

数据库系统 课程实验4

存储过程与事务处理

计科210X甘晴void 202108010XXX

目录

数据库系统 课程实验4
存储过程与事务处理

实验目的

实验环境

实验准备

表设计

初始数据

实验内容

4.1 存储过程实验

实验内容与要求

实验重点和难点

实验过程

(0) 模板：存储过程

(1) 无参数的存储过程

(2) 有参数的存储过程

(3) 有局部变量的存储过程

(4) 有输出参数的存储过程

(5) 修改存储过程

(6) 删除存储过程

4.2 自定义函数实验

实验内容与要求

实验重点和难点

实验过程

(0) 模板：函数

(1) 无参数的自定义函数

(2) 有参数的自定义函数

(3) 有局部变量的自定义函数

(4) 修改自定义函数

(5) 删除自定义函数

函数和过程小结

4.3 游标实验

实验内容与要求

实验重点和难点

实验过程

(0) 游标基础知识

(1) 普通游标

效果展示

<1> 游标定义与变量定义先后

<2> 定义必须在最前面

<3> 游标遍历结束处理

<4> 不需要显式释放游标空间

(2) REFCURSOR类型游标

(3) 带参数的游标

4.4 事务处理实验

实验内容与要求

实验重点和难点

实验过程

(0) 事务基础知识

(1) 数据库准备

(2) 设计转账过程p_transfer

(3) 开始转账

(4) 分析原因

参考文献

实验感悟

实验目的

掌握数据库存储过程的设计及使用方法，掌握自定义函数与游标的定义和使用方法；掌握利用存储过程进行事务处理相关操作的方法。

实验环境

DBMS: 8.0.33 MySQL Community Server - GPL

可视化: Navicat Premium 16.1.6

命令行: Navicat自带命令列

实验准备

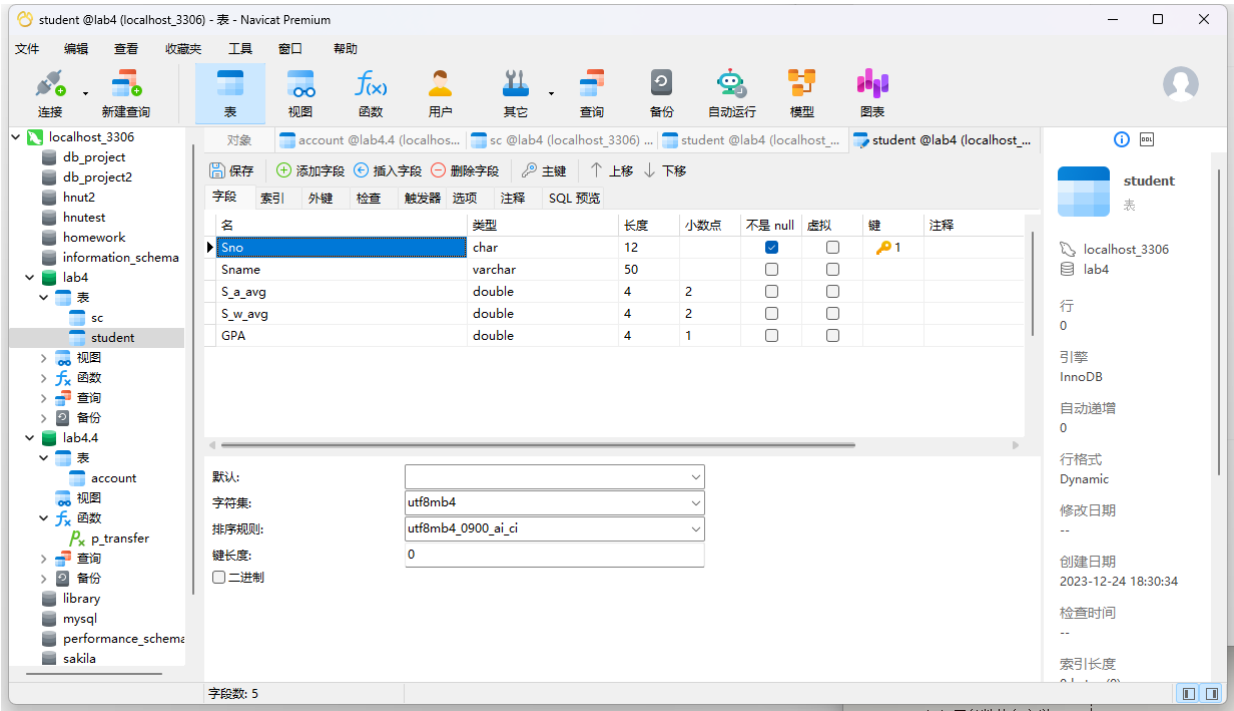
实验4.1-4.3使用我自己建立的数据库进行。实验4.4另外使用指导书提供的同款数据库进行，再述。

下面展示实验4.1-4.3的数据库。

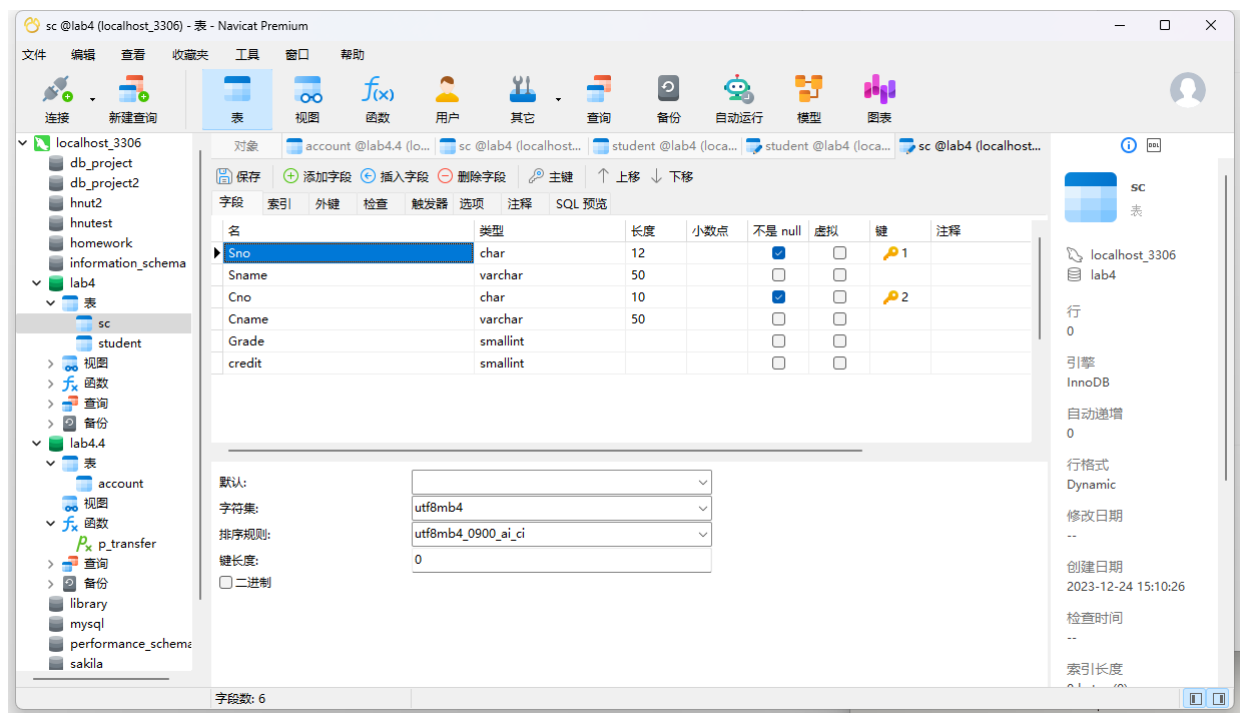
共有两张表，sc和student，是简单的学生课程分数模型。

表设计

student表设计：



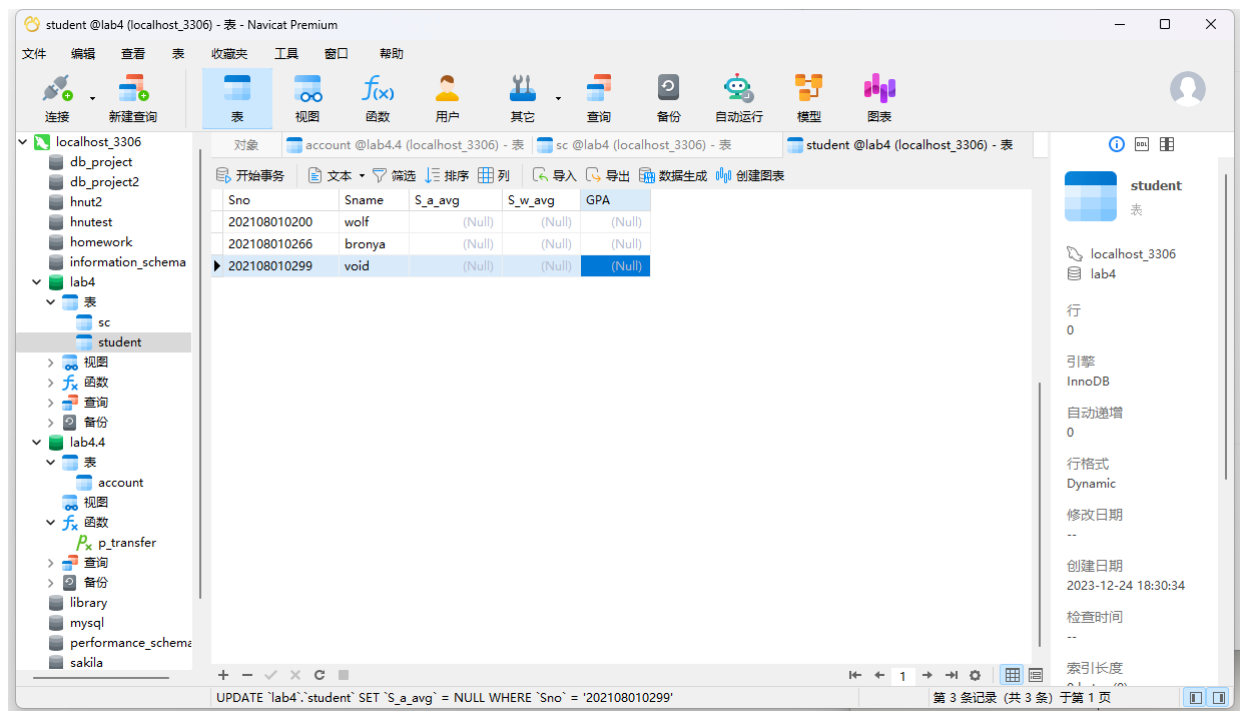
sc表设计：



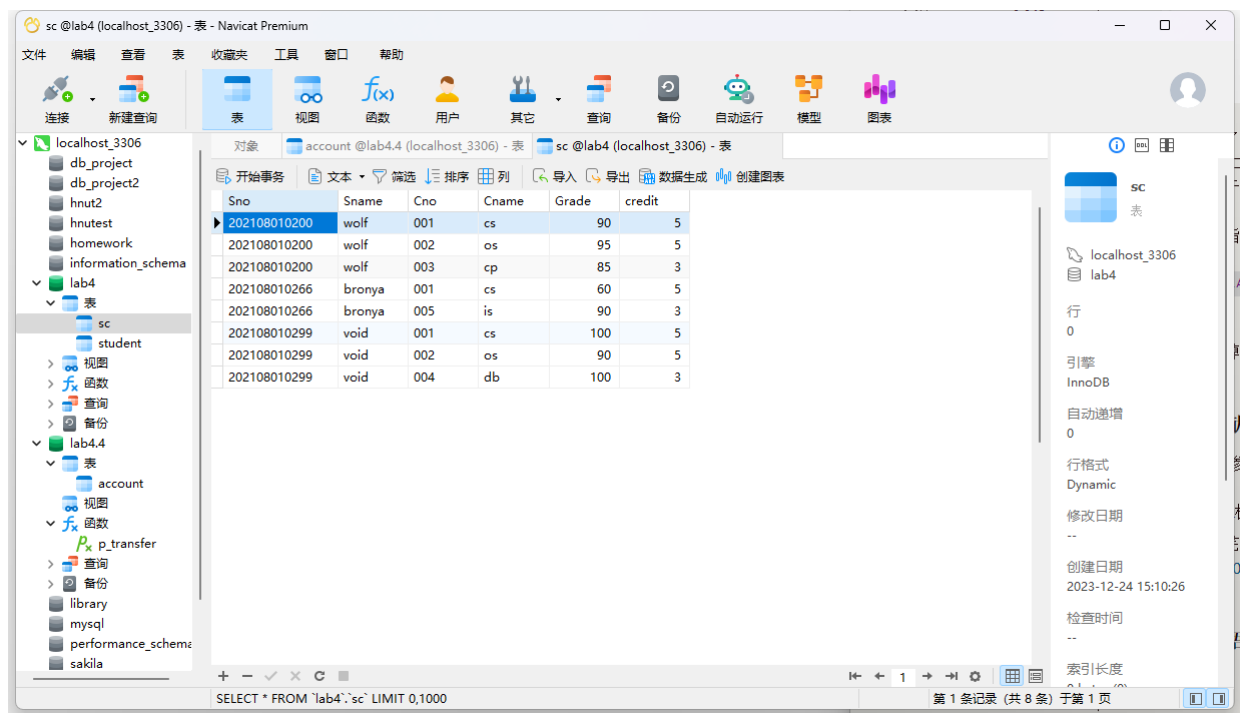
另外我自己给定了一些初始数据

初始数据

student表数据:



sc表数据:



实验内容

4.1 存储过程实验

实验内容与要求

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。

实验重点和难点

- 实验重点：存储过程的定义与运行。
- 实验难点：存储过程的参数传递方法。

实验过程

(0) 模板：存储过程

一个过程的模板如下

```

/*存储过程*/
CREATE PROCEDURE proc_example(

```

```

sname CHAR(20), #输入参数
out result REAL #如果要输出，这里写，否则这里不写
)
BEGIN
    /*局部变量声明*/
    DECLARE c1 CHAR(9);
    DECLARE f1 FLOAT;
    /*函数体*/
    SELECT sc.grade INTO result #这种方式为返回值赋值
    FROM student
    WHERE student.sname = sname;
END;

```

(1) 无参数的存储过程

定义一个无参数的存储过程，更新所有学生的算术平均成绩。

```

CREATE PROCEDURE Proc4_1_1_calculate_Avg()
BEGIN
    UPDATE student
    SET S_a_avg = (
        SELECT AVG(Grade)
    FROM SC
    WHERE student.Sno = sc.Sno
    );
END;

```

调用函数前，学生成绩表中无信息

sc@lab4 (localhost_3306) - 表					student @lab4 (localhost_3306) - 表			
Sno	Sname	Cno	Cname	Grade	Sno	Sname	S_a_avg	S_w_avg
202108010200	wolf	001	cs	90	202108010200	wolf	(Null)	(Null)
202108010200	wolf	002	os	95	202108010266	boronya	(Null)	(Null)
202108010200	wolf	003	cp	85	202108010299	void	(Null)	(Null)
202108010266	bronya	001	cs	60				
202108010266	bronya	005	is	90				
202108010299	void	001	cs	100				
202108010299	void	002	os	90				
202108010299	void	004	db	100				

调用函数前

调用函数之后，学生算术平均成绩被更新

sc@lab4 (localhost_3306) - 表					student @lab4 (localhost_3306) - 表			
Sno	Sname	Cno	Cname	Grade	Sno	Sname	S_a_avg	S_w_avg
202108010200	wolf	001	cs	90	202108010200	wolf	90	(Null)
202108010200	wolf	002	os	95	202108010266	boronya	75	(Null)
202108010200	wolf	003	cp	85	202108010299	void	97	(Null)
202108010266	bronya	001	cs	60				
202108010266	bronya	005	is	90				
202108010299	void	001	cs	100				
202108010299	void	002	os	90				
202108010299	void	004	db	100				

调用函数计算后

(2) 有参数的存储过程

定义一个有参数的存储过程，更新某一个学生的算术平均成绩。

```
CREATE PROCEDURE Proc4_1_2_Calculate_target_Avg(
    target varchar(50)
)
BEGIN
    UPDATE student
    SET S_a_avg = (
        SELECT AVG(Grade)
    FROM SC
    WHERE student.Sno = sc.Sno AND
        student.Sname = target
    );
END;
```

使用如下指令调用函数，或使用可视化界面直接调用函数

```
Proc4.1.2_Calculate_target_Avg('bronya')
```

调用函数之后，只有特定的学生算术平均成绩被更新

The screenshot shows a database management tool interface. On the left, the 'sc' table is displayed with columns: Sno, Sname, Cno, Cname, Grade. It contains 10 rows of data. On the right, the 'student' table is displayed with columns: Sno, Sname, S_a_avg, S_w_avg. It contains 3 rows of data. The row for 'bronya' (Sno: 202108010266) is highlighted, showing an updated S_a_avg of 75. A red box highlights the 'student' table title, and red text on the right says '调用函数后 只有目标对象发生了更新' (After calling the function, only the target object was updated).

Sno	Sname	Cno	Cname	Grade
202108010200	wolf	001	cs	90
202108010200	wolf	002	os	95
202108010200	wolf	003	cp	85
202108010266	bronya	001	cs	60
202108010266	bronya	005	is	90
202108010299	void	001	cs	100
202108010299	void	002	os	90
202108010299	void	004	db	100

Sno	Sname	S_a_avg	S_w_avg
202108010200	wolf	(Null)	(Null)
202108010266	bronya	75	(Null)
202108010299	void	(Null)	(Null)

(3) 有局部变量的存储过程

定义一个有局部变量的存储过程，更新某个学生的对学分加权平均成绩。

修改sc表，新增一列credit（学分），并赋予相应的课程相应的学分。

这里使用局部变量target_sno存储target_sname对应的学生学号。

```
CREATE PROCEDURE Proc4_1_3_Calculate_target_weighted_Avg(  
    target_sname varchar(50)  
)  
BEGIN  
    DECLARE target_sno char(12);  
    SELECT student.sno INTO target_sno  
        FROM student  
        WHERE student.sname = target_sname;  
    UPDATE student  
        SET S_w_avg = (  
            SELECT SUM(Grade*credit) / SUM(credit)  
            FROM SC  
            WHERE student.Sno = sc.Sno AND sc.sno = target_sno  
        );  
END;
```


使用如下指令调用函数，或使用可视化界面直接调用函数

```
Proc4.1.3_Calculate_target_weighted_Avg('bronya')
```

调用函数之后，只有特定的学生加权平均成绩被更新

Sno	Sname	Cno	Cname	Grade	credit
202108010200	wolf	001	cs	90	5
202108010200	wolf	002	os	95	5
202108010200	wolf	003	cp	85	3
202108010266	bronya	001	cs	60	5
202108010266	bronya	005	is	90	3
202108010299	void	001	cs	100	5
202108010299	void	002	os	90	5
202108010299	void	004	db	100	3

Sno	Sname	S_a_avg	S_w_avg
202108010200	wolf	0	(Null)
202108010266	bronya	0	71.25
202108010299	void	0	(Null)

调用函数之后

(4) 有输出参数的存储过程

在刚刚的基础上，再输出该学生的加权平均成绩。

只要在参数列表加上再加上返回值即可

```
out result REAL
```

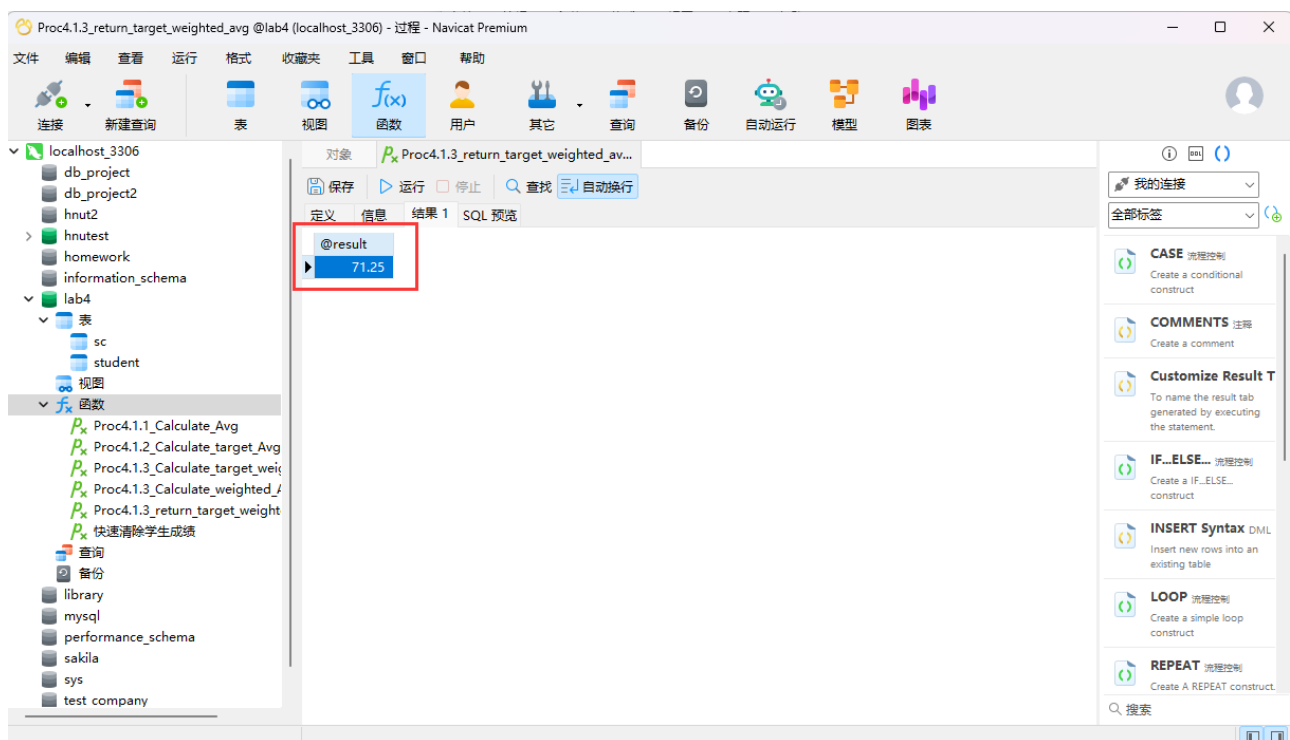
代码如下

```

CREATE PROCEDURE Proc4_1_4_return_target_weighted_avg(
    target_sname varchar(50),
    out result REAL
)
BEGIN
    DECLARE target_sno char(12);
    SELECT student.sno INTO target_sno
        FROM student
        WHERE student.sname = target_sname;
    SELECT SUM(Grade*credit) / SUM(credit) INTO result
        FROM SC,student
        WHERE student.Sno = sc.Sno AND sc.sno = target_sno;
END;

```

使用可视化界面调用该函数，结果如下。



（5）修改存储过程

在MySQL中，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，再以相同的命名创建新的存储过程；如果要修改存储过程的名称，可以先删除原存储过程，再以不同的命名创建新的存储过程。

MySQL 中修改存储过程的语法格式如下：

```

ALTER PROCEDURE 存储过程名 [ 特征 ... ]

```

特征指定了存储过程的特性，支持的操作有：

- 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 表示定义者有对存储过程中对象的访问权限，调用者就能够执行。
- INVOKER 表示调用者必须拥有存储过程中的对象的访问权限，才能执行。
- COMMENT 'string' 表示注释信息。

尝试：

【注意】似乎修改和删除的对象都不能是我们上面使用DEFINER方法生成的过程，因此我现在重新生成一个过程，并为其添加注释信息，然后删除。

```
CREATE PROCEDURE Proc4_1_5_example()  
BEGIN  
    UPDATE student  
    SET S_a_avg = (  
        SELECT AVG(Grade)  
    FROM SC  
    WHERE student.Sno = sc.Sno  
    );  
END;
```

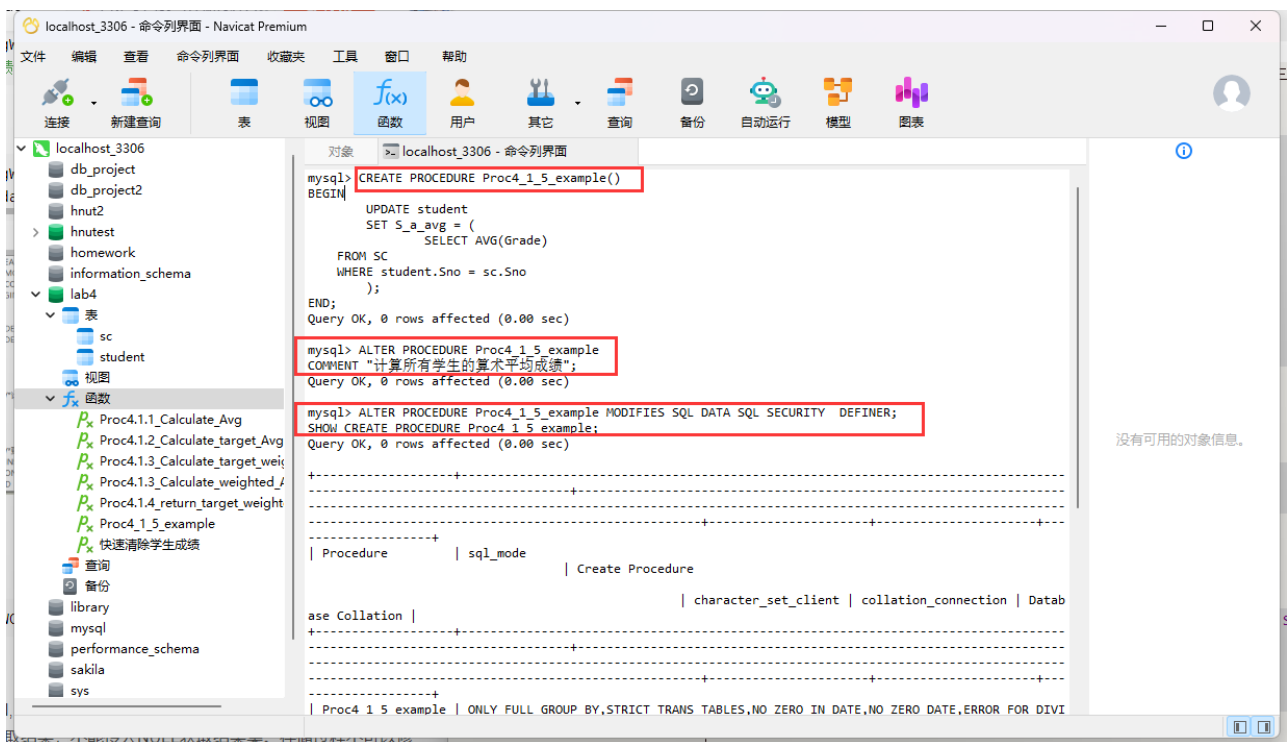
为存储过程添加注释信息：

```
ALTER PROCEDURE Proc4_1_5_example  
COMMENT "计算所有学生的算术平均成绩";
```

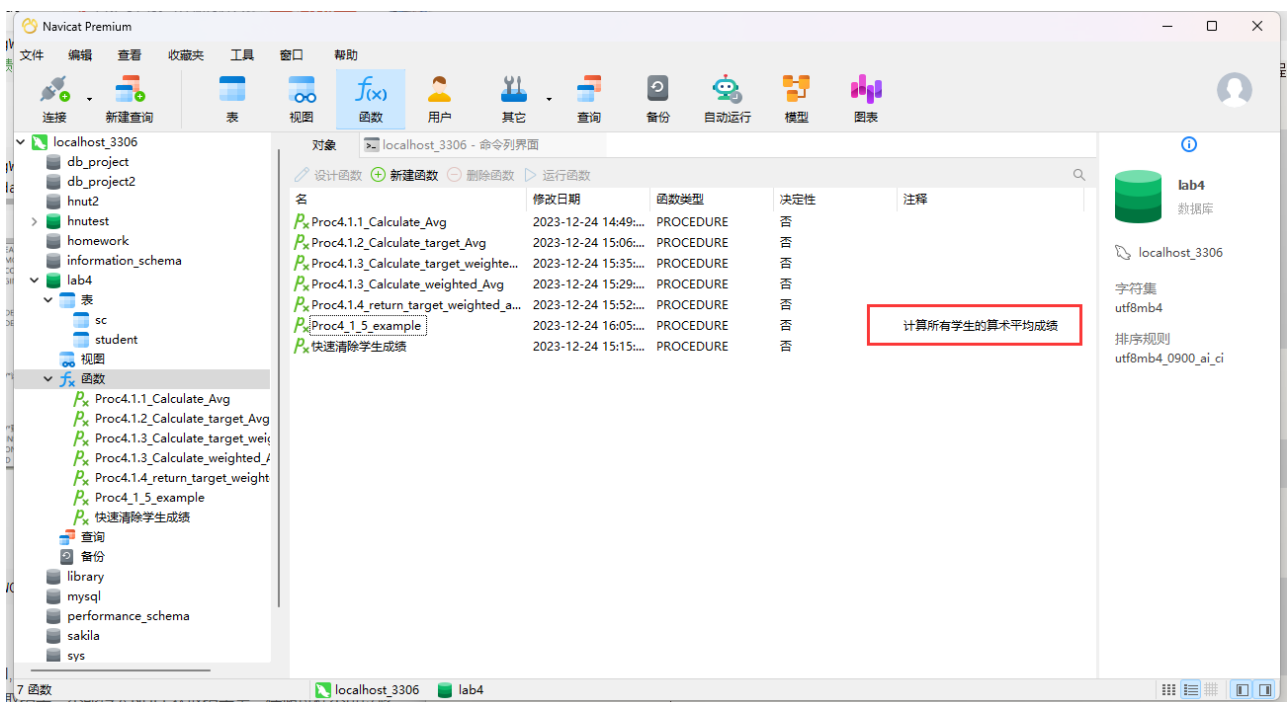
指定只有定义者权限：

```
ALTER PROCEDURE Proc4_1_5_example MODIFIES SQL DATA SQL SECURITY  
DEFINER;  
SHOW CREATE PROCEDURE Proc4_1_5_example;
```

命令如下



结果如下，可见其注释信息：



(6) 删除存储过程

DROP PROCEDURE Proc4_1_5_example;

4.2 自定义函数实验

实验内容与要求

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 和编程规范，规范设计自定义函数。

实验重点和难点

- 实验重点：自定义函数的定义与运行。
- 实验难点：自定义函数的参数传递方法。

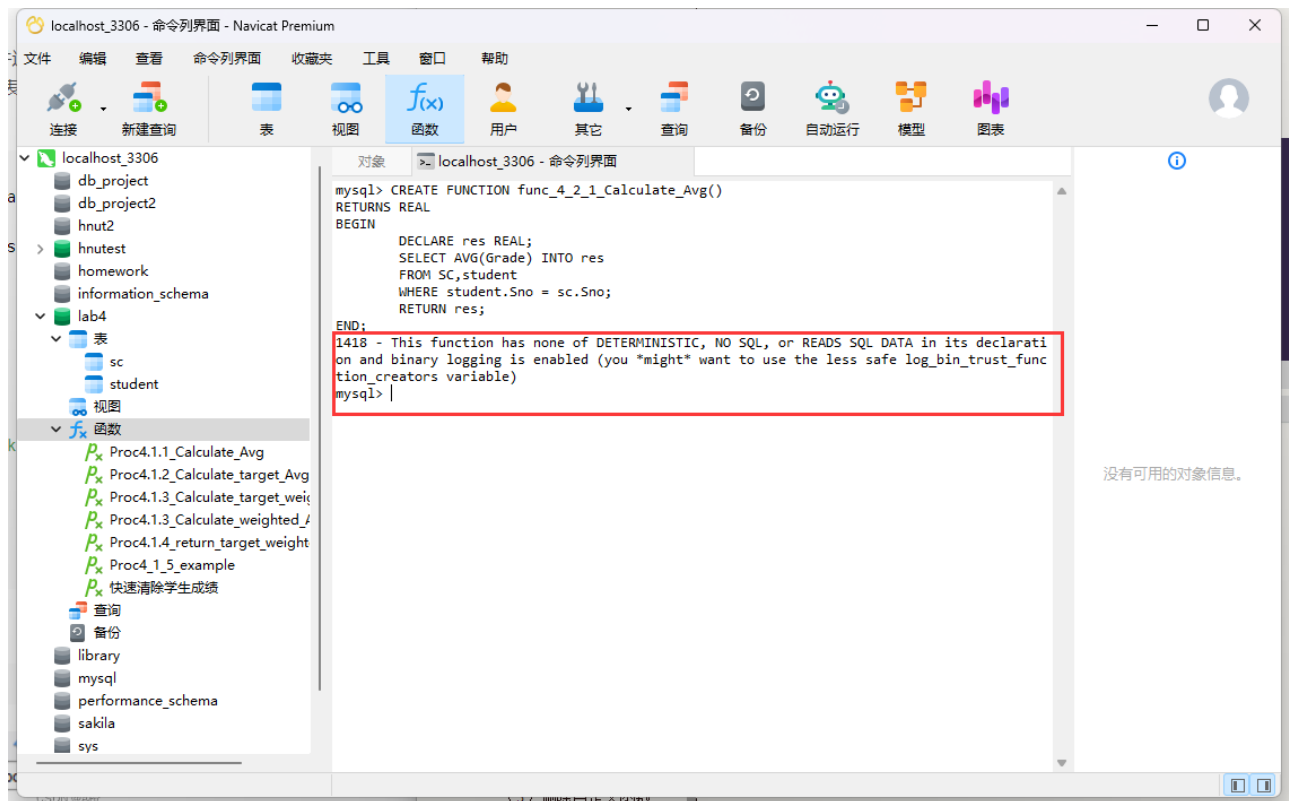
实验过程

（0）模板：函数

一个函数的模板如下

```
set global log_bin_trust_function_creators=TRUE;
/*存储过程*/
CREATE FUNCTION func_example(
    sname CHAR(20), #输入参数
)
RETURNS REAL      #显式指定返回类型
BEGIN
    /*局部变量声明*/
    DECLARE c1 CHAR(9);
    DECLARE f1 FLOAT;
    /*函数体*/
    SELECT sc.grade INTO f1 #这种方式为返回值赋值
    FROM student
    WHERE student.sname = sname;
    RETURN f1; #显式返回
END;
```

直接定义函数会出现如下问题

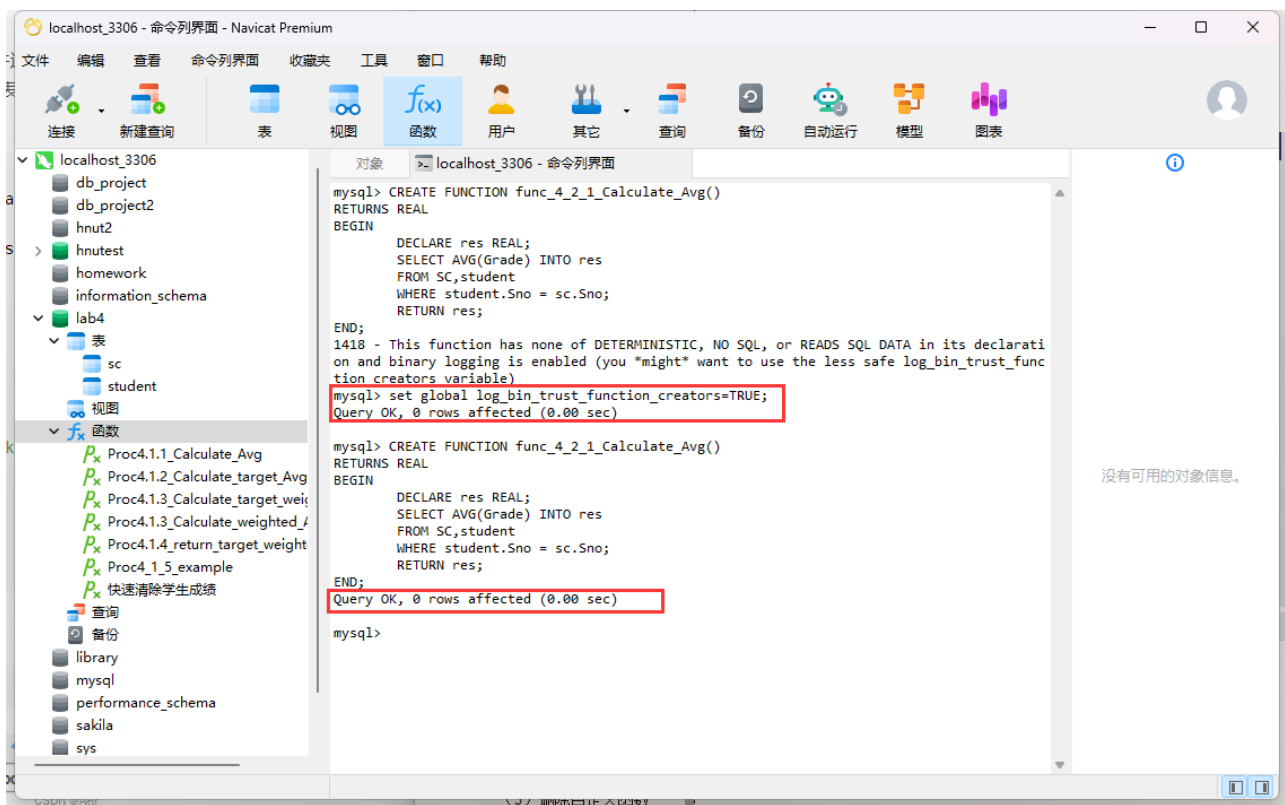


这是由于MYSQL启用了二进制日志，需要修改`log_bin_trust_function_creators`为`TRUE`，表示当信任存储函数创建者，不会创建写入二进制日志引起不安全事件的存储函数。

使用这个

```
set global log_bin_trust_function_creators=TRUE;
```

之后可以看见函数就可以正常创建了。（这个使用一次即可一直生效）



(1) 无参数的自定义函数

定义一个无参数的函数，获取所有学生所有科目的算术平均成绩。

```
CREATE FUNCTION func_4_2_1_calculate_Avg()
RETURNS REAL
BEGIN
    DECLARE res REAL;
    SELECT AVG(Grade) INTO res
    FROM SC,student
    WHERE student.Sno = sc.Sno;
    RETURN res;
END;
```

这个函数可以实现函数的效果，获取值并返回所有学生所有科目的算术平均成绩。

我如果不仅想要获取信息，也要造成改变，实现4.1（1）的内容，是否可以呢。

如下：

更新所有学生的算术平均成绩，获取所有学生所有科目的算术平均成绩。

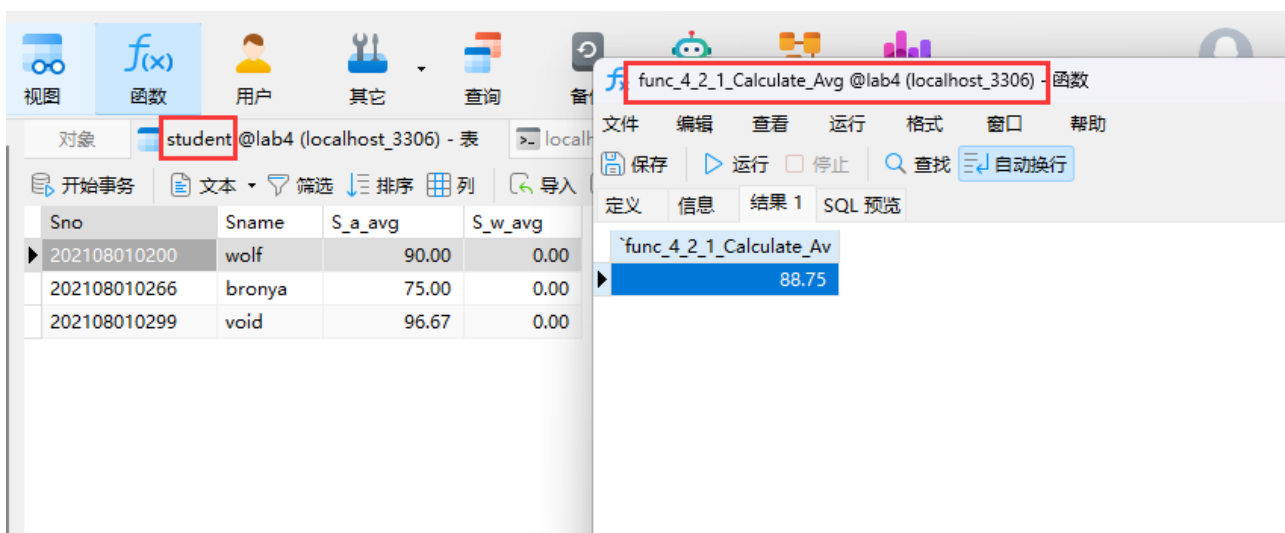
```
CREATE FUNCTION func_4_2_1_calculate_Avg()
RETURNS REAL
```

```

BEGIN
    DECLARE res REAL;
    UPDATE student
        SET S_a_avg = (
            SELECT AVG(Grade)
            FROM SC
            WHERE student.Sno = sc.Sno
        );
    SELECT AVG(Grade) INTO res
    FROM SC,student
    WHERE student.Sno = sc.Sno;
    RETURN res;
END;

```

运行截图如下：



这次不仅可以返回值，也可以同时修改student表中的内容。

(2) 有参数的自定义函数

更新某一个学生的算术平均成绩，并返回这个结果

```

CREATE FUNCTION func_4_2_2_calculate_target_Avg(
    target varchar(50)
)
RETURNS REAL
BEGIN
    DECLARE res REAL;
    UPDATE student
        SET S_a_avg = (

```

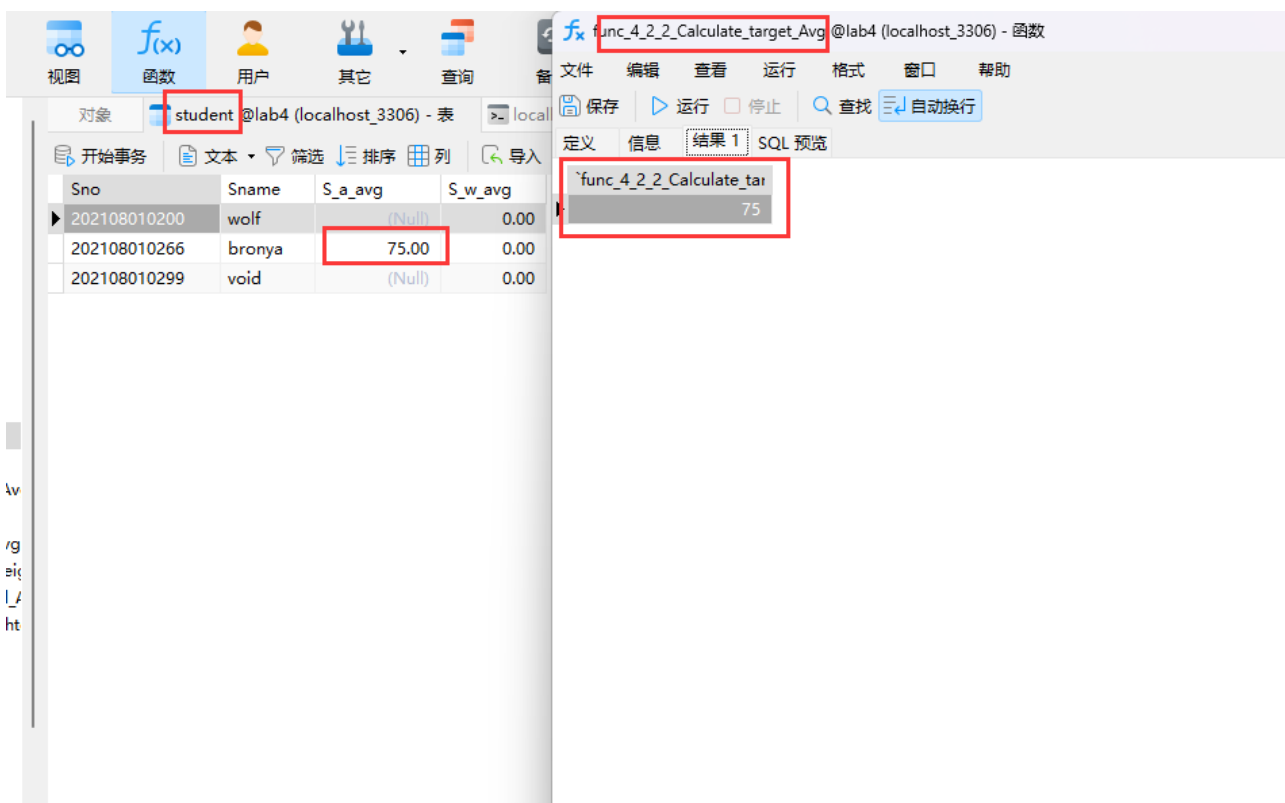


```

SELECT AVG(Grade)
FROM SC
WHERE student.Sno = sc.Sno AND
      student.Sname = target
);
SELECT AVG(Grade) INTO res
FROM SC,student
WHERE student.Sno = sc.Sno AND
      student.Sname = target;
RETURN res;
END;

```

运行结果如下：



Sno	Sname	S_a_avg	S_w_avg
202108010200	wolf	(Null)	0.00
202108010266	bronya	75.00	0.00
202108010299	void	(Null)	0.00

(3) 有局部变量的自定义函数

定义一个有局部变量的自定义函数，更新某个学生的对学分加权平均成绩，并返回这个结果。

```

CREATE FUNCTION func_4_2_3_Calculate_target_weighted_Avg(
    target_sname varchar(50)
)
RETURNS REAL
BEGIN

```

```

DECLARE res REAL;
DECLARE target_sno char(12);
SELECT student.sno INTO target_sno
    FROM student
    WHERE student.sname = target_sname;
UPDATE student
    SET S_w_avg = (
        SELECT SUM(Grade*credit) / SUM(credit)
    FROM SC
    WHERE student.Sno = sc.Sno AND sc.sno = target_sno
    );
SELECT SUM(Grade*credit) / SUM(credit) INTO res
    FROM SC,student
    WHERE student.Sno = sc.Sno AND sc.sno = target_sno;
RETURN res;
END;

```

运行截图如下：

The screenshot shows the SQL Server Enterprise Manager interface. On the left, a table named 'student' is displayed with the following data:

Sno	Sname	S_a_avg	S_w_avg
202108010200	wolf	0.00	(Null)
202108010266	bronya	0.00	71.25
202108010299	void	0.00	(Null)

On the right, a window titled 'func_4_2_3_Calculate_target_weighted_Avg @lab4 (localhost_3306) - 函数' is open, showing the execution result of the function 'func_4_2_3_Calculate_tar' as 71.25.

(4) 修改自定义函数

```
/*添加注释*/  
ALTER FUNCTION func_example COMMENT "欢迎来到湖南大学";  
/*修改函数为定义者权限*/  
ALTER FUNCTION func_example MODIFIES SQL DATA SQL SECURITY  
DEFINER;
```

这个跟过程类似。

（5）删除自定义函数

```
DROP FUNCTION func_example;
```

函数和过程小结

函数和存储过程有很多相似点，但也有很多不同的地方。

根据我之前的经验。在PASCAL语言中的“过程”和“函数”。而在c++语言中，过程的实现是返回值为“void”的函数。“过程”和“函数”几乎是可以相互替代的。

而在MYSQL语言中，过程和函数还有一些区别：

- 存储过程只能单独使用CALL函数调用，而函数在SELECT语句中，像内置函数一样调用。
- 存储过程可以有IN, OUT参数，返回多个值。而函数只能返回一个值。
- 存储过程可以调用函数，函数不能使用存储过程。

这使得过程和函数之间的选择，在有的时候还是不同的。

4.3 游标实验

实验内容与要求

游标定义、游标使用。掌握各种类型游标的特点、区别与联系。

实验重点和难点

- 实验重点：游标的定义和使用。
- 实验难点：游标类型。

实验过程

（0）游标基础知识

游标的设计是一种数据缓冲区的思想，用来存放SQL语句执行的结果。游标是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。尽管游标能遍历结果中的所有行，但一次只指向一行。游标的作用就是用于对查询数据库所返回的记录进行遍历，以便进行相应的操作。

游标具有三个属性：

- 不敏感（**Asensitive**）：数据库可以不复制结果集
- 只读（**Read only**）
- 不滚动（**Nonscrollable**）：游标只能向一个方向前进，并且不可以跳过任何一行数据。

游标的主要缺点是性能不高。

游标的开销与游标中进行的操作相关，如果在游标中进行复杂的操作，开销会非常高。如果采用面向集合的SQL语句，扫描成本为 $O(N)$ ；但如果采用面向集合的SQL语句的扫描成本为 $O(N*N)$ ，则使用游标有可能会带来性能上的提升。游标只能一行一行操作。在数据量大的情况下，速度过慢。数据库大部分是面对集合的，业务会比较复杂，而游标使用会有死锁，影响其他的业务操作，不可取。当数据量大时，使用游标会造成内存不足现象。

个人理解：游标类似于c++的迭代器（这下看懂了），但是这个迭代器只能单向逐个迭代。用于每次处理一个数据，如果SELECT的返回结果不是一个值而是一个向量的话，使用游标配合loop循环可以逐个处理这些结果。这个功能还是很强大的。

接下来是游标的使用方法

```
DECLARE cursor_name CURSOR FOR select_statement #定义游标
OPEN cursor_name;                               #打开游标
FETCH cursor_name INTO var_name [, var_name]... #取游标数据并向后移动游标
CLOSE cursor_name;                              #关闭游标
DEALLOCATE cursor_name;                         #释放游标
#【注意】mysql中不要写这句DEALLOCATE
```

关键点就在游标的定义和取游标数据这两步，接下来实验过程中我会涉及。

（1）普通游标

定义一个游标，使用游标的方法计算班级某个同学的课程对学分加权平均绩点。

【为了贴近实际，特别使用湖南大学同款绩点系统】

证明 Certificate

兹证明湖南大学教务处采用下列学生成绩评分系统。

This is to certify that in the grading system adopted by the Academic Affairs Office,
Hunan University, the following equivalences apply:

绩点 GP	百分制 100-mark grading	平均成绩 Average value	五级分制 Five-point scale	五级制 Five-level scale	两级制 Two-level scale
4.0	[90,100]	95	A	优秀 Excellent	
3.7	[85,90)	87	A ⁻		
3.3	[80,85)	82	B ⁺		
3.0	[75,80)	77	B	良好 Very Good	合格 Satisfactory
2.7	[70,75)	72	B ⁻		
2.3	[67,70)	68	C ⁺		
2.0	[65,67)	65	C	中等 Good	
1.7	[62,65)	63	C ⁻		
1.0	[60,62)	60	D	及格 Pass	
0	[0,60)	30	F	不及格 Fail	不合格 Unsatisfactory

备注：如果计算平均成绩，则依照平均成绩取值。免修课程计算平均成绩时按60分计算。

NB: In the case of the 100-mark grading system, the average scores are calculated in accordance with the average values. The average scores of exempted courses are calculated as 60 out of 100.

湖南大学教务处
Academic Affairs Office, Hunan University

To authenticate this document, please contact the Academic Affairs Office, Hunan University
Tel & Fax: 86-731-88823270 Website: <http://jwc.hnu.edu.cn>
Academic Affairs Office, Room 104, Executive Building, Hunan University, Changsha, P. R. China 410082

```
CREATE PROCEDURE PROC_CURSOR(  
    target_sname VARCHAR(50),  
    out result REAL)
```

```

BEGIN
    DECLARE total_credit INT;
    DECLARE total_GPA FLOAT;
    DECLARE temp_grade INT;
    DECLARE temp_credit INT;
    DECLARE temp_GPA FLOAT;
    DECLARE done INT DEFAULT 0;
    DECLARE mycursor CURSOR FOR
        SELECT Grade,credit
        FROM sc
        WHERE sc.sname = target_sname;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN mycursor;
    SET total_GPA = 0;
    SET total_credit = 0;
    my_loop:LOOP
        FETCH mycursor INTO temp_grade,temp_credit;
        IF done THEN LEAVE my_loop;
        END IF;
        IF temp_grade BETWEEN 90 AND 100 THEN
            SET temp_GPA = 4.0;
        ELSEIF temp_grade BETWEEN 85 AND 89 THEN
            SET temp_GPA = 3.7;
        ELSEIF temp_grade BETWEEN 80 AND 84 THEN
            SET temp_GPA = 3.3;
        ELSEIF temp_grade BETWEEN 75 AND 79 THEN
            SET temp_GPA = 3.0;
        ELSEIF temp_grade BETWEEN 70 AND 74 THEN
            SET temp_GPA = 2.7;
        ELSEIF temp_grade BETWEEN 67 AND 69 THEN
            SET temp_GPA = 2.3;
        ELSEIF temp_grade BETWEEN 65 AND 66 THEN
            SET temp_GPA = 2.0;
        ELSEIF temp_grade BETWEEN 62 AND 64 THEN
            SET temp_GPA = 1.7;
        ELSEIF temp_grade BETWEEN 60 AND 61 THEN
            SET temp_GPA = 1.0;
        ELSEIF temp_grade BETWEEN 0 AND 59 THEN
            SET temp_GPA = 0;
        END IF;
        SET total_GPA = total_GPA + temp_GPA * temp_credit;
        SET total_credit = total_credit + temp_credit;
    END LOOP;
END

```

```

END LOOP;

CLOSE mycursor;

# DEALLOCATE mycursor; #Mysql中不写这句

SET result = total_GPA/total_credit;

END;

```

效果展示

The screenshot shows a database management tool interface. On the left, a table named 'sc @lab4 (localhost_3306) - 表' is displayed. The table has columns: Sno, Sname, Cno, Cname, Grade, and credit. The data is as follows:

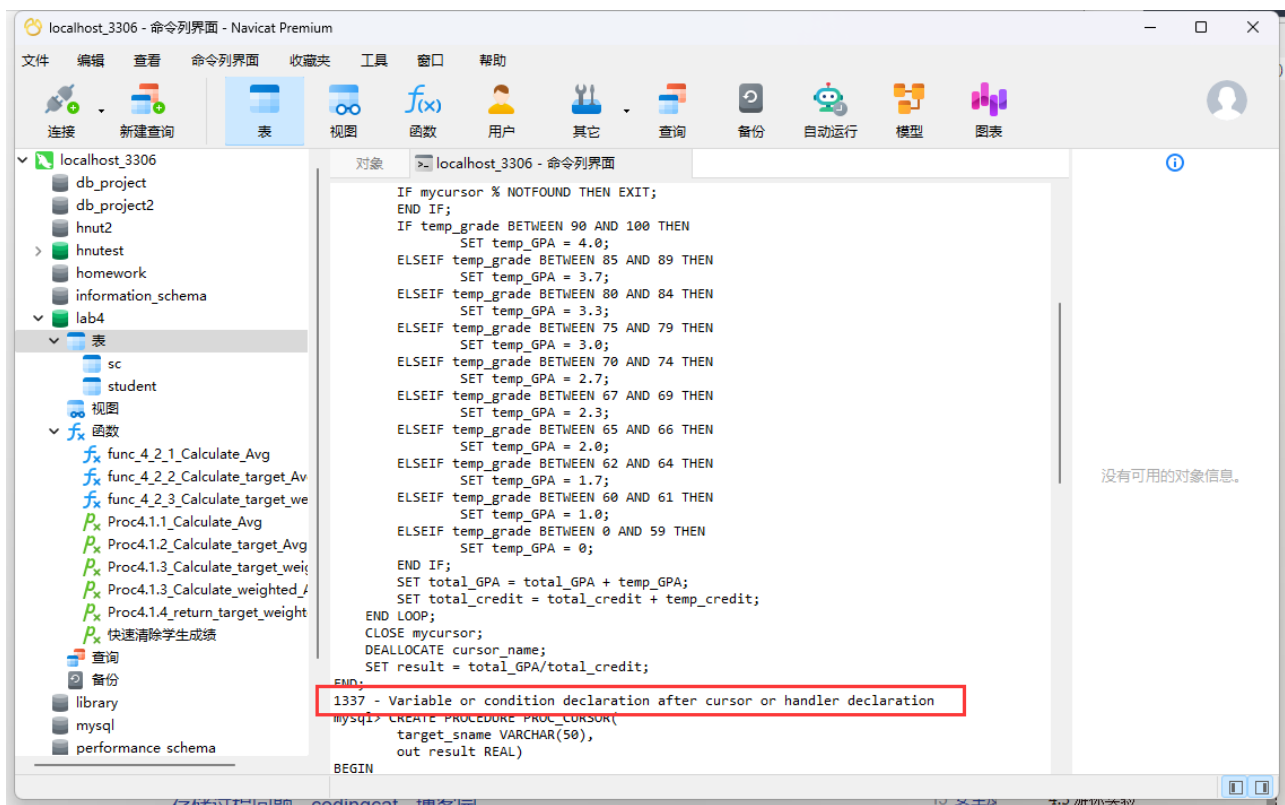
Sno	Sname	Cno	Cname	Grade	credit
202108010200	wolf	001	cs	90	5
202108010200	wolf	002	os	95	5
202108010200	wolf	003	cp	85	3
202108010266	bronya	001	cs	60	5
202108010266	bronya	005	is	90	3
202108010299	void	001	cs	100	5
202108010299	void	002	os	90	5
202108010299	void	004	db	100	3

On the right, a window titled 'PROC_CURSOR @lab4 (localhost_3306) - 过程' shows the execution result of a stored procedure. The result is a variable '@result' with the value '3.930000066757202'.

写这个的时候遇到了很多的问题，我把它记录下来：

<1> 游标定义与变量定义先后

游标必须定义在所有定义定义完之后的最后面，否则会出错，报错像这样：



<2> 定义必须在最前面

游标定义出错，严格来讲，这不是游标定义出错，而是定义出错。

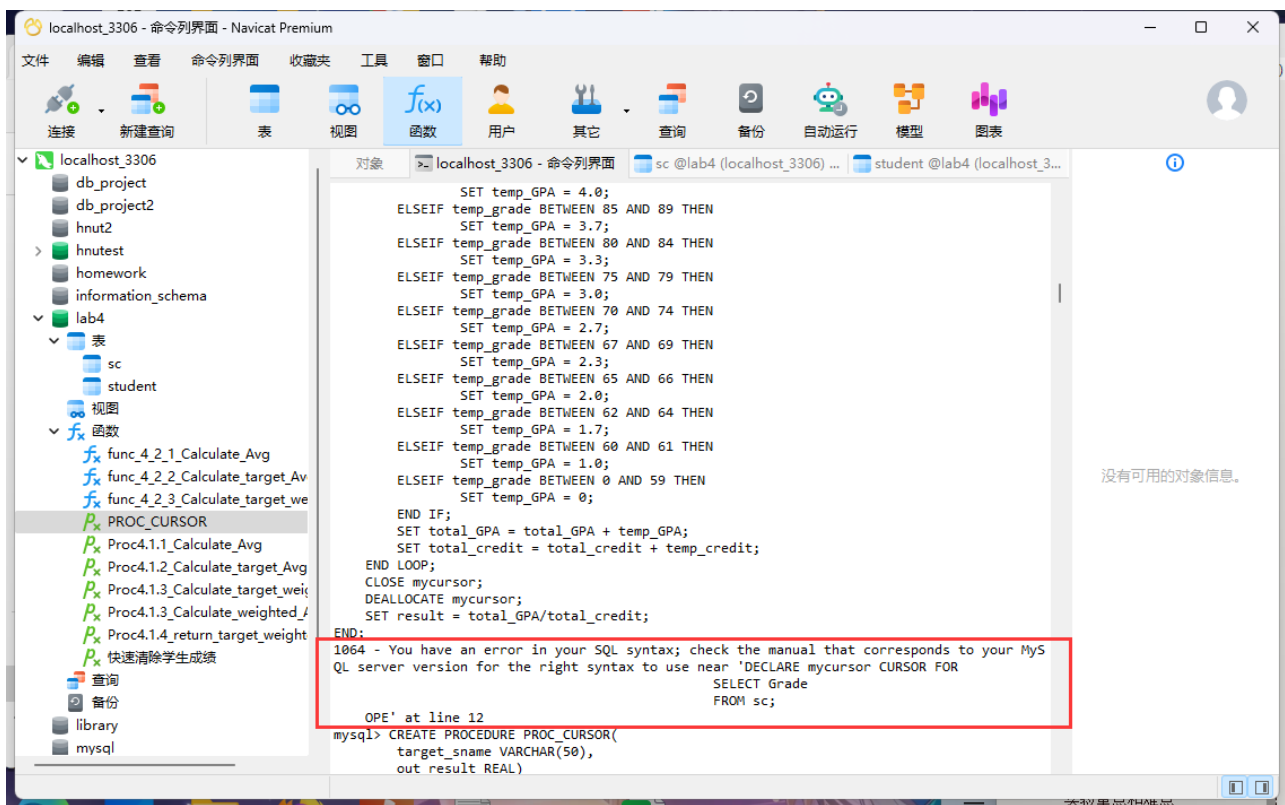
在MySQL中，要求定义全部紧跟BEGIN之后，而不是像c++一样可以随用随定义，这一点倒有点像PASCAL要先定义在函数/过程前面，再使用。不过MySQL可以定义在函数/过程体中，但是所有定义都要在任何一句其它语句之前。

简单来说，声明必须紧跟在 **BEGIN** block 之后，中间不能跟SET。

出错报错像这样：

```
1064 - You have an error in your SQL syntax; check the manual that  
corresponds to your MySQL server version for the right syntax to  
use near 'DECLARE mycursor CURSOR FOR  
                SELECT Grade  
                FROM sc;  
OPE' at line 12
```

截图如下：



这真的很令人绝望，浪费了半个小时找不到错误，我也试着查manual手册，结果发现手册上写的真的很少，几乎是什么都没有。万幸，直到我在网上遇到了一个相同的问题。

CURSOR错误解决: <https://www.coder.work/article/493073>

<3> 游标遍历结束处理

这也是一个大坑，书上写的不适用于mysql数据库。

解决方法是用这个结构，相当于构造了一个标记flag用来表示遍历结束。

(A橙是用的这种方法，另外一位貌似有问题)

```
DECLARE done INT DEFAULT 0;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
my_loop:LOOP
    FETCH mycursor INTO temp_grade,temp_credit;
    IF done THEN LEAVE my_loop;
    END IF; #若结束，就跳出循环
# .....
END LOOP;
```

另外这里不能用EXIT，第6版教材P255页使用EXIT退出不适用于mysql数据库。

书上的`EXIT WHEN mycursor % NOTFOUND;`也不适用。

<4> 不需要显式释放游标空间

```
# DEALLOCATE mycursor; #Mysql中不写这句
```

（2）REFCURSOR类型游标

MYSQL中游不能使用动态游标。

REF CURSOR是动态游标，一般的CURSOR是静态游标。一般的CURSOR在定义时结果集的类型就是确定的，不可以再更改，而动态游标可以打开任何形式的结果集，取出任何类型的记录数据。在MYSQL中只能使用静态游标，定义时的SELECT语句必须确定，不可以再更改。

（3）带参数的游标

Mysql中游标不可以携带参数。

4.4 事务处理实验

实验内容与要求

掌握利用存储过程进行事务处理相关操作的方法。

实验重点和难点

- 实验重点：创建存储过程，设置事务管理。
- 实验难点：利用存储过程进行事务处理的相关操作。

实验过程

（0）事务基础知识

定义：数据库事务(transaction)是访问并可能操作各种数据项的一个数据库操作序列，这些操作要么全部执行,要么全部不执行，是一个不可分割的工作单位。

事务具有ACID特性：

- 原子性(Atomicity): 事务中的全部操作在数据库中是不可分割的, 要么全部完成, 要么全部不执行。
- 一致性(Consistency): 几个并行执行的事务, 其执行结果必须与按某一顺序 串行执行的结果相一致。
- 隔离性(Isolation): 事务的执行不受其他事务的干扰, 事务执行的中间结果对其他事务必须是透明的。
- 持久性(Durability): 对于任意已提交事务, 系统必须保证该事务对数据库的改变不被丢失, 即使数据库出现故障。

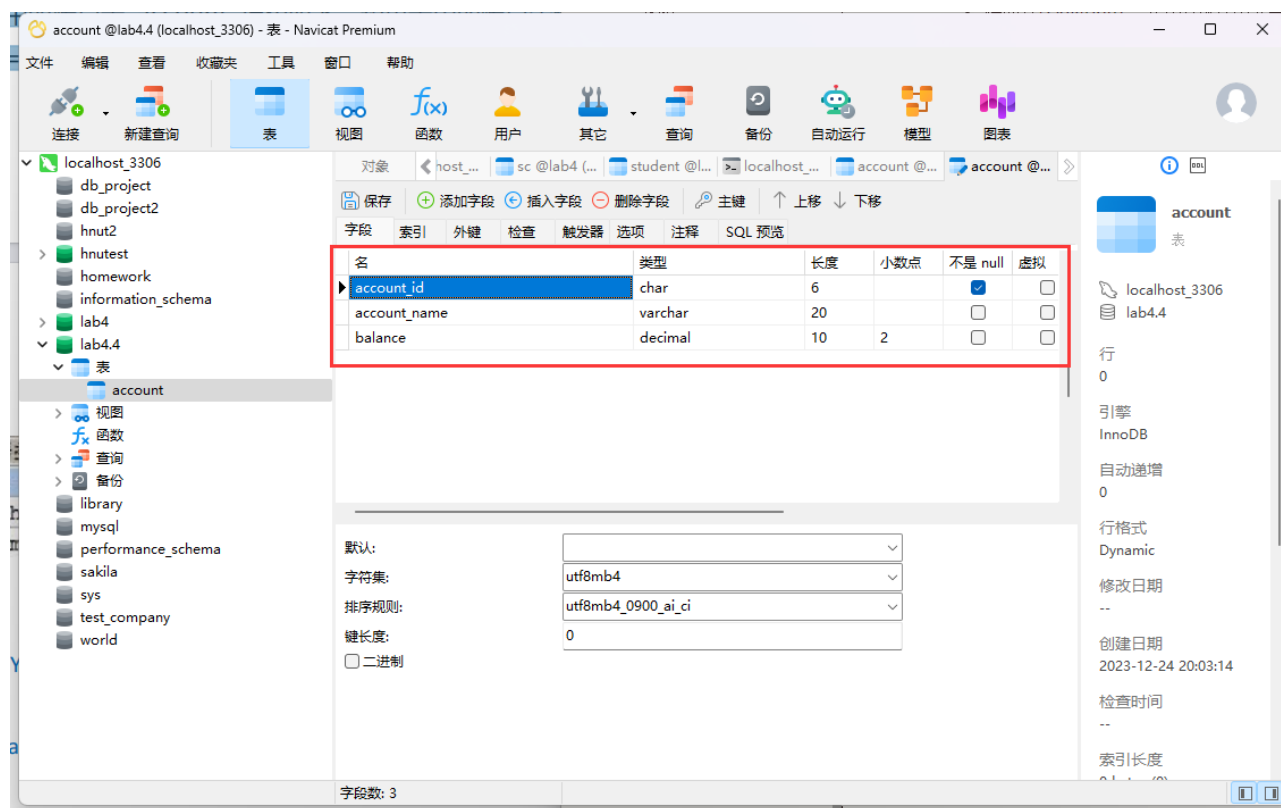
接下来在具体实验中加以说明。(本次实验采用指导书给出的银行转账案例)

(1) 数据库准备

以银行账户之间的转账为例, 首先构造某银行数据库 bank 中的帐户表 account表, 约束表中的帐户余额不能为负值: CHECK(balance>=0)。

```
CREATE TABLE account(
account_id CHAR(6) PRIMARY KEY, account_name VARCHAR(20), balance
DECIMAL(10,2) CHECK(balance>=0)
);
```

建表如下:



(2) 设计转账过程 `p_transfer`

设计一个存储过程 `p_transfer` 用于转账，转账中进行事务控制，出现 SQL 异常时能撤消事务，保证帐户数据的一致性。

指导书上提供的有点小问题，修改如下可以用。

```
# DROP PROCEDURE IF EXISTS p_transfer    #这一句不要加
CREATE PROCEDURE p_transfer(
    from_account CHAR(6),
    to_account CHAR(6),
    amount DECIMAL(10,2)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION ROLLBACK;
    START TRANSACTION;
        UPDATE account
            SET balance = balance + amount
            WHERE account_id = to_account;
        UPDATE account
            SET balance = balance - amount
            WHERE account_id = from_account;
    COMMIT;
END;
```

(3) 开始转账

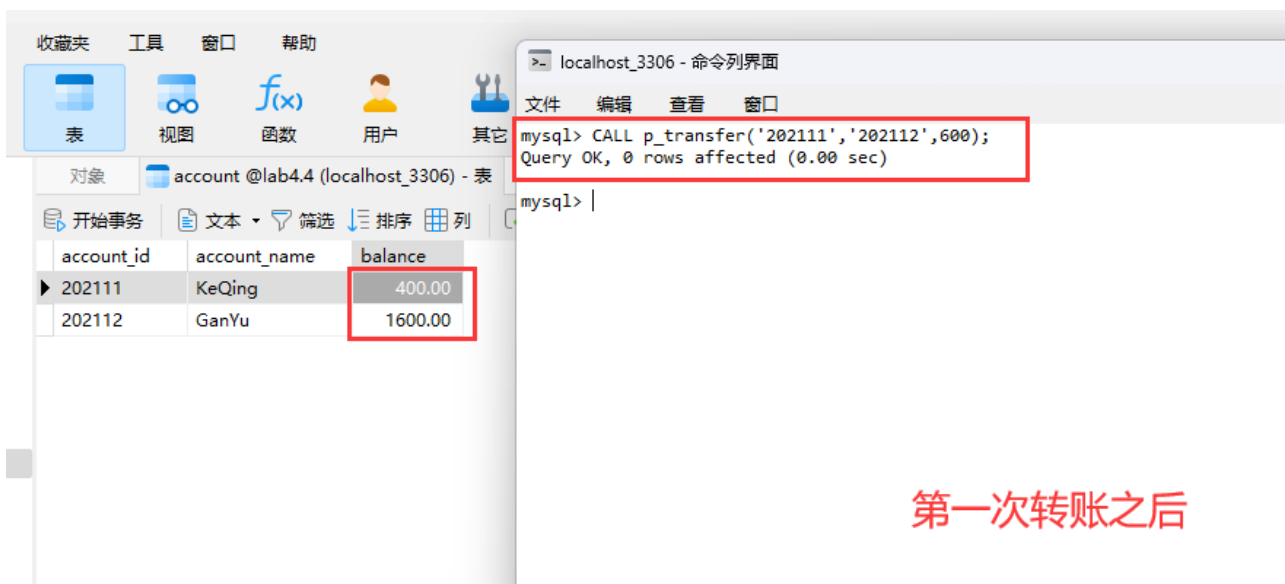
若在 `bank` 表有两条记录 `('202112', 'GanYu', 1000)`、`('202111', 'keQing', 1000)`，两次调用存储过程 `p_transfer` 进行转账，'张一' 每次转 600 元 给 '王二'，观察表中余额的变化。

调用刚刚的过程开始转账。

```
CALL p_transfer('202111', '202112', 600);
```

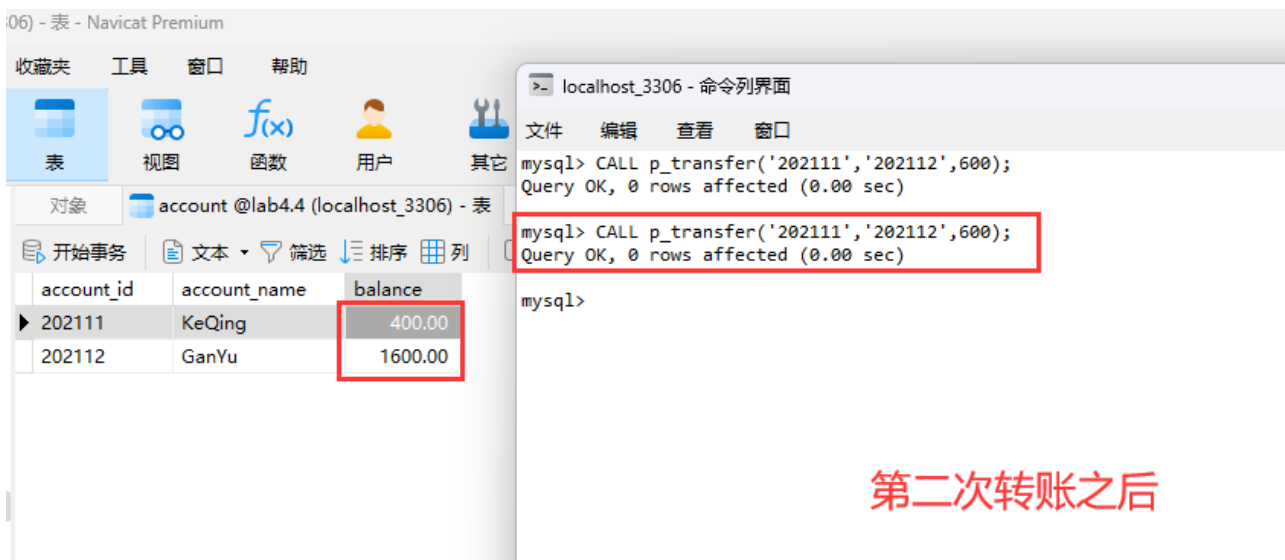
关注转账过程：

第一次转账之后



第一次转账之后

第二次转账之后



第二次转账之后

(4) 分析原因

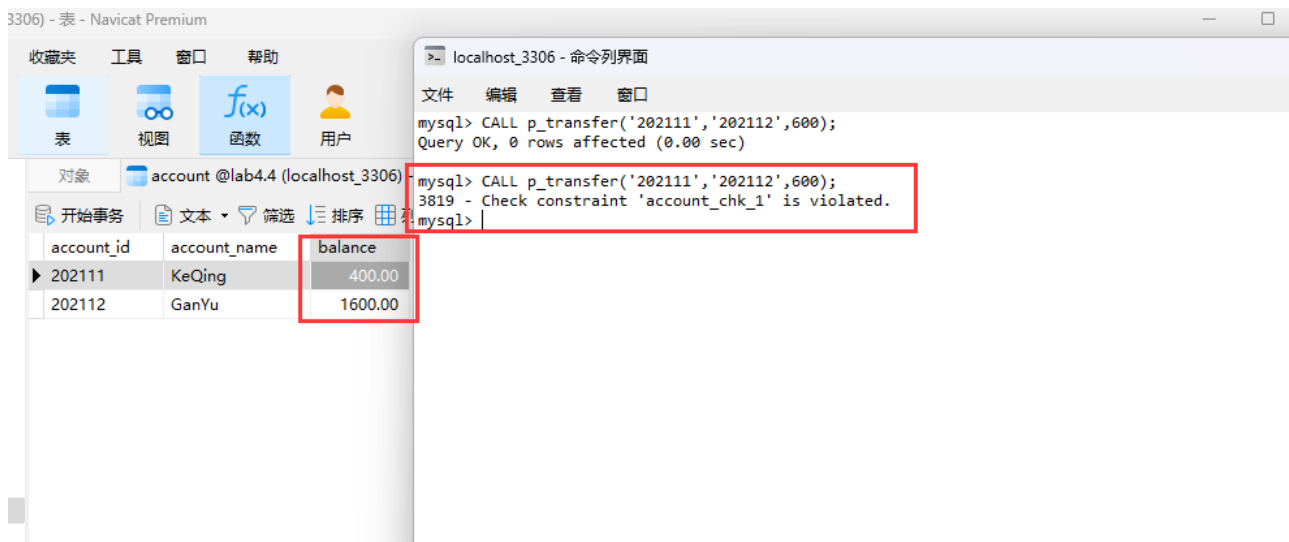
是不是感觉很奇怪？金额并没有变化，难道是出bug了？

再关注一下刚刚的代码，看看这句话在干什么。

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION ROLLBACK;
```

如果出现异常，就按ROLLBACK处理，出现了异常，所以这整个事务被自动撤销了。那么出现了什么异常呢？

我们把这句话删掉，重新运行这个转账程序两次。



现在发现了问题，规定金额必须要不能是负数（这是建立表格的时候给的约束）。KeQing第二次向GanYu转账时，如果这个转账事务成功通过，会导致KeQing的账户金额小于0，这是不被允许的异常。所以按照错误处理例程回滚。

这句就是指示错误处理例程为回滚，撤销刚刚的事务。

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION ROLLBACK;
```

这句被删掉之后，错误就暴露出来了。

参考文献

4.1-4.3 有参考学习（实现没有参考，使用自建的数据库）

- A橙: <https://blog.csdn.net/Aaron503/article/details/128280271>
- 芜湖韩金轮: https://blog.csdn.net/qq_51684393/article/details/128414550

实验感悟

这个实验大概做了6个小时吧。主要是在调bug上花了太多的时间。还有就是花了点时间看了看书，把之前忘记的知识补了一下。还有就是没有按照指导书上去做4.1-4.3，因为感觉那样就太无聊了，就是纯粹复制代码然后截图，没有趣味。

通过本次实验，我进一步掌握了数据库编程，尤其是过程，函数，游标的写法。也借此机会了解了事务的原子性是怎么体现的（实验4.4）。此外再次注意到Mysql与书本的差距是巨大的，不仅体现在语法上，甚至有些组件上Mysql有自己的实现方法或者干脆不实现。所以在这个区别上还是要特别注意，比较花时间的。

总体感觉是收获到了很多知识，自己debug了好几个错误，虽然很折磨但结果还是可以解决的。特别是游标的那个实验自己写的很满意，也很贴近实际生活，能直接应用。如果不是期末周的话，我可能还能再深入一些。

梅炳寅 2023.12.24晚 于天马学生公寓