# 存储过程实验实验

16337113

劳马东

计算机科学与技术（超算方向）

## 实验目的

掌握数据库存储过程的设计和使用方法。

## 实验内容和要求

存储过程定义、运行、更名、删除、参数传递。

## 实验环境

|  |  |
| --- | --- |
| 系统 | Windows 10 |
| SQL | MySQL 8.0 |
| 工具 | MySQL Workbench |

## 实验过程

###### 无参数的存储过程

###### 定义一个存储过程，更新所有订单的（含税折扣价）总价。

默认情况下，MySQL的语句结束符是分号，而存储过程是含有许多MySQL语句的语句块，它们也是以分号结束的。这样，MySQL编译器就会认为第一个分号出现的地方就是存储过程定义的结束，这显然会出现语法错误。因此，需要用DELIMITER语句修改存储过程结束符为//或$$，而让普通MySQL语句的结束符为分号。

下面的代码创建一个存储过程，它使用SUM聚集函数计算同一个orderkey的订单的总价，并将这个总价更新到orders表对应orderkey的每一条记录中。

DELIMITER //

CREATE PROCEDURE Proc\_CalTotalPrice()

BEGIN

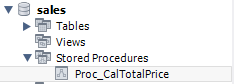
UPDATE orders SET totalprice =

         (SELECT SUM(extendedprice \* (1-discount) \* (1+tax))

FROM lineitem WHERE orders.orderkey = lineitem.orderkey);

END//

创建完毕后，在MySQL workbench中点开sales模式的Stored Procedures查看所有已定义的存储过程，可以看到出现了Proc\_CalTotalPrice，说明存储过程创建成功。

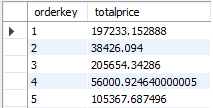


###### 执行存储过程Proc\_CalTotalPrice()。

使用CALL关键字调用存储过程，它会更新orders表每条记录的totalprice属性值。

CALL Proc\_CalTotalPrice();

使用SELECT查询语句显示每种orderkey的订单对应的totalprice，结果如下，每种订单的总价从NULL变成了对应的含税折扣价之和。



###### 有参数的存储过程

###### 定义一个存储过程，更新给定订单的总价。

该存储过程与上一个的区别是多了一个p\_okey参数，它是需要更新总价的订单的orderkey，在该存储过程中WHERE条件判断因此也加上lineitem.orderkey = p\_okey，这有p\_okey订单会被计算总价。

DELIMITER //

CREATE PROCEDURE Proc\_CalTotalPrice4Order(p\_okey INTEGER)

BEGIN

UPDATE orders SET totalprice =

         (SELECT SUM(extendedprice \* (1-discount) \* (1+tax))

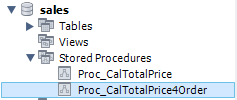
FROM lineitem

WHERE orders.orderkey=lineitem.orderkey

AND lineitem.orderkey = p\_okey

);

END//



###### 执行存储过程Proc\_CalTotalPrice4Order()。

调用存储过程更新orderkey为100的订单。

CALL Proc\_CalTotalPrice4Order(100);

为了去掉上一个存储过程的影响，显示这个存储过程的效果，CALL之前先将orders表中orderkey为100的订单的totalprice修改为0，如下左图。调用存储过程之后，totalprice变为198978.322212，说明存储过程执行成功了。

###### 

###### 有局部变量的存储过程

###### 定义一个存储过程，更新某个顾客的所有订单的（含税折扣价）总价。

该存储过程的不同之处是利用DECLARE语句声明一个L\_custkey局部变量，它存储名为custname的用户的custkey，之后利用L\_custkey变量更新orders表对应custkey的订单的总价。

DELIMITER //

CREATE PROCEDURE Proc\_CalTotalPrice4Customer(custname CHAR(25))

BEGIN

DECLARE L\_custkey INTEGER;

SELECT custkey INTO L\_custkey FROM customer WHERE name=TRIM(custname);

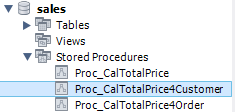
UPDATE orders SET totalprice =

     (SELECT SUM(extendedprice \* (1-discount) \* (1+tax))

FROM lineitem WHERE orders.orderkey=lineitem.orderkey

AND orders.custkey=L\_custkey);

END//



###### 执行存储过程Proc\_CalTotalPrice4Customer()。

该存储过程使用客户名作为参数而不是客户主码，下面的代码更新名为Customer#000000100的客户的订单总价，这个客户的custkey是100。

CALL Proc\_CalTotalPrice4Customer('Customer#000000100');

从下图可以看出，该客户的订单总价从0变成了69020.707626，说明存储过程调用成功了。

###### 有输出参数的存储过程

###### 定义一个存储过程，更新某个顾客的所有订单的（含税折扣价）总价。

该存储过程的不同之处是有一个OUT参数，它在存储过程中被赋值（SET、SELECT…INTO）会影响到外部对应的一个变量，相当于C++中的引用类型。参数默认是IN类型，作为输入参数，相当有C/C++中的值传递；此外还有INOUT类型，既作为输入参数，又作为输出参数。

存储过程在最后将orders表中的所有总价求和，赋值给p\_totalprice。

DELIMITER //

CREATE PROCEDURE Proc\_CalTotalPrice4Customer2

(custname CHAR(25), OUT p\_totalprice REAL)

BEGIN

DECLARE L\_custkey INTEGER;

SELECT custkey INTO L\_custkey FROM customer WHERE name=TRIM(custname);

UPDATE orders SET totalprice =

     (SELECT SUM(extendedprice \* (1-discount) \* (1+tax))

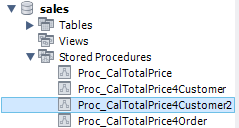
FROM lineitem WHERE orders.orderkey=lineitem.orderkey

AND orders.custkey=L\_custkey);

SELECT SUM(totalprice) INTO p\_totalprice

FROM orders WHERE custkey=L\_custkey;

END//



###### 执行存储过程Proc\_CalTotalPrice4Customer2()。

OUT类型参数必须有一个对应的外部变量，不能是NULL，因此教材中将NULL传递给p\_totalprice的做法是不正确的。下面的代码使用SET定义了一个全局变量@p\_totalprice，初始化为0，将其传入Proc\_CalTotalPrice4Customer2过程，调用完毕后用SELECT语句查看它的值。此外，在orders表上使用查询语句进一步验证@p\_totalprice的值是否正确，与总价之和一致。

SET @p\_totalprice = 0;

CALL Proc\_CalTotalPrice4Customer2('Customer#000000100', @p\_totalprice);

SELECT @p\_totalprice;

SELECT SUM(totalprice) FROM orders WHERE custkey=

(SELECT custkey FROM customer WHERE name='Customer#000000100');

从下图可以看出，@p\_totalprice的值从0变成2690838.346334，且与求和语句的的结果相同，说明OUT参数起作用了，存储过程也正确执行。

如果将NULL传递到OUT参数，会发生如下错误：Error Code: 1414. OUT or INOUT argument 2 for routine sales.Proc\_CalTotalPrice4Customer2 is not a variable or NEW pseudo-variable in BEFORE trigger，意思是第2个是OUT/INOUT参数，必须将一个变量传递给它。

###### 修改存储过程

###### 修改存储过程名Proc\_CalTotalPrice4Order为CalTotalPrice4Order。

MySQL中没有RENAME语句，因此为了重命名存储过程，可以修改系统的配置数据库mysql中对应的表格——proc，将name和specific\_name修改为新的名字。

此外，还有一种比较粗暴的方法是先删除Proc\_CalTotalPrice4Order，然后再创建一个叫CalTotalPrice4Order的新存储过程。

UPDATE `mysql`.`proc`

SET name='CalTotalPrice4Order', specific\_name='CalTotalPrice4Order'

WHERE name='Proc\_CalTotalPrice4Order';

###### 修改后，在MySQL workbench中刷新，就能看到CalTotalPrice4Order存储过程，说明重命名成功。

###### 

###### 删除存储过程

删除存储过程CalTotalPrice4Order。

DROP PROCEDURE CalTotalPrice4Order;

## 实验总结

这次的实验整体而言比较顺利，因为MySQL的存储过程或者函数与其他语言的过程/函数的结构、术语类似，只要稍加类比，很快就能理解并打出对应的代码。

不过实验中还是发生了一些小问题。由于这个实验只需选做其中的一个，一开始我选择做函数部分，遇到的第一个问题是FUNCTION的定义需要加上DETERMINISTIC、NO SQL或READS SQL DATA，因为MySQL为了保证系统一致性，需要函数在修改表格时申请SUPER权限。

遇到的第二个跨不过去的问题是返回多个值，即使用OUT类型参数。我查阅MySQL的官方文档，发现FUNCTION没有OUT参数这种东西，于是为了实验能顺利做下去，我转为选做存储过程，PROCEDURE和FUNCTION二者在定义语法和逻辑上相差无几，因此也能利用已经打好的代码。

但是，选做存储过程也并非一帆风顺。存储过程的ALTER操作没有重命名选项，查阅文档得到的几个选项也没有例子，因此不知道如何用ALTER语句重命名。后来，我在Stack Overflow上找到了一种终极方法——修改mysql.proc，这和之前使用过的user表一样是系统的配置信息表格，修改它就能达到重命名的目的。

总而言之，这次实验得益于类比方法，我很快完成了实验，遇到的只是一些小问题。