

MySQL 5.0 版本开始支持存储过程。

存储过程（Stored Procedure）是一种在数据库中存储复杂程序，以便外部程序调用的一种数据库对象。

存储过程是为了完成特定功能的SQL语句集，经编译创建并保存在数据库中，用户可通过指定存储过程的名字并给定参数(需要时)来调用执行。

存储过程思想上很简单，就是数据库 SQL 语言层面的代码封装与重用。

## 优点

- 存储过程可封装，并隐藏复杂的商业逻辑。
- 存储过程可以回传值，并可以接受参数。
- 存储过程无法使用 SELECT 指令来运行，因为它是子程序，与查看表，数据表或用户定义函数不同。
- 存储过程可以用在数据检验，强制实行商业逻辑等。

## 缺点

- 存储过程，往往定制化于特定的数据库上，因为支持的编程语言不同。当切换到其他厂商的数据库系统时，需要重写原有的存储过程。
- 存储过程的性能调校与撰写，受限于各种数据库系统。

# 一、存储过程的创建和调用

---

- 存储过程就是具有名字的一段代码，用来完成一个特定的功能。
- 创建的存储过程保存在数据库的数据字典中。

## 创建存储过程

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type
```

```

characteristic:
    COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement

[begin_label:] BEGIN
    [statement_list]
    .....
END [end_label]

```

## MYSQL 存储过程中的关键语法

声明语句结束符，可以自定义:

```

DELIMITER $$
或
DELIMITER //

```

声明存储过程:

```
CREATE PROCEDURE demo_in_parameter(IN p_in int)
```

存储过程开始和结束符号:

```
BEGIN .... END
```

变量赋值:

```
SET @p_in=1
```

变量定义:

```
DECLARE l_int int unsigned default 4000000;
```

创建mysql存储过程、存储函数:

```
create procedure 存储过程名(参数)
```

存储过程体:

```
create function 存储函数名(参数)
```

## 实例

创建数据库，备份数据表用于示例操作：

```
create database db1;
use db1;
create table PLAYERS as select * from TENNIS.PLAYERS;
create table MATCHES as select * from TENNIS.MATCHES;
```

下面是存储过程的例子，删除给定球员参加的所有比赛：

```
mysql> delimiter $$      #将语句的结束符号从分号;临时改为两个$$ (可以是自定义)
mysql> CREATE PROCEDURE delete_matches(IN p_playerno INTEGER)
-> BEGIN
->     DELETE FROM MATCHES
->     WHERE playerno = p_playerno;
-> END$$
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter;      #将语句的结束符号恢复为分号
```

**解析：**默认情况下，存储过程和默认数据库相关联，如果想指定存储过程创建在某个特定的数据库下，那么在过程名前面加数据库名做前缀。在定义过程时，使用 **DELIMITER \$\$** 命令将语句的结束符号从分号 ; 临时改为两个 \$\$，使得过程体中使用的分号被直接传递到服务器，而不会被客户端（如mysql）解释。

调用存储过程：

```
call sp_name[(传参)];
```

```
mysql> select * from MATCHES;
+-----+-----+-----+-----+-----+
| MATCHNO | TEAMNO | PLAYERNO | WON | LOST |
+-----+-----+-----+-----+-----+
|      1 |      1 |      6 |    3 |    1 |
|      7 |      1 |     57 |    3 |    0 |
|      8 |      1 |      8 |    0 |    3 |
|      9 |      2 |     27 |    3 |    2 |
|     11 |      2 |    112 |    2 |    3 |
```

```

+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> call delete_matches(57);
Query OK, 1 row affected (0.03 sec)

mysql> select * from MATCHES;
+-----+-----+-----+-----+
| MATCHNO | TEAMNO | PLAYERNO | WON | LOST |
+-----+-----+-----+-----+
|      1 |      1 |      6 |   3 |   1 |
|      8 |      1 |      8 |   0 |   3 |
|      9 |      2 |     27 |   3 |   2 |
|     11 |      2 |    112 |   2 |   3 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

**解析：**在存储过程中设置了需要传参的变量p\_playerno，调用存储过程的时候，通过传参将57赋值给p\_playerno，然后进行存储过程里的SQL操作。

### 存储过程体

- 存储过程体包含了在过程调用时必须执行的语句，例如：dml、ddl语句，if-then-else和while-do语句、声明变量的declare语句等
- 过程体格式：以begin开始，以end结束(可嵌套)

```

BEGIN
  BEGIN
    BEGIN
      statements;
    END
  END
END

```

**注意：**每个嵌套块及其中的每条语句，必须以分号结束，表示过程体结束的begin-end块(又叫做复合语句compound statement)，则不需要分号。

为语句块贴标签:

```

[begin_label:] BEGIN
  [statement_list]
END [end_label]

```

例如：

标签有两个作用：

- 1、增强代码的可读性
- 2、在某些语句(例如:leave和iterate语句)，需要用到标签

## ## 二、存储过程的参数

MySQL存储过程的参数用在存储过程的定义，共有三种参数类型，**IN**，**OUT**，**INOUT**，形式如：

```
```mysql
CREATEPROCEDURE 存储过程名([[IN |OUT |INOUT ] 参数名 数据类型...])
```

- IN 输入参数：表示调用者向过程传入值（传入值可以是字面量或变量）
- OUT 输出参数：表示过程向调用者传出值(可以返回多个值)（传出值只能是变量）
- INOUT 输入输出参数：既表示调用者向过程传入值，又表示过程向调用者传出值（值只能是变量）

## 1、in 输入参数

```
mysql> delimiter $$
mysql> create procedure in_param(in p_in int)
-> begin
->     select p_in;
->     set p_in=2;
->     select P_in;
-> end$$
mysql> delimiter ;
```

```
mysql> set @p_in=1;
```

```
mysql> call in_param(@p_in);
```

```
+-----+
| p_in |
+-----+
|    1 |
+-----+
```

```
+-----+
| P_in |
+-----+
|    2 |
```

```

+-----+

mysql> select @p_in;
+-----+
| @p_in |
+-----+
|      1 |
+-----+

```

以上可以看出，p\_in 在存储过程中被修改，但并不影响 @p\_id 的值，因为前者为局部变量、后者为全局变量。

以上可以看出，p\_in 在存储过程中被修改，但并不影响 @p\_id 的值，因为前者为局部变量、后者为全局变量。

## 2、out输出参数

```

mysql> delimiter //
mysql> create procedure out_param(out p_out int)
-> begin
->     select p_out;
->     set p_out=2;
->     select p_out;
-> end
-> //
mysql> delimiter ;

mysql> set @p_out=1;

mysql> call out_param(@p_out);
+-----+
| p_out |
+-----+
|  NULL |
+-----+

#因为out是向调用者输出参数，不接收输入的参数，所以存储过程里的p_out为null
+-----+
| p_out |
+-----+
|      2 |
+-----+

mysql> select @p_out;

```

```

+-----+
| @p_out |
+-----+
|      2 |
+-----+

```

#调用了out\_param存储过程，输出参数，改变了p\_out变量的值

### 3、inout输入参数

```

mysql> delimiter $$
mysql> create procedure inout_param(inout p_inout int)
-> begin
->     select p_inout;
->     set p_inout=2;
->     select p_inout;
-> end
-> $$
mysql> delimiter ;

```

```
mysql> set @p_inout=1;
```

```
mysql> call inout_param(@p_inout);
```

```

+-----+
| p_inout |
+-----+
|      1 |
+-----+

```

```

+-----+
| p_inout |
+-----+
|      2 |
+-----+

```

```
mysql> select @p_inout;
```

```

+-----+
| @p_inout |
+-----+
|      2 |
+-----+

```

#调用了inout\_param存储过程，接受了输入的参数，也输出参数，改变了变量

注意：

1、如果过程没有参数，也必须在过程名后面写上小括号例：

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

2、确保参数的名字不等于列的名字，否则在过程体中，参数名被当做列名来处理

建议：

- 输入值使用in参数。
- 返回值使用out参数。
- inout参数就尽可能的少用。

## 三、变量

### 1. 变量定义

局部变量声明一定要放在存储过程体的开始：

```
DECLARE variable_name [,variable_name...] datatype [DEFAULT value];
```

其中，datatype 为 MySQL 的数据类型，如: int, float, date, varchar(length)

例如:

```
DECLARE l_int int unsigned default 4000000;
DECLARE l_numeric number(8,2) DEFAULT 9.95;
DECLARE l_date date DEFAULT '1999-12-31';
DECLARE l_datetime datetime DEFAULT '1999-12-31 23:59:59';
DECLARE l_varchar varchar(255) DEFAULT 'This will not be padded';
```

### 2. 变量赋值

```
SET 变量名 = 表达式值 [,variable_name = expression ...]
```

### 3. 用户变量



在MySQL客户端使用用户变量:

```
mysql > SELECT 'Hello World' into @x;
mysql > SELECT @x;
+-----+
| @x    |
+-----+
| Hello World |
+-----+

mysql > SET @y='Goodbye Cruel World';
mysql > SELECT @y;
+-----+
| @y    |
+-----+
| Goodbye Cruel World |
+-----+

mysql > SET @z=1+2+3;
mysql > SELECT @z;
+-----+
| @z    |
+-----+
| 6     |
+-----+
```

在存储过程中使用用户变量

```
mysql > CREATE PROCEDURE GreetWorld( ) SELECT CONCAT(@greeting, ' World');

mysql > SET @greeting='Hello';
mysql > CALL GreetWorld( );
+-----+
| CONCAT(@greeting, ' World') |
+-----+
| Hello World                  |
+-----+
```

在存储过程间传递全局范围的用户变量

```
mysql> CREATE PROCEDURE p1() SET @last_procedure='p1';
mysql> CREATE PROCEDURE p2() SELECT CONCAT('Last procedure was
',@last_procedure);
mysql> CALL p1( );
mysql> CALL p2( );
+-----+
| CONCAT('Last procedure was ',@last_proc |
+-----+
| Last procedure was p1 |
+-----+
```

注意:

- 1、用户变量名一般以@开头
- 2、滥用用户变量会导致程序难以理解及管理

## 四、注释

MySQL 存储过程可使用两种风格的注释

两个横杆--：该风格一般用于单行注释。

**c 风格**：一般用于多行注释。

例如：

```
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc1 --name存储过程名
-> (IN parameter1 INTEGER)
-> BEGIN
-> DECLARE variable1 CHAR(10);
-> IF parameter1 = 17 THEN
-> SET variable1 = 'birds';
-> ELSE
-> SET variable1 = 'beasts';
-> END IF;
-> INSERT INTO table1 VALUES (variable1);
-> END
-> //
mysql > DELIMITER ;
```

## MySQL存储过程的调用

用call和你过程名以及一个括号，括号里面根据需要，加入参数，参数包括输入参数、输出参数、输入输出参数。具体的调用方法可以参看上面的例子。

## MySQL存储过程的查询

我们像知道一个数据库下面有那些表，我们一般采用 `showtables;` 进行查看。那么我们要查看某个数据库下面的存储过程，是否也可以采用呢？答案是，我们可以查看某个数据库下面的存储过程，但是是另一种方式。

我们可以用以下语句进行查询：

```
selectname from mysql.proc where db='数据库名';
```

或者

```
selectroutine_name from information_schema.routines where  
routine_schema='数据库名';
```

或者

```
showprocedure status where db='数据库名';
```

如果我们想知道，某个存储过程的详细，那我们又该怎么做呢？是不是也可以像操作表一样用describe 表名进行查看呢？

答案是：我们可以查看存储过程的详细，但是需要用另一种方法：

```
SHOWCREATE PROCEDURE 数据库.存储过程名;
```

就可以查看当前存储过程的详细。

## MySQL存储过程的修改

```
ALTER PROCEDURE
```

更改用 CREATE PROCEDURE 建立的预先指定的存储过程，其不会影响相关存储过程或存储功能。

## MySQL存储过程的删除

删除一个存储过程比较简单，和删除表一样：

```
DROP PROCEDURE
```

从 MySQL 的表格中删除一个或多个存储过程。

## MySQL存储过程的控制语句

### (1). 变量作用域

内部的变量在其作用域范围内享有更高的优先权，当执行到 end。变量时，内部变量消失，此时已经在其作用域外，变量不再可见了，应为在存储过程外再也不能找到这个声明的变量，但是你可以通过 out 参数或者将其值指派给会话变量来保存其值。

```
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc3()
    -> begin
    -> declare x1 varchar(5) default 'outer';
    -> begin
    -> declare x1 varchar(5) default 'inner';
    -> select x1;
    -> end;
    -> select x1;
    -> end;
    -> //
mysql > DELIMITER ;
```

### (2). 条件语句

#### 1. if-then-else 语句

```
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc2(IN parameter int)
    -> begin
    -> declare var int;
    -> set var=parameter+1;
    -> if var=0 then
    -> insert into t values(17);
    -> end if;
    -> if parameter=0 then
    -> update t set s1=s1+1;
    -> else
    -> update t set s1=s1+2;
```

```

-> end if;
-> end;
-> //
mysql > DELIMITER ;

```

## 2. case语句:

```

mysql > DELIMITER //
mysql > CREATE PROCEDURE proc3 (in parameter int)
-> begin
-> declare var int;
-> set var=parameter+1;
-> case var
-> when 0 then
-> insert into t values(17);
-> when 1 then
-> insert into t values(18);
-> else
-> insert into t values(19);
-> end case;
-> end;
-> //
mysql > DELIMITER ;
case
    when var=0 then
        insert into t values(30);
    when var>0 then
    when var<0 then
    else
end case

```

## (3). 循环语句

### 1. while ... end while

```
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc4()
    -> begin
    -> declare var int;
    -> set var=0;
    -> while var<6 do
    -> insert into t values(var);
    -> set var=var+1;
    -> end while;
    -> end;
    -> //
mysql > DELIMITER ;
```

```
while 条件 do
    --循环体
endwhile
```

## 2. repeat... end repea

它在执行操作后检查结果，而 while 则是执行前进行检查。

```
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc5 ()
    -> begin
    -> declare v int;
    -> set v=0;
    -> repeat
    -> insert into t values(v);
    -> set v=v+1;
    -> until v>=5
    -> end repeat;
    -> end;
    -> //
mysql > DELIMITER ;
```

```
repeat
    --循环体
until 循环条件
end repeat;
```

## 3. loop .....endloop

loop 循环不需要初始条件，这点和 while 循环相似，同时和 repeat 循环一样不需要结束条件，leave 语句的意义是离开循环。

```
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc6 ()
    -> begin
    -> declare v int;
    -> set v=0;
    -> LOOP_LABEL:loop
    -> insert into t values(v);
    -> set v=v+1;
    -> if v >=5 then
    -> leave LOOP_LABEL;
    -> end if;
    -> end loop;
    -> end;
    -> //
mysql > DELIMITER ;
```

#### 4. LABELS 标号：

标号可以用在 begin repeat while 或者 loop 语句前，语句标号只能在合法的语句前面使用。可以跳出循环，使运行指令达到复合语句的最后一步。

#### (4). ITERATE 迭代

ITERATE 通过引用复合语句的标号,来从新开始复合语句:

```
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc10 ()
    -> begin
    -> declare v int;
    -> set v=0;
    -> LOOP_LABEL:loop
    -> if v=3 then
    -> set v=v+1;
    -> ITERATE LOOP_LABEL;
    -> end if;
    -> insert into t values(v);
    -> set v=v+1;
    -> if v>=5 then
    -> leave LOOP_LABEL;
    -> end if;
    -> end loop;
```

```

-> end;
-> //
mysql > DELIMITER ;

```

## 例子代码

```

CREATE TABLE `player` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `playerName` varchar(20) DEFAULT NULL,
  `playerPosition` varchar(20) DEFAULT NULL,
  `playerNumber` int(11) DEFAULT NULL,
  `teamId` int(11) DEFAULT NULL,
  `championNumber` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4

SELECT * FROM tbl_user

CREATE VIEW v_user AS SELECT * FROM tbl_user WHERE id =4

CREATE VIEW v_user AS SELECT count(*) FROM tbl_user

DROP VIEW v_user

CREATE VIEW v_user (id1,`name`) AS SELECT id,`NAME` FROM tbl_user

SELECT * FROM v_user WHERE id1 =4
UPDATE v_user SET `name`='majian' WHERE id1 =4

DROP VIEW v_user

SELECT `name` FROM tbl_user WHERE id=4

-- 创建自定义函数

```



```

CREATE FUNCTION getuser()
RETURNS VARCHAR(20) -- 定义返回值的类型
RETURN -- 指定返回值 为后面的 select 语句结果
(SELECT `name` FROM tbl_user WHERE id=4);

SELECT getuser(); -- 调用函数

-- 创建带参数的函数 参数格式 参数名字 类型

DELIMITER ;; -- 指定函数的结束标记为;; 默认是; ,但是我们的函数中可能会有多次; 为
了防止误结束,所以先修改结束标记
CREATE FUNCTION    addPlayer(playerName VARCHAR(20) ,playerPosition
VARCHAR(20) ,playerNumber INT UNSIGNED ,teamId INT UNSIGNED
,championNumber INT)
    RETURNS INT    UNSIGNED
BEGIN
    INSERT INTO player
VALUES(NULL,playerName,playerPosition,playerNumber,teamId,championNumber)
;
    RETURN LAST_INSERT_ID();
END ;;
DELIMITER ;

-- 调用带参数的函数
SELECT addPlayer('世纪佳缘','程序员的最终归宿',334,2423,235234)
-- 钱多,话少,死的早
-- 删除函数
DROP FUNCTION IF EXISTS getuser

-- 创建存储过程,无参数, 其实就是一个函数
DELIMITER ;;
CREATE PROCEDURE getuser()
BEGIN -- 下面是要执行的内容
SELECT `name` FROM tbl_user WHERE id=4;
END ;;
-- 调用过程
CALL getuser();

-- 定义带参数的过程,参数类型是输入参数
DELIMITER ;;

```

```

CREATE PROCEDURE    addPlayer(in playerName VARCHAR(20) ,in playerPosition
VARCHAR(20) ,in playerNumber INT UNSIGNED ,in teamId INT UNSIGNED
,in championNumber  INT)
BEGIN
    INSERT INTO player
VALUES(NULL,playerName,playerPosition,playerNumber,teamId,championNumber)
;
    SELECT LAST_INSERT_ID();
END  ;;
DELIMITER ;

```

```

CALL addPlayer('content', '几分伤心几分痴', 321, 312, 3123)

```

-- 创建带输出参数的过程, 注意输出参数不能直接在函数内调用

```

CREATE PROCEDURE getuser1(out out_a int)
BEGIN
-- SELECT `name` FROM tbl_user WHERE id=4;
INSERT INTO player
VALUES(NULL, 'dasdas', '打扫打扫大厦', 213, 31231, 3131);
SELECT LAST_INSERT_ID() INTO out_a; -- 赋值给输出参数
SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动 select 一下参数
END  ;;

```

-- 声明变量

```

SET @adasdas=1;

```

```

-- SELECT LAST_INSERT_ID() INTO @adasdas;

```

```

-- SELECT @adasdas;

```

-- 盗用参数, 将声明的变量作为参数传递

```

CALL getuser1(@adasdas);

```

-- 创建带输入/输出参数的过程, 这个参数可以作为输入参数也可以作为输出参数

```

CREATE PROCEDURE getuser2(inout out_a int)
BEGIN
-- SELECT `name` FROM tbl_user WHERE id=4;
INSERT INTO player

```

```

VALUES(NULL, 'dasdas', '打扫打扫大厦', out_a, 31231, 3131); -- 取值
SELECT LAST_INSERT_ID() INTO out_a; -- 赋值
SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动 select 一下参数
END ;;
-- 调用过程
SET @adasdas=999;
CALL getuser2(@adasdas);

-- 创建带 if 条件的过程
CREATE PROCEDURE getuser3(inout out_a int)
BEGIN
-- SELECT `name` FROM tbl_user WHERE id=4;
IF out_a=0 THEN -- if 条件
    INSERT INTO player
    VALUES(NULL, 'dasdas', '打扫打扫大厦', out_a, 000000, 0000000000);
    SELECT LAST_INSERT_ID() INTO out_a;
    SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动 select 一下参数
END IF;
IF out_a =1 THEN
    INSERT INTO player
    VALUES(NULL, 'ewaqe', '每次来上海', out_a, 11111111, 111111);
    SELECT LAST_INSERT_ID() INTO out_a;
    SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动 select 一下参数
ELSE
    INSERT INTO player
    VALUES(NULL, 'dasdas', '就要来台风', out_a, 666666, 666666);
    SELECT LAST_INSERT_ID() INTO out_a;
    SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动 select 一下参数
END IF;
END ;;
-- 调用过程
SET @bianliang1=0;
CALL getuser3(@bianliang1)

```

-- 创建带 switch case 的过程

```
CREATE PROCEDURE getuser4(inout out_a int)
BEGIN
    CASE out_a
    WHEN 0 THEN
        INSERT INTO player
        VALUES(NULL, 'xiangyao', '汤臣一品', out_a, 312310000, 310031);
        SELECT LAST_INSERT_ID() INTO out_a;
        SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动
select 一下参数
    WHEN 1 THEN
        INSERT INTO player
        VALUES(NULL, 'zhuanche', '地铁2号线', out_a, 111, 312111);
        SELECT LAST_INSERT_ID() INTO out_a;
        SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动
select 一下参数
    WHEN 2 THEN
        INSERT INTO player
        VALUES(NULL, 'duanzi', '踩刹车, 踩刹车, 刹车是哪个', out_a, 222, 222);
        SELECT LAST_INSERT_ID() INTO out_a;
        SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动
select 一下参数
    ELSE -- 默认操作
        INSERT INTO player
        VALUES(NULL, 'majian', '马健有一个特殊爱好', out_a, 333, 333);
        SELECT LAST_INSERT_ID() INTO out_a;
        SELECT out_a; -- 因为默认是将 select 的结果当做返回值, 所以我们必须手动
select 一下参数
    END CASE;
END ;;

SET @bianliang1=3;
CALL getuser4(@bianliang1);
```

-- 常见循环操作的过程

```
CREATE PROCEDURE getuser5(inout out_a int)
BEGIN
    DECLARE var int DEFAULT 0;
    WHILE var < out_a DO
        INSERT INTO player
        VALUES(NULL, 'majian', '马健有一个特殊爱好', var, 333, 333);
        SET var = var + 1;
    END WHILE;
```

```
END ;;  
-- 执行操作  
SET @cishu =10;  
CALL getuser5(@cishu)
```