



OceanBase 0.5 SQL 参考指南

文档版本: Beta 03

发布日期: 2014.06.06

支付宝（中国）网络技术有限公司·OceanBase 团队

前言

概述

本文档主要介绍OceanBase 0.5支持的SQL语言、语法规则和使用方法等。

读者对象

本文档主要适用于：

- 开发工程师。
- 数据库管理工程师。

通用约定

在本文档中可能出现下列标志，它们所代表的含义如下。

标志	说明
 警告：	表示可能导致设备损坏、数据丢失或不可预知的结果。
 注意：	表示可能导致设备性能降低、服务不可用。
 小窍门：	可以帮助您解决某个问题或节省您的时间。
 说明：	表示正文的附加信息，是对正文的强调和补充。

在本文档中可能出现下列格式，它们所代表的含义如下。

格式	说明
宋体	表示正文。
黑体	标题、警告、注意、小窍门、说明等内容均采用黑体。
Calibri	表示代码或者屏幕显示内容。
粗体	表示命令行中的关键字（命令中保持不变、必须照输的部分）或者正文中强调的内容。

格式	说明
<i>斜体</i>	用于变量输入。
{ a b ... }	表示从两个或多个选项中选取一个。
[]	表示用“[]”括起来的部分在命令配置时是可选的。

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本。

版本和发布日期	说明
Beta 03（2014-06-06）	<p>第一次发布Beta 03版本，适用于OceanBase 0.5。</p> <ul style="list-style-type: none"> 增加了CREATE TABLE时的<code>option</code>。 增加部分数据类型和函数。 增加向量比较运算。 增加OceanBase支持的转义字符。 增加索引操作。 DELETE和UPDATE支持多行操作。 增加ALTER TABLE和ALTER SYSTEM语句。 设置字符集编码的方法改变。 增加部分hint语句。
Beta 02（2014-02-24）	<p>第一次发布Beta版本，适用于OceanBase 0.4.2。</p> <ul style="list-style-type: none"> CREATE TABLE时支持添加注释信息和指定表ID。 增加SHOW processlist和KILL语句。 增加STRICT_CURRENT_TIMESTAMP()函数。 增加OceanBase支持的转义字符。 增加设置字符集编码。 删除聚合操作下压条件：没有ORDER BY子句。
01（2013-10-30）	第一次正式发布，适用于OceanBase 0.4.1。

联系我们

如果您有任何疑问或是想了解 OceanBase 的最新开源动态消息，请联系我们：

支付宝（中国）网络技术有限公司·OceanBase 团队

地址：杭州市万塘路 18 号黄龙时代广场 B 座；邮编：310099

北京市朝阳区东三环中路 1 号环球金融中心西塔 14 层；邮编：100020

邮箱：alipay-oceanbase-support@list.alibaba-inc.com

新浪微博：<http://weibo.com/u/2356115944>

技术交流群（阿里旺旺）：853923637

目 录

1 OceanBase SQL 简介	- 1 -
1.1 支持语句	- 1 -
1.2 数据类型	- 1 -
1.2.1 基本数据类型	- 1 -
1.2.2 高精度数值类型	- 3 -
1.3 函数	- 4 -
1.3.1 系统函数	- 4 -
1.3.2 聚集函数	- 17 -
1.4 运算符和优先级	- 18 -
1.4.1 逻辑运算符	- 18 -
1.4.2 算数运算符	- 20 -
1.4.3 比较运算符	- 22 -
1.4.4 向量比较运算符	- 25 -
1.4.5 拼接运算符	- 26 -
1.4.6 优先级	- 26 -
1.5 转义字符	- 27 -
1.6 内部表	- 28 -
2 数据定义语言	- 29 -
2.1 CREATE TABLE 语句	- 29 -
2.2 ALTER TABLE 语句	- 32 -
2.3 DROP TABLE 语句	- 35 -
2.4 CREATE INDEX 语句	- 35 -
2.5 DROP INDEX 语句	- 36 -
2.6 ALTER SYSTEM 语句	- 36 -
3 数据操作语言	- 42 -
3.1 INSERT 语句	- 42 -
3.2 REPLACE 语句	- 42 -
3.3 UPDATE 语句	- 43 -
3.4 DELETE 语句	- 44 -

3.5 SELECT 语句.....	- 44 -
3.5.1 基本查询	- 44 -
3.5.2 JOIN 句法.....	- 48 -
3.5.3 集合操作	- 49 -
3.5.4 DUAL 虚拟表	- 51 -
3.5.5 SELECT ... FOR UPDATE 句法.....	- 52 -
3.5.6 IN 和 OR.....	- 52 -
4 事务处理.....	- 54 -
5 数据库管理语句.....	- 57 -
5.1 用户及权限管理.....	- 57 -
5.1.1 新建用户	- 57 -
5.1.2 删除用户	- 58 -
5.1.3 修改密码	- 58 -
5.1.4 修改用户名	- 59 -
5.1.5 锁定用户	- 60 -
5.1.6 用户授权	- 60 -
5.1.7 撤销权限	- 61 -
5.1.8 查看权限	- 62 -
5.2 修改用户变量	- 62 -
5.3 修改系统变量	- 63 -
5.4 修改系统配置项.....	- 64 -
6 预备执行语句.....	- 67 -
7 其他 SQL 语句.....	- 69 -
7.1 SHOW 语句.....	- 69 -
7.2 KILL 语句	- 70 -
7.3 DESCRIBE 语句	- 70 -
7.4 EXPLAIN 语句	- 70 -
7.5 WHEN 语句	- 70 -
7.6 hint 语法.....	- 71 -
8 SQL 优化.....	- 73 -
8.1 执行计划	- 73 -
8.2 内部优化规则	- 73 -

8.2.1 主键索引	- 74 -
8.2.2 并发执行	- 74 -

1 OceanBase SQL 简介

SQL (Structured Query Language) 是一种组织、管理和检索数据库存储数据的计算机语言。由于 OceanBase 完全兼容 MySQL 的网络协议, 所以 OceanBase SQL 用户可以使用 MySQL 客户端、Java 客户端和 C 客户端连接 OceanBase。

1.1 支持语句

OceanBase SQL 语句中的关键字、表名、列名、函数名等均大小写不敏感。表名和列名都转换为小写之后存入 Schema 中, 所以即使用户建表时候列名是大写的, 查询的时候获得的列名也是小写。如果您需要保存大写字母, 请使用双引号, 例如: "Info"。

OceanBase SQL 语法遵循 SQL92 标准, 单引号表示字符串; 双引号表示表名、列名或函数名。双引号内可以出现 SQL 保留的关键字。

OceanBase 没有 Database 的概念, 可以理解为一个 OceanBase 集群只有一个 Database, 所以用户不需要也不能使用 "USE DATABASE" 语句来指定 Database。

目前版本支持的语句有 CREATE TABLE, DROP TABLE, ALTER TABLE, ALTER SYSTEM, SELECT, INSERT, REPLACE, DELETE, UPDATE, SET, SHOW 等。

1.2 数据类型

OceanBase 支持常用的基本数据类型和高精度数值类型。

1.2.1 基本数据类型

目前 OceanBase 支持的基本数据类型如[表 1-1](#)所示

表 1-1 数据类型

数据类型	说明	字段
BIGINT/INT/INTEGER/MEDIUMINT/SMALLINT/TINYINT	在 OceanBase 中, BIGINT、INT、INTEGER、MEDIUMINT、SMALLINT 和 TINYINT 无论语义还是实现都是等价的, 存储为 8 字节有符号整型。	MYSQL_TYPE_LONGLONG

数据类型	说明	字段
BINARY/CHAR/VARBINARY/VARCHAR	字符串，使用单引号。 取值范围：[0, 65536]。 在 OceanBase 中，BINARY、CHAR、VARBINARY 和 VARCHAR 等价，均存储为 VARCHAR 类型。这种类型的比较使用的是字节序。	MYSQL_TYPE_VAR_STRING
BOOL	布尔类型，表示 True 或者 False。	MYSQL_TYPE_TINY
CREATETIME	OceanBase 数据库提供的特殊的数据类型，用于记录本行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。	-
DATETIME/TIMESTAMP	时间戳类型。OceanBase 不支持 TIME、DATE 等类型。 OceanBase 的时间戳格式必须为“YYYY-MM-DD HH:MI:SS”或者“YYYY-MM-DD HH:MI:SS.SSSSSS”，否则插入的时间戳可能不正确。	MYSQL_TYPE_DATETIME
DOUBLE/REAL	表示 8 字节浮点数。 在 OceanBase 中，DOUBLE 和 REAL 等价，均存储为 DOUBLE 类型。	MYSQL_TYPE_DOUBLE
FLOAT	表示 4 字节浮点数。	MYSQL_TYPE_FLOAT
MODIFYTIME	OceanBase 数据库提供的特殊的数据类型，用于记录本行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。	-

与 MySQL 的区别如下：

- 在 MySQL 中，型如“1.2345”的字面量是作为 DECIMAL 类型处理的，型如“1.2345e18”的字面量才作为浮点数处理。而目前版本的 OceanBase 中，两者都作为浮点数类型 DOUBLE 处理。

- 在 MySQL 中,在做表达式运算时,两个整数类型相除,结果是 DECIMAL 类型。而目前版本的 OceanBase 中,两者相除的结果为 DOUBLE 类型。

1.2.2 高精度数值类型

OceanBase 0.5 支持高精度数值类型: DECIMAL(p,s)和 NUMERIC(p,s)。在 OceanBase 中, DECIMAL(p,s)和 NUMERIC(p,s)等效。

- “ p ”表示 precision, 与“ s ”的差值为整数位数的限制。
 - $(p-s) > 0$: 表示整数部分的位数不能超过“ $p-s$ ”, 否则将报错。
 - $(p-s) \leq 0$: 表示整数部分必须为“0”, 小数点后“ $-(p-s)$ ”位也必须为“0”, 否则将报错。
- “ s ”表示 scale。
 - $s > 0$: 表示精度限制在小数点后 s 位, 超过 s 位后面的小数将被四舍五入。
 - $s \leq 0$: 小数部分被舍去, 且小数点前 s 位, 将被四舍五入。

最大取值范围:

- 正数: $1e-135 \sim 9.999...999e+134$ (45 个 9 后边带 90 个 0)
- 0
- 负数: $-1e-135 \sim -9.999...999e+134$ (45 个 9 后边带 90 个 0)

DECIMAL 可以表示的十进制位数最大为 45 位。



说明:

计算 DECIMAL 的位数时, 结尾的 0 不占位数。例如, “102400000” 的位数是 4。

精度溢出后处理方法:

- 整数数值超过表示范围时, 直接报错。
- 小数数值超过表示范围时, 四舍五入。
- 十进制位数超过表示范围时, 四舍五入。

DECIMAL 类型举例:

- DECIMAL(3, 2) 1.2345 ==> 1.23 小数部分四舍五入保留 2 位
12.3 ==> Error 整数部分大于 1 位
- DECIMAL(3, -2) 345.6 ==> 300 小数点前 2 位被四舍五入
45.6 ==> 0 小数点前 2 位被四舍五入

- 123456.7 ==> Error 整数部分大于 5 位
- DECIMAL(2, 3) 1.2 ==> Error 整数部分不为 0
- 0.1 ==> Error 小数部分前 1 位不为 0
- 0.02345 ==> 0.023 小数部分超过 3 位后面的被四舍五入

1.3 函数

OceanBase 支持的函数可以分为系统函数和聚集函数。

1.3.1 系统函数

介绍 OceanBase 支持的系统函数的格式和用法。

* CAST(*expr* AS *type*)

将 *expr* 字段值转换为 *type* 数据类型。数据类型如[表 1-1](#)所示。

```
mysql> SELECT CAST(123 AS BOOL);
+-----+
| CAST(123 AS BOOL) |
+-----+
| 1 |
+-----+
1 row in set (0.03 sec)
```

* COALESCE(*expr*, *expr*, ...)

依次参考各参数表达式，遇到非 NULL 值即停止并返回该值。如果所有的表达式都是空值，最终将返回一个空值。

```
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 (0.06 sec)
```

* CONCAT(*str1*, *str2*)

把两个字符串连接成一个字符串。左右参数都必须是 VARCHAR 类型或 NULL，否则报错。如果执行成功，则返回连接后的字符串；参数中有一个值是 NULL 结果就是 NULL。

```
mysql> SELECT CONCAT('hello', 'world');
```

```
+-----+
| CONCAT('hello', 'world') |
+-----+
|helloworld                |
+-----+
1 row in set (0.01 sec)
```

* **CURRENT_TIME()**和 **CURRENT_TIMESTAMP()**

用于获取系统当前时间，精确到微秒。

格式为“YYYY-MM-DD HH:MI:SS.SSSSSS”。

```
mysql> SELECT CURRENT_TIME(), CURRENT_TIMESTAMP();
```

```
+-----+-----+
| CURRENT_TIME() | CURRENT_TIMESTAMP() |
+-----+-----+
| 2014-02-14 16:58:12.795944 | 2014-02-14 16:58:12.795944 |
+-----+-----+
1 row in set (0.03 sec)
```

* **DATE_ADD(*date*, INTERVAL *expr unit*)**

这个函数用来执行时间的算术计算。将 *date* 值作为基数，对 *expr* 进行相加计算，*expr* 的值允许为负数。DATE_ADD() 计算时间值是否使用夏令时，由操作系统根据其内部配置和相应时区设定来决定。

- *date* 参数类型只能为 Time 类型(DATETIME, TIMESTAMP 等)或者代表时间的一个字符串，不接受其它类型。
 - *date* 参数的期望日期类型为“YYYY-MM-DD HH:MM:SS.SSSSSS”格式，但 OceanBase 解析日期类型字符串时允许“不严格”语法，如果一个字符串中包含数字和非数字，OceanBase 将解析出被非数字隔断的数字序列作为时间序列，依次赋给年月日。例如“Ywwe1990d07 09,12:45-08&900”该字符串和“1990-07-09 12:45:08.900”在表示时间值上是等价的。
 - 在 *date* 字符串中，日期部分是必须的，而时间部分是可以缺省的。例如“1990-07-09”是合法的，这种情况下后面的时间部分将默认填充为 0，其等价于“1990-07-09 00:00:00.000000”；而“1990-07”和“1990”这样的格式都是非法的。
 - *date* 字符串不支持例如“990309”这种 TIMESTAMP 类型字符串。
 - OceanBase 中的其它系统函数调用结果可以作为的 *date* 参数进行计算。
 - OceanBase 不支持对两位数的年份进行模糊匹配，例如如 12 年在 MySQL 中匹配为 2012 年，而在 OceanBase 中就代表 12 年。

- *expr* 的值允许为负，对一个负值相加功能等同于对一个正值相减。允许系统函数的调用结果作为该参数，但是所有结果都将作为字符串结果。
- *unit* 为单位，支持 MICROSECOND、SECOND、MINUTE、HOUR、DAY、WEEK、MONTH、QUARTER、YEAR、SECOND_MICROSECOND、MINUTE_MICROSECOND、MINUTE_SECOND、HOUR_MICROSECOND、HOUR_SECOND、HOUR_MINUTE、DAY_MICROSECOND、DAY_SECOND、DAY_MINUTE、DAY_HOUR 和 YEAR_MONTH。其中 QUARTER 代表季度。
- *unit* 为复合单位时，*expr* 必须加单引号。



小窍门：

当单行显示过长，造成阅读困难时，可在 SELECT 结尾使用 “\G”，将查询结果垂直排列。

```
mysql> SELECT DATE_ADD(now(), INTERVAL 5 DAY),
               DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND),
               DATE_ADD('2014-01-10', INTERVAL 5 SECOND),
               DATE_ADD('2014-01-10', INTERVAL 5 MINUTE),
               DATE_ADD('2014-01-10', INTERVAL 5 HOUR),
               DATE_ADD('2014-01-10', INTERVAL 5 DAY),
               DATE_ADD('2014-01-10', INTERVAL 5 WEEK),
               DATE_ADD('2014-01-10', INTERVAL 5 MONTH),
               DATE_ADD('2014-01-10', INTERVAL 5 QUARTER),
               DATE_ADD('2014-01-10', INTERVAL 5 YEAR),
               DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND),
               DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND),
               DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND),
               DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND),
               DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND),
               DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE),
               DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND),
               DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND),
               DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE),
               DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR),
               DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH) \G
***** 1. row *****
DATE_ADD(now(), INTERVAL 5 DAY): 2014-02-19 17:18:36.423538
DATE_ADD('2014-01-10', INTERVAL 5 MICROSECOND): 2014-01-10 00:00:00.000005
DATE_ADD('2014-01-10', INTERVAL 5 SECOND): 2014-01-10 00:00:05
DATE_ADD('2014-01-10', INTERVAL 5 MINUTE): 2014-01-10 00:05:00
DATE_ADD('2014-01-10', INTERVAL 5 HOUR): 2014-01-10 05:00:00
DATE_ADD('2014-01-10', INTERVAL 5 DAY): 2014-01-15 00:00:00
DATE_ADD('2014-01-10', INTERVAL 5 WEEK): 2014-02-14 00:00:00
DATE_ADD('2014-01-10', INTERVAL 5 MONTH): 2014-06-10 00:00:00
DATE_ADD('2014-01-10', INTERVAL 5 QUARTER): 2015-04-10 00:00:00
DATE_ADD('2014-01-10', INTERVAL 5 YEAR): 2019-01-10 00:00:00
DATE_ADD('2014-01-10', INTERVAL '5.000005' SECOND_MICROSECOND) : 2014-01-10
00:00:05.000005
```

```

DATE_ADD('2014-01-10', INTERVAL '05:05.000005' MINUTE_MICROSECOND): 2014-01-10
00:05:05.000005
DATE_ADD('2014-01-10', INTERVAL '05:05' MINUTE_SECOND): 2014-01-10 00:05:05
DATE_ADD('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND): 2014-01-10
05:05:05.000005
DATE_ADD('2014-01-10', INTERVAL '05:05:05' HOUR_SECOND): 2014-01-10 05:05:05
DATE_ADD('2014-01-10', INTERVAL '05:05' HOUR_MINUTE): 2014-01-10 05:05:00
DATE_ADD('2014-01-10', INTERVAL '01 05:05:05.000005' DAY_MICROSECOND): 2014-01-11
05:05:05.000005
DATE_ADD('2014-01-10', INTERVAL '01 05:05:05' DAY_SECOND): 2014-01-11 05:05:05
DATE_ADD('2014-01-10', INTERVAL '01 05:05' DAY_MINUTE): 2014-01-11 05:05:00
DATE_ADD('2014-01-10', INTERVAL '01 05' DAY_HOUR): 2014-01-11 05:00:00
DATE_ADD('2014-01-10', INTERVAL '1-01' YEAR_MONTH): 2015-02-10 00:00:00
1 row in set (0.07 sec)

```

* DATE_FORMAT(*date*, *format*)

DATE_FORMAT()是 STR_TO_DATE()的逆函数，DATE_FORMAT()接受一个时间值 *date*，将其按 *format* 的格式格式化成一个时间字符串。

- *date* 参数给出了被格式化的时间值，*date* 只接受 **time** 类型和时间字符串作为参数，其具体描述参考 DATE_ADD()的 *date* 参数描述。
- *format* 的格式如[表 1-2](#)所示。

表 1-2 *format* 格式

格式	含义	返回格式
%a	星期。	Sun..Sat
%b	月份的缩写名称。	Jan, ..., Dec
%c	月份，数字形式。	1, ..., 12
%D	带有英语后缀的日期。	1st, 2nd, ..., 31st
%d	日期，数字形式。	01, ..., 31
%e	日期，数字形式。	1, ..., 31
%f	微秒。	000000, ..., 999999
%H	小时。	00, ..., 23
%h	小时。	01, ..., 12
%l	小时。	01, ..., 12

格式	含义	返回格式
%i	分钟。	00, ..., 59
%j	一年中的第几天。	000, ..., 366
%k	小时。	0, ..., 23
%l	小时。	01, ..., 12
%M	月份名称。	January, ..., December
%m	月份, 数字形式。	01, ..., 12
%p	上午或下午。	AM, PM
%r	12 小时制时间。	hh:mm:ss AM/PM
%S	秒。	00, ..., 59
%s	秒。	00, ..., 59
%T	24 小时制时间。	hh:mm:ss
%U	一年中的第几周, 其中周日为每周的第一天。	00, ..., 53
%u	一年中的第几周, 其中周一为每周的第一天。	00, ..., 53
%V	<p>一年中的第几周, 其中周日为每周的第一天, 和%X 同时使用。</p> <p>说明: 在一年中的第一周或者最后一周产生跨年时 (以 2014-01-01 星期三为例), %U 和%u 时, 该天为 2014 年的第 00 周, %V 和%v 时为 2013 年的第 52 周。</p>	01, ..., 53
%v	一年中的第几周, 其中周一为每周的第一天, 和%x 同时使用。	01, ..., 53
%W	星期。	Sunday, ..., Saturday
%w	一周中的第几天。	0=Sunday, ..., 6=Saturday

格式	含义	返回格式
%X	某一周所属的年份，其中周日为每周的第一天，数字形式，4 位数，和%V 同时使用。	-
%x	某一周所属的年份，其中周一为每周的第一天，数字形式，4 位数，和%v 同时使用。	-
%Y	年，用四位数字表示。	-
%y	年，用两位数字表示。	-
%%	文字字符，输出一个%。	-

注：“-”表示无。

```
mysql> SELECT DATE_FORMAT('2014-01-01', '%Y-%M-%d'),
                DATE_FORMAT('2014-01-01', '%X-%V'),
                DATE_FORMAT('2014-01-01', '%U') \G
***** 1. row *****
DATE_FORMAT('2014-01-01', '%Y-%M-%d') : 2014-January-01
DATE_FORMAT('2014-01-01', '%X-%V') : 2013-52
DATE_FORMAT('2014-01-01', '%U') : 00
1 row in set (0.00 sec)
```

* **DATE_SUB(*date*, INTERVAL *expr unit*)**

对时间进行算数计算。将 *date* 作为基数，对 *expr* 进行相减计算，*expr* 允许为负，结果相当于做取反做加法。

参数说明参考 DATE_ADD()。

```
mysql> SELECT DATE_SUB('2014-01-10', INTERVAL 5 HOUR),
                DATE_SUB('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND) \G
***** 1. row *****
DATE_SUB('2014-01-10', INTERVAL 5 HOUR): 2014-01-09 19:00:00
DATE_SUB('2014-01-10', INTERVAL '05:05:05.000005' HOUR_MICROSECOND): 2014-01-09
18:54:54.999995
1 row in set (0.00 sec)
```

* **EXTRACT(*unit* FROM *date*)**

提取 *date* 表达式中被 *unit* 指定的时间组成单元的值。

参数参考 DATE_ADD()。

- EXTRACT 函数返回的结果类型为 ObIntType，即 64 位 int 类型。

- 对于 MICROSECOND~YEAR 这种 Single unit，将直接返回对应的 int 值。
- *unit* 为 WEEK 返回的是 date 表达式中指定的日期在该年所对应的周数，而 OceanBase 将一年的第一个星期日作为该年第一周的开始，如果某年的第一个星期日不是 1 月 1 日，那么该星期日之前的日期处于第 0 周。例如，2013 年第一个星期日是 1 月 6 日，所以 SELECT EXTRACT(WEEK FROM '2013-01-01')返回的结果为 0，而 SELECT EXTRACT(WEEK FROM '2013-01-06')返回的结果是 1。
- 对于 SECOND_MICROSECOND 这种 combinative unit，OceanBase 将各个值拼接在一起作为返回值。例如：EXTRACT(YEAR_MONTH FROM '2012-03-09')返回的结果将是“201203”。

```
mysql> SELECT EXTRACT(WEEK FROM '2013-01-01'),
               EXTRACT(WEEK FROM '2013-01-06'),
               EXTRACT(YEAR_MONTH FROM '2012-03-09'),
               EXTRACT(DAY FROM NOW()) \G
***** 1. row *****
      EXTRACT(WEEK FROM '2013-01-01'): 0
      EXTRACT(WEEK FROM '2013-01-06'): 1
      EXTRACT(YEAR_MONTH FROM '2012-03-09'): 201203
      EXTRACT(DAY FROM NOW()): 15
1 row in set (0.00 sec)
```

* HEX(str)

将字符串转化为十六进制数显示。

str 必须是整数或者字符串。当输入是整数（进制不限）的时候，输出整数的十六进制表示；当输入是字符串的时候，输出是字符串的字节流的十六进制表示。



注意：

目前 OceanBase 中，16 进制数两个数位表示一个字符，因此需要用偶数位数来表示，如 0x0123 正确，0x123 失败。

```
mysql> SELECT HEX('OceanBase'), HEX(123), HEX(0x0123);
+-----+-----+-----+
| HEX('OceanBase') | HEX(123) | HEX(0x0123) |
+-----+-----+-----+
| 4F63655616E42617365 | 7B      | 0123        |
+-----+-----+-----+
1 row in set (0.00 sec)
```

* INT2IP(int_value)

将一个整数转换成 IP 地址。

- 输入数据类型必须为 **INT**。若输入为 **NULL**，则输出为 **NULL**。若输入的数字大于 **MAX_INT32** 或小于 0 则输出为 **NULL**。
- **int_value** 的内码格式符合“低高高低”的规则。即高位的数据，出现在结果的低位。

```
mysql> SELECT INT2IP(16777216),HEX(16777216),INT2IP(1);
+-----+-----+-----+
| INT2IP(16777216) | HEX(16777216) | INT2IP(1) |
+-----+-----+-----+
| 0.0.0.1          | 1000000       | 1.0.0.0   |
+-----+-----+-----+
1 row in set (0.00 sec)
```

* **IP2INT('ip_addr')**

将字符串表示的 IP 地址转换成整数内码表示。

- 输入数据类型必须为 **VARCHAR**。若输入为 **NULL**，则输出为 **NULL**。若输入的 IP 地址不是一个正确的 IP 地址(包含非数字字符，每一个 ip segment 的数值大小超过 256 等)，则输出为 **NULL**。
- 仅支持 **ipv4** 地址，暂不支持 **ipv6** 地址。
- 内码格式符合“低高高低”的规则。即高位的数据，出现在结果的低位。

```
mysql> SELECT IP2INT('0.0.0.1'),
              HEX(IP2INT('0.0.0.1')),
              HEX(IP2INT('1.0.0.0')),
              IP2INT('1.0.0.257') \G
***** 1. row *****
      IP2INT('0.0.0.1'): 16777216
      HEX(IP2INT('0.0.0.1')): 1000000
      HEX(IP2INT('1.0.0.0')): 1
      IP2INT('1.0.0.257'): NULL
1 row in set (0.00 sec)
```

* **LENGTH(str)**

返回字符串的长度，单位为字节。参数必须是 **VARCHAR** 类型或 **NULL**，否则报错。如果执行成功，结果是 **INT** 型整数，表示字符串长度；当参数是 **NULL** 结果为 **NULL**。

```
mysql> SELECT LENGTH('text');
+-----+
| LENGTH('text') |
+-----+
|                4 |
+-----+
1 row in set (0.00 sec)
```

* **str1 [NOT] LIKE str2 [ESCAPE str3]**

字符串匹配函数。左右参数都必须是 **VARCHAR** 类型或 **NULL**，否则报错。如果执行成功，结果是 **TRUE** 或者 **FALSE**，或某一个参数是 **NULL** 结果就是 **NULL**。

通配符包括“%”和“_”：

- “%”表示匹配任何长度的任何字符，且匹配的字符可以不存在。
- “_”表示只匹配单个字符，且匹配的字符必须存在。

如果你需要查找“a_c”，而不是“abc”时，可以使用 OceanBase 的转义字符“\”，即可以表示为“a_c”。

ESCAPE 用于定义转义符，即表示如果 **str2** 中包含 **str3**，那么在匹配时，**str3** 后的字符为普通字符处理，例如：**LIKE 'abc%' ESCAPE 'c'**，此时“c”为转义符，而“%”为普通字符，不再作为转义字符，本语句匹配的字符串为“ab%”。

```
mysql> SELECT 'ab%' LIKE 'abc%' ESCAPE 'c';
+-----+
| 'ab%' LIKE 'abc%' ESCAPE 'c' |
+-----+
|                               1 |
+-----+
1 row in set (0.00 sec)
```

* **LOWER(str)**

将字符串转化为小写字母的字符。参数必须是 **VARCHAR** 类型。若为 **NULL**，结果总为 **NULL**。

由于中文编码的字节区间与 **ASCII** 大小写字符不重合，对于中文，**UPPER** 可以很好的兼容。

```
mysql> SELECT LOWER('OceanBase 您好！');
+-----+
| LOWER('OceanBase 您好！') |
+-----+
| oceanbase 您好！          |
+-----+
1 row in set (0.02 sec)
```

* **NOW()**

用于获取系统当前时间，精确到微秒。格式为“YYYY-MM-DD HH:MI:SS.SSSSSS”。

```
mysql> SELECT NOW();
+-----+
| NOW()                |
+-----+
| 2014-02-15 09:28:48.674548 |
+-----+
1 row in set (0.00 sec)
```

* **NVL(str1,replace_with)**

如果 *str1* 为 NULL，则替换成 *replace_with*。

```
mysql> SELECT NVL(NULL, 0), NVL(NULL, 'a');
+-----+-----+
| NVL(NULL, 0) | NVL(NULL, 'a') |
+-----+-----+
|          0 | a              |
+-----+-----+
1 row in set (0.00 sec)
```

* STR_TO_DATE(*date*, *format*)

该函数接受一个时间字符串 *date*，将按照 *format* 中指定的格式进行扫描，并将扫描结果转换为 Time 类型返回。

- *date* 为字符串类型参数，其它类型将报错；*date* 的格式可以也是一个时间字符串，同样也支持不严格的时间语法，其具体描述参考 DATE_ADD() 函数中 *date* 参数的描述。
- *format* 的格式如[表 1-3](#)所示。

表 1-3 *format* 格式

格式	含义	返回格式
%b	月份的缩写名称。	Jan, ..., Dec
%c	月份，数字形式。	1, ..., 12
%D	带有英语后缀的日期。	1st, 2nd, ..., 31st
%d	日期，数字形式。	01, ..., 31
%e	日期，数字形式。	1, ..., 31
%f	微秒。	000000, ..., 999999
%H	小时。	00, ..., 23
%h	小时。	01, ..., 12
%l	小时。	01, ..., 12
%i	分钟。	00, ..., 59
%k	小时。	0, ..., 23
%l	小时。	01, ..., 12
%M	月份名称。	January, ..., December

格式	含义	返回格式
%m	月份，数字形式。	01, ..., 12
%p	上午或下午。	AM, PM
%r	12 小时制时间。	hh:mm:ss AM/PM
%S	秒。	00, ..., 59
%s	秒。	00, ..., 59
%T	24 小时制时间。	hh:mm:ss
%Y	年，用四位数字表示。	-

注：“-”表示无。

```
mysql> SELECT STR_TO_DATE('2014-Jan-1st 5:5:5 pm', '%Y-%b-%D %r');
+-----+
| STR_TO_DATE('2014-Jan-1st 5:5:5 pm', '%Y-%b-%D %r') |
+-----+
| 2014-01-01 17:05:05                                |
+-----+
1 row in set (0.00 sec)
```

* **STRICT_CURRENT_TIMESTAMP()**

当系统运行多 MergeServer 时，由于每个 MergeServer 可能存在时间误差，将导致 NOW() 的运算结果不一致，而 STRICT_CURRENT_TIMESTAMP() 从 UpdateServer 上获取时间，从而保证各个 MergeServer 上运行该函数时，获取的时间都是一致的。

```
mysql> select STRICT_CURRENT_TIMESTAMP();
+-----+
| STRICT_CURRENT_TIMESTAMP() |
+-----+
| 2014-02-15 12:27:44.742050   |
+-----+
1 row in set (0.01 sec)
```

* **SUBSTR(str,pos,len)、SUBSTR(str,pos)和 SUBSTR(str FROM pos)**

这三个函数均用于返回一个子字符串。起始于位置 *pos*，长度为 *len*。使用 FROM 的格式为标准 SQL 语法。

- *str* 必须是 VARCHAR，*pos* 和 *len* 必须是整数。任意参数为 NULL，结果总为 NULL。
- *str* 中的中文字符被当做字节流看待。

- 不带有 *len* 参数的时,则返回的子字符串从 *pos* 位置开始到原字符串结尾。
- *pos* 值为负数时, *pos* 的位置从字符串的结尾的字符数起; 为零 0 时, 可被看做 1。
- 当 *len* 小于等于 0, 或者 *pos* 指示的字符串位置不存在字符时, 返回结果为空字符串。

```
mysql> SELECT SUBSTR('abcdefg',3),
               SUBSTR('abcdefg',3,2),
               SUBSTR('abcdefg',-3),
               SUBSTR('abcdefg',3,-2);
```

SUBSTR('abcdefg',3)	SUBSTR('abcdefg',3,2)	SUBSTR('abcdefg',-3)	SUBSTR('abcdefg',3,-2)
cdefg	cd	efg	

1 row in set (0.01 sec)

* TIME_TO_USEC(*date*)

将 OceanBase 的内部时间类型转换成一个微秒数计数。表示 *date* 所指的时刻距离“1970-01-01 00:00:00”的微秒数, 这是一个 UTC 时间, 不带时区信息。

- *date* 为被计算的时刻, 且这个时刻附带时区信息, 而时区信息是用户当前系统设置的时区信息。该参数为 **TIMESTAMP** 类型或者时间格式的字符串。
- **TIME_TO_USEC** 能够接受其它函数的调用结果作为参数, 但是其的结果类型必须为 **TIMESTAMP** 或者时间格式的字符串。
- 该函数返回值为微秒计数, 返回类型为 **INT**。

```
mysql> SELECT TIME_TO_USEC('2014-03-25'), TIME_TO_USEC(now());
```

TIME_TO_USEC('2014-03-25')	TIME_TO_USEC(now())
1395676800000000	1395735415207794

1 row in set (0.00 sec)

* TRIM([[{**BOTH** | **LEADING** | **TRAILING**}] [*remstr*] FROM] *str*)

删除字符串所有前缀和（或）后缀。

- *remstr* 和 *str* 必须为 **VARCHAR** 或 **NULL** 类型。当参数中有 **NULL** 时结果总为 **NULL**。
- 若未指定 **BOTH**、**LEADING** 或 **TRAILING**, 则默认为 **BOTH**。
- *remstr* 为可选项, 在未指定情况下, 删除空格。

```
mysql> SELECT TRIM(' bar '),
              TRIM(LEADING 'x' FROM 'xxxbarxxx'),
              TRIM(BOTH 'x' FROM 'xxxbarxxx'),
              TRIM(TRAILING 'x' FROM 'xxxbarxxx') \G
***** 1. row *****
              TRIM(' bar '): bar
              TRIM(LEADING 'x' FROM 'xxxbarxxx'): barxxx
              TRIM(BOTH 'x' FROM 'xxxbarxxx'): bar
              TRIM(TRAILING 'x' FROM 'xxxbarxxx'): xxxbar
1 row in set (0.00 sec)
```

* UNHEX(*str*)

HEX(*str*)的反向操作，即将参数中的每一对十六进制数字理解为一个数字，并将其转化为该数字代表的字符。结果字符以二进制字符串的形式返回。

str 必须是 VARCHAR 或 NULL。当 *str* 是合法的十六进制值时将按照十六进制到字节流的转换算法进行，当 *str* 不是十六进制字符串的时候返回 NULL。当 *str* 为 NULL 的时候输出是 NULL。

```
mysql> SELECT HEX('OceanBase'),
              UNHEX('4f6365616e42617365'),
              UNHEX(HEX('OceanBase')),
              UNHEX(NULL) \G
***** 1. row *****
              HEX('OceanBase'): 4F6365616E42617365
              UNHEX('4f6365616e42617365'): OceanBase
              UNHEX(HEX('OceanBase')): OceanBase
              UNHEX(NULL): NULL
1 row in set (0.00 sec)
```

* UPPER(*str*)

将字符串转化为大写字母的字符。参数必须是 VARCHAR 类型。若为 NULL，结果总为 NULL。

由于中文编码的字节区间与 ASCII 大小写字符不重合，对于中文，UPPER 可以很好的兼容。

```
mysql> SELECT UPPER('OceanBase 您好! ');
+-----+
| UPPER('OceanBase 您好! ') |
+-----+
| OCEANBASE 您好!          |
+-----+
1 row in set (0.00 sec)
```

* USEC_TO_TIME(*usec*)

该函数为 TIME_TOUSEC(*date*)的逆函数，表示“1970-01-01 00:00:00”增加 *usec* 后的时间，且附带了时区信息。例如在东八区调用该函数“USEC_TO_TIME(1)”，返回值为“1970-01-01 08:00:01”。

- *usec* 为一个微秒计数值。
- 返回值为 **TIMESTAMP** 类型。

```
mysql> SELECT USEC_TO_TIME(1);
+-----+
| USEC_TO_TIME(1) |
+-----+
| 1970-01-01 08:00:00.000001 |
+-----+
1 row in set (0.00 sec)
```

1.3.2 聚集函数

介绍 OceanBase 支持的聚集函数的格式和用法。

在 OceanBase 的聚集函数中，Value 表达式只能出现一个。例如：不支持 COUNT(c1, c2)，仅支持 COUNT(c1)。

* AVG()

返回指定组中的平均值，空值被忽略。

假设表 a 有三行数据：id=1, num=10; id=2, num=20; id=3, num=30。

```
mysql> SELECT AVG(num) from a;
+-----+
| AVG(num) |
+-----+
|      20 |
+-----+
1 row in set (0.01 sec)
```

* COUNT()

返回指定组中的行数。

假设表 a 有三行数据：id=1, num=10; id=2, num=20; id=3, num=30。

```
mysql> SELECT COUNT(num) FROM a;
+-----+
| COUNT(num) |
+-----+
|          3 |
+-----+
1 row in set (0.00 sec)
```

* MAX()

返回指定数据中的最大值。

假设表 a 有三行数据：id=1, num=10; id=2, num=20; id=3, num=30。


```
mysql> SELECT MAX(num) FROM a;
```

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

```
| MAX(num) |
```

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

```
|      30 |
```

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

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

* MIN()

返回指定数据中的最小值。

假设表 **a** 有三行数据：id=1, num=10; id=2, num=20; id=3, num=30。

```
mysql> SELECT MIN(num) FROM a;
```

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

```
| MIN(num) |
```

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

```
|      10 |
```

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

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

* SUM()

返回指定组中的和。

假设表 **a** 有三行数据：id=1, num=10; id=2, num=20; id=3, num=30。

```
mysql> SELECT SUM(num) FROM a;
```

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

```
| SUM(num) |
```

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

```
|      60 |
```

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

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

1.4 运算符和优先级

主要介绍 OceanBase 支持多种类型的运算符，主要包括算数运算符、比较运算符、向量比较运算符、逻辑运算符和位运算符，以及运算的优先级。

1.4.1 逻辑运算符

在 OceanBase 中，逻辑操作符会把左右操作数都转成 **BOOL** 类型进行运算。逻辑运算时返回“Error”表示计算错误。

OceanBase 各数据类型转换 **BOOL** 类型的规则如下：

- 字符串只有是“True”、“False”、“1”和“0”才能够转换到 **BOOL** 类型，其中字符串“True”和“1”为“True”，字符串“False”和“0”为“False”。
- “INT”、“FLOAT”、“DOUBLE”和“DECIMAL”转换 **BOOL** 类型时，数值不为零时为“True”，数值为零时为“False”。

* NOT

逻辑非，操作类型对照表如下：

INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
True/False	True/False	True/False	Error	True/False/Error	True/False	NULL

```
mysql> SELECT NOT 0, NOT 1, NOT NULL;
```

```
+-----+-----+-----+
| NOT 0 | NOT 1 | NOT NULL |
+-----+-----+-----+
|    1 |    0 |   NULL |
+-----+-----+-----+
1 row in set (0.00 sec)
```

* AND

逻辑与，操作类型对照表如下：

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	True/False	True/False	True/False	Error	True/False/Error	True/False	False/NULL
FLOAT		True/False	True/False	Error	True/False/Error	True/False	False/NULL
DOUBLE			True/False	Error	True/False/Error	True/False	False/NULL
TIMESTAMP				Error	Error	True/False	Error
VARCHAR					True/False/Error	True/False/Error	False/NULL
BOOL						True/False	False/NULL
NULL							NULL

```
mysql> SELECT (0 AND 0), (0 AND 1), (1 AND 1), (1 AND NULL);
```

```
+-----+-----+-----+-----+
| (0 AND 0) | (0 AND 1) | (1 AND 1) | (1 AND NULL) |
+-----+-----+-----+-----+
|        0 |        0 |        1 |        NULL |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

* OR

逻辑或，操作类型对照表如下：

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	True/False	True/False	True/False	Error	True/False/Error	True/False	True/NULL
FLOAT		True/False	True/False	Error	True/False/Error	True/False	True/NULL
DOUBLE			True/False	Error	True/False/Error	True/False	True/NULL
TIMESTAMP				Error	Error	Error	Error
VARCHAR					True/False/Error	True/False/Error	True/NULL
BOOL						True/False	True/NULL
NULL							NULL

```
mysql> SELECT (0 OR 0), (0 OR 1), (1 OR 1), (1 AND NULL);
```

```
+-----+-----+-----+-----+
| (0 OR 0) | (0 OR 1) | (1 OR 1) | (1 AND NULL) |
+-----+-----+-----+-----+
|      0 |      1 |      1 |      NULL |
+-----+-----+-----+-----+
```

```
1 row in set (0.01 sec)
```

1.4.2 算数运算符

OceanBase 中，数值计算只允许在数值类型和 VARCHAR 直接进行，其它类型直接报错。字符串在做算术运算时，如果无法转成 DOUBLE 类型则报错，比如“3.4he' + 3”。字符串只有在内容全为数字或者开头是“+”或者“-”，且后面跟数字的形式才能转成 DOUBLE 型。

OceanBase 支持的运算符如[表 1-4](#)所示。

表 1-4 算数运算符

表达式	含义	举例
+	加法。	SELECT 2+3;
-	减法。	SELECT 2-3;

表达式	含义	举例
*	乘法。	SELECT 2*3;
/	除法，返回商。如果除数为“0”，则返回结果为“NULL”。	SELECT 2/3;
%或 MOD	除法，返回余数。如果除数为“0”，则返回结果为“NULL”	SELECT 2%3, 2 MOD 3;

“+”、“-”、“*”的操作类型对照表如下：

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	INT	DOUBLE	DOUBLE	Error	DOUBLE/Error	Error	NULL
FLOAT		DOUBLE	DOUBLE	Error	DOUBLE/Error	Error	NULL
DOUBLE			DOUBLE	Error	DOUBLE/Error	Error	NULL
TIMESTAMP				Error	Error	Error	Error
VARCHAR					DOUBLE/Error	Error	NULL/Error
BOOL						Error	Error
NULL							NULL

“/”的操作类型对照表如下，如果除数为零则报错：

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	DOUBLE/Error	DOUBLE/Error	DOUBLE/Error	Error	DOUBLE/Error	Error	NULL/Error
FLOAT		DOUBLE/Error	DOUBLE/Error	Error	DOUBLE/Error	Error	NULL/Error
DOUBLE			DOUBLE/Error	Error	DOUBLE/Error	Error	NULL/Error
TIMESTAMP				Error	Error	Error	Error

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
VARCHAR					DOUBLE/Error	Error	NULL/Error
BOOL						Error	Error
NULL							NULL

“%”和“MOD”的操作类型对照表如下：

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	INT	DOUBLE	DOUBLE	Error	DOUBLE/Error	Error	NULL
FLOAT		DOUBLE	DOUBLE	Error	DOUBLE/Error	Error	NULL
DOUBLE			DOUBLE	Error	DOUBLE/Error	Error	NULL
TIMESTAMP				Error	Error	Error	Error
VARCHAR					DOUBLE/Error	Error	NULL/Error
BOOL						Error	Error
NULL							NULL

1.4.3 比较运算符

OceanBase 比较的策略是，先将操作数转换为相同的类型，然后进行比较。所有比较运算符的返回类型为 **BOOL** 或者 **NULL**。比较结果为真则返回“1”，为假则返回“0”，不确定则返回“NULL”。

* 数值比较

比较运算符用于比较两个数值的大小。OceanBase 支持的运算符如[表 1-5](#)所示。

表 1-5 比较运算符

表达式	含义	举例
=	等于。	SELECT 1=0, 1=1, 1=NULL;
>=	大于等于。	SELECT 1>=0, 1>=1, 1>=2, 1>=NULL;

表达式	含义	举例
>	大于。	SELECT 1>0, 1>1, 1>2, 1>NULL;
<=	小于等于。	SELECT 1<=0, 1<=1, 1<=2, 1<=NULL;
<	小于。	SELECT 1<0, 1<1, 1<2, 1<NULL;
!=或<>	不等于。	SELECT 1!=0, 1!=1, 1<>0, 1<>1, 1!=NULL, 1<>NULL;

数值比较运算符的转换规则如下表。

	INT	FLOAT	DOUBLE	TIMESTAMP	VARCHAR	BOOL	NULL
INT	INT	FLOAT	DOUBLE	Error	?INT	Error	NULL
FLOAT		FLOAT	DOUBLE	Error	?FLOAT	Error	NULL
DOUBLE			DOUBLE	Error	?DOUBLE	Error	NULL
TIMESTAMP				TIMESTAMP	?TIMESTAMP	Error	NULL
VARCHAR					VARCHAR	?BOOL	NULL
BOOL						BOOL	NULL
NULL							NULL

注:

- “?”开头的类型表示会在执行的时候尝试转换到指定类型，如果转换失败则报错。
- *BOOL* 类型比较时，*False* 小于 *True*。

* [NOT] BETWEEN ... AND ...

判断是否存在或者不存在于指定范围。

```
mysql> SELECT 2 BETWEEN 1 AND 2,
              3 NOT BETWEEN 1 AND 2,
              1 BETWEEN null AND 0,
              1 NOT BETWEEN null AND 0 \G
***** 1. row *****
      2 BETWEEN 1 AND 2: 1
      3 NOT BETWEEN 1 AND 2: 1
      1 BETWEEN null AND 0: 0
      1 NOT BETWEEN null AND 0: 1
1 row in set (0.00 sec)
```

* [NOT] IN

判断是否存在于指定集合。

```
mysql> SELECT 2 IN (1, 2), 3 IN (1, 2);
+-----+-----+
| 2 IN (1, 2) | 3 IN (1, 2) |
+-----+-----+
|          1 |          0 |
+-----+-----+
1 row in set (0.00 sec)
```

* IS [NOT] NULL | TRUE | FALSE | UNKNOWN

判断是否为 NULL、真、假或未知。如果执行成功，结果是 True 或者 False，不会出现 NULL。



注意：

左参数必须是 BOOL 类型，或者是 NULL，否则报错。

```
mysql> SELECT 0 IS NULL,
              NULL IS NULL,
              NULL IS TRUE,
              (0>1) IS FALSE,
              NULL IS UNKNOWN,
              0 IS NOT NULL,
              NULL IS NOT NULL,
              NULL IS NOT TRUE,
              (0>1) IS NOT FALSE,
              NULL IS NOT UNKNOWN \G
***** 1. row *****
      0 IS NULL: 0
      NULL IS NULL: 1
      NULL IS TRUE: 0
      (0>1) IS FALSE: 1
      NULL IS UNKNOWN: 1
      0 IS NOT NULL: 1
      NULL IS NOT NULL: 0
      NULL IS NOT TRUE: 1
      (0>1) IS NOT FALSE: 0
      NULL IS NOT UNKNOWN: 0
1 row in set (0.00 sec)
```

1.4.4 向量比较运算符

向量比较运算符对两个向量（ROW）进行比较，支持“<”、“>”、“=”、“<=”、“>=”、“!=”、“in”和“not in”等 8 个操作符。这几个操作符都是二元操作符，被比较的两个向量的维度要求相同。

向量比较操作符和普通操作符相比主要有两点不同：

- 如果两个操作数的第 i 个标量的比较就决定了比较结果，则不再继续比较后续的数值。
- 在向量比较中，NULL 等于 NULL，且小于其他普通值。例如 `NULL = NULL < " < 'a'`。除此以外，标量值之前的比较和类型转换的规则同普通比较操作符相同。

向量比较操作符需要注意以下几点：

- 多元向量比较，关键字 ROW 可以省略。例如，`ROW(1,2,3) < ROW(1,3,5)` 等价于 `(1,2,3) < (1,3,4)`。
- 1 元向量比较，关键字 ROW 不能省略。例如，`ROW(1) < ROW(2)` 不等价于 `(1) < (2)`。前者是向量比较操作，后者是普通比较操作。
- in/not in 操作一定是向量操作，表示左参数（不）在右集合内，集合用“（）”括起。例如，`Row(1) in (Row(2), Row(3), Row(1))`，等价与 `1 in (2, 3, 1)`。
- in/not in 操作需要左右操作数对应位置的标量都是可以比较的，否则返回错误。例如，`ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4))`成功，`ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(1,3,4))`失败。


```
mysql> SELECT ROW(1,2) < ROW(1, 3),
        ROW(1,2,10) < ROW(1, 3, 0),
        ROW(1,null) < ROW(1,0),
        ROW(null) = ROW(null),
        ROW(null, 1) < ROW(null, 2),
        Row(1) in (Row(2), Row(3), Row(1)),
        ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)),
        1 in (1,2,3),
        1 not in (2,3,4),
        ROW(1,2) not in (ROW(2,1),ROW(2,3), ROW(3,4)) \G
***** 1. row *****
          ROW(1,2) < ROW(1, 3): 1
        ROW(1,2,10) < ROW(1, 3, 0): 1
          ROW(1,null) < ROW(1,0): 1
          ROW(null) = ROW(null): 1
        ROW(null, 1) < ROW(null, 2): 1
        Row(1) in (Row(2), Row(3), Row(1)): 1
ROW(1,2) in (ROW(1,2), ROW(2,3), ROW(3,4), ROW(4,5)): 1
          1 in (1,2,3): 1
          1 not in (2,3,4): 1
        ROW(1,2) not in (ROW(2,1), ROW(2,3), ROW(3,4)): 1
1 row in set (0.00 sec)
```

1.4.5 拼接运算符

OceanBase 支持的运算符如[表 1-6](#)所示。

表 1-6 拼接运算符

表达式	含义	举例
	将值联结到一起构成单个值。	SELECT 'a' 'b';

1.4.6 优先级

当我们需要对 OceanBase 的操作符进行混合运算时，我们需要了解这些操作符的优先级。

OceanBase 中操作符的优先级由高到低，如[表 1-7](#)所示。

表 1-7 优先级

优先级	运算符
1	*, /, %, MOD
2	+, -
3	=, >, >=, <, <=, <>, !=, IS, LIKE, IN

优先级	运算符
4	BETWEEN
5	NOT
6	AND
7	OR



小窍门：

在实际运用的时，我们可以用“()”将需要优先的操作括起，这样既起到优先操作的作用，又使其他用户便于阅读。

1.5 转义字符

转义字符是在字符串中，某些序列前添加反斜线“\”，用于表示特殊含义。转义字符对大小写敏感。例如“\b”表示退格，而“\B”还是表示“B”。

OceanBase 识别的转义字符如[表 1-8](#)所示。

表 1-8 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。

转义字符	含义
\%	%字符。
\0	空字符(NULL)。

1.6 内部表

OceanBase 内部表都以 “__” 开头，普通用户表请不要使用这种格式的名字。

原则上，OceanBase 开发人员保留在不同版本间增删内部表和修改它们

Schema 的权利，所以普通用户和应用程序请不要依赖于这些表的任何内容。

我们可以使用 **SHOW ALL TABLES** 命令查看数据库中有哪些表，如[图 1-1](#)所示。

图 1-1 内部表

```
mysql> show all tables;
+-----+
| table_name |
+-----+
| __first_root_table |
| __all_table |
| __all_column |
| __all_join_info |
| __all_ddl_operation |
| __all_sys_stat |
| __all_sys_param |
| __all_user |
| __all_table_privilege |
| __all_cluster |
| __all_trigger_event |
| __all_client |
| __all_sys_config |
| __all_sys_config_stat |
| __all_server |
| __all_server_stat |
| __all_server_session |
| __all_statement |
| __rootserver_ups_stat |
| __rootserver_sstable_dist |
| __rootserver_ms_stat |
| __rootserver_cs_stat |
| __rootserver_status |
| __updateserver_session_info |
| __updateserver_memtable_info |
| __chunkserver_tablet_image_info |
+-----+
26 rows in set (0.00 sec)
```

2 数据定义语言

DDL（Data Definition Language）的主要作用是建立数据库基本组件的，例如建立表，删除表和修改表等。

OceanBase 支持的 DDL 主要有 CREATE TABLE，DROP TABLE，ALTER TABLE，CREATE INDEX，DROP INDEX 和 ALTER SYSTEM。

2.1 CREATE TABLE 语句

该语句用于在 OceanBase 数据库中创建新表。

* 格式

```
CREATE TABLE [IF NOT EXISTS] table_name
    (column_name data_type [NOT NULL | NULL] [DEFAULT default_value]
    [AUTO_INCREMENT],
    ...,
    PRIMARY KEY (column_name1, column_name2...)
    [table_options_list];
```



注意：

OceanBase 内部数据以 b 树为索引，按照 Primary Key 排序，因此建表的时候，必须指定 Primary Key。Primary Key 有两种指定方式，详细请参见“举例”部分。另外，OceanBase 不允许用户创建只有主键列的表。

- 使用“IF NOT EXISTS”时，即使创建的表已经存在，也不会报错，如果不指定时，则会报错。
- “*data_type*”请参见“1.2 数据类型”章节。
- NOT NULL，DEFAULT，AUTO_INCREMENT 用于列的完整性约束。其中 AUTO_INCREMENT 功能暂时没有实现。
- “*table_option_list*”内容请参见[表 2-1](#)，各子句间用“,”隔开。

表 2-1 表选项

参数	含义	举例
CHARACTER SET	指定该表所有字符串的编码，用于对外提供元数据信息。目前仅支持 UTF-8。	CHARACTER SET = 'utf8'

参数	含义	举例
COMMENT	添加注释信息。	COMMENT='create by Bruce'
COMPRESS_METHOD	<p>存储数据时使用的压缩方法名，目前提供的方法有以下三种：</p> <ul style="list-style-type: none"> • none（默认值，表示不作压缩） • lz4_1.0 • lzo_1.0 • snappy_1.0 	COMPRESS_METHOD = 'none'
CONSISTENT_MODE	<p>指定表的默认一致性级别。</p> <ul style="list-style-type: none"> • static • frozen • weak • strong 	CONSISTENT_MODE = 'static'
EXPIRE_INFO	<p>在 UpdateServer 中的动态数据和 ChunkServer 中的静态数据合并时，满足表达式的行会自动删除。</p> <p>一般可用于自动删除过期数据。</p>	EXPIRE_INFO = '\$SYS_DATE > c1 + 24*60*60'，表示自动删除 c1 列的值比当前时间小 24 小时的行，其中“24*60*60”表示过期的微秒数。
JOIN_INFO	<p>假设创建表 a 时 JOIN_INFO 表 b，那么当 a.a1=b.b1 时，a.a2=b.b2。</p> <p>其中 a.a1 和 b.b1 必须为主键列。</p>	<p>JOIN_INFO = '[a1\$b1]%b:a2\$b2'</p> <p>详细例子可参见“例 2”。</p>
REPLICA_NUM	这个表的 Tablet 副本数，默认值为 3。	REPLICA_NUM = 3
TABLE_ID	指定表的 ID。如果指定的 table_id 小于 1000，需要打开 RootServer 的配置项开关“ddl_system_table_switch”。	TABLE_ID = 4000

参数	含义	举例
TABLET_BLOCK_SIZE	设置组成 Tablet 的宏块的大小。	一般设置为 2M。
TABLET_MAX_SIZE	这个表的 Tablet 最大尺寸，单位为字节，默认为 256MB。	TABLET_MAX_SIZE = 268435456
USE_BLOOM_FILTER	对本表读取数据时，是否使用 Bloom Filter。 <ul style="list-style-type: none"> 0: 默认值，不使用。 1: 使用。 	USE_BLOOM_FILTER = 0

* 举例

例 1:

1. 执行以下命令，创建数据库表。
CREATE TABLE test (c1 int primary key, c2 varchar)
REPLICA_NUM = 3, COMPRESS_METHOD = 'none';
或者
CREATE TABLE test (c1 int, c2 varchar, primary key(c1))
REPLICA_NUM = 3, COMPRESS_METHOD = 'none';
2. 执行命令查看表信息，如[图 2-1](#)所示。
SHOW tables;
DESCRIBE test;

图 2-1 CREATE TABLE

```
mysql> SHOW tables;
+-----+
| table_name |
+-----+
| test      |
+-----+
1 row in set (0.00 sec)

mysql> DESCRIBE test;
+-----+-----+-----+-----+-----+-----+
| field | type          | nullable | key | default | extra |
+-----+-----+-----+-----+-----+-----+
| c1    | int           | YES     | 1  | NULL    |      |
| c2    | varchar(65536) | YES     | 0  | NULL    |      |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

例 2:

1. 执行以下命令，创建表 b。
CREATE TABLE b(b1 INT PRIMARY KEY,b2 INT);
2. 执行以下命令，创建表 a，并连接表 b。
CREATE TABLE a(a1 INT PRIMARY KEY, a2 INT)
JOIN_INFO='[a1\$b1]%'b:a2\$b2';

3. 执行以下命令，向表 **b** 中插入数据。
INSERT INTO b VALUES(1,2);
4. 执行以下命令，向表 **a** 插入 **a1** 列数据。
INSERT INTO a(a1) VALUES(1);
5. 执行以下命令，查看表 **a** 中 **a2** 有数据且等于 **b2**。
SELECT * FROM a;

2.2 ALTER TABLE 语句

该语句用于修改已存在的表的设计。



小窍门：

当执行多个 ALTER TABLE 语句时，可以进行合并，详细请参见举例中的“例 2”。

* 增加列

```
ALTER TABLE table_name
  ADD [COLUMN] column_name data_type
  [NOT NULL | NULL]
  [DEFAULT default_value]
  [PRIMARY KEY]
```

- 如果使用了 NOT NULL 属性则必须指定 DEFAULT。
- *data_type* 请参见“1.2 数据类型”。

* 修改列属性

```
ALTER TABLE table_name
  ALTER [COLUMN] column_name
  [SET NOT NULL | DROP NOT NULL | DROP DEFAULT | SET
  DEFAULT default_value]
```

不允许将某一列的属性从 NULL 改为 NOT NULL。

* 删除列

```
ALTER TABLE table_name
  DROP [COLUMN] column_name
  [CASCADE | RESTRICT]
```

- 不允许删除主键列或者包含索引的列。
- 参数 CASCADE 和 RESTRICT，暂时不起作用。

* 表重命名

```
ALTER TABLE table_name
    RENAME TO new_table_name
```

* 列重命名

```
ALTER TABLE table_name
    RENAME [COLUMN] column_name
    TO new_column_name
```

* 设置过期数据删除

```
ALTER TABLE table_name
    SET EXPIRE_INFO [=] 'express_string'
```

如果一个数据表中包含了若干索引，那么通过 **expire info** 指定的列将必须在主表和所有索引表中都存在，不然会导致数据的不一致，

例如：有数据表 t1(pk, c1, c2, c3, PK<pk>), 索引表 idx1 在 c2 上有索引 idx1(c2, pk, PK<c2, pk>), 索引表 idx2 在 c3 上有索引 idx2(c3, pk, PK<c3, pk>)。

那么 ALTER TABLE t1 SET EXPIRE_INFO c2 > 1, 将会失败。因为 c2 在索引表 idx2 中不存在。

* 设置该表 **Tablet** 最大尺寸

```
ALTER TABLE table_name
    SET TABLET_MAX_SIZE [=] tablet_max_size
```

* 设置组成 **Tablet** 的宏块大小

```
ALTER TABLE table_name
    SET TABLET_BLOCK_SIZE [=] tablet_block_size
```

* 设置该表的副本数

```
ALTER TABLE table_name
    SET REPLICA_NUM [=] replica_num
```

* 设置该表的压缩方式

- ALTER TABLE *table_name*
SET COMPRESS_METHOD [=] {'none' | 'lz4_1.0' | 'lzo_1.0' | 'snappy_1.0'}

* 设置是否使用 **BloomFilter**set

```
ALTER TABLE table_name
    SET USE_BLOOM_FILTER [=] {0 | 1}
```


其中 0 表示关闭，1 表示使用。

* 设置一致性类型

```
ALTER TABLE table_name
    SET CONSISTENT_MODE [=] {static | frozen | weak | strong}
```

* 设置注释信息

```
ALTER TABLE table_name
    SET COMMENT [=] 'comment_string'
```

* 举例

例 1:

1. 增加列前，执行以下命令查看表信息，如[图 2-2](#)所示。

DESCRIBE test;

图 2-2 ADD 前

field	type	nullable	key	default	extra
c1	int		1	NULL	
c2	varchar(-1)		0	NULL	

2 rows in set (0.00 sec)

2. 执行以下命令增加 c3 列。

ALTER TABLE test ADD c3 int;

3. 增加列后，执行以下命令查看表信息，如[图 2-3](#)所示。

DESCRIBE test;

图 2-3 ADD 后

field	type	nullable	key	default	extra
c1	int		1	NULL	
c2	varchar(-1)		0	NULL	
c3	int		0	NULL	

3 rows in set (0.00 sec)

4. 执行以下命令删除 c3 列。

ALTER TABLE test DROP c3;

5. 删除列后，执行以下命令查看表信息，如[图 2-4](#)所示。

DESCRIBE test;

图 2-4 DROP 后

field	type	nullable	key	default	extra
c1	int		1	NULL	
c2	varchar(-1)		0	NULL	

2 rows in set (0.00 sec)

例 2:

当执行多个 ALTER TABLE 语句时，可以进行合并。

```
ALTER TABLE
test
SET REPLICA_NUM=2,
ADD COLUMN c5 int;
```

2.3 DROP TABLE 语句

该语句用于删除 OceanBase 数据库中的表。

* 格式

DROP TABLE [IF EXISTS] *table_name*;

- 使用“IF EXISTS”时，即使要删除的表不存在，也不会报错，如果不指定时，则会报错。
- 同时删除多个表时，用“,”隔开。

* 举例

```
DROP TABLE IF EXISTS test;
```

2.4 CREATE INDEX 语句

索引是创建在表上的，对数据库表中的一列或多列的值进行排序的一种结构。其作用主要在于提高查询的速度，降低数据库系统的性能开销。

OceanBase 中，新创建的索引需要等待一次每日合并后，才生效。

* 格式

```
CREATE INDEX index_name
ON table_name (columnname_list);
```

- *columnname_list* 中，每个列名后都要制定 ASC（升序）或者 DESC（降序）。若不指定，默认为升序。
- 本语句建立索引的排列方式为：首先以 *columnname_list* 中第一个列的值排序；该列值相同的记录，按下一列名的值排序；以此类推。

- 执行“**SHOW INDEX FROM *table_name***”可以查看创建的索引。

* 举例

1. 执行以下命令，创建表 `test`。
CREATE TABLE test (c1 int primary key, c2 varchar);
2. 执行以下命令，创建表 `test` 的索引。
CREATE INDEX test_index ON test (c1, c2 DESC);
3. 执行以下命令，查看表 `test` 的索引。
SHOW INDEX FROM test;

2.5 DROP INDEX 语句

当索引过多时，维护开销增大，因此，需要删除不必要的索引。

* 格式

```
DROP INDEX index_name
      ON table_name;
```

* 举例

```
DROP INDEX test_index ON test;
```

2.6 ALTER SYSTEM 语句

ALTER SYSTEM 语句主要对 OceanBase 发送命令，执行某项指定操作。

* 主备 RootServer 切换

```
ALTER SYSTEM [FORCE]
      SWITCH ROOTSERVER {MASTER|SLAVE}
      [SERVER='ip:port']
```

- 主 RootServer 切备
 - 必须指定主 RootServe 的 IP 和 PORT 进行切备。如果不指定，直接报错退出。
 - 如果集群中不存在主 RootServer，则不进行切换。
- 备 RootServer 切主
 - 可以指定备 RootServe 的 IP 和 PORT 进行切主。如果不指定，则从集群中任意选择备 RootServer 进行切主。
 - 如果集群中不存在备 RootServer，则直接报错退出。
 - 如果集群中已经有主 RootServer，则须使用 **FORCE** 参数进行强制切换，否则将报错退出。

* 主备 UpdateServer 切换

```
ALTER SYSTEM [FORCE]
    SWITCH UPDATESERVER {MASTER|SLAVE}
    [SERVER='ip:port']
```

- 主 UpdateServer 切备

不进行任何操作，也不存在什么场景需要使用这个命令。假设集群中存在两个主 RootServer 和两个主 UpdateServerS。只需要将主 RootServer 设置成备，那么主 UpdateServer 也会因为 Lease 失效而变成备 UpdateServer。

- 备 UpdateServer 切主

- 集群中存在主 UpdateServer

不容许添加 FORCE 标志符。

可以指定备 UpdateServer 的 IP 和 PORT 进行切主。如果不指定，则从集群中任意选择备 UpdateServer 进行切主。

- 集群中不存在主 UpdateServer

必须加 FORCE 标志符，并且必须指定新主 UpdateServer 的地址。

* 修改集群状态

```
ALTER SYSTEM [FORCE]
    {START|STOP} CLUSTER
    CLUSTER_ID=cluster_id
```

- 修改 cluster_id 对应的集群的状态，同时通知 RootServer 重新加载集群信息。主要在新增或者升级的时候需要用到该语法。
- 可以在 __all_cluster 表中看到集群相应的状态，列名为 status，其中 0 表示启动中，1 表示可以提供服务，2 表示停止服务，3 表示无效集群。
- 当集群中包含主 RootServer 或者主 UpdateServer 时，不容许执行 STOP 命令，用户只能先进行 RootServer/UpdateServer 的切换以后，再进行 STOP。

* 重新加载集群信息和每日合并信息

```
ALTER SYSTEM
    RELOAD CLUSTER
```

通知 RootServer 重新加载集群信息和每日合并信息。

* 清除每日合并过程报错

```
ALTER SYSTEM
    CLEAN MERGE ERROR
```

通知 RootServer 清除每日合并过程中产生的错误，否则，RootServer 将不断进行每日合并错误的报警提醒。

* 暂停每日合并

```
ALTER SYSTEM
    PAUSE MERGE CLUSTER
    CLUSTER_ID=cluster_id
```

- 暂停指定集群的 ChunkServer 的每日合并。
- RootServer 的每日合并过程不受这个控制。

* 继续每日合并

```
ALTER SYSTEM
    RESUME MERGE CLUSTER
    CLUSTER_ID=cluster_id
```

- 继续指定集群的 ChunkServer 的每日合并。
- RootServer 的每日合并过程不受这个控制。

* 流量配置

```
ALTER SYSTEM [FORCE]
    SWITCH FLOW
    FLOW_PERCENT=percent
    CLUSTER_ID=cluster_id
```

- 设置指定集群的流量百分比。
- *percent* 的取值范围[0, 100]。

* 上下线管理

```
ALTER SYSTEM
    [CANCEL]
    {SHUTDOWN | RESTART}
    SERVER 'ip:port'
    [CLUSTER_ID=cluster_id]
```

- 通知 RootServer 指定集群的某个 Server 上下线。
- 如果 SHUTDOWN 命令用来通知 RootServer 某个 ChunkServer 需要下线，RootServer 会尽量把这个 ChunkServer 上的 Tablet 迁移到其他机器。是否迁移结束可以在__all_trigger 表中查询。
- RESTART 命令不常用，实现也很简单，只是通知 RootServer 修改 ChunkServer 的一个标志位，不做任何其他工作。

* 创建空 Tablet

```
ALTER SYSTEM
  CREATE {ROOT | META | NORMAL} TABLET
    RANGE 'tablet_range'
    VERSION ver_num
    {SERVER = 'ip:port' | CLUSTER_ID=cluster_id | SERVER =
      'ip:port' CLUSTER_ID=cluster_id}
```

- 在 ChunkServer 上面创建空 Tablet, 需要指定 Tablet 的 Range 和 Version。
- *tablet_range* 的格式为 “table_id:1001,(start_key;end_key)” ; key 的形式为 “type:value”。

* 删除 Tablet

```
ALTER SYSTEM
  DROP TABLET
  RANGE 'tablet_range'
  VERSION ver_num
  SERVER = 'ip:port' [CLUSTER_ID=cluster_id]
```

删除指定 ChunkServer 上的 Tablet, 必须指定 Tablet 的 Range。

* 迁移/复制 Tablet

```
ALTER SYSTEM
  {MIGRATE|COPY} TABLET
  RANGE 'tablet_range'
  VERSION ver_num
  SOURCE='ip:port'
  DESTINATION='ip:port'
  [CLUSTER_ID=cluster_id]
```

迁移或者复制 Tablet, 需要指定源 ChunkServer 和目的 ChunkServer, 以及 Tablet 的 Range 和 Version。

* 汇报 Tablet

```
ALTER SYSTEM
  REPORT TABLET
  {SERVER = 'ip:port' CLUSTER_ID=cluster_id | SERVER = 'ip:port'
  [CLUSTER_ID=cluster_id]
```

强制要求某个 ChunkServer 或者某个集群内的 ChunkServer 进行 Tablet 汇报。

* 安装磁盘

```
ALTER SYSTEM
  INSTALL DISK 'disk_mount_point'
  SERVER = 'ip:port'
  [CLUSTER_ID=cluster_id]
```

- 指定 ChunkServer 安装磁盘。
- *disk_mount_point* 为磁盘挂载点，如“/data/3”。

* 卸载磁盘

```
ALTER SYSTEM
    UNINSTALL DISK disk_no
    SERVER='ip:port'
    [CLUSTER_ID=cluster_id]
```

- 指定 ChunkServer 和磁盘号，卸载磁盘。
- *disk_no* 为需要卸载的磁盘号。

* 冻结 UpdateServer 内存表

```
ALTER SYSTEM
    {MINOR FREEZE | MAJOR FREEZE}
```

- 冻结 UpdateServer 内存表分大版本冻结和小版本冻结。
- 小版本冻结达到一定次数后会触发大版本冻结。小版本冻结次数由配置项“minor_num_limit”控制。
- 大版本冻结会引起每日合并。

* 删除 UpdateServer 内存表

```
ALTER SYSTEM
    DROP MEMTABLE
    SERVER = 'ip:port'
    [CLUSTER_ID=cluster_id]
```

* 清除 UpdateServer 内存

```
ALTER SYSTEM
    CLEAR ACTIVE MEMTABLE
    SERVER = 'ip:port'
    [CLUSTER_ID=cluster_id]
```

清除 UpdateServer 中活跃内存表所占的内存。

* 结束本地 Session

```
ALTER SYSTEM
    KILL [GLOBAL|LOCAL] SESSION 'session_id'
    SERVER = 'ip:port'
    [CLUSTER_ID=cluster_id]
```

结束 MergeServer 或者 UpdateServer 的本地连接。

* 添加/删除 **UpdateServer** 服务器

```
ALTER SYSTEM
    {ADD|DELETE} UPDATESERVER
    SERVER='ip:port;ip:port'
```

* 强制切换 **Logfile**

```
ALTER SYSTEM
    SWITCH LOGFILE
```

强制切换 UpdateServer 的 CommitLog，即当前 Logfile 写满之前，开启新的 Logfile 作为当前的 log 文件。

* 刷新 **Schema**

```
ALTER SYSTEM
    REFRESH SCHEMA
    CLUSTER_ID=cluster_id
```

用户用修改核心表内容时，需要刷新 Schema。RootServer 收到命令后，会从内部表中读出最新的 Schema 并缓存起来。

* 打印 **RootTable** 内容

```
ALTER SYSTEM
    DUMP ROOTTABLE
```

* 打印不合要求的 **Tablet** 信息

```
ALTER SYSTEM
    DUMP UNUSUAL TABLET
```

打印副本或者版本不满足要求的 Tablet 信息。

3 数据操作语言

DML（Data Manipulation Language）的主要作用是根据需要写入、删除、更新数据库中的数据。

OceanBase 支持的 DML 主要有 INSERT，REPLACE，SELECT，UPDATE 和 DELETE。

3.1 INSERT 语句

该语句用于添加一个或多个记录到表。

* 格式

```
INSERT INTO table_name [(column_name,...)]  
VALUES (column_values,...);
```

- [(*column_name*,...)]用于指定插入数据的列。
- 同时插入多列时，用“,” 隔开。

* 举例

1. 执行以下命令，插入行。
INSERT INTO test VALUES (1, 'hello alipay'),(2, 'hello ob');
2. 执行以下命令查看插入的行，如[图 3-1](#)所示。
SELECT * FROM test;

图 3-1 INSERT

```
+-----+-----+  
| c1    | c2                |  
+-----+-----+  
|      1 | hello alipay      |  
|      2 | hello ob           |  
+-----+-----+  
2 rows in set (0.01 sec)
```

3.2 REPLACE 语句

REPLACE 语句的语法和 INSERT 相同，语义有别：如果本行已经存在，则修改对应列的值为新值；如果不存在，则插入。

由于 OceanBase 系统架构的特点,REPLACE 语句的性能要显著优于 INSERT: REPLACE 语句的执行不需要向 Chunk Server 请求数据,而 INSERT 则需要。所以,如果你的应用程序在数据操作的语义上可以使用 REPLACE 或 INSERT,请优先使用 REPLACE 语句。

* 格式

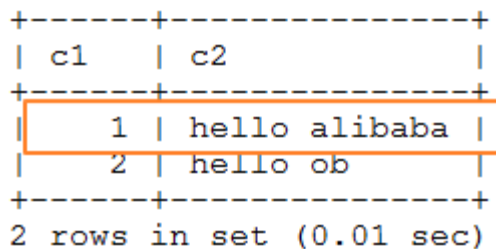
```
REPLACE INTO table_name [(column_name,...)]  
VALUES (column_values,...);
```

- [(*column_name*,...)]用于指定插入数据的列。
- 同时替换多列时,用“,”隔开。

* 举例

1. 执行以下命令,替换表 test 中的行。
REPLACE INTO test VALUES (1, 'hello alibaba'),(2, 'hello ob');
2. 执行以下命令查看表中的行,如[图 3-2](#)所示。
SELECT * FROM test;

图 3-2 REPLACE



c1	c2
1	hello alibaba
2	hello ob

2 rows in set (0.01 sec)

3.3 UPDATE 语句

该语句用于修改表中的字段值。

* 格式

```
UPDATE table_name  
SET column_name1=column_values  
[column_name2=column_values] ...  
WHERE where_condition;
```

* 举例

1. 执行以下命令,修改“hello ob”为“hello oceanbase”。
UPDATE test SET c2='hello oceanbase' WHERE c1=2;
2. 执行以下命令查看修改后的信息。如[图 3-3](#)所示。
SELECT * FROM test;

图 3-3 UPDATE

```
+-----+-----+
| c1    | c2                |
+-----+-----+
|      1 | hello alibaba    |
|      2 | hello oceanbase  |
+-----+-----+
2 rows in set (0.00 sec)
```

3.4 DELETE 语句

该语句用于删除表中符合条件的行。

* 格式

```
DELETE FROM table_name
        WHERE where_condition;
```

* 举例

1. 执行以下命令删除“c1=2”的行。其中 c1 列为表 test 中的 Primary Key。
DELETE FROM test WHERE c1 = 2;
2. 执行以下命令查看删除行后的表信息，如[图 3-4](#)所示。
SELECT * FROM test;

图 3-4 DELETE

```
+-----+-----+
| c1    | c2                |
+-----+-----+
|      1 | hello alibaba    |
+-----+-----+
1 row in set (0.01 sec)
```

3.5 SELECT 语句

该语句用于查询表中的内容。

3.5.1 基本查询

* 格式

```

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}];

```

SELECT 子句说明如[表 3-1](#)所示。

表 3-1 子句说明

子句	说明
ALL DISTINCT	在数据库表中，可能会包含重复值。指定“DISTINCT”，则在查询结果中相同的行只显示一行；指定“ALL”，则列出所有的行；不指定时，默认为“ALL”。
select_list	列出要查询的列名，用“,”隔开。也可以用“*”表示所有列。
AS other_name	为输出字段重新命名。
FROM table_name	必选项，指名了从哪个表中读取数据。
WHERE where_conditions	可选项，WHERE 字句用来设置一个筛选条件，查询结果中仅包含满足条件的数据。 where_conditions 为表达式。
GROUP BY group_by_list	用于进行分类汇总。
HAVING search_confitions	HAVING 字句与 WHERE 字句类似，但是 HAVING 字句可以使用累计函数(如 SUM, AVG 等)。
ORDER BY order_list [ASC DESC]	用来按升序 (ASC) 或者降序 (DESC) 显示查询结果。不指定 ASC 或者 DESC 时，默认为 ASC。

子句	说明
<code>[LIMIT {[offset,] row_count row_count OFFSET offset}]</code>	<p>强制 SELECT 语句返回指定的记录数。 LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。</p> <ul style="list-style-type: none"> 如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)。 如果只给定一个参数，它表示返回记录行的最大数目，偏移量为 0。

* 举例

假设现有表 a 如[图 3-5](#)所示。

图 3-5 表 a

表a

id	name	num
1	a	100
2	b	200
3	a	50

ALL 和 **DISTINCT**：在数据库表中，可能会包含重复值。指定“**DISTINCT**”，则在查询结果中相同的行只显示一行；指定“**ALL**”，则列出所有的行；不指定时，默认为“**ALL**”。

SELECT name FROM a;

或者

SELECT ALL name FROM a;

```
+-----+
| name |
+-----+
| a    |
| b    |
| a    |
+-----+
3 rows in set (0.01 sec)
```

SELECT DISTINCT name FROM a;

```
+-----+
| name |
+-----+
| a    |
| b    |
+-----+
2 rows in set (0.01 sec)
```

AS: 为输出字段重新命名。

SELECT id, name, num/2 AS avg FROM a;

```
+-----+-----+-----+
| id    | name | avg  |
+-----+-----+-----+
| 1     | a    | 50   |
| 2     | b    | 100  |
| 3     | a    | 25   |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

WHERE: 用来设置一个筛选条件，查询结果中仅包含满足条件的数据。

SELECT id, name, num FROM a WHERE name = 'a';

```
+-----+-----+-----+
| id    | name | num  |
+-----+-----+-----+
| 1     | a    | 100  |
| 3     | a    | 50   |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

GROUP BY: 用于进行分类汇总。以下语句是按“name”求“num”的总和。

SELECT id, name, SUM(num) FROM a GROUP BY name;

```
+-----+-----+-----+
| id    | name | SUM(num) |
+-----+-----+-----+
| 1     | a    | 150      |
| 2     | b    | 200      |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

HAVING: HAVING 字句与 WHERE 字句类似，但是 HAVING 字句可以使用累计函数（如 SUM，AVG 等）。以下语句是查询 num 总和小于 160 的行。

SELECT id, name, SUM(num) as sum FROM a GROUP BY name HAVING SUM(num) < 160;

```
+-----+-----+-----+
| id    | name | sum  |
+-----+-----+-----+
| 1     | a    | 150  |
+-----+-----+-----+
1 row in set (0.01 sec)
```

ORDER BY: 用来按升序（ASC）或者降序（DESC）显示查询结果。不指定 ASC 或者 DESC 时，默认为 ASC。

SELECT * FROM a ORDER BY num ASC;

```
+-----+-----+-----+
| id    | name  | num   |
+-----+-----+-----+
|      3 | a     | 50    |
|      1 | a     | 100   |
|      2 | b     | 200   |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

SELECT * FROM a ORDER BY num DESC;

```
+-----+-----+-----+
| id    | name  | num   |
+-----+-----+-----+
|      2 | b     | 200   |
|      1 | a     | 100   |
|      3 | a     | 50    |
+-----+-----+-----+
3 rows in set (0.01 sec)
```

LIMIT: 强制 **SELECT** 语句返回指定的记录数。以下语句为从第 2 行开始，返回两行。

SELECT * FROM a LIMIT 1,2;

```
+-----+-----+-----+
| id    | name  | num   |
+-----+-----+-----+
|      2 | b     | 200   |
|      3 | a     | 50    |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

3.5.2 JOIN 句法

JOIN 连接分为内连接和外连接。外连接又分为左连接、右连接和全连接。两个表联接后，可以使用 **ON** 指定条件进行筛选。

目前，OceanBase 的 **JOIN** 不支持 **USING** 子句，并且 **JOIN** 的连接条件中必须至少有一个等值连接条件。

假设现有表 a 和表 b 如[图 3-6](#)所示。

图 3-6 表 a 和表 b

表a		表b	
id	name	id	colour
1	a	1	red
2	b	2	blue
3	c		

内连接：结果中只包含两个表中同时满足条件的行。

SELECT a.id, a.name, b.colour FROM a INNER JOIN b ON a.id = b.id;

```

+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
+-----+-----+-----+
2 rows in set (0.01 sec)

```

左连接：结果中包含位于关键字 **LEFT [OUTER] JOIN** 左侧的表中的所有行，以及该关键字右侧的表中满足条件的行。

SELECT a.id, a.name, b.colour FROM a LEFT OUTER JOIN b ON a.id = b.id;

```

+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
| 3     | a     | NULL   |
+-----+-----+-----+
3 rows in set (0.00 sec)

```

右连接：结果中包含位于关键字 **[RIGHT] [OUTER] JOIN** 右侧的表中的所有行，以及该关键字左侧的表中满足条件的行。

SELECT a.id, a.name, b.colour FROM a RIGHT OUTER JOIN b ON a.id = b.id;

```

+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
+-----+-----+-----+
2 rows in set (0.01 sec)

```

全连接：结果中包含两个表中的所有行。

SELECT a.id, a.name, b.colour FROM a FULL OUTER JOIN b ON a.id = b.id;

```

+-----+-----+-----+
| id    | name  | colour |
+-----+-----+-----+
| 1     | a     | red    |
| 2     | b     | blue   |
| 3     | a     | NULL   |
+-----+-----+-----+
3 rows in set (0.00 sec)

```

3.5.3 集合操作

OceanBae 中的集合操作主要包括 **UNION**、**EXCEPT** 和 **INTERSECT**。

* **UNION** 句法

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。使用 UNION 需要注意以下几点：

- UNION 内部的 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每条 SELECT 语句中的列的顺序必须相同。
- 默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。
- UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

UNION 指令的目的是将两个或多个 SELECT 语句的结果合并起来。从这个角度来看，UNION 跟 JOIN 有些类似，因为这两个指令都可以由多个表格中撷取资料。但是 UNION 只是将两个结果联结起来一起显示，并不是联结两个表。

假设现有表 a 和表 c 如[图 3-7](#)所示。

图 3-7 表 a 和表 c

表a			表c		
id	name	num	xuhao	mingzi	shumu
1	a	100	4	c	150
2	b	200	5	b	200
3	a	50			

执行以下两个命令，比较 UNION 和 UNION ALL 的区别：

SELECT name, num FROM a UNION SELECT mingzi, shumu FROM c;

```
+-----+-----+
| name | num |
+-----+-----+
| a    | 50  |
| a    | 100 |
| b    | 200 |
| c    | 150 |
+-----+-----+
4 rows in set (0.01 sec)
```

SELECT name, num FROM a UNION ALL SELECT mingzi, shumu FROM c;

```
+-----+-----+
| name | num |
+-----+-----+
| a    | 100 |
| b    | 200 |
| a    | 50  |
| c    | 150 |
| b    | 200 |
+-----+-----+
5 rows in set (0.01 sec)
```

*** EXCEPT 句法**

EXCEPT 用于查询第一个集合中存在，但是不存在于第二个集合中的数据。

假设现有表 a 和表 d 如图 3-8 所示。

图 3-8 表 a 和表 d

表a			表d		
id	name	num	id	name	num
1	a	100	1	a	100
2	b	200	4	d	20
3	a	50			

执行以下命令，查询表 a 中存在，但是不存在与表 d 的数据。

SELECT * FROM a EXCEPT SELECT * FROM d;

```
+-----+-----+-----+
| id   | name | num   |
+-----+-----+-----+
| 2    | b    | 200   |
| 3    | a    | 50    |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

*** INTERSECT 句法**

INTERSECT 用于查询在两个集合中都存在的数据。

假设现有表 a 和表 d 如图 3-8 所示。

执行以下命令，查询表 a 和表 d 中均存在的数据。

SELECT * FROM a INTERSECT SELECT * FROM d;

```
+-----+-----+-----+
| id   | name | num   |
+-----+-----+-----+
| 1    | a    | 100   |
+-----+-----+-----+
1 row in set (0.01 sec)
```

3.5.4 DUAL 虚拟表

DUAL 是一个虚拟的表，可以视为一个一行零列的表。当我们不需要从具体的表来取得表中数据，而是单纯地为了得到一些我们想得到的信息，并要通过 SELECT 完成时，就要借助一个对象，这个对象就是 DUAL。一般可以使用这种特殊的 SELECT 语法获得用户变量或系统变量的值。

当 SELECT 语句没有 FROM 子句的时候，语义上相当于 FROM DUAL，此时，表达式中只能是常量表达式。

* 格式

```
SELECT
    [ALL | DISTINCT]
    select_list
    [FROM DUAL [WHERE where_condition]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}];
```

参数说明请参见“3.2.1 基本查询”。

* 举例

执行以下命令，计算 2 和 3 的积。

```
SELECT 2*3;
```

或者

```
SELECT 2*3 FROM DUAL;
```

```
+-----+
| 2*3 |
+-----+
|    6 |
+-----+
1 row in set (0.00 sec)
```

3.5.5 SELECT ... FOR UPDATE 句法

SELECT ... FOR UPDATE 可以用来对查询结果所有行上排他锁，以阻止其他事务的并发修改，或阻止在某些事务隔离级别时的并发读取。即使用 FOR UPDATE 语句将锁住查询结果中的元组，这些元组将不能被其他事务的 UPDATE，DELETE 和 FOR UPDATE 操作，直到本事务提交。

注意的是，目前 OceanBase 实现有如下限制：必须是单表查询。

例如：SELECT * FROM a FOR UPDATE;

3.5.6 IN 和 OR

OceanBase 支持逻辑运算“IN”和“OR”，其中 IN 可以直接定位到查询的数据，而 OR 需要进行全表扫描，因此我们推荐使用 IN 语句。

假设现有表 a 如[图 3-9](#)所示。

图 3-9 表 a

表a

id	name	num
1	a	100
2	b	200
3	a	50

以下两个句法等价，但第一个句法执行效率高。

SELECT * FROM a WHERE id IN (1,2);

SELECT * FROM a WHERE id = 1 OR id = 2;

id	name	num
1	a	100
2	b	200

2 rows in set (0.00 sec)

4 事务处理

数据库事务（Database Transaction）是指作为单个逻辑工作单元执行的一系列操作。事务处理可以用来维护数据库的完整性，保证成批的 **SQL** 操作全部执行或全部不执行。

* 格式

1. 开启事务语句格式如下：

```
START TRANSACTION  
    [WITH CONSISTENT SNAPSHOT];
```

或者

```
BEGIN [WORK];
```

- **WITH CONSISTENT SNAPSHOT** 子句用于启动一个一致的读取。该子句的效果与发布一个 **START TRANSACTION**，后面跟一个来自任何 OceanBase 表的 **SELECT** 的效果一样。
- **BEGIN** 和 **BEGIN WORK** 被作为 **START TRANSACTION** 的别名受到支持，用于对事务进行初始化。**START TRANSACTION** 是标准的 **SQL** 语法，并且是启动一个 **ad-hoc** 事务的推荐方法。一旦开启事务，则随后的 **SQL** 数据操作语句（即 **INSERT**, **UPDATE**, **DELETE**，不包括 **REPLACE**）直到显式提交时才会生效。

2. 提交当前事务语句格式如下：

```
COMMIT [WORK];
```

3. 回滚当前事务语句格式如下：

```
ROLLBACK [WORK];
```

* 举例

假设现有表 **a** 如[图 4-1](#)所示。执行事务：将 **id** 为 3 的的 **name** 改为 **c**，并插入一行当前卖出 **a** 的记录。

图 4-1 表 a

表a

id	name	num	sell_date
1	a	100	2013-06-21 10:06:43
2	b	200	2013-06-21 13:07:21
3	a	50	2013-06-21 13:08:15

- 依次执行以下命令开始执行事务。
START TRANSACTION;
UPDATE a SET name = 'c' WHERE id = 3;
INSERT INTO a VALUES (4, 'a', 30, '2013-06-21 16:09:13');
COMMIT;

- 事务提交后，执行查看表 a 信息，如[图 4-2](#)所示。

SELECT * FROM a;

图 4-2 TRANSACTION

```

+-----+-----+-----+-----+
| id    | name  | num   | sell_date                |
+-----+-----+-----+-----+
| 1     | a     | 100   | 2013-06-21 10:06:43     |
| 2     | b     | 200   | 2013-06-21 13:07:21     |
| 3     | c     | 50    | 2013-06-21 13:08:15     |
| 4     | a     | 30    | 2013-06-21 16:09:13     |
+-----+-----+-----+-----+
4 rows in set (0.01 sec)

```



说明：

在事务还没有 COMMIT 之前，您可以查看下本事务中的操作是否已经生效，比如可以在 COMMIT 之前，加一句“**SELECT * FROM a;**”。不过结果肯定是没有生效，在事务还没有 COMMIT 之前，你之前做的操作是不可见的。如果你想回滚该事务的话，直接用“**ROLLBACK**”替代“**COMMIT**”。

执行事务时，要注意以下几点：

- 我们采用 **datetime** 类型的值作为主键，在进行 **INSERT** 操作的时候，不能直接在 **SQL INSERT** 语句中采用 **CURRENT_TIME()** 函数来获取时间作为主键列的值，比如 **INSERT INTO a VALUES(4, 'a', 30, CURRENT_TIME())**，则会报错的。我们只能在 **INSERT** 之前，先用 **CURRENT_TIME()** 函数获取当前时间，然后在 **INSERT** 语句中直接用该确定的时间值。
- 由于事务有一个超时时间，所以当我们手动输入事务中的多个 **SQL** 语句的时候，由于打字时间太长，会导致整个事务超时。解决方法是：修改当

前 session 中的系统变量 `ob_tx_timeout`，使得该值尽可能大，而不至于导致超时。修改系统参数的方法请参见“5.4 修改系统配置项”。

- 目前在 OceanBase 的事务中执行 `SELECT` 语句，不能读取当前事务中未提交的数据。用户如果有这个需求，可以暂时用 `SELECT ... FOR UPDATE` 语句代替。

5 数据库管理语句

数据库管理包括用户及权限管理、修改用户变量、系统变量和系统配置。

5.1 用户及权限管理

数据库用户权限管理包括新建用户、删除用户、修改密码、修改用户名、锁定用户、用户授权和撤销授权等。

5.1.1 新建用户

CREATE USER 用于创建新的 OceanBase 用户。创建新用户后，可以使用该用户连接 OceanBase。

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。

* 格式

```
CREATE USER 'user'  
    [IDENTIFIED BY [PASSWORD] 'password'];
```

- 必须拥有全局的 CREATE USER 权限或对 “__all_user” 表的 INSERT 权限，才可以使用 CREATE USER 命令。
- 新建用户后，“__all_user” 表会新增一行该用户的表项。如果同名用户已经存在，则报错。
- 使用自选的 IDENTIFIED BY 子句，可以为账户给定一个密码。
- 此处密码为明文，存入 “__all_user” 表后，服务器端会变为密文存储下来。
- 同时创建多个用户时，用 “,” 隔开。

* 举例

1. 执行以下命令创建“sqluser01”和“sqluser02”用户，密码均为“123456”。
CREATE USER 'sqluser01' IDENTIFIED BY '123456', 'sqluser02' IDENTIFIED BY '123456';
2. 执行以下命令查看创建的用户，如[图 5-1](#)所示。
SELECT user_name FROM __all_user;

图 5-1 创建用户

```
+-----+
| user_name |
+-----+
| admin     |
| sqluser01 |
| sqluser02 |
+-----+
3 rows in set (0.00 sec)
```

5.1.2 删除用户

DROP USER 语句用于删除一个或多个 OceanBase 用户。

* 格式

DROP USER 'user';

- 必须拥有全局的 CREATE USER 权限或对 “__all_user” 表的 DELETE 权限，才可以使用 DROP USER 命令。
- 成功删除用户后，这个用户的所有权限也会被一同删除。
- 同时删除多个用户时，用 “,” 隔开。

* 举例

执行以下命令，删除 “sqluser02” 用户。

DROP USER 'sqluser02';

5.1.3 修改密码

用于修改 OceanBase 登录用户的密码。

*格式

SET PASSWORD [FOR 'user' =] 'password';

或者

ALTER USER 'user' IDENTIFIED BY 'password';

- 如果没有 For user 子句，则修改当前用户的密码。任何成功登陆的用户都可以修改当前用户的密码。
- 如果有 For user 子句，或使用第二种语法，则修改指定用户的密码。必须拥有对 “__all_user” 表的 UPDATE 权限，才可以修改制定用户的密码。

* 举例

执行以下命令将“sqluser01”的密码修改为“abc123”。

```
SET PASSWORD FOR 'sqluser01' = 'abc123';
```

或者

```
ALTER USER 'sqluser01' IDENTIFIED BY 'abc123';
```

5.1.4 修改用户名

用于修改 OceanBase 登录用户的用户名。

* 格式

```
RENAME USER 'old_user'  
TO 'new_user';
```

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

* 举例

1. 修改前，执行以下命令查看用户，如[图 5-2](#)所示。

```
SELECT user_name FROM __all_user;
```

图 5-2 修改前

```
+-----+  
| user_name |  
+-----+  
| admin     |  
| sqluser01 |  
+-----+  
2 rows in set (0.00 sec)
```

2. 执行以下命令修改用户名。
RENAME USER 'sqluser01' TO 'obsqluser01';
3. 修改后，执行以下命令查看用户，如[图 5-3](#)所示。

```
SELECT user_name FROM __all_user;
```

图 5-3 修改后

```
+-----+  
| user_name |  
+-----+  
| admin     |  
| obsqluser01 |  
+-----+  
2 rows in set (0.00 sec)
```

5.1.5 锁定用户

锁定或者解锁用户。被锁定的用户不允许登陆。

* 格式

锁定用户：ALTER USER 'user' LOCKED;

解锁用户：ALTER USER 'user' UNLOCKED;

必须拥有对 “__users” 表的 UPDATE 权限，才可以执行本命令。

* 举例

锁定用户：ALTER USER 'obsqluser01' LOCKED;

解锁用户：ALTER USER 'obsqluser01' UNLOCKED;

5.1.6 用户授权

GRANT 语句用于系统管理员授予 OceanBase 用户操作权限。

* 格式

GRANT *priv_type*
ON *table_name*
TO 'user';

- 给特定用户授予权限。如果用户不存在则报错。
- 当前用户必须拥有被授予的权限（例如，user1 把表 t1 的 SELECT 权限授予 user2，则 user1 必须拥有表 t1 的 SELECT 的权限），并且拥有 GRANT OPTION 权限，才能授予成功。
- 用户授权后，该用户只有重新连接 OceanBase，权限才能生效。
- 用 “*” 代替 *table_name*，表示赋予全局权限，即对数据库中的所有表赋权。
- 同时把多个权限赋予用户时，权限类型用 “,” 隔开。
- 同时给多个用户授权时，用户名用 “,” 隔开。
- *priv_type* 的如[表 5-1](#)所示。

表 5-1 权限类型

权限	说明
ALL PRIVILEGES	除 GRANT OPTION 以外所有权限。
ALTER	ALTER TABLE 的权限。

权限	说明
CREATE	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 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。

本用户自动拥有自己创建的对象（目前基本上只有表）。例如，用户 **user1** 创建了表 **t1** 和 **t2**，那用户 **user1** 应该自动就有对 **t1** 和 **t2** 的 **ALL PRIVILEGES** 及 **GRANT OPTION** 权限，不再需要额外去授权。

* 举例

执行以下命令给“**obsqluser01**”赋予所有权限。

GRANT ALL PRIVILEGES, GRANT OPTION ON * TO 'obsqluser01';

5.1.7 撤销权限

REVOKE 语句用于系统管理员撤销 OceanBase 用户的操作权限。

* 格式

```
REVOKE priv_type
      ON table_name
      FROM 'user';
```

- 用户必须拥有被撤销的权限（例如，**user1** 要撤销 **user2** 对表 **t1** 的 **SELECT** 权限，则 **user1** 必须拥有表 **t1** 的 **SELECT** 的权限），并且拥有 **GRANT OPTION** 权限。

- 撤销“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。
- 撤销操作不会级联。例如，用户 user1 给 user2 授予了某些权限，撤回 user1 的权限不会同时也撤回 user2 的相应权限。
- 用“*”代替 *table_name*，表示撤销全局权限，即撤销对数据库中所有表的操作权限。
- 同时对用户撤销多个权限时，权限类型用“,”隔开。
- 同时撤销多个用户的授权时，用户名用“,”隔开。
- priv_type 的如[表 5-1](#)所示。

* 举例

执行以下命令撤销“obsqluser01”的所有权限。

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'obsqluser01';

5.1.8 查看权限

SHOW GRANTS 语句用于系统管理员查看 OceanBase 用户的操作权限。

* 格式

SHOW GRANTS [FOR *user*];

- 如果不指定用户名，则缺省显示当前用户的权限。对于当前用户，总可以查看自己的权限。
- 如果要查看其他指定用户的权限，必须拥有对“__all_user”的 SELECT 权限。

* 举例

SHOW GRANTS FOR 'sqluser01';

5.2 修改用户变量

用户变量用于保存一个用户自定义的值，以便于在以后引用它，这样可以将该值从一个语句传递到另一个语句。用户变量与连接有关，即一个客户端定义的用户变量不能被其它客户端看到或使用，当客户端退出时，该客户端连接的所有变量将自动释放。

* 格式

SET @var_name = expr;

- 用户变量的形式为@var_name，其中变量名 var_name 可以由当前字符集的文字数字字符、“.”、“_”和“\$”组成。
- 每个变量的 expr 可以为整数、实数、字符串或者 NULL 值。
- 同时定义多个用户变量时，用“,”隔开。

* 举例

1. 执行以下命令设置用户变量。

```
SET @a=2, @b=3;  
SET @c = @a + @b;
```

2. 执行以下命令，查看用户变量，如[图 5-4](#)所示。

```
SELECT @a, @b, @c;
```

图 5-4 用户变量

```
+-----+-----+-----+  
| @a    | @b    | @c    |  
+-----+-----+-----+  
| 2      | 3      | 5      |  
+-----+-----+-----+  
1 row in set (0.00 sec)
```

5.3 修改系统变量

系统变量和 SQL 功能相关，存放在“__all_sys_param”表中，如 autocommit, tx_isolation 等。系统变量的参数说明请参见《OceanBase 0.5 参考指南》的“3.12 __all_sys_param”。

OceanBase 维护两种变量：

- 全局变量
影响 OceanBase 整体操作。当 OceanBase 启动时，它将所有全局变量初始化为默认值。修改全局变量，必须具有 SUPER 权限。
- 会话变量
影响当前连接到 OceanBase 的客户端。在客户端连接 OceanBase 时，使用相应全局变量的当前值对该客户端的会话变量进行初始化。设置会话变量不需要特殊权限，但客户端只能更改自己的会话变量，而不能更改其它客户端的会话变量。



说明：

全局变量的更改不影响目前已经连接的客户端的会话变量，即使客户端执行 SET GLOBAL 语句也不影响。

* 格式

设置全局变量的格式：

```
SET GLOBAL system_var_name = expr;
```

或者

```
SET @@GLOBAL.system_var_name = expr;
```

设置会话变量的格式：

```
SET [SESSION | @@SESSION. | LOCAL | LOCAL. | @@]system_var_name  
= expr;
```

查看系统变量的格式：

如果指定 GLOBAL 或 SESSION 修饰符，则分别打印全局或当前会话的系统变量值；如果没有修饰符，则显示当前会话值。

```
SHOW [GLOBAL | SESSION]  
VARIABLES  
[LIKE 'system_var_name' | WHERE expr];
```

* 举例

1. 执行以下命令，修改会话变量中的 SQL 超时时间。

```
SET @@SESSION.ob_tx_timeout = 900000;
```

2. 执行以下命令，查看 SQL 超时时间。

```
SHOW SESSION VARIABLES LIKE 'ob_tx_timeout';
```

5.4 修改系统配置项

配置项存放在“__all_sys_config”表中，一般针对每一类 Server 进行配置，一般影响某个 Server 的行为，比如 MergeServer 的线程池大小，RootServer 的负载均衡策略等，当然也可以通过指定 IP 对某个 Server 单独进行配置。

* 格式

修改系统配置项的格式：

```
ALTER SYSTEM SET param_name = expr  
[COMMENT 'text']  
[SCOPE = conf_scope]  
SERVER_TYPE = server_type  
[CLUSTER_ID = cluster_id | SERVER_IP = 'server_ip'  
SERVER_PORT = server_port];
```

- 修改系统配置项说明如[表 5-2](#)所示。

表 5-2 子句说明

子句	说明
<i>param_name = expr</i>	配置项请参见《OceanBase 0.5 参考指南》的“3.11 __all_sys_config”和“4 配置项参考”。
COMMENT ' <i>text</i> '	可选，用于添加关于本次修改的注释。建议不要省略。
SCOPE = <i>conf_scope</i>	<p>SCOPE 用来指定本次配置项修改的生效范围。它的值主要有以下三种：</p> <ul style="list-style-type: none"> MEMORY：表明只修改内存中的配置项，修改立即生效，且本修改在 Server 重启以后会失效（目前暂时没有配置项支持这种方式）。 SPFILE：表明只修改配置表中的配置项值，当 Server 重启以后才生效。 BOTH：表明既修改配置表，又修改内存值，修改立即生效，且 Server 重启以后配置值仍然生效。 <p><i>说明</i>：SCOPE 默认值为 BOTH。对于不能立即生效的配置项，如果 SCOPE 使用 BOTH 或 MEMORY，会报错。</p>
SERVER_TYPE = <i>server_type</i>	服务器类型， ROOTSERVER\UPDATESERVER\CHUNKSERVER\MERGESERVER。
CLUSTER_ID = <i>cluster_id</i>	表明本配置项的修改正对指定集群的特定 Server 类型，否则，针对所有集群的特定 Server 类型。
SERVER_IP = <i>'server_ip'</i> SERVER_PORT = <i>server_port</i>	只修改指定 Server 实例的某个配置项。

- 同时修改多个系统配置项时，用“,”隔开。

查看系统配置项的格式：

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

* 举例

- 执行以下命令，修改 Tablet 副本数。

ALTER SYSTEM SET ups_waiting_register_time='15s' COMMENT

**'Modify by Bruce' SCOPE = SPFILE SERVER_TYPE =
ROOTSERVER SERVER_IP= '10.10.10.2' SERVER_PORT= 1234;**

2. 查看 “tablet_replicas_num”。

SHOW PARAMETERS LIKE 'ups_waiting_register_time';

6 预备执行语句

OceanBase 实现了服务器端的真正的 Prepared statement。用户先通过客户端发送一个预备语句把要执行的 SQL 数据操作语句发给服务器，服务器端会解析这个语句，产生执行计划并返回给客户端一个句柄（名字或者 ID）。随后，用户可以使用返回的句柄和指定的参数反复执行一个预备好的语句，省去了每次执行都解析 SQL 语句的开销，可以极大地提高性能。

由于目前 OceanBase SQL 引擎的优化工作还做的不够，SQL 解析并产生执行计划的过程效率不高。而使用预备执行语句，可以省掉这一过程，直接用已经预备好的执行计划和用户指定的参数执行语句，这样可以极大的提高性能。所以，我们强烈推荐应用尽可能多地使用预备执行语句。

* PREPARE 语句

```
PREPARE stmt_name
      FROM preparable_stmt;
```

- *preparable_stmt* 为 SQL 数据操作语句，预备好的语句在整个 SQL 会话期间可以使用 *stmt_name* 这个名字来执行。
- 数据操作语句（即 SELECT, REPLACE, INSERT, UPDATE, DELETE）都可以被预备执行。
- 在被预备的 SQL 语句中，可以使用问号 (?) 表明一个之后执行时才绑定的参数。问号只能出现在 SQL 语句的常量中。一个被预备的语句也可以不包含问号。

* EXECUTE 语句

```
EXECUTE stmt_name
      [USING @var_name [, @var_name] ...];
```

- 一个使用 PREPARE 语句预备好的 SQL 语句，可以使用 EXECUTE 语句执行。
- 如果预备语句中有问号指明的绑定变量，需要使用 USING 子句指明相同个数的执行时绑定的值。USING 子句后只能使用 SET 语句定义的用户变量。

* DEALLOCATE 语句

```
DEALLOCATE PREPARE stmt_name;
```

或者

```
DROP PREPARE stmt_name;
```

删除一个指定的预备语句。一旦删除，以后就不能再执行。

* 举例

依次使用 PREPARE 查询表 a 中 id=2 的行。

```
PREPARE stmt1 FROM SELECT name FROM a WHERE id=?;  
SET @id = 2;  
EXECUTE stmt1 USING @id;  
DEALLOCATE PREPARE stmt1;
```

7 其他 SQL 语句

主要介绍 SHOW、KILL、DESCRIBE、EXPLAIN、WHEN 和 hint 等语句。

7.1 SHOW 语句

主要用于查看 OceanBase 的各类信息。

1. 查看可以用来建立指定表格的建表语句。

```
SHOW CREATE TABLE table_name
```

2. 查看指定表的列的信息。

```
SHOW COLUMNS {FROM | IN} table_name  
[LIKE 'pattern' | WHERE expr]
```

3. 统计上一条语句执行过程中产生的 “Warning” 信息。

```
SHOW COUNT(*) WARNINGS
```

4. 查看 OceanBase 用户的操作权限。

```
SHOW GRANTS [FOR user]
```

- 如果不指定用户名，则缺省显示当前用户的权限。对于当前用户，总可以查看自己的权限。
- 如果要查看其他指定用户的权限，必须拥有对__all_user 的 SELECT 权限。

5. 查看某个表的索引。

```
SHOW INDEX  
FROM table_name
```

6. 查看各个配置项在各个 Server 实例上的值。

```
SHOW PARAMETERS  
[LIKE 'pattern' | WHERE expr]
```

7. 查看所有用户的当前连接。

```
SHOW PROCESSLIST
```

8. 查看数据库中是否存在哪些表。

```
SHOW [ALL] TABLES  
[LIKE 'pattern' | WHERE expr]
```

加 ALL，则查询所有表，不加 ALL，则只查询用户创建的表。

9. 打印系统变量值。

```
SHOW [GLOBAL | SESSION]
    VARIABLES
    [LIKE 'system_var_name' | WHERE expr]
```

- 如果指定 GLOBAL 或 SESSION 修饰符，则分别打印全局或当前会话的系统变量值。
- 如果没有修饰符，则显示当前会话值。

10. 获得上一条语句执行过程中产生的“Warning”信息，不包含“Error”。

```
SHOW WARNINGS
    [LIMIT [offset,] row_count];
```

7.2 KILL 语句

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



说明：

每个与 OceanBase 的连接都在一个独立的线程里运行，您可以使用 **SHOW PROCESSLIST** 语句查看哪些线程正在运行，并使用 **KILL *thread_id*** 语句终止一个线程。Index 列为 *thread_id*。

- KILL CONNECTION 与不含修改符的 KILL 一样：它会终止与给定的 *thread_id*。
- KILL QUERY 会终止连接当前正在执行的语句，但是会保持连接的原状。

7.3 DESCRIBE 语句

```
DESCRIBE table_name [col_name | wild];
```

或者

```
DESC table_name [col_name | wild];
```

这个语句等同于 SHOW COLUMNS FROM 语句。

7.4 EXPLAIN 语句

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

这个语句可以输出 SELECT, INSERT, UPDATE, DELETE, REPLACE 等 DML 语句内部的物理执行计划。VERBOSE 模式也会输出逻辑执行计划。DBA 和开发人员可以根据 EXPLAIN 的输出来优化 SQL 语句。

7.5 WHEN 语句

statement WHEN *expr*|ROW_COUNT(*statement*) *op* *expr*;

WHEN 子句是 OceanBase 扩展的 SQL 语法，其含义为：当 WHEN 子句的条件满足的时候，才执行 WHEN 前的主句。

- *statement* 为 SELECT、FOR UPDATE、UPDATE、DELETE、INSERT、REPLACE 等语句。
- WHEN 子句可以嵌套和组合。

例如：

```
SELECT * FROM t1 WHERE c1 = 1000 FOR UPDATE WHEN  
ROW_COUNT(UPDATE t2 SET c1 = c1 + 1 WHERE c0 = 1000) >= 0;
```

7.6 hint 语法

hint 是一种特殊的 SQL 注释，SQL 注释的一般语法和 C 语言的注释语法相同，即/* ... */。而 hint 的语法在此基础上加了一个加号，型如/*+ ... */。既然是注释，那么如果 Server 端不认识你 SQL 语句中的 hint，它可以直接忽略而不必报错。这样做的一个好处是，同一条 SQL 发给不同的数据库产品，不会因为 hint 引起语法错误。hint 只影响数据库服务器端内部优化的逻辑，而不影响 SQL 语句本身的语义。

OceanBase 支持的 hint 有以下几个特点：

- 不带参数的，如/*+ KAKA */。
- 带参数的，如/*+ HAHA(param) */。
- 多个 hint 可以写到同一个注释中，用逗号分隔，如/*+ KAKA, HAHA(param) */。
- SELECT 语句的 hint 必须近接在关键字 SELECT 之后，其他词之前。如：
SELECT /*+ KAKA */ ...。
- UPDATE 语句的 hint 必须紧接在关键字 UPDATE 之后。

OceanBase 支持的 hint 如[表 7-1](#)所示。

表 7-1 hint

hint	参数	适用语句	含义
READ_CONSISTENCY	WEAK, STRONG, FROZEN, STATIC	SELECT	表明是强一致性读还是弱一致性读，如果 SQL 语句中不指定，默认值根据系统变量 ob_read_consistency 的值决定。STATIC 表示只读静态数据，如果访问跨多 Tablet，那么数据可能不是来自同一个快照点。 FROZEN 表示读最近一次冻结点的数据，冻结版本号由系统自动选择。

hint	参数	适用语句	含义
FROZEN_VERSION	冻结版本号	SELECT	只读指定版本的数据。
READ_CLUSTER	SLAVE 或 MASTER	SELECT	限制请求发到指定集群。 read_cluster(slave) 是尽量选备集群，不存在备集群也没关系。这个 hint 仅控制 OceanBase 客户端对 SQL 语句的路由策略，和 READ_CONSISTENCY 没有关联。
HOTSPOT	无	SELECT	在 WHEN 连接的复合语句中，表明本 UPDATE 所更新的行是“热点行”，执行计划在 UpdateServer 端会根据热点行排队，以减少锁冲突。
QUERY_TIMEOUT	超时时间，单位微秒	SELECT 和 UPDATE	设定 Server 端执行语句的超时时间，超时以后 Server 端中断执行并返回超时错误码。
USE_NL	USE_NL(右物理表名)	SELECT	使用 NESTED LOOP JOIN 对两表 JOIN，要求至少有一个 JOIN 条件，可以没有非 WHERE 条件。参考 Oracle 用法。
INDEX	INDEX(表名 索引名)	SELECT	和 Oracle 类似，设定对于指定表的查询强制走索引名，如果索引不存在或者不可用，也不报错。例如 SELECT /*+ INDEX(t1 i1) INDEX(t2 i2)*/ from t1, t2 WHERE t1.c1=t2.c1;

8 SQL 优化

主要介绍 OceanBase 的 SQL 优化，提高 OceanBase 的执行效率。

8.1 执行计划

Explain 语句，可以查看一个 DML 语句的执行计划。执行计划是一个物理运算符组成的树状结构。常见的物理运算符有 TableScan, Filter, Sort, GroupBy, Join, Project 等。例如，执行以下语句：

```
EXPLAIN
  SELECT name, value1, value2
    FROM __all_sys_config_stat
   WHERE name = 'location_cache_timeout';
```

可以得到如下一个 Plan：

```
Project(columns=[expr=[COL],expr=[COL],expr=[COL]])
TableRpcScan(rpc_scan==[COL|varchar:location_cache_timeout|EQ|])
Project(columns=[expr=[COL],expr=[COL],expr=[COL]])
```

这个 Plan 由 Project 和 TableRpcScan 这两个物理运算符组成：

- Project 运算符负责投影和表达式计算，它的输入数据由下层的 TableRpcScan 提供。
 - Columns 参数列出了三个表达式，表示这个投影操作会产生 3 列数据。
 - expr=[COL]代表一个后缀表达式，这个表达式会产生一个 TABLE_ID 为 NULL，COLUMN_ID 为 65519 的 cell。这个 cell 的值根据后缀表达式[COL]运算后产生，这里这个表达式是一个列引用，就是取 TABLE_ID 为 12，COLUMN_ID 为 25 的数据值。
- TableRpcScan 实际上由 RpcScan 操作符实现。这个物理运算符实际上是在 ChunkServer 上执行。它读取 TABLE_ID 为 12 的表中通过 Project(columns=[expr=[COL],expr=[COL],expr=[COL]])中指定 COLUMN_ID 为 25,27,28 的 3 列数据。然后经过一个 Filter 操作符进行过滤，过滤条件即后缀表达式[COL|varchar:location_cache_timeout|EQ|]，它表示过滤 TABLE_ID 为 12，COLUMN_ID 为 25 的 cell，等于 varchar:location_cache_timeout 的行。

8.2 内部优化规则

OceanBase 为 SQL 语句产生执行计划的时候，实现了一些基于规则的优化策略，这里简单描述一些目前实现的规则。

8.2.1 主键索引

OceanBase 表格中数据存储都是按照主键排序的。如果一个 **SELECT** 查询的 **WHERE** 条件中限定了主键的所有列，那么查询可以使用主键索引进行优化。

假设有一个表 **t1**：

```
mysql> DESC t1;
```

field	type	null	key	default	extra	comment
c1	INT	NO	1	NULL		
c2	INT	NO	2	NULL		
c3	INT	NO	3	NULL		
c4	INT	NO	0	NULL		

4 rows in set (0.14 sec)

下面举例说明优化规则：

- 使用等值条件限定所有主键的查询，会使用 **MultiGet** 操作进行单行查询。如“**SELECT * FROM t1 WHERE c1 = 1 and c2 = 2 and c3 = 3**”。注意，这里的等值条件必须写成列在等号左边，例如 **1 = c1** 则不满足本规则。
- 使用 **IN** 表达式限定所有主键的查询，会使用 **MultiGet** 操作进行多行查询。如“**SELECT * FROM t1 WHERE (c1, c2, c3) IN ((1,2,3), (4,5,6))**”。
- 使用简单比较操作限定主键前缀列取值范围的查询，会转换为对某些特定 **Tablet** 的扫描。如“**SELECT * FROM t1 WHERE c1 >= 1 and c1 < 10 and c2 > 100 and c2 <= 200 and c3 > 300**”，会转换为对主键范围(**,**)所有 **Tablet** 的扫描。
- 不满足上面三种情况的，则需要对整个表进行全表扫描。特别的，目前执行计划生产过程没有做表达式变化。所以“**SELECT * FROM t1 WHERE (c1 = 1 and c2 = 2 and c3 = 3) OR (c1 = 10 and c2 = 20 and c3 = 30)**”这个语句不会使用 **MultiGet** 优化，应用应该使用 **IN** 表达式执行这种多行查询。

8.2.2 并发执行

MergeServer 向 **ChunkServer** 读取基本表数据的时候，是并发查询多个 **ChunkServer** 的。除此之外，如果满足某些条件，**MergeServer** 会把聚合操作分发到拥有数据的相关 **ChunkServer** 上执行，然后把 **ChunkServer** 汇总后的结果再做汇总。我们把这个优化叫做“聚合操作下压”，它需要查询满足以下一些条件：

- 单表查询。
- 没有 **UNION**, **INTERSECT**, **EXCEPT** 等集合操作。
- 有 **GROUP BY** 子句，或者有聚集函数，且任何聚集函数不能有 **DISTINCT** 修饰符。

- 系统变量开关 `ob_group_agg_push_down_param` 为 `true`（默认值）。

此外，`Limit` 操作也可以下压到 `ChunkServer` 端执行，以减少需要网络传输的数据。适用这个优化的查询需要满足以下条件：

- 单表查询。
- 没有 `UNION`, `INTERSECT`, `EXCEPT` 等集合操作。
- 没有 `ORDER BY` 子句。
- 没有 `GROUP BY` 子句以及聚集函数。
- 当然需要有 `LIMIT` 子句。