# Mysql存储过程

1. 什么是存储过程

存储例程是存储在数据库服务器中的一组sql语句，通过在查询中调用一个指定的名称来执行这些sql语句命令

1. 为什么要使用mysql存储过程

我们都知道应用程序分为两种，一种是基于web，一种是基于桌面，他们都和数据库进行交互来完成数据的存取工作。假设现在有一种应用程序包含了这两 种，现在要修改其中的一个查询sql语句，那么我们可能要同时修改他们中对应的查询sql语句，当我们的应用程序很庞大很复杂的时候问题就出现这，不易维 护！另外把sql查询语句放在我们的web程序或桌面中很容易遭到sql注入的破坏。而存储例程正好可以帮我们解决这些问题。

1. 存储过程(stored procedure)、存储例程(store routine)、存储函数区别

Mysql存储例程实际包含了存储过程和存储函数，它们被统称为存储例程。其中存储过程主要完成在获取记录或插入记录或更新记录或删除记录，即完成select insert delete update等的工作。而存储函数只完成查询的工作，可接受输入参数并返回一个结果。

1. 变量的声明

设置全局变量

set @a='一个新变量';

在函数和储存过程中使用的变量declear

declear a int unsigned default 1;

这种变量需要设置变量类型 而且只存在在 begin..end 这段之内

select .. into..  直接将表内内容赋值到指定变量当中

select name,bid into @a,@b from bank limit 1;

要注意一点就是变量名不能和字段名一致

1. 局部变量

### **变量定义**

局部变量声明一定要放在存储过程体的开始   
DECLAREvariable\_name [,variable\_name…] datatype [DEFAULT value];   
其中，datatype为MySQL的数据类型，如:int, float, date,varchar(length)

Eg:

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';

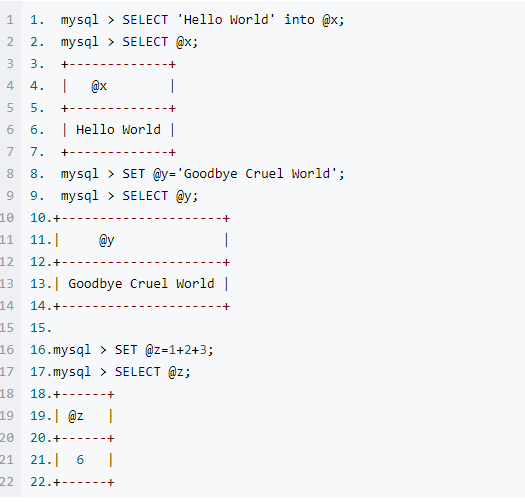
### **变量赋值**

SET 变量名 = 表达式值 [,variable\_name = expression …]

## **用户变量**

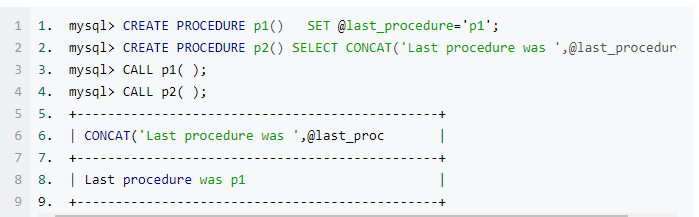
### **在MySQL客户端使用用户变量**

### **在存储过程中使用用户变量**



### **在存储过程中使用用户变量**

### **在存储过程间传递全局范围的用户变量**



注：

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

显示所有存储过程

显示所有存储过程

show procedure status;

删除指定存储过程

drop procedure 过程名;

**举例**

create procedure yanshi(in arg tinyint)

begin

declare age tinyint default 0;

set age=arg;

if age<20 then

select '小于20的数';

elseif age>20 then

select '大于20的数';

end if;

end

$

//调用过程

set @num=12$

call yanshi(@num)$

call yanshi(21)$

1. 存储过程的传参

分为三种 in,out,inout三种

in 可以输出从外部传入的变量 不会改变传进变量本来的值

create procedure a(in id int)  
begin  
 select id;  
 set id = 100;  
end  
$  
set @id=1$  
call a(@id)$  //输出1 即从外部传进来的@id 的值

select $id$  //输出1 说明存储过程中没有改变传进的值

out 不能输出从外部传进的值。 会改变传进变量本来的值

create procedure b(out id int)  
begin  
 select id;  
 set id = 100;  
end  
$  
set @id=1$  
call b(@id)$   //输入null

select @id$ //输出100

inout 就是又能输出传入变量又能改变传入变量咯

下面是检验你电脑硬件性能的时候了

还记得当年的bank表吗? 就是他保留住 然后执行以下命令:

create procedure addbank()  
begin  
 declare i int default 0;  
 set i = 5000000;  
 while i > 0 do  
 insert into bank (name) values (i);  
 set i = i - 1;  
 end while;  
end  
$

call addbank()$

## **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，但是是令一钟方式。   
我们可以用   
select name from mysql.proc where db=’数据库名’;   
或者   
select routine\_name from information\_schema.routines where routine\_schema=’数据库名’;   
或者   
show procedure status where db=’数据库名’;   
进行查询。   
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。

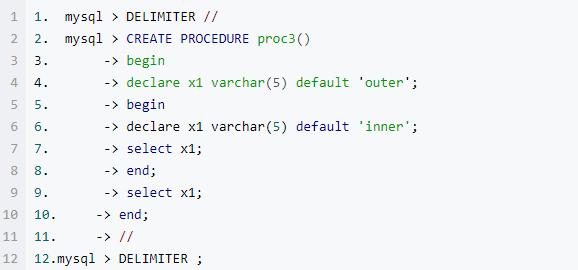
## **MySQL存储过程的删除**

删除一个存储过程比较简单，和删除表一样：   
DROP PROCEDURE procedure\_name(存储过程名称)   
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**

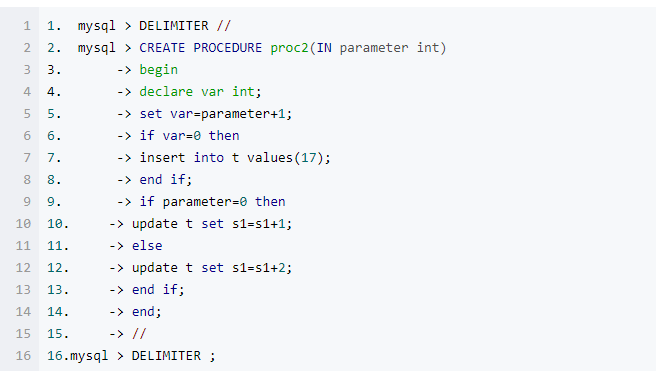
### **变量作用域**

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申明的变量，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。

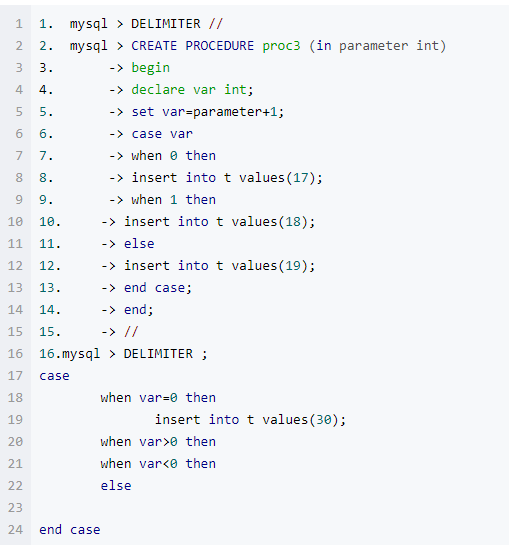


### **条件语句**

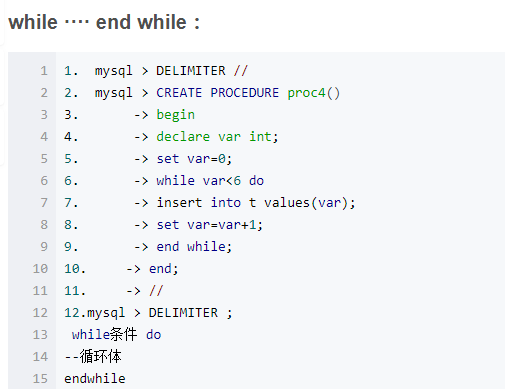
#### **if-then -else语句**



#### **case语句：**



### **循环语句**



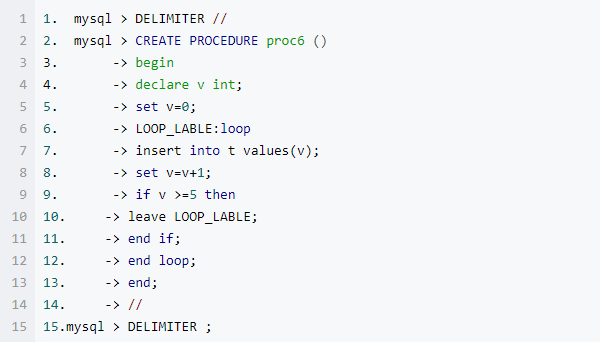
#### **repeat···· end repeat：**

它在执行操作后检查结果，而while则是执行前进行检查。



#### **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



## **LABLES 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。

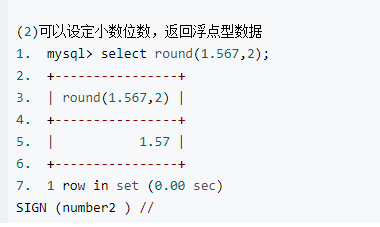
## **MySQL存储过程的基本函数**

### **字符串类**

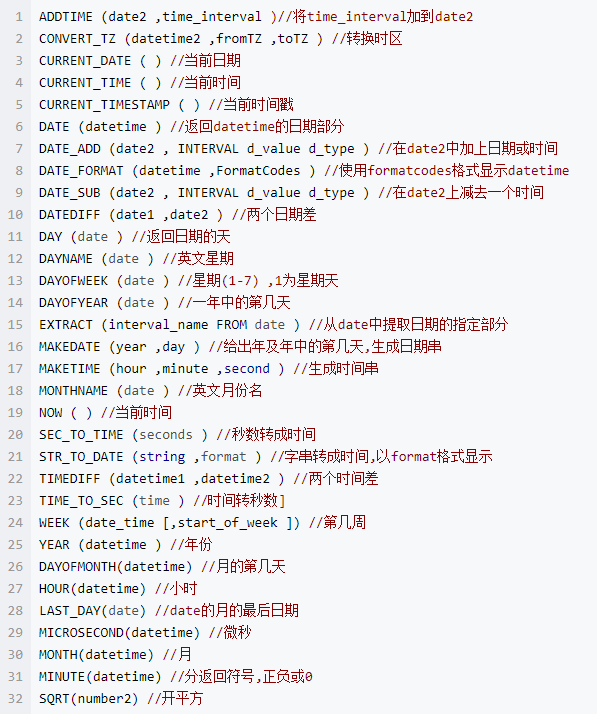


### **数学类**





### **日期时间类**



## **MySql分页存储过程**

DROP PROCEDURE IF EXISTS pr\_pager; //判断是否存在该存储过程函数存在则删除

//创建存储过程函数

CREATE PROCEDURE pr\_pager(

IN p\_table\_name VARCHAR(1024), //表名

IN p\_fields VARCHAR(1024), //字段名

IN p\_page\_size INT, //每页显示多少条

IN p\_page\_now INT, //现在的页数

IN p\_order\_string VARCHAR(128), //表排序规则

IN p\_where\_string VARCHAR(1024), //表查询条件

OUT p\_out\_rows INT //返回的数据行数

)

NOT DETERMINISTIC //表示存储过程对同样的输入参数产生相同的结果;NOT DETERMINISTIC，则表示会产生不确定的结果（默认）

SQL SECURITY DEFINER //

COMMENT '分页存储过程' //对此存储过程的描述

BEGIN

DECLARE m\_begin\_row INT DEFAULT 0;

DECLARE m\_limit\_string CHAR(64);

SET m\_begin\_row = (p\_page\_now - 1) \* p\_page\_size;

SET m\_limit\_string = CONCAT(' LIMIT ', m\_begin\_row, ', ', p\_page\_size); SET @COUNT\_STRING = CONCAT('SELECT COUNT(\*) INTO @ROWS\_TOTAL FROM ', p\_table\_name, ' ', p\_where\_string);

SET @MAIN\_STRING = CONCAT('SELECT ', p\_fields, ' FROM ', p\_table\_name, ' ', p\_where\_string, ' ', p\_order\_string,m\_limit\_string);

PREPARE count\_stmt FROM @COUNT\_STRING; //解析sql语句

EXECUTE count\_stmt;//设置参数

DEALLOCATE PREPARE count\_stmt;

SET p\_out\_rows = @ROWS\_TOTAL;

PREPARE main\_stmt FROM @MAIN\_STRING;

EXECUTE main\_stmt;

DEALLOCATE PREPARE main\_stmt;

END;

mysql> call pr\_pager("t","var",3,3,"","",@result);

mysql> call pr\_pager("t","var",3,2,"","",@result);

注：Characteristic:存储过程的某些特征设定，分别介绍  
1 COMMENT'string':用于对存储过程的描述，其中string为描述内容,comment为关键字。  
2 LANGUAGE SQL:指明编写这个存储过程的语言为SQL语言。这个选项可以不指定。  
3 DETERMINISTIC:表示存储过程对同样的输入参数产生相同的结果;NOT DETERMINISTIC，则表示会产生不确定的结果（默认）。  
4 contains sql | no sql | reads sql data | modifies sql data Contains sql表示存储过程包含读或写数据的语句（默认）  
No sql表示不包含sql语句  
Reads sql data表示存储过程只包含读数据的语句  
Modifies sql data 表示存储过程只包含写数据的语句  
5 sql security:这个特征用来指定存储过程使用创建该存储过程的用户(definer)的许可来执行，还是使用调用者(invoker)的许可来执行。默认是definer  
Routine\_body:存储过程的主体部分，包含了在过程调用的时候必须执行的sql语句。以begin开始，以end结束。如果存储过程体中只有一条sql语句,可以省略begin-end标志。

PREPARE的作用：

使用PAREPARE STATEMENT，可以减少每次执行SQL的语法分析，  
比如用于执行带有WHERE条件的SELECT和DELETE，或者UPDATE，或者INSERT，只需要每次修改变量值即可。  
同样可以防止SQL注入，参数值可以包含转义符和定界符。  
适用在应用程序中，或者SQL脚本中均可。

更多用法：  
同样PREPARE ...FROM可以直接接用户变量

EXECUTE和DEALLOCATE的作用：

每一次执行完EXECUTE时，养成好习惯，须执行DEALLOCATE PREPARE … 语句，这样可以释放执行中使用的所有数据库资源（如游标）。  
不仅如此，如果一个session的预处理语句过多，可能会达到max\_prepared\_stmt\_count的上限值。  
  
预处理语句只能在创建者的会话中可以使用，其他会话是无法使用的。  
而且在任意方式（正常或非正常）退出会话时，之前定义好的预处理语句将不复存在。

如果在存储过程中使用，如果不在过程中DEALLOCATE掉，在存储过程结束之后，该预处理语句仍然会有效。

**存储过程和函数区别**  
1）一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。存储过程，功能强大，可以执行包括修改表等一系列数据库操作；用户定义函数不能用于执行一组修改全局数据库状态的操作。

2）对于存储过程来说可以返回参数，如记录集，而函数只能返回值或者表对象。函数只能返回一个变量；而存储过程可以返回多个。存储过程的参数可以有IN,OUT,INOUT三种类型，而函数只能有IN类~~存储过程声明时不需要返回类型，而函数声明时需要描述返回类型，且函数体中必须包含一个有效的RETURN语句。

3）存储过程，可以使用非确定函数，不允许在用户定义函数主体中内置非确定函数。

4）存储过程一般是作为一个独立的部分来执行（ EXECUTE 语句执行），而函数可以作为查询语句的一个部分来调用（SELECT调用），由于函数可以返回一个表对象，因此它可以在查询语句中位于FROM关键字的后面。 SQL语句中不可用存储过程，而可以使用函数。

**光标（游标）**  
**3.1 定义**     查询语句可能查询出多条记录，在存储过程和函数中使用光标标来逐条读取查询结果集中的记录。光标的使用包括声明光标、打开光标、使用光标和关闭光标。光标必须声明光标、打开光标、使用光标和关闭光标。光标必须声明在处理程序之前，并且声明在变量和条件之后。

**声明光标**

**Declare** cursor\_name **cursor** forselect\_statement;

Cursor\_name:光标名称

Select\_statement:**select**语句的内容

**Declare** cur\_employee **cursor** forselect **name**,age **from** employee;

**打开光标**

Mysql中使用fetch关键字来使用光标，语法形式

**Fetch** cur\_name intovar\_name[,var\_name…];

Cur\_name表示光标的名称

Var\_name表示将光标中的**select**语句查询出来的信息存入该参数。Var\_name必须在声明光标前就定义好。

**Fetch** cur\_employee intoemp\_name,emp\_age;

**关闭光标**

**Close** cursor\_name;

**Close** cur\_employee;

每个光标不再需要时都应该被关闭，使用close语句将会释放光标所使用的全部资源。在一个光标被关闭后，如果没有重新被打开，则不能被使用。对于声明过的光标，则不需要再次声明，可直接使用open语句打开。

**使用范例**（将表test\_cur1数据复制到test\_cur2）

**CREATE** **TABLE** `test\_cur1` (

`id` **int**(11) NOT NULL auto\_increment,

`type` **char**(11) **default** NULL,

`order1` **char**(11) **default** NULL,

**PRIMARY** **KEY** (`id`)

)

**INSERT** **INTO** `test\_cur1` **VALUES** (1, '145', 'd1');

**INSERT** **INTO** `test\_cur1` **VALUES** (2, '134', '1d');

**INSERT** **INTO** `test\_cur1` **VALUES** (3, '123', '1ad');

**INSERT** **INTO** `test\_cur1` **VALUES** (4, '121', '1as');

**CREATE** **TABLE** `test\_cur2` (

`id` **int**(11) NOT NULL auto\_increment,

`type` **char**(11) **default** NULL,

`order1` **char**(11) **default** NULL,

**PRIMARY** **KEY** (`id`)

然后写光标了：

**create** **procedure** get\_cur ()

**BEGIN**

**DECLARE** done **INT** **DEFAULT** 0;

**DECLARE** ID **int**(11);

**DECLARE** type **char**(11);

**DECLARE** order1 **char**(11);

**DECLARE** mycur **CURSOR** **FOR** **SELECT** \* **FROM** test\_cur1;//定义光标

**DECLARE** **CONTINUE** HANDLER **FOR** SQLSTATE '02000' **SET** done = 1;

//打开光标

**OPEN** mycur;

//开始循环

REPEAT

**FETCH** mycur **INTO** ID,type,order1;//取出光标的内容到临时变量

IF NOT done **THEN**

**INSERT** **INTO** test\_cur2 **VALUES** (ID,type,order1);//插入到另一张表

**END** IF;

UNTIL done **END** REPEAT;//当done=1时结束循环

//关闭光标

**CLOSE** mycur;

**END**

//运行

call get\_cur()