**OceanBase SQL用户手册[[1]](#footnote-2)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **编号** | **文档版本** | **修订章节** | **修订原因** | **修订日期** | **修订人** |
| **1** | 0.1 |  | 新建文档 | 2012/10/16 | 竹翁 |
| **2** | 0.2 |  | 添加0.4.1版本新特性 | 2013/1/13 | 竹翁 |
| **3** | 0.3 |  | 细节说明；添加SELECT FOR UPDATE说明；说明事务中SELECT的BUG陷阱 | 2013/1/21 | 竹翁 |
| **4** | 0.4 |  | 陷阱和缺陷添加对varchar(32)的说明；  陷阱和缺陷添加REPLACE返回affected\_rows的说明；  附录中添加PS语句API示例程序 | 2013/1/23 | 竹翁 |
| **5** | 0.5 | 4.1,5.1 | SQL语句大小写不敏感；双引号转义说明；新增alter table语句 | 2013/3/14 | 竹翁 |
| **6** | 0.6 | 4.2 | 增加对返回类型的描述 | 2013/3/22 | 方戟 |

目录

[1. OceanBase介绍 4](#_Toc351040757)

[2. 快速上手 4](#_Toc351040758)

[步骤一 4](#_Toc351040759)

[步骤二 5](#_Toc351040760)

[步骤三 5](#_Toc351040761)

[3. 使用API 6](#_Toc351040762)

[3.1. C程序 6](#_Toc351040763)

[3.1.1. 编译方法一： 8](#_Toc351040764)

[3.1.2. 编译方法二： 8](#_Toc351040765)

[3.2. Java程序 8](#_Toc351040766)

[4. 陷阱和缺陷 11](#_Toc351040767)

[4.1. 支持的语句 11](#_Toc351040768)

[4.2. 数据类型 11](#_Toc351040769)

[4.3. 浮点数 11](#_Toc351040770)

[4.4. 函数 12](#_Toc351040771)

[4.5. Join的限制 12](#_Toc351040772)

[4.6. update和delete的限制 12](#_Toc351040773)

[4.7. 事务中SELECT的BUG 12](#_Toc351040774)

[4.8. REPLACE语句返回的affected\_rows值 12](#_Toc351040775)

[5. SQL语法参考 13](#_Toc351040776)

[5.1. 数据定义语句 13](#_Toc351040787)

[5.1.1. create table语句 13](#_Toc351040788)

[5.1.2. drop table语句 14](#_Toc351040789)

[5.1.3. alter table语句 14](#_Toc351040790)

[5.2. 数据操作语句 15](#_Toc351040791)

[5.2.1. select语句 15](#_Toc351040792)

[5.2.2. replace语句 18](#_Toc351040793)

[5.2.3. insert语句 18](#_Toc351040794)

[5.2.4. update语句 18](#_Toc351040795)

[5.2.5. delete语句 18](#_Toc351040796)

[5.3. 事务相关语句 19](#_Toc351040797)

[5.3.1. 开始事务 19](#_Toc351040798)

[5.3.2. 结束事务 19](#_Toc351040799)

[5.4. 变量相关语句 19](#_Toc351040800)

[5.4.1. set语句 19](#_Toc351040801)

[5.4.2. show variables语句 20](#_Toc351040802)

[5.5. 预备执行语句 20](#_Toc351040803)

[5.5.1. prepare语句 20](#_Toc351040804)

[5.5.2. execute语句 20](#_Toc351040805)

[5.5.3. deallocate语句 21](#_Toc351040806)

[5.5.4. 一个例子 21](#_Toc351040807)

[6. 常见问题 21](#_Toc351040808)

[6.1. 用replace还是insert？ 21](#_Toc351040810)

[6.2. 什么情况下需要使用预备执行语句？ 21](#_Toc351040811)

[附录 22](#_Toc351040812)

[Prepared Statement API示例 22](#_Toc351040813)

[SELECT示例 22](#_Toc351040814)

[REPLACE示例 25](#_Toc351040815)

# OceanBase介绍



RootServer/

UpdateServer（主）

RootServer/

UpdateServer（备）

Chunkserver/

MergeServer

OceanBase是一个由RootServer, UpdateServer, ChunkServer, MergeServer等角色组成的分布式数据库系统。RootServer是管理服务器，用于管理UpdateServer、ChunkServer和MergeServer等。ChunkServer是OceanBase的基线数据存储子系统，由多台机器构成，基线数据通常保存2~3副本并且保存在不同的ChunkServer上。UpdateServer是OceanBase的增量更新子系统，记录用户的修改，通常由一主一备或一主多备构成。MergeServer是OceanBase的用户接口子系统，接收用户的查询请求，并从对应ChunkServer上获得基线数据以及从UpdateServer上获得更新的增量，然后融合二者后返回用户。

SQL用户只需要和MergeServer交互，每一个MergeServer都可以视为一个SQL服务器。用户连接任何一台MergeServer都可以对全部表格数据执行SQL语句。

# 快速上手

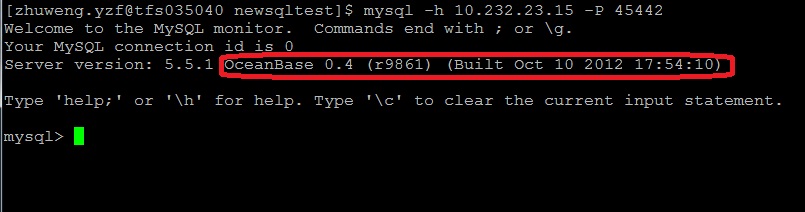
OceanBase完全兼容MySQL的网络协议，SQL用户可以使用mysql终端客户端、以及使用libmysqlclient的任意程序连接OceanBase。只要你熟悉mysql，那么你就可以直接使用OceanBase SQL服务了。下面我们以mysql终端客户端为例演示一下。

## 步骤一

请询问你的OceanBase DBA任意一台MergeServer的IP地址和SQL服务端口（默认端口号是**2828**）。@TODO改用LD\_PRELOAD=libobsql.so启动mysql的方式

## 步骤二

使用mysql终端连接MergeServer，如下图。



注意红色部分，如果你看到类似这样的字样，说明你已经成功登陆了OceanBase！像使用MySQL一样体验一下吧。

## 步骤三

我们下面可以创建一个表格，插入一些数据，然后查询出来。很简单，是不是：）



# 使用API

## C程序

OceanBase为应用程序提供了libobsql函数库[[2]](#footnote-3)。这个库在**二进制接口**上与mysql的libmysqlclient库完全兼容。也就是说，一个程序（例如mysql, mysqltest等）之前用libmysqlclient编译好，使用PRELOAD技术**无需重新编译**就可以使用来连接OceanBase。应用程序可以使用与libmysqlclient提供的API（mysql.h）完全相同的接口直接访问OceanBase SQL服务。下面是一个简单的例子程序的完整清单。

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <mysql/mysql.h> //是的，你没看错，就是mysql.h

**int** main(**int** argc, **char** \*argv[])

{

(**void**)argc;

(**void**)argv;

**const** **char**\* HOST = "127.0.0.1";

**int** PORT = 2828;

// 1. 初始化libmysqlclient

**if** (0 != mysql\_library\_init(0, NULL, NULL))

{

fprintf(stderr, "could not init mysql library\n");

exit(1);

}

MYSQL my\_;

ASSERT\_TRUE(NULL != mysql\_init(&my\_)); // 此处用ASSERT省略错误检查

fprintf(stderr, "Connecting server %s:%d...\n", HOST, PORT);

// 2. 连接OceanBase SQL服务

ASSERT\_TRUE(NULL != mysql\_real\_connect(&my\_, HOST, "", "", "test", PORT, NULL, 0));

// 3. 建表

**int** ret = 0;

ret = mysql\_query(&my\_, "drop table if exists ob\_new\_sql\_test");

**if** (0 != ret)

{

fprintf(stderr, "%s\n", mysql\_error(&my\_));

}

ASSERT\_EQ(0, ret);

ret = mysql\_query(&my\_, "create table if not exists ob\_new\_sql\_test "

"(c0 varchar(64), c1 int primary key, c2 int, "

"c3 int, c4 int, c5 int, c6 int, "

"c7 int, c8 int, c9 int, c10 int, "

"c11 int, c12 int, c13 int, c14 int,"

"c15 int)");

**if** (0 != ret)

{

fprintf(stderr, "%s\n", mysql\_error(&my\_));

}

ASSERT\_EQ(0, ret);

// 4. 插入数据

ret = mysql\_query(&my\_, "insert into ob\_new\_sql\_test (c0, c1, c2)values ('hello', 1, 2)");

**if** (0 != ret)

{

fprintf(stderr, "%s\n", mysql\_error(&my\_));

}

ASSERT\_EQ(0, ret);

// 5. 查询数据

ASSERT\_EQ(0, mysql\_query(&my\_, "select \* from ob\_new\_sql\_test"));

MYSQL\_RES \*res = mysql\_use\_result(&my\_); // 取结果集

ASSERT\_TRUE(NULL != res);

**unsigned** **int** num\_fields = mysql\_num\_fields(res); // 结果列数

ASSERT\_EQ(16U, num\_fields);

**unsigned** **long** \*lengths = NULL;

MYSQL\_ROW row;

row = mysql\_fetch\_row(res);

ASSERT\_TRUE(NULL != row);

lengths = mysql\_fetch\_lengths(res); // row中每个值的长度，0表示NULL

ASSERT\_TRUE(NULL != lengths);

**for** (**unsigned** **int** j = 0; j < num\_fields; ++j)

{

**if** (0 != j)

{

fprintf(stdout, ",");

}

fprintf(stdout, "%.\*s", row[j] ? **static\_cast**<**int**>(lengths[j])

: **static\_cast**<**int**>(**sizeof**("NULL")-1), row[j] ? row[j] : "NULL");

}

fprintf(stdout, "\n");

row = mysql\_fetch\_row(res);

ASSERT\_TRUE(NULL == row); // no more rows

mysql\_free\_result(res);

// 6. 删表

ret = mysql\_query(&my\_, "drop table if exists ob\_new\_sql\_test");

**if** (0 != ret)

{

fprintf(stderr, "%s\n", mysql\_error(&my\_));

}

// 7. destroy

mysql\_library\_end();

**return** 0;

}

恭喜你，你已经完成了一个从OceanBase中读写数据的程序。关于API的完整说明，请参考**MySQL C API**文档[[3]](#footnote-4)。

### 编译方法一：

首先，需要保证你的开发机上已经安装了mysql应用开发包，在RHEL或者fedora上可以使用如下命令安装：@TODO obsql如何安装

yum install mysql-devel

然后，拷贝上面的程序到一个文件ob\_sql\_example.cpp中，用如下命令编译：

g++ -o ob\_sql\_example ob\_sql\_example.cpp –lmysqlclient

程序编译好后，用下面的命令运行：

LD\_LIBRARY\_PATH=<path\_to\_libobsql> LD\_PRELOAD=libobsql.so ./ob\_sql\_example

### 编译方法二：

你也可以在编译的时候直接使用libobsql代替libmysqlclient，这样产生的可执行程序无需使用LD\_PRELOAD的方式来执行。应用程序源码不需要任何修改，还是包含mysql.h和使用其中的API。@TODO补充libobsql执行示例的环境变量设置和头文件的

g++ -o ob\_sql\_example ob\_sql\_example.cpp –lobsql

./ob\_sql\_example

## Java程序

OceanBase提供了符合Java标准的DataSource[[4]](#footnote-5)，OBGroupDataSource是javax.sql.DataSource的一个实现，Java应用程序可以使用OBGroupDataSource获得与OceanBase服务器交互的连接。这个库在mysql-connector-java的基础上，实现了OceanBase多集群间流量分配，多MergeServer间的负载均衡，以及在单个MergeServer发生失效的情况下自动进行切换容灾等功能。下面是一个简单的例子程序的完整清单。@TODO Java PS的例子

**package** com.alipay.oceanbase;

**import** java.io.\*;

**import** java.util.\*;

**import** java.sql.\*;

**import** javax.sql.DataSource;

**import** com.alipay.oceanbase.OBGroupDataSource;

**public** **class** ObJDBCExample

{

**public** **static** **void** main(String[] argv)

{

**try** {

// init the data source

OBGroupDataSource ds = **new** OBGroupDataSource();

List<String> instanceList = **new** ArrayList<String>();

instanceList.add("10.232.36.185:45447"); // OceanBase RootServer

ds.setInstanceList(instanceList); //推荐使用setConfigURL的方式，见后

ds.setUserName("user");

ds.setPasswd("password");

ds.init();

// JDBC stuff

Connection conn = ds.getConnection();

Statement stmt = conn.createStatement();

stmt.executeUpdate("drop table if exists t1");

stmt.executeUpdate("create table t1 (c1 int primary key, c2 varchar)");

stmt.executeUpdate("insert into t1 values (1, 'hello ob'), (2, 'hello java')");

ResultSet rs = stmt.executeQuery("select \* from t1");

ResultSetMetaData rsmd = rs.getMetaData();

**int** columnCount = rsmd.getColumnCount();

System.out.println("column count=" + columnCount);

**while** (rs.next()){

System.out.println(rs.getInt("c1") + "|" + rs.getString("c2"));

}

System.out.println("finish");

}

**catch** (SQLException ex) {

System.out.println(ex);

}

**catch** (Exception ex){

System.out.println(ex);

}

}

}

Java程序也可以通过Spring等框架来使用OBGroupDataSource。例如使用Spring JdbcTemplate：

OBGroupDataSource ds = **new** OBGroupDataSource();

ds.setUserName("user");

ds.setPasswd("pass");

ds.setDbName("test");

ds.setConfigURL("http://10.232.102.182:8080/diamond-server/config.co?dataId=xxx"); //这个URL由OceanBase服务方提供

ds.init();

JdbcTemplate jtp = **new** JdbcTemplate();

jtp.setDataSource(ds);

String sql = "select 1 from dual";

**int** ret = jtp.queryForInt(sql);

通过Bean使用的Spring配置：

<bean id="ds" **class**="com.alipay.oceanbase.OBGroupDataSource" init-method="init">

<property name="userName" value="user" />

<property name="passwd" value="pass"/>

<property name="dbName" value="test" />

<property name="configURL" value="http://10.232.102.182:8080/diamond-server/config.co?dataId=xxx" />

</bean>

OBDataSource库主要的maven依赖关系如下（请在你的pom.xml中添加）：

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

</dependency>

<dependency>

<groupId>com.taobao.tbdatasource</groupId>

<artifactId>tbdatasource</artifactId>

<version>2.0.3-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>com.alipay.oceanbase</groupId>

<artifactId>obdatasource</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

# 陷阱和缺陷

## 支持的语句

OceanBase SQL语法在参考MySQL的语法同时，尽量遵循SQL92标准，在必要的时候还参考了Oracle。

SQL语句中的关键字，以及表名、列名、函数名等都是大小写不敏感的。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。

遵循SQL92标准，字符串字面量要用单引号表示，见5.2.1.7。而双引号扩住的名字被解释为表名、列名或函数名，而**不是**字符串。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, select, insert, replace, delete, update, set, show等，具体语法请见SQL语法参考。权限控制等管理语句参考《OceanBase SQL管理员手册》。

## 数据类型

OceanBase不支持time，date类型，只支持datetime和timestamp类型（这两者等价）。用户可以使用datetime, timestamp类型的字面量（literal，见5.2.1.6）。

OceanBase暂不支持decimal类型，将在下一个版本提供。

OceanBase提供了两个特殊的数据类型：Createtime类型的列中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；Modifytime类型的列中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。这两个类型的列也不能作为主键的组成部分。

OceanBase没有字符集的概念。Char，Varchar，Binary，Varbinary在OceanBase内部都存储为不解释内容的二进制变长字符串，类似于Varbinary。这种类型的比较使用的是**字节序**。此外，Create table时定义的varchar列的最大长度也是不起作用的。例如varchar(32)，实际上可以插入大于32字节的串，不认为是错误。

在OceanBase内部，Tinyint, Smallint, Mediumint, Int, Integer, Bigint无论语义还是实现都是等价的，存储为8字节有符号整型，即int64\_t。

OceanBase支持float和double浮点数类型，分别表示4字节和8字节的浮点数，其语义等同于IEEE 754浮点数。

OceanBase 返回给客户端的数据类型目前只可能是如下几种，客户端在绑定变量的时候需要特别注意。

|  |  |
| --- | --- |
| Data Type | MySQL Data Type |
| int | MYSQL\_TYPE\_LONGLONG |
| float | MYSQL\_TYPE\_FLOAT |
| double | MYSQL\_TYPE\_DOUBLE |
| datetime | MYSQL\_TYPE\_DATETIME |
| string | MYSQL\_TYPE\_VAR\_STRING |
| bool | MYSQL\_TYPE\_TINY |
| decimal | MYSQL\_TYPE\_NEWDECIMAL |

## 浮点数

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。

## 函数

OceanBase遵循SQL标准，不支持MySQL COUNT(DISTINCT expr, expr, [expr...])的语法，即聚集函数中只能出现一个value表达式，不支持多个（例如count(distinct c1, c2)）。

OceanBase目前支持的系统函数非常有限，只有length、substr、cast、current\_time、current\_date、current\_timestamp、trim、lower、upper、coalesce等。我们会根据用户的需求优先级不断添加新的函数。

## Join的限制

目前，Join不支持USING子句，见后面的Join语法。并且，Join的连接条件中必须**至少有一个**等值连接条件。例如，下面的两个例子是支持的：

select \* from t1 inner join t2 on t1.c1 = t2.c1 and t1.c2 < 1000;

select \* from t1, t2 where t1.c1 = t2.c1 and t1.c2 < 1000;

而下面的语句是不支持的：

select \* from t1 inner join t2 on t1.c1 > t2.c1 and t1.c2 < 1000;

## update和delete的限制

目前，update和delete语句必须指定primary key的所有列作为限制条件，并且只支持单行。换言之，update、delete操作必须指定单行的主键。例如

create table t1 (c1 int, c2 int, c3 int, primary key(c1, c2));

delete from t1 where c1 = 1 and c2 = 2;

update set c3 = c1 + 1 where c1 = 1 and c2 = 2;

而下面的语句都是不支持的：

delete from t1;

delete from t1 where c1 = 1;

update set c3 = c1 + 1;

update set c3 = c1 + 1 where c1 > 1 and c2 > 2;

## 事务中SELECT的BUG

目前在事务中（start transaction或者begin开启事务）执行SELECT语句，是不能读取当当前事务中未提交的数据的。这是一个已知的功能BUG。用户如果有这个需求，可以暂时用SELECT … FOR UPDATE语句代替。

## REPLACE语句返回的affected\_rows值

使用API执行REPLACE语句时，执行完成后可以获得affected\_rows值。但是，目前OB返回的affected\_rows的值是错误的。应用程序**不能**依赖于这个值来判断行是否已经存在，或者replace是否成功（成功的情况下，replace返回的affected\_rows可能也为0）。

# SQL语法参考



## 数据定义语句

### create table语句

CREATE TABLE [IF NOT EXISTS] ***tbl\_name***

    (***create\_definition***,...)

    [***table\_options***]

***create\_definition***:

***col\_name*** ***column\_definition***

  | PRIMARY KEY (***index\_col\_name***,...)

***column\_definition***:

***data\_type*** [NOT NULL | NULL] [DEFAULT ***default\_value***]

      [AUTO\_INCREMENT] [[PRIMARY] KEY]

***data\_type***:

TINYINT[(***length***)]

  | SMALLINT[(***length***)]

  | MEDIUMINT[(***length***)]

  | INT[(***length***)]

  | INTEGER[(***length***)]

  | BIGINT[(***length***)]

  | REAL[(***length***,***decimals***)]

  | DOUBLE[(***length***,***decimals***)]

  | FLOAT[(***length***,***decimals***)]

  | DECIMAL[(***length***[,***decimals***])]

  | NUMERIC[(***length***[,***decimals***])]

  | TIMESTAMP

  | DATETIME

  | CHAR[(***length***)]

  | VARCHAR(***length***)

  | BINARY[(***length***)]

  | VARBINARY(***length***)

  | CREATETIME

  | MODIFYTIME

***table\_option***:

    EXPIRE\_INFO [=] ***‘expire info’***

  | TABLET\_MAX\_SIZE [=] ***value***

  | REPLICA\_NUM [=] ***value***

  | COMPRESS\_METHOD [=] ‘compress method name’

  | USE\_BLOOM\_FILTER [=] {0 | 1}

***table\_option***中，EXPIRE\_INFO是OceanBase提供的自动删除过期数据的功能配置。TABLET\_MAX\_SIZE是这个表的Tablet最大尺寸，单位为字节，默认为256MB。REPLICA\_NUM为这个表的tablet复本数，默认值为3。COMPRESS\_METHOD为存储数据时使用的压缩方法名，目前提供的方法有none（默认值，表示不作压缩）、lzo\_1\_0、snappy\_1\_0等。USE\_BLOOM\_FILTER是配置对于本表OceanBase读取数据时是否使用Bloom Filter，默认值为0。

注意，NOT NULL，DEFAULT，AUTO\_INCREMENT选项可以出现，但目前**尚未实现**。

建表的时候，必须指定Primary Key，因为OB内部数据以btree为索引，按照Primary Key排序。同样的原因，插入数据时，Primary Key的列值不允许为NULL。

如果指定IF NOT EXISTS，则如果同名表已经存在，不报错。

### drop table语句

DROP TABLE [IF EXISTS]

***tbl\_name*** [, ***tbl\_name***] ...

如果指定IF EXISTS，则如果某个表不存在，不报错。

### alter table语句

ALTER TABLE

***tbl\_name*** ***alter\_column\_action*** [***,alter\_column\_action***] ...

***alter\_column\_action:***

ADD [COLUMN] ***col\_name*** ***column\_definition***

| DROP [COLUMN] ***col\_name***

目前只支持增加和删除列。

## 数据操作语句

### select语句

#### 基本句法

SELECT

[ALL | DISTINCT]

***select\_expr*** [, ***select\_expr*** ...]

FROM ***table\_references***

[WHERE ***where\_condition***]

[GROUP BY {***col\_name*** | ***expr*** | ***position***}

[ASC | DESC], ...]

[HAVING ***where\_condition***]

[ORDER BY {***col\_name*** | ***expr*** | ***position***}

[ASC | DESC], ...]

[LIMIT {[***offset***,] ***row\_count*** | ***row\_count*** OFFSET ***offset***}]

#### Join句法

SELECT语法中，FROM子句中的table\_references可以用下面的语法来表达Join关系：

***table\_references:***

***table\_reference*** [, ***table\_reference***] ...

***table\_reference***:

***table\_factor***

| ***join\_table***

***table\_factor***:

***tbl\_name*** [[AS] ***alias***]

| ***table\_subquery*** [AS] ***alias***

| ( ***table\_references*** )

***join\_table***:

***table\_reference*** [INNER] JOIN ***table\_factor*** ***join\_condition***

| ***table\_reference*** {LEFT|RIGHT} [OUTER] JOIN ***table\_reference*** ***join\_condition***

***join\_condition***:

ON ***conditional\_expr***

#### 集合操作

SELECT ...

***set\_op*** SELECT ...

...

[ORDER BY {***col\_name*** | ***expr*** | ***position***}

[ASC | DESC], ...]

[LIMIT {[***offset***,] ***row\_count*** | ***row\_count*** OFFSET ***offset***}]

***set\_op:***

UNION [ALL | DISTINCT]

| INTERSECT [ALL | DISTINCT]

| EXCEPT [ALL | DISTINCT]

#### DUAL虚拟表

SELECT

[ALL | DISTINCT]

***select\_expr*** [, ***select\_expr*** ...]

[FROM DUAL

[WHERE ***where\_condition***]]

[LIMIT {[***offset***,] ***row\_count*** | ***row\_count*** OFFSET ***offset***}]

DUAL是一个虚拟的表，可以视为一个一行零列的表。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。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。

#### SELECT … FOR UPDATE

在5.2.1.1小节所示的简单select语句上可以添加for update子句。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。

注意的是，目前实现有如下限制：

* 必须是单表查询；
* Where条件中必须限定单行，同4.6小节。

#### 表达式

***expr***:

***expr*** OR ***expr***

| ***expr*** AND ***expr***

| NOT ***expr***

| ***boolean\_primary*** IS [NOT] {TRUE | FALSE | UNKNOWN}

| ***boolean\_primary***

***boolean\_primary***:

***boolean\_primary*** IS [NOT] NULL

| ***boolean\_primary*** ***comparison\_operator*** ***predicate***

| ***predicate***

***comparison\_operator***: = | >= | > | <= | < | !=

***predicate***:

***bit\_expr*** [NOT] IN (***subquery***)

| ***bit\_expr*** [NOT] IN (***expr*** [, ***expr***] ...)

| ***bit\_expr*** [NOT] BETWEEN ***bit\_expr*** AND ***predicate***

| ***bit\_expr*** [NOT] LIKE ***simple\_expr***

| ***bit\_expr***

***bit\_expr***:

***bit\_expr*** + ***bit\_expr***

| ***bit\_expr*** - ***bit\_expr***

| ***bit\_expr*** \* ***bit\_expr***

| ***bit\_expr*** / ***bit\_expr***

| ***bit\_expr*** MOD ***bit\_expr***

| ***bit\_expr*** % ***bit\_expr***

| ***simple\_expr***

***simple\_expr***:

***literal***

| ***identifier***

| ***function\_call***

| ***simple\_expr*** || ***simple\_expr***

| + ***simple\_expr***

| - ***simple\_expr***

| (***expr*** [, ***expr***] ...)

| (***subquery***)

| ***case\_expr***

#### 字面量

* 字符串，型如’a string’，注意不能使用双引号””
* 整数，型如12或0或-1234
* 浮点数，型如12.34或-1.234e+10或13.00
* Timestamp类型，型如Timestamp’2012-10-12 14:49:00’或Timestamp’2012-10-12 14:49:00.’或Timestamp’2012-10-12 14:49:00.123456’
* 十六进制表示的字节串，型如X’A23FEF’
* 布尔类型，型如true, false, TRUE, FALSE
* NULL值，型如NULL

### replace语句

REPLACE INTO ***tbl\_name*** [(***col\_name***,...)]

VALUES (***expr***,...),(...),...

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。

### insert语句

INSERT INTO ***tbl\_name*** [(***col\_name***,...)]

VALUES (***expr***,...),(...),...

注意，因为OceanBase中所有的表格都以主键为唯一索引，所以insert的行必须包含所有主键列的值，且主键列不能为NULL。

### update语句

UPDATE ***table\_reference***

SET ***col\_name1***={***expr1***|DEFAULT} [, ***col\_name2***={***expr2***|DEFAULT}] ...

[WHERE ***where\_condition***]

注意，update目前只支持指定primary key条件的单行更新，且DEFAULT暂不支持。

### delete语句

DELETE FROM ***tbl\_name***

[WHERE ***where\_condition***]

注意，delete目前只支持指定primary key条件的单行删除。

## 事务相关语句

### 开始事务

START TRANSACTION [WITH CONSISTENT SNAPSHOT]

BEGIN [WORK]

开始一个事务。若不在一个本语句显式开始的事务内，一般单条语句的执行都作为一个独立事务立即提交[[5]](#footnote-6)。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。

### 结束事务

COMMIT [WORK]

ROLLBACK [WORK]

COMMIT提交当前事务。ROLLBACK回滚当前事务。

## 变量相关语句

### set语句

SET ***variable\_assignment*** [, ***variable\_assignment***] ...

***variable\_assignment***:

***user\_var\_name*** = ***expr***

| [GLOBAL | SESSION] ***system\_var\_name*** = ***expr***

| [@@global. | @@session. | @@]***system\_var\_name*** = ***expr***

OceanBase SQL中，变量有两类：用户变量和系统变量，其中，系统变量又可以是全局系统变量或会话系统变量。如下表所示：

|  |  |  |
| --- | --- | --- |
| 变量类型\作用域 | 全局 | 会话 |
| 系统 | 变量名前加GLOBAL或@@global. | 变量名前加SESSION或@@session.或@@或LOCAL或@@local.，或什么都不加 |
| 用户 | 变量名前加@ |  |

修改全局系统变量需要当前用户拥有super\_priv权限，而修改用户变量或系统会话变量不需要任何权限。修改后的会话系统变量只在当前会话内有效。

### show variables语句[[6]](#footnote-7)

SHOW [GLOBAL | SESSION] VARIABLES

[LIKE '***pattern***' | WHERE ***expr***]

本语句可以获得当前所有系统变量的值。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。

## 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。

注意，本节介绍的预备执行语句SQL语法层面直接支持的预备执行，普通用户请**不要使用**。因为接口和协议的缘故，这组语句不能高效执行。最高效执行预备语句的方法是通过程序调用OceanBase提供的编程API中预备执行相关的接口。Java语言程序可以使用OceanBase data source JDBC库，C语言类程序可以使用libobmysql函数库，详见附录中Prepared statement的例子。

### prepare语句

PREPARE ***stmt\_name*** FROM ***preparable\_stmt***

预备***preparable\_stmt***中的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。5.2小节中列出的所有数据操作语句，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。注意，问号只能出现在SQL语句中常量出现的位置。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子句后只能使用用set语句定义的用户变量。

### deallocate语句

{DEALLOCATE | 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。

### 一个例子

prepare stmt1 from select \* from t1 where uid=? and gid=?;

set @uid = 1000, @gid = 2000;

execute stmt1 using @uid, @gid;

set @uid = 1001;

execute stmt1 using @uid, @gid;

deallocate prepare stmt1;

# 常见问题



## 用replace还是insert？

因为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Chunkserver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，则优先使用replace。例如，在很多准备数据的场景中，应用可以保证数据主键的唯一性，此时推荐使用replace。

## 什么情况下需要使用预备执行语句？

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。

# 附录

## Prepared Statement API示例[[7]](#footnote-8)

对Prepared statement有一组API来专门提供支持。对于反复执行的SQL语句，使用prepared statement可以极大地提高性能。目前可以被预备执行的SQL语句包括所有5种DML语句：select, replace, insert, update, delete。

### SELECT示例

**void** ObPSTest::select\_test(**int** exp\_row\_count)

{

MYSQL\_STMT \*stmt = mysql\_stmt\_init(&my\_);

ASSERT\_TRUE(stmt);

**static** **const** **char**\* SELECT\_QUERY = "select \* from ob\_ps\_test where c1 >= ?";

// 1. prepare

**int** ret = mysql\_stmt\_prepare(stmt, SELECT\_QUERY, strlen(SELECT\_QUERY));

ASSERT\_QUERY\_RET(ret);

**int** param\_count = mysql\_stmt\_param\_count(stmt);

ASSERT\_EQ(1, param\_count);

MYSQL\_RES \*prepare\_meta = mysql\_stmt\_result\_metadata(stmt);

ASSERT\_TRUE(prepare\_meta);

**int** num\_fields = mysql\_num\_fields(prepare\_meta);

ASSERT\_EQ(3, num\_fields);

MYSQL\_FIELD \*fields = mysql\_fetch\_fields(prepare\_meta);

ASSERT\_TRUE(fields);

// print detailed field info

**for** (**int** i = 0; i < num\_fields; ++i)

{

fprintf(stdout, "field=%u name=%s org\_name=%s table=%s org\_table=%s db=%s catalog=%s length=%ld max\_length=%ld type=%d\n",

i, fields[i].name, fields[i].org\_name,

fields[i].table, fields[i].org\_table,

fields[i].db, fields[i].catalog,

fields[i].length, fields[i].max\_length,

fields[i].type);

}

// 2. bind params

MYSQL\_BIND bind[1];

memset(bind, 0, **sizeof**(bind));

/\* INTEGER PARAM \*/

/\* This is a number type, so there is no need

to specify buffer\_length \*/

**int** int\_data = 0;

bind[0].buffer\_type= MYSQL\_TYPE\_LONG;

bind[0].buffer= (**char** \*)&int\_data;

bind[0].is\_null= 0;

bind[0].length= 0;

ret = mysql\_stmt\_bind\_param(stmt, bind);

ASSERT\_QUERY\_RET(ret);

// 3. execute

int\_data = 2;

ret = mysql\_stmt\_execute(stmt);

ASSERT\_QUERY\_RET(ret);

// 4. bind results

MYSQL\_BIND row[3];

my\_bool is\_null[3];

my\_bool error[3];

**unsigned** **long** length[3];

memset(row, 0, **sizeof**(row));

// int column

**int** c1\_data;

row[0].buffer\_type = MYSQL\_TYPE\_LONG;

row[0].buffer = (**char**\*)&c1\_data;

row[0].is\_null = &is\_null[0];

row[0].length = &length[0];

row[0].error = &error[0];

**int** c2\_data;

row[1].buffer\_type = MYSQL\_TYPE\_LONG;

row[1].buffer = (**char**\*)&c2\_data;

row[1].is\_null = &is\_null[1];

row[1].length = &length[1];

row[1].error = &error[1];

// string column

**static** **const** **int** STRING\_SIZE = 64;

**char** c3\_data[STRING\_SIZE];

row[2].buffer\_type = MYSQL\_TYPE\_VAR\_STRING;

row[2].buffer = c3\_data;

row[2].buffer\_length = STRING\_SIZE;

row[2].is\_null = &is\_null[2];

row[2].length = &length[2];

row[2].error = &error[2];

ret = mysql\_stmt\_bind\_result(stmt, row);

ASSERT\_QUERY\_RET(ret);

// 5. fetch rows

fprintf(stdout, "Fetching results...\n");

**int** row\_count = 0;

**while** (!mysql\_stmt\_fetch(stmt))

{

++row\_count;

fprintf(stdout, "row=%d ", row\_count);

**if** (is\_null[0])

{

fprintf(stdout, "c1=NULL ");

}

**else**

{

fprintf(stdout, "c1=%d(%ld) ", c1\_data, length[0]);

ASSERT\_EQ(row\_count+1, c1\_data);

}

**if** (is\_null[1])

{

fprintf(stdout, "c2=NULL ");

}

**else**

{

fprintf(stdout, "c2=%d(%ld) ", c2\_data, length[1]);

}

**if** (is\_null[2])

{

fprintf(stdout, "c3=NULL\n");

}

**else**

{

fprintf(stdout, "c3=%s(%ld)\n", c3\_data, length[2]);

}

} // end while

ASSERT\_EQ(exp\_row\_count, row\_count);

mysql\_free\_result(prepare\_meta);

ret = mysql\_stmt\_close(stmt);

ASSERT\_QUERY\_RET(ret);

}

### REPLACE示例

**void** ObPSTest::replace\_test()

{

MYSQL\_STMT \*stmt = mysql\_stmt\_init(&my\_);

ASSERT\_TRUE(stmt);

**const** **char**\* QUERY = "replace into ob\_ps\_test values (?, ?, ?)";

// 1. prepare

**int** ret = mysql\_stmt\_prepare(stmt, QUERY, strlen(QUERY));

ASSERT\_QUERY\_RET(ret);

**int** param\_count = mysql\_stmt\_param\_count(stmt);

ASSERT\_EQ(3, param\_count);

// 2. bind params

MYSQL\_BIND bind[3];

memset(bind, 0, **sizeof**(bind));

/\* INTEGER PARAM \*/

/\* This is a number type, so there is no need

to specify buffer\_length \*/

**int** int\_data = 0;

bind[0].buffer\_type= MYSQL\_TYPE\_LONG;

bind[0].buffer= (**char** \*)&int\_data;

bind[0].is\_null= 0;

bind[0].length= 0;

my\_bool is\_null = 0;

bind[1].buffer\_type= MYSQL\_TYPE\_LONG;

bind[1].buffer= 0;

bind[1].is\_null= &is\_null;

bind[1].length= 0;

**static** **const** **int** STRING\_SIZE = 64;

**char** str\_data[STRING\_SIZE];

**unsigned** **long** str\_length;

bind[2].buffer\_type= MYSQL\_TYPE\_STRING;

bind[2].buffer= str\_data;

bind[2].is\_null= 0;

bind[2].length= &str\_length;

ret = mysql\_stmt\_bind\_param(stmt, bind);

ASSERT\_QUERY\_RET(ret);

// 3. execute

int\_data = 4;

is\_null = 1;

strncpy(str\_data, "ps1", STRING\_SIZE);

str\_length = strlen(str\_data);

ret = mysql\_stmt\_execute(stmt);

ASSERT\_QUERY\_RET(ret);

// 4. results

my\_ulonglong affected\_rows = mysql\_stmt\_affected\_rows(stmt);

ASSERT\_EQ(0U, affected\_rows); // this is a known bug

int\_data = 5;

is\_null = 1;

strncpy(str\_data, "ps2", STRING\_SIZE);

str\_length = strlen(str\_data);

ret = mysql\_stmt\_execute(stmt);

ASSERT\_QUERY\_RET(ret);

affected\_rows = mysql\_stmt\_affected\_rows(stmt);

ASSERT\_EQ(0U, affected\_rows);

int\_data = 5;

is\_null = 1;

strncpy(str\_data, "ps3", STRING\_SIZE);

str\_length = strlen(str\_data);

ret = mysql\_stmt\_execute(stmt);

ASSERT\_QUERY\_RET(ret);

affected\_rows = mysql\_stmt\_affected\_rows(stmt);

ASSERT\_EQ(0U, affected\_rows);

// 5. clean up

ret = mysql\_stmt\_close(stmt);

ASSERT\_QUERY\_RET(ret);

}

1. 本手册适用于OceanBase 0.4.1 [↑](#footnote-ref-2)
2. 目前0.4.1版本没有发布libobsql，用户暂时可以用libmysql代替。 [↑](#footnote-ref-3)
3. 见http://dev.mysql.com/doc/refman/5.5/en/c.html [↑](#footnote-ref-4)
4. 见http://docs.oracle.com/javase/6/docs/api/javax/sql/DataSource.html [↑](#footnote-ref-5)
5. 目前暂未实现默认语句不自动提交，即autocommit=false的功能，后续会支持。 [↑](#footnote-ref-6)
6. 其他SHOW语句的变种，如show columns等，请参考《OceanBase SQL管理员手册》。 [↑](#footnote-ref-7)
7. 还可以参考mysql手册中的示例，见<http://dev.mysql.com/doc/refman/5.0/en/mysql-stmt-execute.html>和<http://dev.mysql.com/doc/refman/5.0/en/mysql-stmt-fetch.html>。 [↑](#footnote-ref-8)