存储过程实验实验

16337113
劳马东
计算机科学与技术（超算方向）

一、 实验目的

掌握数据库存储过程的设计和使用方法。

二、 实验内容和要求

存储过程定义、运行、更名、删除、参数传递。

三、 实验环境

系统	Windows 10
SQL	MySQL 8.0
工具	MySQL Workbench

四、 实验过程

1. 无参数的存储过程

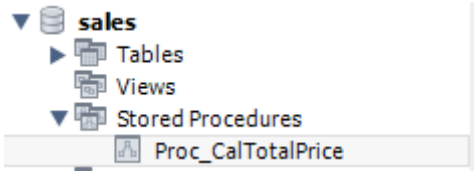
1) 定义一个存储过程，更新所有订单的（含税折扣价）总价。

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。



2） 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。

orderkey	totalprice
1	197233.152888
2	38426.094
3	205654.34286
4	56000.924640000005
5	105367.687496

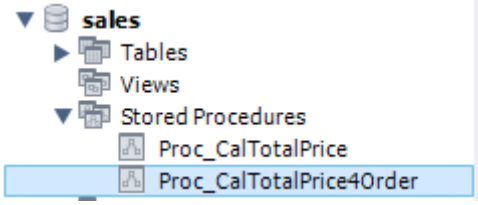
2. 有参数的存储过程

1） 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//
```



2） 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。

	orderkey	totalprice
▶	100	0

	orderkey	totalprice
▶	100	198978.32221199997

3. 有局部变量的存储过程

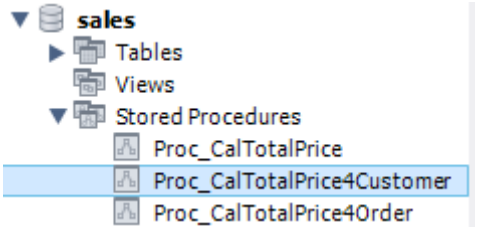
1） 定义一个存储过程，更新某个顾客的所有订单的（含税折扣价）总价。

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//
```



2） 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。

	custkey	totalprice
▶	100	0

	custkey	totalprice
▶	100	69020.707626

4. 有输出参数的存储过程

1) 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。

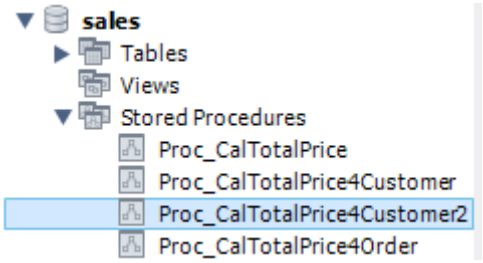
该存储过程的不同之处是有一个 OUT 参数，它在存储过程中被赋值（SET、SELECT…INTO）会影响到外部对应的一个变量，相当于 C++中的引用类型。参数默认是 IN 类型，作为输入参数，相当有 C/C++中的值传递；此外还有 INOUT 类型，既作为输入参数，又作为输出参数。

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//
```



2) 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。

	@p_totalprice
▶	2690838.346334

	SUM(totalprice)
▶	2690838.346334

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。

5. 修改存储过程

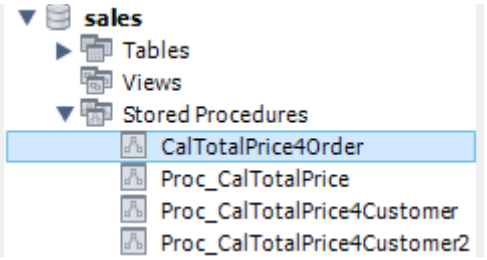
1) 修改存储过程名 Proc_CalTotalPrice4Order 为 CalTotalPrice4Order。

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。



6. 删除存储过程

删除存储过程 CalTotalPrice4Order。

```
DROP PROCEDURE CalTotalPrice4Order;
```

五、 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。