实验报告

题目：存储过程实验	姓名	郭梦鸽	日期	2022/5/12

实验 6.1 存储过程实验

1. 实验目的

掌握数据库 PL/SQL 编程语言，以及数据库存储过程的设计和使用方法。

2. 实验内容和要求

存储过程定义，存储过程运行，存储过程更名，存储过程删除，存储过程的参数传递。掌握 PL/SQL 编程语言和编程规范，规范设计存储过程。

3. 实验过程

创建 product 表和 orders 表，并通过 INSERT INTO 语句插入数据：

```
CREATE table product(
    pid INT PRIMARY KEY AUTO_INCREMENT,#产品编号
    pname VARCHAR(10), #产品名称
    price NUMERIC(5,2),#产品价格
    pnum INT,          #产品数量
    discount DOUBLE    #产品折扣
);
CREATE table orders(
    oid INT PRIMARY KEY AUTO_INCREMENT,#订单编号
    pid INT,
    FOREIGN KEY(pid) REFERENCES product(pid),
    oprice NUMERIC(5,2),
    onum INT,                  #数量
    totalprice NUMERIC(10,2)#订单总价
);
```

pid	pname	price	pnum	discount
1	零件A	2.4	1000	0.8
2	零件B	5.6	1500	0.75
3	零件C	1.7	800	0.95

oid	pid	onum	totalprice	oprice
1	1	240	(Null)	(Null)
2	3	320	(Null)	(Null)
3	2	540	(Null)	(Null)
4	2	200	(Null)	(Null)

步骤 1：无参数的存储过程

(1)定义一个存储过程，更新 orders 表中所有订单的单价和总价。

```
CREATE PROCEDURE CalTotalPrice()
BEGIN
    UPDATE orders SET oprice =
        (SELECT price * discount
        FROM product
```

```
        WHERE product.pid = orders.pid);
    UPDATE orders SET totalprice = oprice * onum;
END;
```

(2)执行存储过程 CalTotalPrice()。

```
CALL CalTotalPrice();
```

oid	pid	onum	totalprice	oprice
1	1	240	460.8	1.92
2	3	320	518.4	1.62
3	2	540	2268	4.2
4	2	200	840	4.2

步骤 2：有参数的存储过程

(1)定义一个存储过程，更新 orders 表中折扣有变化的订单的单价和总价。

```
UPDATE product SET discount=0.6 WHERE pid=3;
```

pid	pname	price	pnum	discount
1	零件A	2.4	1000	0.8
2	零件B	5.6	1500	0.75
3	零件C	1.7	800	0.6

```
CREATE PROCEDURE CalTotalpriceOrder(ono INT)
BEGIN
    UPDATE orders SET oprice=
        (SELECT price*discount
        FROM product
        WHERE product.pid=orders.pid);
UPDATE orders SET totalprice=oprice*onum WHERE orders.oid=ono;
END;
```

(2)执行存储过程 CalTotalpriceOrder()，带参数的调用。

```
CALL CalTotalpriceOrder(2);
```

oid	pid	oprice	onum	totalprice
1	1	1.92	240	460.8
2	3	1.02	320	326.4
3	2	4.2	540	2268
4	2	4.2	200	840

步骤 3：有局部变量的存储过程

(1)定义一个存储过程，更新 orders 表中某个零件的订单的单价和总价。

```
UPDATE product SET discount=0.7 WHERE pname='零件 B';
```

pid	pname	price	pnum	discount
1	零件A	2.4	1000	0.8
2	零件B	5.6	1500	0.7

oid	pid	oprice	onum	totalprice
1	1	1.92	240	460.8
2	3	1.02	320	326.4
3	2	4.2	540	2268
4	2	4.2	200	840

```
CREATE PROCEDURE CalTotalpricePro(pro_name VARCHAR(10))
BEGIN
    DECLARE pno INT;
    SELECT pid INTO pno
    FROM product
    WHERE pname=pro_name;
    UPDATE orders SET oprice=
        (SELECT price*discount
        FROM product
        WHERE product.pid=orders.pid);
UPDATE orders SET totalprice=oprice*onum WHERE orders.pid=pno;
END;
```

(2)执行存储过程 CalTotalpricePro()，带参数的调用。

```
CALL CalTotalpricePro('零件 B');
```

oid	pid	oprice	onum	totalprice
1	1	1.92	240	460.8
2	3	1.02	320	326.4
3	2	3.92	540	2116.8
4	2	3.92	200	784

步骤 4：有输出参数的存储过程

(1)定义一个存储过程，更新 orders 表中某个零件的订单的单价和总价。

```
UPDATE product SET discount=0.9 WHERE pname='零件 A';
```

pid	pname	price	pnum	discount
1	零件A	2.4	1000	0.9
2	零件B	5.6	1500	0.7
3	零件C	1.7	800	0.6

```
CREATE PROCEDURE CalTotalpriceOut(pro_name VARCHAR(10),OUT a_
totalprice DOUBLE)
BEGIN
    DECLARE pno INT;
    SELECT pid INTO pno
    FROM product
    WHERE pname=pro_name;
    UPDATE orders SET oprice=
        (SELECT price*discount
        FROM product
        WHERE product.pid=orders.pid);
```

```
UPDATE orders SET totalprice=oprice*onum WHERE orders.pid=pno;
SELECT totalprice INTO a_totalprice FROM orders WHERE orders.pid
=pno;
END;
```

(2)执行存储过程 CalTotalpricePro()。

```
CALL CalTotalpriceOut('零件 A',@A_totalprice);
```

oid	pid	oprice	onum	totalprice
1	1	2.16	240	518.4
2	3	1.02	320	326.4
3	2	3.92	540	2116.8
4	2	3.92	200	784

```
SELECT @A_totalprice;
```

@A_totalprice
518.4

步骤 5：修改存储过程

查看存储过程：

```
SHOW PROCEDURE STATUS LIKE "%CalTotalpriceOut";
```

Db	Name	Type	Definer	Modified	Created	Security_type
business	CalTotalpriceOut	PROCEDURE	root@localhost	2022-05-12	2022-05-12	DEFINER

修改存储过程：

```
ALTER PROCEDURE CalTotalpriceOut MODIFIES SQL DATA SQL SECURITY
INVOKER;
SHOW PROCEDURE STATUS LIKE "%CalTotalpriceOut";
```

Db	Name	Type	Definer	Modified	Created	Security_type
business	CalTotalpriceOut	PROCEDURE	root@localhost	2022-05-13	2022-05-12	INVOKER

步骤 6：删除存储过程

```
DROP PROCEDURE CalTotalPrice;
DROP PROCEDURE CalTotalpriceOrder;
DROP PROCEDURE CalTotalpricePro;
DROP PROCEDURE CalTotalpriceOut;
```

思考题：

(1)试总结几种调试存储过程的方法。

- 利用 CREATE TEMPORARY TABLE 语句创建一张临时表来记录调试过程；
- 在存储过程中，通过 select @xxx 语句在控制台查看结果；
- 打开控制台，在控制台中查看结果，根据输出结果修改代码即可。

(2)存储过程中的 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，可以将查询的结果赋给存储过程中的变量，而

普通的 SELECT 语句中无法使用；
存储过程是由过程化 SQL 语句书写的过程，由于存储过程不像解释执行的 SQL 语句那样在提出操作请求时才进行语法分析和优化工作，因而运行效率高，提供了在服务器端快速执行 SQL 语句的有效途径。

实验 6.2 自定义函数实验

1. 实验目的
掌握数据库 PL/SQL 编程语言以及数据库自定义函数的设计和使用方法。

2. 实验内容和要求
自定义函数定义，自定义函数运行，自定义函数更名，自定义函数删除，自定义函数的参数传递。掌握 PL/SQL 编程语言和编程规范，规范设计自定义函数。

3. 实验过程

步骤 1：无参数的自定义函数

(1)定义一个自定义函数，更新 orders 表中所有订单的单价和总价，并返回所有订单的总价之和。

```
> 1418 - This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA
in its declaration and binary logging is enabled (you *might* want to use the
less safe log_bin_trust_function_creators variable)
> 时间: 0.002s
```

```
SET global log_bin_trust_function_creators=TRUE;
CREATE FUNCTION FunTotalPrice()
RETURNS DOUBLE
BEGIN
    DECLARE sumprice DOUBLE;
    SELECT SUM(totalprice) INTO sumprice FROM orders;
    RETURN sumprice;
END;
```

(2)执行自定义函数 FunTotalPrice()。

```
SELECT FunTotalPrice();
```

oid	pid	oprice	onum	totalprice
1	1	2.16	240	518.4
2	3	1.02	320	326.4
3	2	3.92	540	2116.8
4	2	3.92	200	784

FunTotalPrice()
3745.6

步骤 2：有参数的自定义函数

(1)定义一个自定义函数，更新并返回给定订单的税费。

```
CREATE FUNCTION FunTotalpriceOrder(ono INT)
RETURNS DOUBLE
```

```
    BEGIN
        DECLARE taxprice DOUBLE;
        SELECT totalprice * 0.05 INTO taxprice FROM orders WHERE
orders.oid=ono;
        RETURN taxprice;
    END;
```

(2)执行自定义函数 FunTotalpriceOrder()。

```
SELECT FunTotalpriceOrder(2);
```

FunTotalpriceOrder(2)
16.32

步骤 3：有局部变量的自定义函数

(1)定义一个自定义函数，计算并返回某种零件的所有订单的总价。

```
CREATE FUNCTION FunTotalpricePro(pro_name VARCHAR(10))
RETURNS DOUBLE
BEGIN
    DECLARE pno INT;
    DECLARE proprice DOUBLE;
    SELECT pid INTO pno
    FROM product
    WHERE pname=pro_name;
SELECT SUM(totalprice) INTO proprice FROM orders WHERE orders.
pid=pno;
    RETURN proprice;
END;
```

(2)执行自定义函数 FunTotalpricePro()。

```
SELECT FunTotalpricePro('零件 B');
```

3	2	3.92	540	2116.8
4	2	3.92	200	784

FunTotalpricePro('零件B')
2900.8

步骤 4：修改自定义函数

查看自定义函数：

```
SHOW FUNCTION STATUS LIKE "%FunTotalPrice";
```

Db	Name	Type	Definer	Modified	Created	Security_type
business	FunTotalPrice	FUNCTION	root@localhost	2022-05-12	2022-05-12	DEFINER

修改自定义函数：

```
ALTER FUNCTION FunTotalPrice MODIFIES SQL DATA SQL SECURITY
INVOKER;
```

```
SHOW FUNCTION STATUS LIKE "%FunTotalPrice";
```

Db	Name	Type	Definer	Modified	Created	Security_type
business	FunTotalPrice	FUNCTION	root@localhost	2022-05-13	2022-05-12	INVOKER

步骤 5：删除自定义函数

```
DROP FUNCTION FunTotalPrice;
DROP FUNCTION FunTotalpriceOrder;
DROP FUNCTION FunTotalpricePro;
```

思考题：

(1)试分析自定义函数与存储过程的区别与联系。

自定义函数中必须包含 RETURN 语句，用来返回一个值，存储过程则不需要 RETURN 语句；

存储过程实现的功能相对复杂，自定义函数针对性较强；

函数可以嵌入到 SQL 语句中使用，可以在 SELECT 语句中调用，而存储过程一般独立执行，使用 CALL 语句；

存储过程可以定义 IN（输入），OUT（输出）、INOUT（输入输出）三种类型的参数，自定义函数不能定义输出参数；

(2)如何使得自定义函数可以返回多个值？如何利用？

RETURN 语句只能返回一个操作数，也就是只能返回一个值，不能一次返回多个值。如果需要返回多个值，可以在函数中定义一个数组，将返回值存储在数组中返回。或使用 concat()函数将多个字符串连接成一个字符串。

实验 6.3 游标实验

1. 实验目的

掌握 PL/SQL 游标的设计、定义和使用方法，理解 PL/SQL 游标按行操作和 SQL 按结果集操作的区别和联系。

2. 实验内容和要求

游标定义、游标使用。掌握各种类型游标的特点、区别与联系。

3. 实验过程

步骤 1：普通游标

(1)定义一个存储过程，用游标实现计算所有订单的总价。

```
CREATE PROCEDURE GetTotalPrice()
BEGIN
    DECLARE sumprice DOUBLE;
    DECLARE proprice DOUBLE;
    DECLARE ono INT;
    DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR SELECT oid FROM orders;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=TRUE;
    SET sumprice=0;
```

```
        OPEN cur;
        read_loop:LOOP
        FETCH cur INTO ono;
        IF done THEN
            LEAVE read_loop;
        END IF;
    SELECT totalprice INTO proprice FROM orders WHERE orders.oid=ono;
        SET sumprice=sumprice+proprice;
        END LOOP;
        CLOSE cur;
        SELECT sumprice;
    END;
```

(2)执行存储过程 GetTotalPrice()。

```
CALL GetTotalPrice();
```

sumprice
3745.60

步骤 2：带参数的游标

(1)定义一个存储过程，用游标实现计算某零件的订单的总价。

```
    CREATE PROCEDURE GetTotalPricePro(pro_name VARCHAR(10))
    BEGIN
        DECLARE proprice DOUBLE;
        DECLARE sumprice DOUBLE;
        DECLARE ono INT;
        DECLARE done INT DEFAULT FALSE;
    DECLARE cur CURSOR FOR SELECT oid FROM orders,product WHERE
orders.pid=product.pid AND pname=pro_name;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=TRUE;
        SET sumprice=0;
        OPEN cur;
        read_loop:LOOP
        FETCH cur INTO ono;
        IF done THEN
            LEAVE read_loop;
        END IF;
    SELECT totalprice INTO proprice FROM orders WHERE orders.oid=ono;
        SET sumprice=sumprice+proprice;
        END LOOP;
        CLOSE cur;
        SELECT sumprice;
    END;
```

(2)执行存储过程 GetTotalPrice()。

```
CALL GetTotalPricePro('零件 A');
```

sumprice
518.4

```
CALL GetTotalPricePro('零件 C');
```

sumprice
326.4

步骤 3：删除存储过程

```
DROP PROCEDURE GetTotalPrice;
DROP PROCEDURE GetTotalPricePro;
```

思考题：

(1)试分析说明 REFCURSOR 类型游标的优点。

REFCURSOR 类型的游标定义一个游标引用变量，只是在打开该类型游标时才指定具体的 SELECT 语句以便产生游标的结果集。因此 REFCURSOR 实质上是定义了一个动态游标，可以灵活方便地根据程序运行时情况动态设置游标的 SELECT 查询结果集。

实验总结与感悟

总结：

1. 存储过程、用户自定义函数可以通过 CALL 和 SELECT 语句调用。

   存储过程、用户自定义函数如果带有 OUT 或 INOUT 参数，则参数对应位置在调用时必须使用 NULL 或其他常量占位。运行所得是一个结果集，结果集由一条或多条 RECORD 组成，每条 RECORD 中字段的顺序是 OUT 或 INOUT 参数对应的字段在前，最后返回 RETURN 语句对应的字段。

   SELECT 调用，就是执行普通的 SELECT 语句。对于存储过程，不能和其他任何常量、函数、存储过程等一并构成表达式使用，只能单独作为一个表达式出现在 SELECT 语句中。对于用户自定义函数，如果没有 OUT 或 INOUT 参数，可以和其他常量、变量、对象名如字段名等组合成表达式使用。带有 OUT 或 INOUT 参数的函数不可以参与表达式的计算。

2. ALTER PROCEDURE 语句用于修改存储过程的某些特征。如果要修改存储过程的内容，可以先删除原存储过程再重新创建。

   ALTER PROCEDURE 存储过程名[特征...]:

   - 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 表示子程序中不包含 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。

3. 存储过程的优点：
   - 由于存储过程不像解释执行的 SQL 语句那样在提出操作请求时才进行语法分析和优化工作，因而运行效率高；
   - 降低了客户机和服务器之间的通信量；
   - 方便实施企业规则。

4. 游标可以实现对数据库记录逐条处理，而不是整个结果集一起处理。

5. 游标操作：
   游标定义：DECLARE cursor_name CURSOR FOR select_statement;
   打开游标：OPEN cursor_name;
   取游标中的数据：FETCH cursor_name INTO var_name [, var_name]...
   关闭游标：CLOSE cursor_name;
   释放游标：DEALLOCATE cursor_name;

感悟：

通过本次实验，我对数据库编程中的存储过程的设计和使用方法、自定义函数的设计和使用方法及游标的设计、定义和使用方法有了更多了解，掌握了不同语句的运用场景；同时通过实例设计，对 CREATE PROCEDURE、CALL、CREATE FUNCTION、SELECT 语句的不同运用方式了解更加清楚。