+---------+-----------+-------------------------------------------+
|       1 | admin     | ed32f974c5e622767438d999101e5d56f6ac5da9 |
+---------+-----------+-------------------------------------------+
1 rows in set (0.00 sec)
```

7.2.3. 修改用户密码

修改用户的登录密码，语法格式为：

SET PASSWORD FOR '**user**'= '**password**';

ALTER USER '**user**' IDENTIFIED BY '**password**';

说明：

- 必须拥有对“__all_user”表的 UPDATE 权限才可以修改用户的密码；

7.2.4. 修改用户名

修改登录用户的用户名，语法格式为：

RENAME USER '**old_name**' TO '**new_name**';

说明：

- 必须拥有全局 CREATE USER 权限或者对“__all_user”表的 UPDATE 权限，才可以使用本命令；
- 同时修改多个用户名时用“,”分隔

```
mysql> select user_id,user_name,pass_word from __all_user;
+---------+-----------+-------------------------------------------+
| user_id | user_name | pass_word                                 |
+---------+-----------+-------------------------------------------+
|       1 | admin     | ed32f974c5e622767438d999101e5d56f6ac5da9 |
|       2 | test_user | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |
+---------+-----------+-------------------------------------------+
2 rows in set (0.00 sec)

mysql> rename user 'test_user' to 'test_user_renamed';
Query OK, 0 rows affected (0.05 sec)

mysql> select user_id,user_name,pass_word from __all_user;
+---------+------------------+-------------------------------------------+
| user_id | user_name        | pass_word                                 |
+---------+------------------+-------------------------------------------+
|       1 | admin            | ed32f974c5e622767438d999101e5d56f6ac5da9 |
|       2 | test_user_renamed | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |
+---------+------------------+-------------------------------------------+
2 rows in set (0.00 sec)
```

7.2.5. 用户锁定/解锁

锁定/解锁用户，被锁定的用户不允许登陆。语法格式为：

ALTER USER '**user**' LOCKED;

ALTER USER '**user**' UNLOCKED;

说明：

- 必须拥有对“__all_user”表的 UPDATE 权限，才可以执行本命令；

```
mysql> select user_id,user_name,pass_word,is_locked from __all_user;
+---------+-----------+-------------------------------------------+-----------+
| user_id | user_name | pass_word                                 | is_locked |
+---------+-----------+-------------------------------------------+-----------+
|       1 | admin     | ed32f974c5e622767438d999101e5d56f6ac5da9 |         0 |
|       2 | dsadmin   | f314cc4760637e4c425528d7af105a6bb07b6654 |         0 |
|       3 | test      | 216e3bd722d99a9659fd9936c2a5eed90e671ae1 |         0 |
|       5 | test_user | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |         0 |
+---------+-----------+-------------------------------------------+-----------+
4 rows in set (0.00 sec)

mysql> alter user 'test_user' locked;
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> select user_id,user_name,pass_word,is_locked from __all_user;
+---------+-----------+------------------------------------------+-----------+
| user_id | user_name | pass_word                                | is_locked |
+---------+-----------+------------------------------------------+-----------+
|       1 | admin     | ed32f974c5e622767438d999101e5d56f6ac5da9 |         0 |
|       2 | test_user | 83915e5dacb2180d3e574a7699ffb52f01a6f60a |         1 |
+---------+-----------+------------------------------------------+-----------+
2 rows in set (0.00 sec)
```

7.2.6. 授予权限

为用户授予操作权限。OBASE 数据库中的权限分为全局权限、库权限和表权限三个级别，全局权限存储在__all_user 系统表中，适用于数据库中的所有对象；库权限存储在__all_database_privilege 系统表中，适用于某个库及该库下的所有表；表权限存储在__all_table_privilege 系统表中，适用于某个库下的表。

语法格式为：

GRANT **priv_type** [,**priv_type**,…] ON **priv_level** TO '**user1**' [,**user2**,…];

说明：

- priv_level 可以设为*、*.*、db_name.*、db_name.tbl_name 和 tbl_name，*和*.*都代表全局权限，db_name.*代表库级权限，db_name.tbl_name 和 tbl_name 代表表级权限，不加 db_name 前缀的表是指当前会话默认库下的表；
- 当前用户必须拥有被授予的权限和 GRANT OPTION 权限才能为其他用户授权；
- 同时把多个权限授予用户时，权限类型不能重复，注意 ALL PRIVILEGES 与除 GRANT OPTION 外的其它权限都重复；
- 被授权用户只有重新连接 OBASE 数据库后，权限才能生效；
- 用户自动拥有自己创建的对象的所有权限，不再需要额外去授权；
- 用“*”代替 tbl_name，表示赋予全局权限，即对数据库中所有表的权限；
- 同时把多个权限授予用户时，权限类型用“,”分隔；同时给多个用户授权时，用户名用“,”分隔；

表 7-4：OBASE 数据库中的权限

权限	说明
ALL PRIVILEGES	除 GRANT OPTION 以外的所有权限；
ALTER	ALTER TABLE 的权限；

权限	说明
CREATE	CREATE DATABASE 和 CREATE TABLE 的权限；
CREATE USER	CREATE USER, DROP USER, RENAME USER 和 REVOKE ALL PRIVILEGES 的权限；
DELETE	DELETE 的权限；
DROP	DROP DATABASE 和 DROP TABLE 的权限；
GRANT OPTION	GRANT OPTION 的权限；
INSERT	INSERT 的权限；
SELECT	SELECT 的权限；
UPDATE	UPDATE 的权限；
REPLACE	REPLACE 的权限；
SUPER	SET GLOBAL 修改全局系统参数的权限；

```
mysql> show grants for 'test_user';
+-------------------------------------------------------------+
| grants                                                      |
+-------------------------------------------------------------+
| GRANT CREATE ON test.* TO 'test_user'                       |
| GRANT CREATE ON test.im TO 'test_user'                      |
| GRANT CREATE ON test.customer TO 'test_user'                |
+-------------------------------------------------------------+
3 rows in set (0.00 sec)
```

7.2.7. 回收权限

回收用户的操作权限，语法格式为：

REVOKE **priv_type** [,**priv_type**,…] ON **tbl_name** FROM '**user1**' [,**user2**,…];

说明：

- 用户必须拥有被撤销的权限 GRANT OPTION 权限才可以回收其他用户的权限；
- 回收“ALL PRIVILEGES”和“GRANT OPTION”权限时，当前用户必须拥有全局 GRANT OPTION 权限，或者对权限表的 UPDATE 及 DELETE 权限；

- 回收权限操作不会级联；
- 用“*”代替 tbl_name 表示回收全局权限，即回收对数据库中所有表的操作权限；
- 同时对用户回收多个权限时，权限类型用“,”分隔；同时回收多个用户的权限时，用户名用“,”分隔；

```
mysql> revoke create on test.* from 'test_user';
Query OK, 0 rows affected (0.05 sec)

mysql> show grants for 'test_user';
Empty set (0.00 sec)
```

7.2.8. 查看权限

查看用户的操作权限，语法格式为：

SHOW GRANTS [FOR ‘**user**‘];

说明：

- 如果不指定用户名则显示当前用户的权限；
- 如果要查看其他用户的权限，必须拥有对“__all_user”的 SELECT 权限；

7.2.9. 权限检查

OBASE 数据库的权限管理有两阶段的验证，第一阶段首先会检查连接数据库的用户名和密码是否正确，以及是否拥有指定数据库的连接权限；第二阶段会检查每个请求是否有足够的权限实施。OBASE 数据库中各种操作需要的权限如表 7-5 所示。

表 7-5：OBASE 数据库中各种操作所需的权限

操作	权限
SELECT	SQL 中的所有表的 SELECT 权限
INSERT	SQL 中的表的 INSERT 权限
REPLACE	SQL 中的表的 REPLACE 权限
UPDATE	SQL 中的表的 UPDATE 权限
DELETE	SQL 中的表的 DELETE 权限

操作	权限
CREATE USER DROP USER	全局 CREATE_USER 权限
CREATE DATABASE	全局 CREATE 权限
DROP DATABASE	库级 DROP 权限
CREATE TABLE	库级 CREATE 权限
DROP TABLE	表级 DROP 权限
ALTER TABLE	表级 ALTER 权限
修改用户名/密码 锁定用户	对__all_user 表的 UPDATE 权限
GRANT	GRANT 语句中的所有权限以及 GRANT OPTION 权限
REVOKE	REVOKE 语句中的所有权限以及 GRANT OPTION 权限
SHOW GRANTS	__all_user/__all_database_privilege/__all_table_privilege 三张系统表的 SELECT 权限
USING DATABASE	指定库下任何对象的任何权限
CONNECT DATABASE	指定库下任何对象的任何权限

7.3. 设置用户变量

SET @**var_name1** = **expr1** [,@**var_name2** = **expr2**, ···];

说明:

- @var_name1 和@var_name2 为用户变量名，用户变量名由字母、数字和下划线组成，首字符不能为数字;
- 变量的值可以设为整数、实数、字符串或者 NULL 值;
- 同时定义多个用户变量时，用','分隔;

```
mysql> set @var1=2+3,@var2=100,@var3=@var1+@var2;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> select @var1,@var2,@var3;
+-------+-------+-------+
| @var1 | @var2 | @var3 |
+-------+-------+-------+
| 5     | 100   | 105   |
+-------+-------+-------+
1 row in set (0.00 sec)
```

7.4. 修改系统变量

OBASE 数据系统中定义了多个系统变量，用于控制会话的环境，如事务隔离级别、查询超时时间等。通过系统表__all_sys_param 可以查询所有的系统变量。系统变量分为全局变量和局部变量两种，前者影响所有新建会话的环境，后者仅影响当前会话。语法格式如下：

设置全局系统变量

SET GLOBAL **sys_var_name** = **expr**;

SET @@GLOBAL.**sys_var_name** = **expr**;

设置局部系统变量

SET SESSION **sys_var_name** = **expr**;

SET @@SESSION.**sys_var_name** = **expr**;

查询系统变量

SHOW [GLOBAL | SESSION] VARIABLES [LIKE '**pattern**' | WHERE **expr**];

说明：

- 全局变量的值仅用于对新建会话的局部变量进行初始化，修改全局变量的值不影响目前已连接会话的局部变量；
- 修改全局变量必须具有 SUPER 权限，修改会话的局部变量不需要特殊权限，但只能修改当前会话的局部变量，不能修改其它会话的局部变量；
- 查询系统变量时如指定 GLOBAL 或 SESSION 选项，则分别输出全局或当前会话的系统变量值，如不指定则显示当前会话的系统变量值；

表 7-6：OBASE 数据库的系统变量

系统变量	数据类型	说明

autocommit	int	是否自动提交事务
ob_charset	varchar	返回结果的字符集
max_allowed_packet	int	网络包大小上限
ob_app_name	varchar	应用名称
ob_group_agg_push_down_param	bool	聚合操作是否下推到 DG 执行
ob_tx_idle_timeout	int	事务开始后无操作的超时时间
ob_read_consistency	int	读一致性级别
ob_tx_timeout	int	事务超时时间
tx_isolation	varchar	事务隔离级别
wait_timeout	int	等待超时时间
ob_query_timeout	int	查询超时时间

8. 其它语句

8.1. 查看当前连接

每个与 OBASE 数据库的连接都在一个独立的线程里运行，可以通过 SHOW PROCESSLIST 语句查看当前系统中所有的用户连接，Index 列为线程 ID。语法格式为：SHOW PROCESSLIST;

```
mysql> show processlist;
+--------------+-------+----------------------+--------+-----------------+------+--------+---
---+-----------------+-------+
| Id           | User  | Host                 | db     | Command         | Time | State  | Info |
CalEng         | Index |
+--------------+-------+----------------------+--------+-----------------+------+--------+---
---+-----------------+-------+
| 218143959232 | admin | 192.168.100.201:49596 | TANG   | NULL            |    0 | SLEEP  | NULL
| 192.168.100.202 |    50 |
| 377057749184 | admin | 192.168.1.2:40309 | sbtest | NULL            |    0 | SLEEP  | NULL |
192.168.100.202 |    87 |
| 398532585664 | admin | 192.168.1.2:41218 | sbtest | show processlist |  418 | ACTIVE | NULL |
192.168.100.202 |    92 |
+--------------+-------+----------------------+--------+-----------------+------+--------+---
---+-----------------+-------+
3 rows in set (0.00 sec)
```

8.2. 中止用户连接

中止用户的连接线程或语句，语法格式为：

KILL [CONNECTION | QUERY] ***thread_id***;

说明：

- KILL CONNECTION 与不含选项的 KILL 一样，会终止指定的连接线程；
- KILL QUERY 会终止指定连接当前正在执行的语句，但不会中止连接；

```
mysql> show processlist;
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
| Id           | User  | Host                 | db     | Command         | Time | State  | Info            | CalEng | Index |
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
| 604691015872 | admin | 192.168.100.202:41982 | sbtest | show processlist |  697 | ACTIVE | NULL            | 192.168.100.202 |   140 |
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
1 row in set (0.00 sec)

mysql> kill query 140;
Query OK, 0 rows affected (0.00 sec)

mysql> show processlist;
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
| Id           | User  | Host                 | db     | Command         | Time | State  | Info            | CalEng | Index |
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
| 604691015872 | admin | 192.168.100.202:41982 | sbtest | show processlist |  727 | ACTIVE | NULL            | 192.168.100.202 |   140 |
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
1 row in set (0.01 sec)

mysql> kill  140;
Query OK, 0 rows affected (0.00 sec)

mysql> show processlist;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    168
Current database: sbtest

+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
| Id           | User  | Host                 | db     | Command         | Time | State  | Info            | CalEng | Index |
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
| 724950100160 | admin | 192.168.100.202:41991 | sbtest | show processlist |  295 | ACTIVE | NULL            | 192.168.100.202 |   168 |
+--------------+-------+----------------------+--------+-----------------+------+--------+-----------------+-------+
1 row in set (0.01 sec)
```

8.3. EXPLAIN 语句

输出数据操作语句的物理执行计划和逻辑执行计划，便于数据库管理员和应用开发人员理解 OBASE 数据库中语句的执行策略。

```
EXPLAIN [VERBOSE] {select_stmt | insert_stmt | update_stmt | delete_stmt | replace_stmt};
```

```
mysql> explain select * from emp where deptno=20;

+| Query Plan                                                   +

| Explain()

Project(rowkey_count=0,
columns=[expr<NULL,65519>=[COL<3041,16>|],expr<NULL,65518>=[COL<3041,17>|],expr<NULL,65517>=[CO
L<3041,18>|],expr<NULL,65516>=[COL<3041,19>|],expr<NULL,65515>=[COL<3041,20>|],expr<NULL,65514>
=[COL<3041,21>|],expr<NULL,65513>=[COL<3041,22>|],expr<NULL,65512>=[COL<3041,23>|]])

TableRpcScan(read_method=SCAN, is use index for storing=0,index tid=-1; is use index without
storing=0,index tid=-1;is_indexed_group_=0; limit=<Limit(limit=expr<0,0>=*, offset=expr<0,0>=*>,
rpc_scan=<RpcScan(Hint=<max_parallel_count=1,   is_get_skip_empty_row_=true,   read_method_=1,
read_consistency=WEAK>,       tid=NULL,       is_indexed_group_=0,       Limit(limit=expr<0,0>=*,
offset=expr<0,0>=*)Filter(filters=[expr<NULL,65511>=[COL<3041,23>|int:20|EQ<2>|],])Project(rowk
ey_count=0,
columns=[expr<3041,16>=[COL<3041,16>|],expr<3041,17>=[COL<3041,17>|],expr<3041,18>=[COL<3041,18
>|],expr<3041,19>=[COL<3041,19>|],expr<3041,20>=[COL<3041,20>|],expr<3041,21>=[COL<3041,21>|],e
xpr<3041,22>=[COL<3041,22>|],expr<3041,23>=[COL<3041,23>|]]))TransID(sd=0,tg=0.0.0.0@0,start=0>)

 |
```

8.4. 语句中的 hint

嵌入在 SQL 语句的提示（hint）是一种特殊的注释，语法格式为/*+ ... */，其作用是影响数据库引擎对 SQL 语句的执行策略，不会影响 SQL 语句自身的语义。多个 hint 写在同一个注释中时，用”,”分隔。在使用 MySQL 客户端连接 OBASE 数据库时，只有设置了-c 参数，语句中的 hint 才会生效，如：

```
mysql -h 192.168.1.4 -P 2504 -uadmin -pxxxxx -Dsales -c
```

表 8-1：OBASE 数据库支持的 hint 语法

hint	参数	语句	含义
READ_STATIC	无	SELECT	指定查询语句只读 取 DG 上的静态数据，不保证数据一致性；
INDEX(tbl idx)	tbl：表名 idx：索引名	SELECT	限制查询执行时使用指定的索引，如该索引不可用，则直接查询源表，且不使用任何索引；
JOIN(join_alg)	si：semi join merge_join bloomfilter_join	SELECT	指定查询语句中所使用的连接算法，semi join 或 merge join
I_MULTI_BATCH	无	INSERT	指定语句采用多批次插入策略，允许插入超过 2M 的数据量，但不保证语句执行的原子性；
UD_MULTI_BATCH	无	UPDATE	指定语句采用多批次更新/删除策略，允

hint	参数	语句	含义
		DELETE	许更新/删除超过 2M 的数据量，但不保证语句执行的原子性；

附录一 OBASE 数据库系统表

OBASE 数据库中所有的系统表表名均以双下划线"__"开头。

1. __first_tablet_entry

记录系统中所有数据表的属性信息，主键为(table_name, db_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
table_name	varchar(256)	数据表名
create_time_column_id	int	gm_create 列的列 ID
modify_time_column_id	int	gm_modify 列的列 ID
table_id	int	数据表 ID
table_type	Int	数据表类型（未启用）
load_type	Int	（未启用）
table_def_type	Int	1-系统表，2-用户数据表
rowkey_column_num	Int	主键列数
column_num	Int	数据表全部列数（包括主键）
max_used_column_id	Int	该表使用过的最大列 ID（列 ID 不重用）
replica_num	Int	Tablet 的副本个数（1~6）
create_mem_version	Int	创建时系统内存表版本（暂留）
tablet_max_size	Int	Tablet 对应 SSTable 文件大小上限(字节)
max_rowkey_length	Int	主键长度上限（字节）

属性	类型	说明
compress_func_name	varchar(256)	存储时所使用压缩方法名称
is_use_bloomfilter	Int	指定是否使用 Bloom Filter
merge_write_sstable_version	Int	合并时写哪 SSTable 的版本
is_pure_update_table	Int	是否属于内存更新表（未启用）
rowkey_split	Int	数据合并过程中 Tablet 的分裂位置
expire_condition	varchar(512)	数据自动过期删除的条件
tablet_block_size	Int	Tablet 文件数据块的大小（字节）
is_read_static	Int	是否仅读取静态数据
schema_version	Int	Schema 的版本
data_table_id	Int	索引所对应数据表的 ID
index_status	Int	索引表的状态
db_name	varchar(16)	数据表所属的数据库名

2. __all_all_column

记录系统中所有数据表的字段定义信息，主键为(table_id, column_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
table_id	Int	所属数据表 ID
column_name	varchar(128)	字段名
table_name	varchar(256)	数据表名
column_id	Int	字段 ID
column_group_id	Int	（未启用）
rowkey_id	Int	0：非主键属性，>0：在主键中的排序
length_in_rowkey	Int	在主键二进制串中占用的字节数
order_in_rowkey	Int	字段的排序顺序
join_table_id	Int	-1：无连接表，>0：连接表的 ID

属性	类型	说明
join_column_id	Int	-1：无连接表，>0：连接表的字段 ID
data_type	Int	数据类型
data_length	Int	整数的字节数或字符串的最大长度
data_precision	Int	Decimal 类型的有效位数
data_scale	Int	Decimal 类型的精度
nullable	Int	是否可以为空

3. __all_join_info

存储数据表之间内部的连接关系，主键为(left_table_id, left_column_id, right_table_id, right_column_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
left_table_id	Int	左表 ID
left_column_id	Int	左表字段 ID
right_table_id	Int	右表 ID
right_column_id	int	右表字段 ID
left_table_name	varchar(256)	左表表名
left_column_name	varchar(256)	左表字段名
right_table_name	varchar(256)	右表表名
right_column_name	varchar(256)	右表字段名

4. __all_cluster

存储系统中所有集群的信息，主键为(cluster_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	int	集群 ID

属性	类型	说明
cluster_vip	varchar(32)	集群主 MG 的 IP 地址
cluster_port	int	集群端口号（未启用）
cluster_role	int	1：主，2：备
cluster_name	varchar(128)	系群名称（未启用）
cluster_info	varchar(128)	集群说明信息（未启用）
cluster_flow_percent	int	流量配比
read_strategy	int	客户端负载均衡策略，0：随机轮转，1：一致性哈希
maneng_port	int	主 MG 的端口

5. __all_server

存储系统中所有服务器的信息，主键为(cluster_id, svr_type, svr_ip, svr_port)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	int	集群 ID
svr_type	varchar(16)	服务器类型
svr_ip	varchar(32)	服务器 IP 地址
svr_port	int	服务器端口号
inner_port	int	内部交互端口号
svr_role	int	0：无关主备，1：主，2：备
svr_version	varchar(64)	服务器程序版本信息

6. __all_database

存储系统中所有数据库的基本信息，主键为(db_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间

属性	类型	说明
db_name	varchar(16)	数据库名
db_id	int	数据库 ID
stat	int	数据库状态

7. __all_server_session

记录当前连接到 OBASE 数据库的所有会话，主键为(id)。

属性	类型	说明
id	int	会话 ID
username	varchar(512)	会话所使用的用户名
host	varchar(128)	会话来源主机 IP 地址和端口
db	varchar(128)	（未启用）
command	varchar(1024)	会话中正在执行的命令
timeelapse	int	命令耗时
state	varchar(128)	ACTIVE：正在使用，SLEEP：暂时未用
info	varchar(128)	会话说明信息
calEng	varchar(128)	会话所连接的 CG
index	int	会话所对应的线程 ID

8. __all_server_stat

记录系统中所有服务器的监控信息，仅维护在内存中，不实际存储，主键为(svr_type, svt_ip, svr_port, name)。

属性	类型	说明
svr_type	varchar(16)	服务器类型
svr_ip	varchar(32)	服务器 IP 地址
svr_port	int	服务器端口
name	varchar(64)	监控项名称
value	int	监控项的值

9. __all_sys_config_stat

记录当前系统中各服务器已生效配置项的设置值，主键为(cluster_id, svr_type, svr_ip, svr_ port, name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	int	所属集群 ID
svr_type	varchar(16)	服务器类型
svr_ip	varchar(32)	服务器 IP 地址
svr_port	int	服务器端口
name	varchar(256)	配置项名称
section	varchar(256)	（未启用）
data_type	varchar(256)	配置项值的类型
value	varchar(256)	配置项的值
value_strict	varchar(512)	（未启用）
info	varchar(512)	配置项的说明

10. __all_sys_param

存储了系统所需的环境变量等参数，主键为(cluster_id, name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	int	集群 ID
name	varchar(256)	参数名称
data_type	int	参数值的数据类型
value	varchar(256)	参数的值
info	varchar(256)	参数的说明

11. __all_sys_stat

存储了系统中的各种状态值，主键为(cluster_id, name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
cluster_id	int	集群 ID
name	varchar(256)	状态名称
data_type	int	状态值的数据类型
value	varchar(256)	状态的值
info	varchar(256)	状态的说明

12. __all_table_privilege

存储了系统中每张数据表的授权信息，主键为(user_id, db_id, table_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
user_id	int	用户 ID
table_id	int	数据表 ID，0 表示所有表
priv_all	int	所有权限
priv_alter	int	ALTER TABLE 权限
priv_create	int	CREATE TABLE 权限
priv_create_user	int	CREATE USER 权限
priv_delete	int	DELETE 权限
priv_drop	int	DROP TABLE 权限
priv_grant_option	int	授权权限
priv_insert	int	INSERT 权限
priv_update	int	UPDATE 权限
priv_select	int	SELECT 权限

属性	类型	说明
priv_replace	int	REPLACE 权限
db_id	int	数据库 ID

13. __all_user

存储了系统中的用户信息及其全局权限，主键为(user_name)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
user_name	varchar(1024)	用户名
user_id	int	用户 ID
pass_word	varchar(1024)	用户密码（密文存储）
info	varchar(1024)	用户信息
priv_all	int	所有权限
priv_alter	int	ALTER 权限
priv_create	int	CREATE TABLE 权限
priv_create_user	int	CREATE USER 权限
priv_delete	int	DELETE 权限
priv_drop	int	DROP TABLE 权限
priv_grant_option	int	授权权限
priv_insert	int	INSERT 权限
priv_update	int	UPDATE 权限
priv_select	int	SELECT 权限
priv_replace	int	REPLACE 权限
is_locked	int	是否被锁定

14. __all_database_privilege

存储了系统中每个数据库的授权信息，主键为(user_id, db_id)。

属性	类型	说明
gm_create	createtime	创建时间
gm_modify	modifytime	修改时间
user_id	int	用户 ID
db_id	int	数据库 ID
priv_all	int	所有权限
priv_alter	int	ALTER TABLE 权限
priv_create	int	CREATE TABLE 权限
priv_create_user	int	CREATE USER 权限
priv_delete	int	DELETE 权限
priv_drop	int	DROP TABLE 权限
priv_grant_option	int	授权权限
priv_insert	int	INSERT 权限
priv_update	int	UPDATE 权限
priv_select	int	SELECT 权限
priv_replace	int	REPLACE 权限
is_locked	int	是否被锁定

15. __all_sequence

存储系统中所有序列的信息，主键为(sequence_name)。

属性	类型	说明
sequence_name	varchar(128)	序列名
data_type	int	数据类型，0：整型，>0：精度
current_value	decimal(31,0)	序列的当前值（prevval）
increment_by	decimal(31,0)	序列的步长
min_value	decimal(31,0)	序列的最小值
max_value	decimal(31,0)	序列的最大值
is_cycle	int	0：不循环，1：循环

属性	类型	说明
cache_num	int	序列值缓存的数量
is_order	int	并发访问是否严格递增(减)
is_valid	int	0：不可用，1：可用
const_start_with	decimal(31,0)	RESTART 设置的起始值
can_use_prevval	int	0：不可用，1：可用
use_quick_path	int	0：无效，1：有效

16. __all_truncate_op

记录系统中清空表操作的执行情况，主键为(rs_trun_time, table_id)。

属性	类型	说明
rs_trun_time	timestamp	MG 上执行的时间戳
table_id	int	数据表 ID
table_name	varchar(273)	数据库名.数据表名
user_name	varchar(32)	执行操作的用户名
info	varchar(512)	说明信息

附录二 OBASE 关键字列表

ACTION	INNER	TABLE
ADD	INSERT	TABLES
ADDDATE	INSTR	TEXT
ALL	INT	THEN
ALTER	INTEGER	TIME
AND	INTERSECT	TIMESTAMP
ANY	INTERVAL	TINYBLOB
AS	INTO	TINYCLOB
ASC	IS	TINYINT
BEGIN	ISOLATION	TINYTEXT
BETWEEN	JOIN	TO
BIGINT	KEY(S)?	TRAILING
BINARY	KILL	TRANSACTION

BIT	LEADING	TRUNCATE
BLOOMFILTER_JOIN	LEFT	UD_ALL_ROWKEY
BOOL(EAN)	LEVEL	UD_MULTI_BATCH
BOTH	LIKE	UD_NOT_PARALLAL
BTREE	LIMIT	UNCOMMITTED
BY	LOCAL	UNION
CACHE	LOCKED	UNIQUE
CASCADE	MASTER	UNSIGNED
CASCADED	MATCH	UPDATE
CASE	MAXVALUE	USE
CHANGE_OBI	MEDIUMINT	USER
CHANGE_VALUE_SIZE	MEMORY	USING
CHAR(ACTER)	MERGE_JOIN	VALUES
CHARSET	MICROSECONDS	VARBINARY
CHECK	MINUTES	VARCHAR2
CLUSTER	MINVALUE	VARCHAR(ACTER)
COLUMN	MOD	VIEW
COLUMNS	MODIFYTIME	WEAK
COMMENT	MONTHS	WHEN
COMMIT	NEXTVAL	WHERE
COMMITTED	NO	WITH
COMPACT	NOT	WITHIN
COMPRESSED	NUMBER	WORK
CONCAT	NUMERIC	YEARS
CONNECTION	OFFSET	
CONSISTENT	ON	
CONSTRAINT	ONLY	
CREATE	OPTION	
CREATETIME	OR	
CURRENT	ORDER	
CURRENT_DATE	OUTER	
CURRENT_TIMESTAMP	OVER	
CURRENT_USER	PARAMETERS	
CYCLE	PARTIAL	
DATABASE	PARTITION	
DATABASES	PASSWORD	
DATE	POSITION	
DATE_ADD	PRECISION	
DATE_SUB	PREPARE	
DATETIME	PREVVAL	
DAYS	PRIMARY	
DEALLOCATE	PRIVILEGES	
DEC(IMAL)	PROCESSLIST	

DEFAULT	QUERY	
DELETE	QUICK	
DESC	RANGE	
DESCRIBE	READ	
DISTINCT	REAL	
DOUBLE	REDUNDANT	
DROP	REFERENCES	
DUAL	RENAME	
DYNAMIC	REPEATABLE	
ELSE	REPLACE	
END	RESTART	
ENGINE	RESTRICT	
ERROR	REVOKE	
EXCEPT	RIGHT	
EXECUTE	ROLLBACK	
EXISTS	ROW_FORMAT	
EXPLAIN	ROW(S)	
FETCH	SCHEMA	
FIRST	SCOPE	
FIXED	SECONDS	
FLOAT	SELECT	
FLOAT4	SEMI	
FLOAT8	SEQUENCE	
FOR	SERIALIZABLE	
FORCE	@@session	
FOREIGN	SESSION	
FROM	SET	
FROZEN	SET_SLAVE_CLUSTER	
FULL	SHOW	
GATHER	SI	
@@global	SIB	
GLOBAL	SIMPLE	
GRANT	SLAVE	
GROUP	SMALLINT	
HASH	SNAPSHOT	
HASH_JOIN_SINGLE	SPFILE	
HAVING	START	
HOURS	STATIC	
IDENTIFIED	STATISTICS	
IF	STORING	
I_MULTI_BATCH	STRONG	
IN	SUBDATE	
INCREMENT	SWITCH_CLUSTER	

INDEX(ES)	SYSTEM	