存储过程使用手册

修订历史

版本	修订日期	修订描述	作者	备注
0.1	2016-1-3	存储过程使用手册	祝君	无

1 简介

存储程序需要在oceanbase数据库中有__all_procedure表。这个表在系统安装时自动创建。oceanbase中的存储过程语法是mysql存储过程语法的子集，其中有一些特性不被支持，在本手册中将详细介绍oceanbase中的存储过程语法规则。

1.1 综述

存储过程是数据库系统的一个功能拓展，将一组带有特定功能的SQL语句以及控制流程语句存储在服务器端，用户通过存储过程的名称以及相应参数进行调用。存储过程支持变量，顺序，条件和循环等控制结构，存储过程可以定义参数这使得存储过程具有很强的模块性同时具有批处理性。虽然存储过程没有返回值，但是存储过程可以通过定义输出参数来实现返回结果的功能。这些特性都使得存储过程成为数据库系统必不可少的功能。
由于存储过程的源码是存储在服务器的，因此客户端可以通过存储过程名以及参数调用，不需要把这些SQL语句集合发送到服务器，这样的话可以减少网络中传输的数据量，降低网络负荷。

2 存储过程的语法

2.1 CREATE PROCEDURE

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])
    routine_body

proc_parameter:
[ IN | OUT | INOUT ] param_name type

type:
Any valid OceanBase data type

routine_body:
    Valid SQL procedure statement or statements
```

> Tips：*routine_body*包含合法的*SQL*过程语句,可以使用复合语句语法，*oceanbase*中一个存储过程只允许一个*BEGIN...END*语句块，暂时不支持多个。

例*1*

```
CREATE PROCEDURE demo_sp(IN @in_param int, OUT @out_patam int)
BEGIN
END
#创建了一个名称为demo_sp并且一个参数是int类型的输入参数，另一个是int类型的输出参数
```

2.2 DROP PROCEDURE

```
DROP PROCEDURE  [IF EXISTS] sp_name
```

例*2*

```
DROP PROCEDURE IF EXISTS demo_sp
```

2.3 CALL语句

```
CALL sp_name([parameter[,...]])
```

例*3*

```
CALL demo_sp(100, @out_param)
#调用名称为demo_sp的存储过程，第一个参数为输入参数，第二个参数为输出参数
```

2.4 BEGIN … END复合语句

```
BEGIN
    [statement_list]
END
```

存储过程可以使用BEGIN … END复合语句来包含多个语句。statement_list 代表一个或多个语句的列表。statement_list之内每个语句都必须用分号（ ； ）来结尾。

2.5 存储过程中的变量

你可以在子程序中声明并使用变量，只能使用在begin-end之间，不能在控制语句内部。

2.5.1 DECLARE局部变量

```
DECLARE var_name[,...] type [DEFAULT value]
```

这个语句被用来声明局部变量。要给变量提供一个默认值，请包含一个DEFAULT子句。值可以被指定为一个表达式，不需要为一个常数。如果没有DEFAULT子句，初始值为NULL(INT类型默认值为0)。

> Tips：*var_name* 使用@+变量名，默认值只支持常量，不支持表达式

例*3*

```
DECLARE @param1 int default 0;           #定义一个变量param1默认值为0
DECLARE @param2 int;                     #定义一个变量param2默认值为NULL
DECLARE @param3, @param4 int default 0; #定义两个变量param3与param4默
认值为0
```

2.5.2 变量SET语句

```
SET var_name = expr [, var_name = expr] ...
```

在存储程序中的SET语句是一般SET语句的扩展版本。被参考变量可能是子程序内声明的变量。

> Tips：*SET*操作不支持+=运算符

例*4*

```
DECLARE @param1 int;
SET @param1 = 10;            #将param1赋值为10
SET @param1 = @param1 + 1;   #将param1的值加1
SET @param1 = 3 + 5;         #将param1的值赋值为等号右边表达式的值
```

2.5.3 SELECT … INTO语句

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

这个SELECT语法把选定的列直接存储到变量。因此，只有单一的行可以被取回。

> Tips：变量一定要声明，类型要匹配。

例*5*

```
DECLARE @email_param varchar; #定义一个变量用来存储SELECT INTO查询出来的
SELECT email INTO @email_param FROM user WHERE id = 1;
#user表由id和email字段组成
```

2.6 游标

简单游标在存储过程内被支持。语法如同在嵌入的SQL中。

2.6.1 声明游标

```
DECLARE cursor_name CURSOR FOR select_statement
```

这个语句声明一个游标。也可以在子程序中定义多个游标，但是一个块中的每一个游标必须有唯一的名字。

> Tips：*SELECT*语句不能有*INTO*子句。

例*6*

```
DECLARE demo_cursor CURSOR FOR SELECT id,email FROM user WHERE i
d=1;
#定义一个名称为demo_cursor的游标
```

2.6.2 游标OPEN语句

```
OPEN cursor_name
```

这个语句打开先前声明的游标。

2.6.3 游标FETCH语句

```
FETCH cursor_name INTO var_name [, var_name] ...

FETCH cursor_name FIRST INTO var_name[, var_name] ...

FETCH cursor_name LAST INTO var_name[, var_name] ...

FETCH cursor_name ABSOLUTE number_val INTO var_name[, var_name] ...

FETCH cursor_name NEXT RELATIVE number_val INTO var_name[, var_nam
e] ...
```

这个语句用指定的打开游标读取下一行（如果有下一行的话），并且前进游标指针。

例*7*

```
DECLARE @id int;
DECLARE @email varchar;
OPEN demo_cursor;                       #打开之前定义的demo_cursor
FETCH demo_cursor INTO @id, @email;     #将FECTH到的值存入声明的变量中
```

2.6.4 游标CLOSE语句

```
CLOSE cursor_name
```

这个语句关闭先前打开的游标。

2.7 流程控制构造

IF, CASE, WHILE 构造被完全实现。这些构造可能每个包含要么一个单独语句，要么是使用BEGIN … END复合语句的一块语句。构造可以被嵌套。

> Tips：目前还不支持*FOR,LOOP,REPEAT,LEAVE*

2.7.1 IF语句

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

IF实现了一个基本的条件构造。如果search_condition求值为真，相应的SQL语句列表被执行。如果没有search_condition匹配，在ELSE子句里的语句列表被执行。statement_list可以包括一个或多个语句。
例*9*

```
DECLARE @cond_param int default 0;
DECLARE @result_param int;
IF @cond_param > 0 THEN      #如果cond_param大于0则把result_param赋值为
100
    SET @result_param = 100;
ELSE
    SET @result_param = -100;
END IF
```

2.7.2 CASE语句

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

存储过程的CASE语句实现一个复杂的条件构造。如果case_value等于when_value 求值为真，相应的SQL被执行。如果没有搜索条件匹配，在ELSE子句里的语句被执行。

> Tips：*when_value*不能使用结果为*bool*类型的表达式

2.7.3 WHILE语句

```
WHILE search_condition DO
    statement_list
END WHILE
```

WHILE语句内的语句或语句群被重复，直至search_condition 为真。

2.8 可以成为statement_list的语句

```
SELECT 语句
UPDATE 语句
INSERT 语句
DELETE 语句
SELECT INTO 语句
IF 构造
WHILE 构造
CASE 构造
SET 语句
DECLARE 语句
游标OPEN 语句
游标FETCH 语句
游标CLOSE 语句
```

3 OceanBase的JDBC使用

3.1 加载驱动

获取oceanbase-jdbc-connector.jar引入项目

```
Class.forName("com.ecnu.ob.ObDriver");
```

使用Class.forName加载驱动，驱动的名称为**com.ecnu.ob.ObDriver**

3.2 获取连接

```
Connection connect = DriverManager.getConnection(
                    "jdbc:oceanbase://10.11.1.190:9004","admin","ad
min");
```

连接数据的url的格式为**jdbc:oceanbase://host:port**

3.3 使用JDBC调用存储过程

```
CallableStatement call= connect.prepareCall("call demo_sp(?,?)");
call.setFloat(1, 2);
call.registerOutParameter(2, Types.VARCHAR);
call.executeQuery();
String out=call.getString(2);
System.out.println(out);
```

4 使用限制条件和注意事项

1. 在客户端使用jdbc调用存储过程，不支持INOUT类型参数
2. WHILE构造中一定要确保有跳出条件，否则程序会陷入死循环
3. 数据类型只支持OceanBase中现有的类型，其他类型不支持
4. 在存储过程中定义了一个游标，在最后一定需要显示的关闭