实验6 存储过程实验

201908010224黄雅妮

6.1 存储过程实验

存储过程就是一组SQL语句集,功能强大,可以实现一些比较复杂的逻辑功能,类似于JAVA语言中的方法,存储过就是数据库SQL与层层面的代码封装与重用。创建的存储过程保存在数据库的数据字典中

特性
1.有输入输出参数,可以声明变量,有if/else/case/while等控制语句,通过编写存储过程,可以实现复杂的逻辑功能

2.函数的普通特性:模块化,封装,代码复用

3.速度快,只有首次执行需要经过编译和优化步骤,后续被调用可以直接执行,省去以上步骤

变量类型

- 用户变量：以”@”开始，形式为”@变量名。”

  用户变量跟mysql客户端是绑定的，设置的变量，只对当前用户使用的客户端生效。
- 全局变量：定义时，以如下两种形式出现，set GLOBAL 变量名 或者 set @@global.变量名。

  对所有客户端生效。只有具有super权限才可以设置全局变量。
- 会话变量：只对连接的客户端有效。一旦客户端失去连接，变量失效。show session variables;
- 局部变量：作用范围在begin到end语句块之间。在该语句块里设置的变量declare语句专门用于定义局部变量。DECLARE l_numeric number(8,2) DEFAULT 9.95;
- 全局变量和会话变量的区别：全局变量在MySQL启动的时候由服务器自动将它们初始化为默认值，这些默认值可以通过更改my.ini这个文件来更改。会话变量在每次建立一个新的连接的时候，由MYSQL来初始化。MYSQL会将当前所有全局变量的值复制一份。来做为会话变量。全局变量与会话变量的区别就在于，对全局变量的修改会影响到整个服务器，但是对会话变量的修改，只会影响到当前的会话（也就是当前的数据库连接）。非root用户在修改全局变量时会报没有权限，在修改会话变量时也需要注意，有些变量是不能修改的，只能由root用户进行修改，例如：event_scheduler。

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：
`CREATE PROCEDURE([[IN |OUT |INOUT ] 参数名 数据类型...])`

- IN 输入参数:表示该参数的值必须在调用存储过程时指定，在存储过程中修改该参数的值不能被返回，为默认值
- OUT 输出参数:该值可在存储过程内部被改变，并可返回
- INOUT 输入输出参数:调用时指定，并且可被改变和返回

6.1.1 无参数的存储过程

定义一个存储过程，更新订单的（含税折扣价）总价。即根据订单明细表，计算每个订单的总价，更新ORDER表

```
-- 1.无参数的存储过程
DROP PROCEDURE  IF EXISTS Proc_CaltotalPrice;
create procedure Proc_CaltotalPrice()
```

```
begin
     update
     orders
set
    O_TOTALPRICE =
     (
    select
        sum(L_EXTENDEDPRICE *(1-L_DISCOUNT) *(1 + L_TAX))
    from
        lineitem
    where
        O_ORDERKEY = L_ORDERKEY
     );
end;

-- 执行存储过程
call Proc_CaltotalPrice();
```

实验结果

可以看到，totalprice的值被全部更新

O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE	O_ORDERPRIORITY	O_CLERK	O_SH
1	37	O	131251.88	1996-01-02	5-LOW	Clerk#000000951	
2	79	O	40183.29	1996-12-01	1-URGENT	Clerk#000000880	
3	124	F	160882.80	1993-10-14	5-LOW	Clerk#000000955	
4	137	O	31084.81	1995-10-11	5-LOW	Clerk#000000124	
5	46	F	86615.26	1994-07-30	5-LOW	Clerk#000000925	
6	56	F	36468.56	1992-02-21	4-NOT SPECIFIED	Clerk#000000058	
7	40	O	171488.79	1996-01-10	2-HIGH	Clerk#000000470	
32	131	O	116923.07	1995-07-16	2-HIGH	Clerk#000000616	
33	67	F	99798.80	1993-10-27	3-MEDIUM	Clerk#000000409	
34	62	O	41670.05	1998-07-21	3-MEDIUM	Clerk#000000223	
35	128	O	148789.56	1995-10-23	4-NOT SPECIFIED	Clerk#000000259	
36	116	O	38988.99	1995-11-03	1-URGENT	Clerk#000000358	
37	88	F	113701.93	1992-06-03	3-MEDIUM	Clerk#000000456	
38	125	O	46366.57	1996-08-21	4-NOT SPECIFIED	Clerk#000000604	
39	82	O	219707.90	1996-09-20	3-MEDIUM	Clerk#000000659	
64	34	F	20065.74	1994-07-16	3-MEDIUM	Clerk#000000661	
65	17	P	65883.94	1995-03-18	1-URGENT	Clerk#000000632	
66	130	F	79258.25	1994-01-20	5-LOW	Clerk#000000743	
67	58	O	116227.10	1996-12-19	4-NOT SPECIFIED	Clerk#000000547	
68	29	O	215135.78	1998-04-18	3-MEDIUM	Clerk#000000440	
69	85	F	162176.28	1994-06-04	4-NOT SPECIFIED	Clerk#000000330	
70	65	F	84651.87	1993-12-18	5-LOW	Clerk#000000322	
71	4	O	178821.78	1998-01-24	4-NOT SPECIFIED	Clerk#000000271	

6.1.2 有参数的存储过程

定义一个存储过程，更新**某条指定的订单**的（含税折扣价）总价。即根据订单明细表，计算指定订单（orderkey）的总价，更新ORDER表

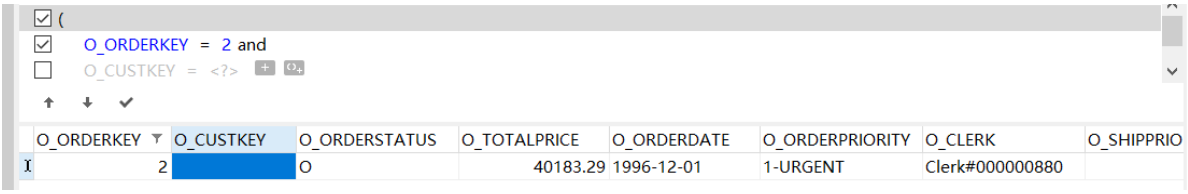
```
-- 2. 有参数的存储过程
DROP PROCEDURE  IF EXISTS Proc_CaltotalPrice1;
create procedure Proc_CaltotalPrice1(orderkey INTEGER)
begin
     update
     orders
set
    O_TOTALPRICE =
     (
```

```
    select
        sum(L_EXTENDEDPRICE *(1-L_DISCOUNT) *(1 + L_TAX))
    from
        lineitem
    where
        O_ORDERKEY = L_ORDERKEY
        and
        O_ORDERKEY = orderkey
     );
end;

call Proc_CaltotalPrice1(2);
```

运行结果

成功更新orderkey = 2的订单总价



6.1.3 有局部变量的存储过程

定义一个存储过程，更新某个顾客所有订单的（含税折扣价）总价。

更新顾客`Customer#000000002`的订单总价

指定参数为custname，定义一个局部变量custkey，记录该顾客的custkey，进而在order表中找到该顾客的所有订单

```
-- 3.有局部变量的存储过程
DROP PROCEDURE  IF EXISTS Proc_CaltotalPrice2;
create procedure Proc_CaltotalPrice2(custname char(25))
begin
DECLARE custkey INTEGER;
SELECT C_CUSTKEY into custkey
from customer
where C_NAME = custname;
     update
     orders
set
    O_TOTALPRICE =
     (
    select
        sum(L_EXTENDEDPRICE *(1-L_DISCOUNT) *(1 + L_TAX))
    from
        lineitem
    where
        O_ORDERKEY = L_ORDERKEY
        and
        O_CUSTKEY = custkey
     );
end;

call Proc_CaltotalPrice2('Customer#000000002');
```

运行结果

查看存储过程执行结果，可以看到，顾客'Customer#000000002'成功更新

```
SELECT *from orders
where O_CUSTKEY = (SELECT C_CUSTKEY from customer where C_NAME =
'Customer#000000002');
```

O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE	O_ORDERPRIORITY	O_CLERK	O_SHIPPF
353	2	F	179984.46	1993-12-31	5-LOW	Clerk#000000449	
896	2	F	169847.68	1993-03-09	1-URGENT	Clerk#000000187	
994	2	F	41433.51	1994-04-20	5-LOW	Clerk#000000497	
1504	2	F	89399.44	1992-08-28	3-MEDIUM	Clerk#000000381	
1603	2	F	29305.49	1993-07-31	4-NOT SPECIFIED	Clerk#000000869	
1669	2	O	24362.39	1997-06-09	3-MEDIUM	Clerk#000000663	
4704	2	O	63873.16	1996-08-16	4-NOT SPECIFIED	Clerk#000000256	
5507	2	O	140363.75	1998-05-28	5-LOW	Clerk#000000692	
5893	2	F	44777.64	1992-07-08	4-NOT SPECIFIED	Clerk#000000560	

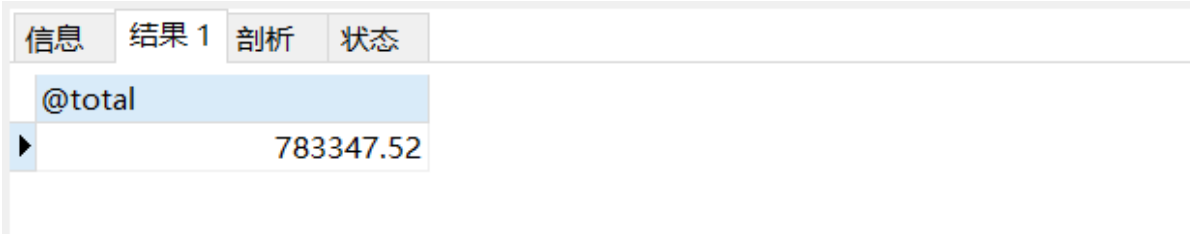
6.1.4 有输出参数的存储过程

定义一个存储过程，更新某个顾客所有订单的（含税折扣价）总价。并将结果输出到参数 totalprice中

```
create procedure Proc_CaltotalPrice3(custname char(25) , OUT totalprice
decimal(15,2) )
begin
DECLARE custkey INTEGER;
SELECT C_CUSTKEY into custkey
from customer
where C_NAME = custname;
     update
     orders
set
    O_TOTALPRICE =
     (
    select
        sum(L_EXTENDEDPRICE *(1-L_DISCOUNT) *(1 + L_TAX))
    from
        lineitem
    where
        O_ORDERKEY = L_ORDERKEY
        and
        O_CUSTKEY = custkey
     );
 select sum(O_TOTALPRICE) into totalprice
 FROM orders where O_CUSTKEY = custkey;
end;
```

调用存储过程，使用select语句查看输出结果，这里需要注意，需要用用户变量@total存放输出结果，@total是用户变量，他的值可以在sql语句之间传递。

```
call Proc_CaltotalPrice3('Customer#000000002',@total);
SELECT @total;
```



6.1.5 修改存储过程

使用 ALTER 语句可以修改存储过程或函数的特性，语法格式如下：

```
ALTER { PROCEDURE | FUNCTION } sp_name [ characteristic ... ]
```

其中，那 sp_name 参数表示存储过程或函数的名称；characteristic 参数指定存储函数的特性，可能的取值有:

- CONTAINS SQL 表示子程序包含 SQL 语句，但不包含读或写数据的语句。
- NO SQL 表示了程序中不包含 SQL 语句。
- READS SQL DATA 表示子程序中包含读数据的语句。
- MODIFIES SQL DATA 表示子程序中包含写数据的语句。
- SQL SECURITY { DEFINER | INVOKER } 指明谁有权限来执行。
- DEFINER 表示只有定义者自己才能够执行。
- INVOKER 表示调用者可以执行。
- COMMENT 'string' 表示注释信息。

修改存储过程Proc_CaltotalPrice2 的定义，将读写权限改为 MODIFIES SQL DATA，并指明调用者可以执行。

```
alter procedure Proc_CaltotalPrice2 MODIFIES SQL DATA SQL SECURITY INVOKER;
```

查看修改后存储过程Proc_CaltotalPrice2 的特性

思考题

> **(1) 试总结几种调试存储过程的方法**

1. 写语句调用

```
call p_next_id('t_factory',2,'0',@result); -- 上面的存储过程含有四个参数，所以这里调用的时候，也需要传递4个参数：输入参数填写值，输出参数用变量表示@result
select @result; -- 这句话是在控制台显示变量值
```

2. 窗口点击

直接点击运行时，在弹出输入框输入：

```
't_factory',2,'0',@result
```

追踪（调试）存储过程执行步骤

mysql不像oracle有plsqldevelper工具用来调试存储过程，所以有两简单的方式追踪执行过程：

- 利用“CREATE TEMPORARY TABLE”语句创建一张临时表，用于记录调试过程
- 直接在存储过程中，增加select @xxx
- 在控制台查看结果，根据输出结果修改代码：

> **(2) 存储过程的select语句与普通的select语句格式有何不同？执行方法有何不同**

- 存储过程中的select语句中的值可以赋给局部变量或者游标，而普通select语句是查询语句，查询结果直接返回给控制台。
- 存储过程负责将select语句等sql语句集合封装起来，并且需要用户显式调用才会执行，而普通select语句直接执行。

6.2 自定义函数实验

sql函数
函数存储着一系列sql语句，调用函数就是一次性执行这些语句。所以函数可以降低语句重复。实现代码重用与封装【但注意的是函数注重返回值，不注重执行过程，所以一些语句无法执行。所以函数并不是单纯的sql语句集合。】

函数与存储过程的区别：**函数只会返回一个值，不允许返回一个结果集。**函数强调返回值，所以函数不允许返回多个值的情况，即使是查询语句。

6.2.1 无参数的自定义函数

定义一个存储过程，更新订单的（含税折扣价）总价。并返回所有订单的总价

```
-- 1.无参数的自定义函数
drop FUNCTION FUN_CalTotalPrice;

CREATE FUNCTION FUN_CalTotalPrice()
RETURNS DECIMAL( 15, 2 ) DETERMINISTIC
BEGIN
    DECLARE res DECIMAL ( 15, 2 );
    UPDATE orders
    SET O_TOTALPRICE = ( SELECT sum( L_EXTENDEDPRICE *( 1-L_DISCOUNT ) *( 1 +
L_TAX )) FROM lineitem WHERE O_ORDERKEY = L_ORDERKEY );

    SELECT
        sum(O_TOTALPRICE) INTO res
    FROM
        orders;
    RETURN res;

END;
```

执行自定义函数FUN_CalTotalPrice()

```
-- 执行自定义函数
SELECT FUN_CalTotalPrice() ;
```

信息 结果 1 剖析 状态

FUN_CalTotalPrice()
151008955.66

6.2.2 有参数的自定义函数

定义一个存储过程，更新**某条指定的订单**的（含税折扣价）总价。即根据订单明细表，计算指定订单（orderkey）的总价，更新ORDER表

```
-- 2.有参数自定义函数
drop FUNCTION FUN_CalTotalPrice1;

CREATE FUNCTION FUN_CalTotalPrice1(orderkey INTEGER)
RETURNs DECIMAL( 15, 2 ) DETERMINISTIC
BEGIN
    DECLARE res DECIMAL ( 15, 2 );
    UPDATE orders
    SET O_TOTALPRICE = ( SELECT sum( L_EXTENDEDPRICE *( 1-L_DISCOUNT ) *( 1 +
L_TAX )) FROM lineitem WHERE O_ORDERKEY = L_ORDERKEY and L_ORDERKEY = orderkey);

    SELECT
        O_TOTALPRICE INTO res
    FROM
        orders
    where O_ORDERKEY = orderkey;

    RETURN res;

END;
```

执行自定义函数FUN_CalTotalPrice1()

```
SELECT FUN_CalTotalPrice1(1);
```

信息 结果 1 剖析 状态

FUN_CalTotalPrice1(
131251.88

思考题

> **(1) 试分析自定义函数与存储过程的区别和联系**

区别

- 存储过程实现的功能相对复杂，函数针对性较强
- 函数强调返回值，所以函数不允许返回多个值的情况，即使是查询语句。存储过程可以返回多个值，函数只能有一个返回值
- 存储过程作为一个独立的部分来执行，函数可作为查询语句的一个部分调用，函数可以嵌在SQL中使用(select count(*) from table_name)，存储过程不行

联系

- 都是一组基础sql语句的集合，能实现复杂的功能，同时实现了功能的封装，需要用户显式调用。
- 在创建时可以指定参数列表，并可以有返回值

> **(2) 如何使得自定义函数可以返回多个值？如何利用？**

1.可以建立一个临时表，将所有想要返回的值存储在临时表中，然后将临时表作为值返回

2.如果需要返回的值是字符串，可以使用concat()函数或者concat_ws()函数，将多个字符串连接成一个字符串。

6.3 游标实验

1、概念
游标(Cursor)它使用户可逐行访问由SQL Server返回的结果集。 使用游标(cursor)的一个主要的原因就是把集合操作转换成单个记录处理方式。

用SQL语言从数据库中检索数据后，结果放在内存的一块区域中，且结果往往是一个含有多个记录的集合。

游标机制允许用户在SQL server内逐行地访问这些记录，按照用户自己的意愿来显示和处理这些记录。

优点
1、允许程序对由查询语句select返回的行集合中的每一行执行相同或不同的操作，而不是对整个行集合执行同一个操作。

2、提供对基于游标位置的表中的行进行删除和更新的能力。

3、游标实际上作为面向集合的数据库管理系统（RDBMS）和面向行的程序设计之间的桥梁，使这两种处理方式通过游标沟通起来。

原理
游标就是把数据按照指定要求提取出相应的数据集，然后逐条进行数据处理。
使用游标的顺序
声名游标、打开游标、读取数据、关闭游标、删除游标。

使用游标(cursor)

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。声明游标后也是单条操作的，但是不能用SELECT语句不能有INTO子句。
2. 游标OPEN语句
   OPEN cursor_name 这个语句打开先前声明的游标。
3. 游标FETCH语句
   FETCH cursor_name INTO var_name [, var_name] ...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。
4. 游标CLOSE语句
   CLOSE cursor_name 这个语句关闭先前打开的游标

6.3.1 loop循环游标

使用游标统计customer表中有多少个顾客来自中国（name = china ， nationkey = 18）

```
-- 1.使用游标统计customer表中有多少个顾客来自中国（name = china ， nationkey = 18）
-- 在windows系统中写存储过程时，如果需要使用declare声明变量，需要添加这个关键字，否则会报错。
delimiter //
drop procedure if exists CalCustomer_Nation;
CREATE PROCEDURE CalCustomer_Nation()
BEGIN
    -- 创建接收游标数据的变量
    declare nationkey int;
    declare name varchar(20);
    -- 创建总数变量
    declare total int default 0;
    -- 创建结束标志变量
    declare done int default false;
    -- 创建游标
    declare cur cursor for
    SELECT C_NAME,C_NATIONKEY from customer
    where C_NATIONKEY = 18;
    -- 指定游标循环结束时的返回值
    declare continue HANDLER for not found set done = true;
    -- 设置初始值
    set total = 0;
    -- 打开游标
    open cur;
    -- 开始循环游标里的数据
    read_loop:loop
    -- 根据游标当前指向的一条数据
    fetch cur into name,nationkey;
    -- 判断游标的循环是否结束
    if done then
        leave read_loop;    -- 跳出游标循环
    end if;
    -- 获取一条数据时，将total值进行累加操作，这里可以做任意你想做的操作，
    set total = total + 1;
    -- 结束游标循环
    end loop;
    -- 关闭游标
    close cur;

    -- 输出结果
    select total;
END;
```

调用存储过程

6.3.2 while 循环游标

使用游标统计customer表中有多少个顾客来自中国（name = china ， nationkey = 18）

```
-- 2 while 循环
delimiter //
drop procedure if exists CalCustomer_Nation1;
CREATE PROCEDURE CalCustomer_Nation1()
BEGIN
    -- 创建接收游标数据的变量
    declare nationkey int;
    declare name varchar(20);
    --
    declare total int default 0;
    -- 创建结束标志变量
    declare done int default false;
    -- 创建游标
    declare cur cursor for
    SELECT C_NAME,C_NATIONKEY from customer
    where C_NATIONKEY = 18;
    -- 指定游标循环结束时的返回值
    declare continue HANDLER for not found set done = true;
    -- 设置初始值
    set total = 0;
    -- 打开游标
    open cur;
    -- 开始循环游标里的数据

    while(not done) do
        fetch cur into name,nationkey;
        set total = total + 1;
    end while;

    -- 关闭游标
    close cur;

    -- 输出结果
    select total;
END;
```

6.3.3 repeat循环游标

使用游标统计customer表中有多少个顾客来自中国（name = china）

```
-- 3.repeat 循环
delimiter //
drop procedure if exists CalCustomer_Nation2;
CREATE PROCEDURE CalCustomer_Nation2()
BEGIN
    -- 创建接收游标数据的变量
    declare nationkey int;
    declare name varchar(20);
    -- 创建总数变量
    declare total int default 0;
```

```
    -- 创建结束标志变量
    declare done int default false;
    -- 创建游标
    declare cur cursor for
    SELECT C_NAME,C_NATIONKEY from customer
    where C_NATIONKEY = 18;
    -- 指定游标循环结束时的返回值
    declare continue HANDLER for not found set done = true;
    -- 设置初始值
    set total = 0;
    -- 打开游标
    open cur;
    -- 开始循环游标里的数据
    REPEAT
    fetch cur into name,nationkey;
    if not done then
        set total = total + 1;
    end if;
    until done end repeat;

    -- 关闭游标
    close cur;
    -- 输出结果
    select total;
END;
-- 调用存储过程
call CalCustomer_Nation2();
```

6.3.4 带参数游标

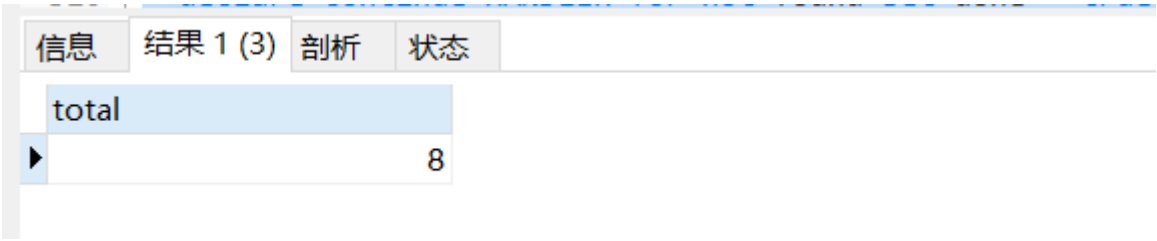
```
delimiter //
drop procedure if exists CalCustomer_Nation3;
CREATE PROCEDURE CalCustomer_Nation3(nname varchar(25))
BEGIN
    -- 创建接收游标数据的变量
    declare nationkey int;
    declare name varchar(20);
    -- 创建总数变量
    declare total int default 0;
    -- 创建结束标志变量
    declare done int default false;
    -- 创建游标
    declare cur cursor for
    SELECT C_NAME,C_NATIONKEY from customer
    where C_NATIONKEY = (select N_NATIONKEY from nation where N_NAME = nname);
    -- 指定游标循环结束时的返回值
    declare continue HANDLER for not found set done = true;
    -- 设置初始值
    set total = 0;
    -- 打开游标
    open cur;
    -- 开始循环游标里的数据
    read_loop:loop
    -- 根据游标当前指向的一条数据
    fetch cur into name,nationkey;
    -- 判断游标的循环是否结束
```

```
    if done then
        leave read_loop;    -- 跳出游标循环
    end if;
    -- 获取一条数据时，将total值进行累加操作，这里可以做任意你想做的操作，
    set total = total + 1;
    -- 结束游标循环
    end loop;
    -- 关闭游标
    close cur;

    -- 输出结果
    select total;
END;
```

调用存储过程

```
-- 调用存储过程
call CalCustomer_Nation3('CHINA');
```



思考题

> **试分析说明REFCURSOR类型游标的优点**

（mysql中没有refcursor类型游标的概念）

优点一：sys_refcursor，可以在存储过程中作为参数返回一个table格式的结构集(我把他认为是table类型，容易理解，其实是一个游标集)， cursor 只能用在存储过程，函数，包等的实现体中，不能做参数使用。

优点二：sys_refcursor 这东西可以使用在包中做参数，进行数据库面向对象开放。

实验总结

存储过程

初学存储过程，很容易联想到C++中的函数，还有上一次实验的触发器。触发器与存储过程非常相似，触发器也是SQL语句集，两者唯一的区别是触发器不能用EXECUTE语句调用，而是在用户执行Transact-SQL语句时自动触发（激活）执行。而存储过程需要用户显式调用。同时两者的应用场景不同。触发器是数据库完整性的一种有效实现方式。存储过程

自定义函数

函数和存储过程也很相似，需要注意两者的区别和联系。mysql自定义函数参数类型没有IN，OUT，INOUT类型之分。默认只能为输入型参数。

函数只能返回一个值，如果想要返回多个值，可以将结果存储在临时表内，然后返回临时表。

游标

还有存储过程中游标的使用。MySQL游标是只读的，不可滚动且不敏感的。

> - **只读**：您无法通过游标更新基础表中的数据。
> - **不可滚动**：您只能按`SELECT`语句确定的顺序获取行。您无法以相反的顺序获取行。此外，您不能跳过行或跳转到结果集中的特定行。
> - **未定型**：有两种光标：未定型游标和不敏感游标。敏感光标指向实际数据，而不敏感光标使用数据的临时副本。敏感性游标比不敏感游标执行得更快，因为它不必创建临时数据副本。但是，对来自其他连接的数据所做的任何更改都将影响敏感光标正在使用的数据，因此，如果不更新敏感光标正在使用的数据，则更安全。MySQL游标是敏感的。

另外，触发器中也可以使用游标。