《数据库系统及应用实践》课程实验报告

实验 3：函数，存储过程与触发器

姓　名: 李鹏达　　学　号: 10225101460　　完成日期: 2024 年 4 月 18 日

1 实验目标

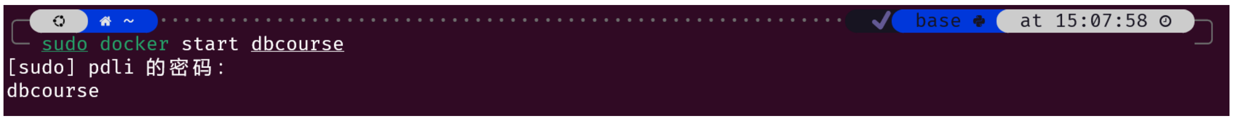
1. 学习 MySQL 数据库管理系统中用户自定义函数、存储过程和触发器的基本概念和语法；
2. 能够根据要求编写正确的自定义函数、存储过程和触发器；
3. 学习 MySQL 数据库管理系统中的递归查询和窗口函数等进阶查询功能；

2 实验过程记录

2.1 使用命令行客户端创建函数与存储过程

1. 打开命令行，启动一个 MySQL 容器示例；

```
sudo docker start dbcourse
```

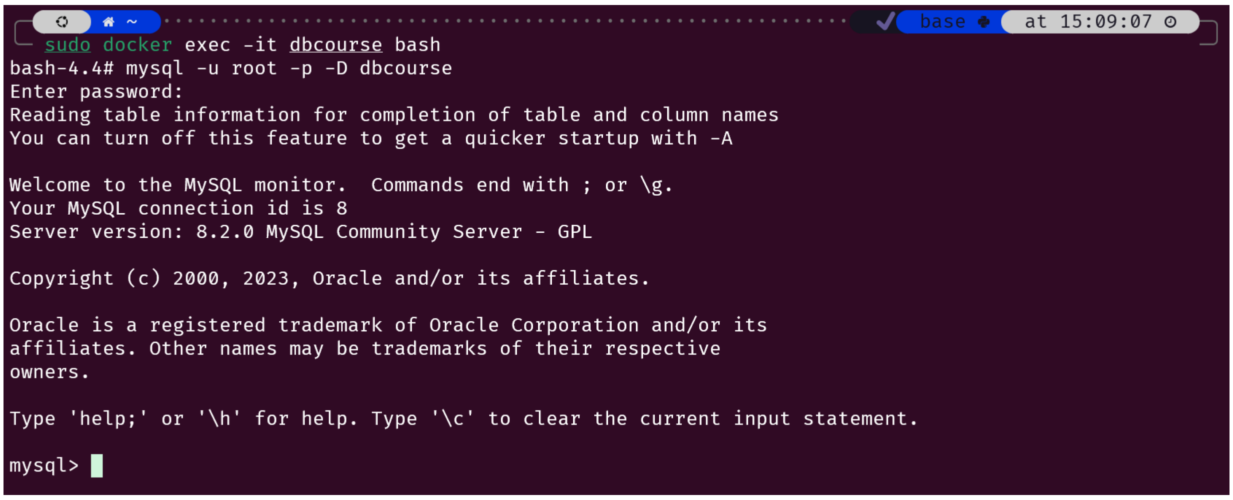


```
 ~                                              base  at 15:07:58
  sudo docker start dbcourse
[sudo] pdli 的密码：
dbcourse
```

图 1: 启动 MySQL 容器示例

2. 在容器内启动一个 bash 终端，使用 mysql 命令登录到 MySQL 数据库;

```
sudo docker exec -it dbcourse bash
mysql -u root -p -D dbcourse
```

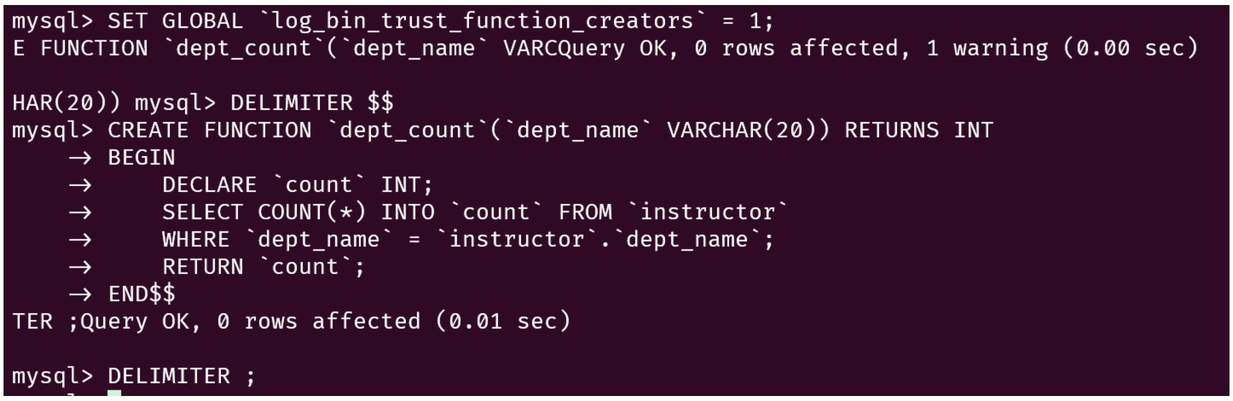


```
sudo docker exec -it dbcourse bash
bash-4.4# mysql -u root -p -D dbcourse
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

图 2: 登录到 MySQL 数据库

3. 参照教材 199 页 Figure 5.6，创建一个名为 `dept_count` 的用户自定义函数，用于统计指定院系的教师数量（返回一个值）；

```sql
SET GLOBAL `log_bin_trust_function_creators` = 1;
DELIMITER $$
CREATE FUNCTION `dept_count`(`dept_name` VARCHAR(20)) RETURNS INT
BEGIN
    DECLARE `count` INT;
    SELECT COUNT(*) INTO `count` FROM `instructor`
    WHERE `dept_name` = `instructor`.`dept_name`;
    RETURN `count`;
END$$
DELIMITER ;
```

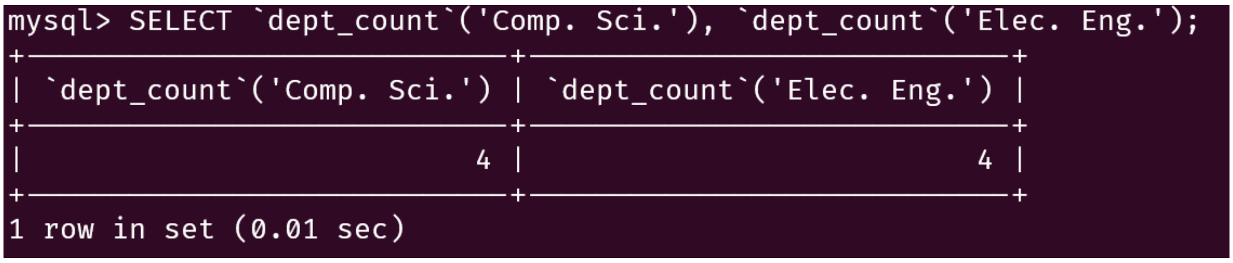


```
mysql> SET GLOBAL `log_bin_trust_function_creators` = 1;
E FUNCTION `dept_count`(`dept_name` VARCQuery OK, 0 rows affected, 1 warning (0.00 sec)

HAR(20)) mysql> DELIMITER $$
mysql> CREATE FUNCTION `dept_count`(`dept_name` VARCHAR(20)) RETURNS INT
    -> BEGIN
    ->     DECLARE `count` INT;
    ->     SELECT COUNT(*) INTO `count` FROM `instructor`
    ->     WHERE `dept_name` = `instructor`.`dept_name`;
    ->     RETURN `count`;
    -> END$$
TER ;Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

图 3: 创建自定义函数

4. 在 SQL 语句中调用自定义函数 `dept_count`，查询计算机科学系（Comp. Sci.）和电子工程系（Elec. Eng.）的教师数量；

```sql
SELECT `dept_count`('Comp. Sci.'), `dept_count`('Elec. Eng.');
```

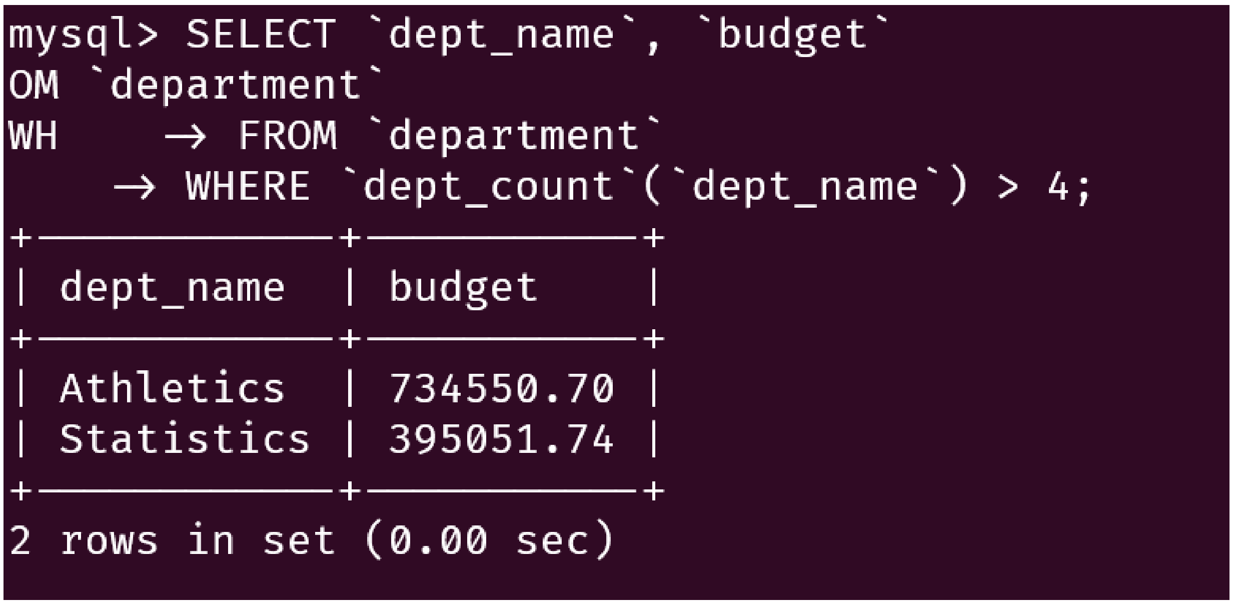


```
mysql> SELECT `dept_count`('Comp. Sci.'), `dept_count`('Elec. Eng.');
+---------------------------+---------------------------+
| `dept_count`('Comp. Sci.') | `dept_count`('Elec. Eng.') |
+---------------------------+---------------------------+
|                         4 |                         4 |
+---------------------------+---------------------------+
1 row in set (0.01 sec)
```

图 4: 运行自定义函数

5. 编写 `SQL` 语句，使用自定义函数 `dept_count` 查询教师数量超过 4 人的院系名称和预算；

```sql
SELECT `dept_name`, `budget`
FROM `department`
WHERE `dept_count`(`dept_name`) > 4;
```

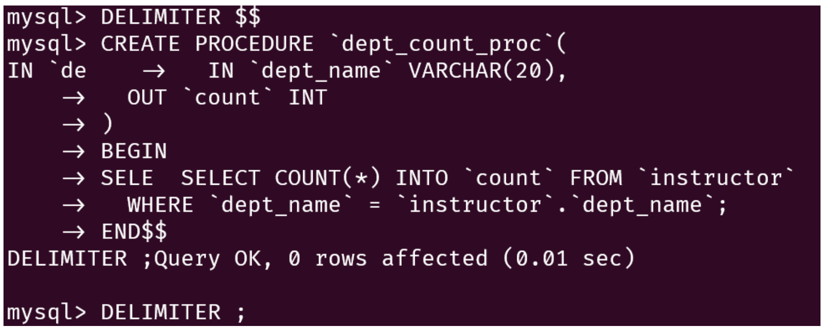


```
mysql> SELECT `dept_name`, `budget`
OM `department`
WH     -> FROM `department`
    -> WHERE `dept_count`(`dept_name`) > 4;
+------------+-----------+
| dept_name  | budget    |
+------------+-----------+
| Athletics  | 734550.70 |
| Statistics | 395051.74 |
+------------+-----------+
2 rows in set (0.00 sec)
```

图 5: 查询结果

6. 将自定义函数 `dept_count` 改写成一个名为 `dept_count_proc` 的存储过程；

```sql
DELIMITER $$
CREATE PROCEDURE `dept_count_proc`(
  IN `dept_name` VARCHAR(20),
  OUT `count` INT
)
BEGIN
  SELECT COUNT(*) INTO `count` FROM `instructor`
  WHERE `dept_name` = `instructor`.`dept_name`;
END$$
DELIMITER ;
```

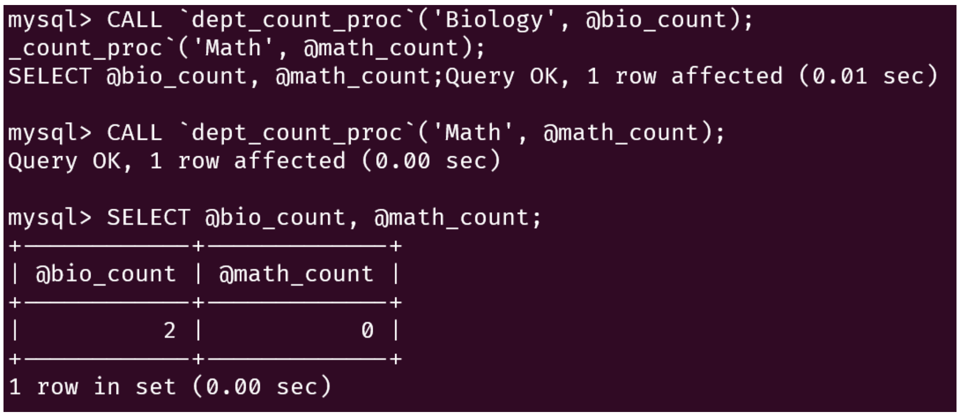


```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE `dept_count_proc`(
IN `de     →    IN `dept_name` VARCHAR(20),
      →    OUT `count` INT
      → )
      → BEGIN
      → SELE  SELECT COUNT(*) INTO `count` FROM `instructor`
      →    WHERE `dept_name` = `instructor`.`dept_name`;
      → END$$
DELIMITER ;Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

图 6: 创建存储过程

7. 通过 call 语句调用存储过程 dept_count_proc，查询生物系（Biology）和数学系（Math）的教师数量；

```sql
CALL `dept_count_proc`('Biology', @bio_count);
CALL `dept_count_proc`('Math', @math_count);
SELECT @bio_count, @math_count;
```

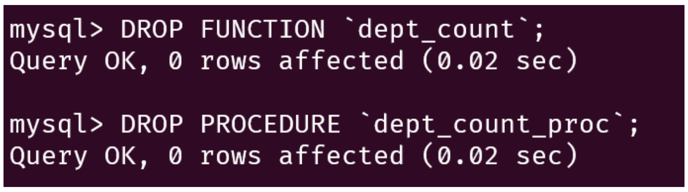


```
mysql> CALL `dept_count_proc`('Biology', @bio_count);
_count_proc`('Math', @math_count);
SELECT @bio_count, @math_count;Query OK, 1 row affected (0.01 sec)

mysql> CALL `dept_count_proc`('Math', @math_count);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @bio_count, @math_count;
+------------+-------------+
| @bio_count | @math_count |
+------------+-------------+
|          2 |           0 |
+------------+-------------+
1 row in set (0.00 sec)
```

图 7: 调用存储过程

8. 删除自定义函数 dept_count 和存储过程 dept_count_proc;

```sql
DROP FUNCTION `dept_count`;
DROP PROCEDURE `dept_count_proc`;
```



```
mysql> DROP FUNCTION `dept_count`;
Query OK, 0 rows affected (0.02 sec)

mysql> DROP PROCEDURE `dept_count_proc`;
Query OK, 0 rows affected (0.02 sec)
```

图 8: 删除自定义函数和存储过程

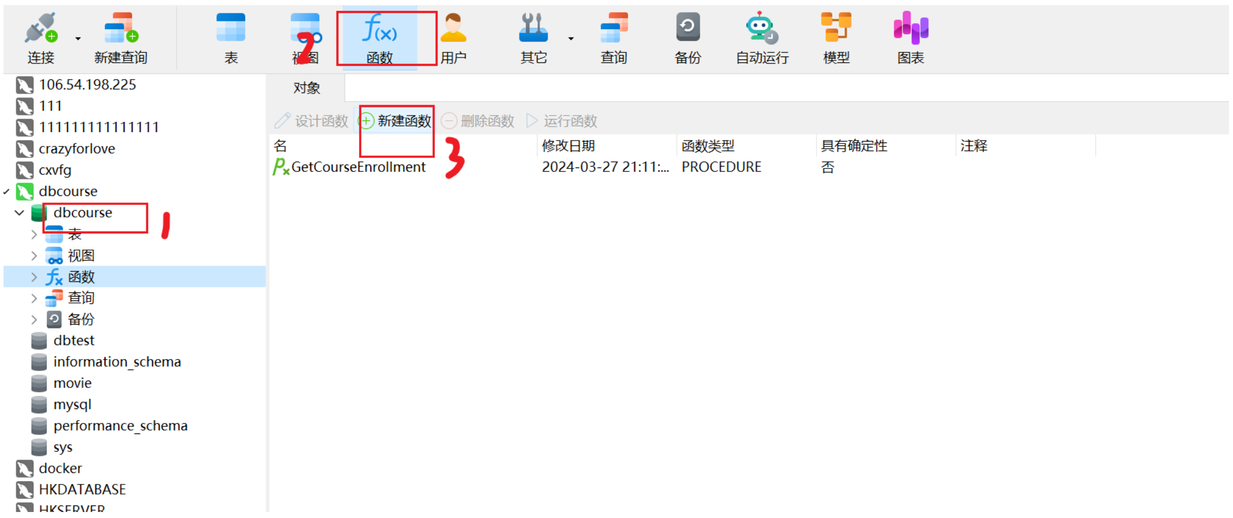
2.2 使用 Navicat 创建函数与存储过程

1. 启动 Navicat 应用，按以下步骤打开“函数向导”对话框

   (a) 连接到 dbcourse 数据库;

   (b) 单击工具栏中的“函数”按钮;

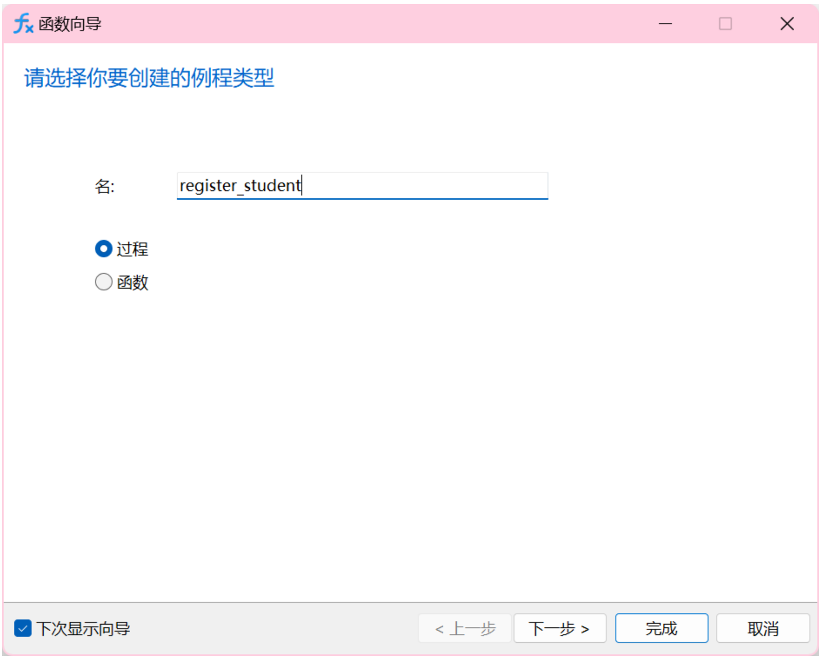
   (c) 单击“新建函数”按钮，打开“函数向导对话框”;



名	修改日期	函数类型	具有确定性	注释
GetCourseEnrollment	2024-03-27 21:11:...	PROCEDURE	否	

图 9: 根据流程创建函数

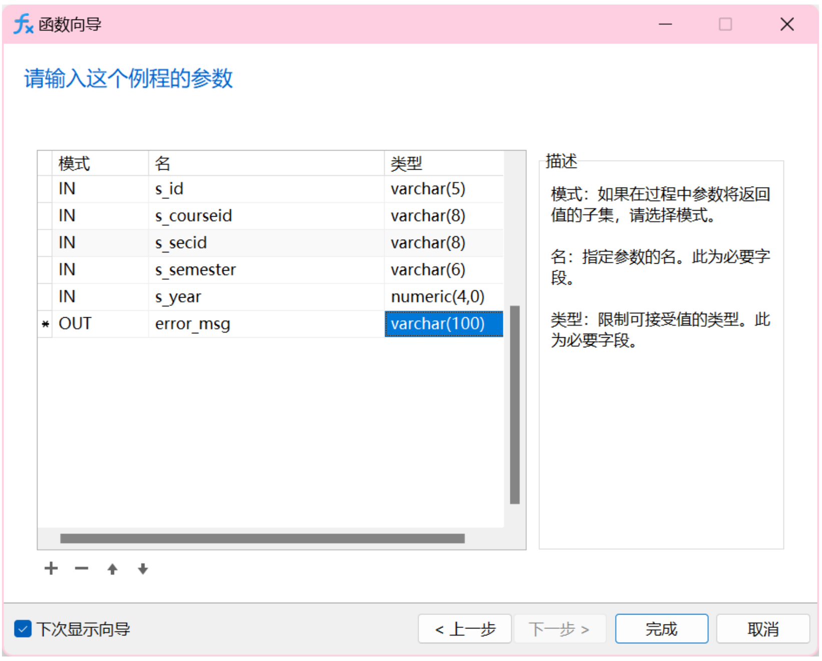
2. 参照教材 203 页 Figure 5.8，创建一个名为 register_student 的存储过程，用于为学生注册课程;



名: register_student

[✓ 过程] 函数

图 10: 创建存储过程



模式	名	类型
IN	s_id	varchar(5)
IN	s_courseid	varchar(8)
IN	s_secid	varchar(8)
IN	s_semester	varchar(6)
IN	s_year	numeric(4,0)
OUT	error_msg	varchar(100)

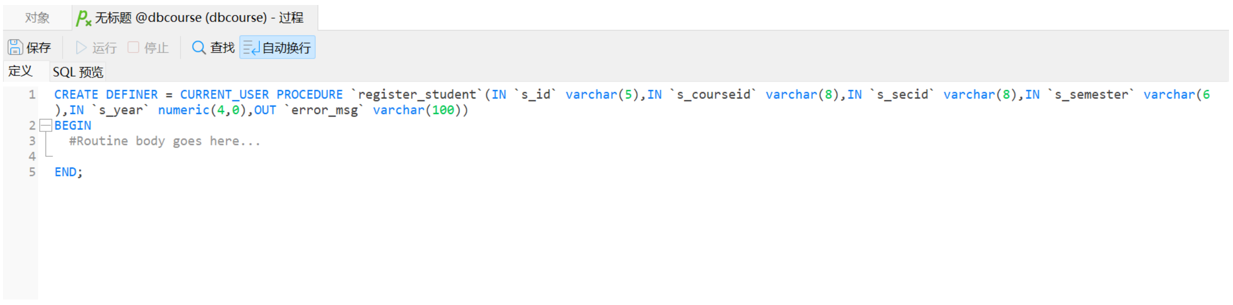
描述

模式：如果在过程中参数将返回值的子集，请选择模式。

名：指定参数的名。此为必要字段。

类型：限制可接受值的类型。此为必要字段。

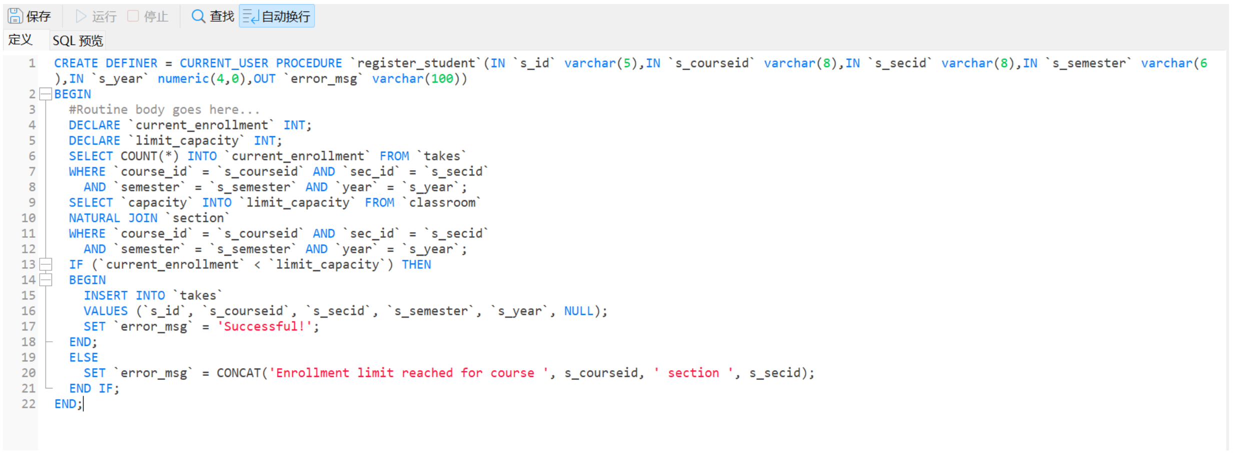
图 11: 设置参数



```
CREATE DEFINER = CURRENT_USER PROCEDURE `register_student`(IN `s_id` varchar(5),IN `s_courseid` varchar(8),IN `s_secid` varchar(8),IN `s_semester` varchar(6
),IN `s_year` numeric(4,0),OUT `error_msg` varchar(100))
BEGIN
  #Routine body goes here...

END;
```

图 12: 编写代码

```
CREATE DEFINER = CURRENT_USER PROCEDURE `register_student`(
  IN `s_id` varchar(5),
  IN `s_courseid` varchar(8),
  IN `s_secid` varchar(8),
  IN `s_semester` varchar(6),
  IN `s_year` numeric(4, 0),
  OUT `error_msg` varchar(100)
)
BEGIN
  -- #Routine body goes here...
  DECLARE `current_enrollment` INT;
  DECLARE `limit_capacity` INT;
  SELECT COUNT(*) INTO `current_enrollment` FROM `takes`
  WHERE `course_id` = `s_courseid` AND `sec_id` = `s_secid`
    AND `semester` = `s_semester` AND `year` = `s_year`;
  SELECT `capacity` INTO `limit_capacity` FROM `classroom`
  NATURAL JOIN `section`
  WHERE `course_id` = `s_courseid` AND `sec_id` = `s_secid`
    AND `semester` = `s_semester` AND `year` = `s_year`;
  IF (`current_enrollment` < `limit_capacity`) THEN
  BEGIN
    INSERT INTO `takes`
    VALUES (`s_id`, `s_courseid`, `s_secid`, `s_semester`, `s_year`, NULL);
    SET `error_msg` = 'Successful!';
  END;
  ELSE
    SET `error_msg` = CONCAT('Enrollment limit reached for course ', `
        s_courseid`, ' section ', `s_secid`);
  END IF;
END;
```



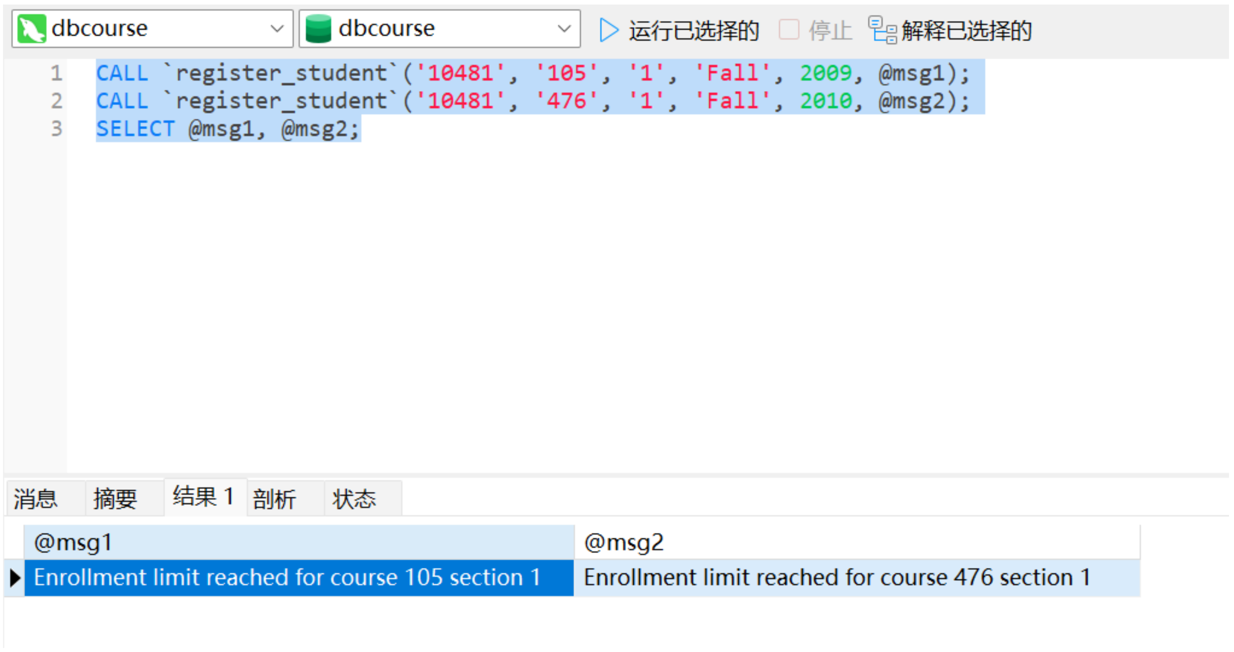
```sql
CREATE DEFINER = CURRENT_USER PROCEDURE `register_student`(IN `s_id` varchar(5),IN `s_courseid` varchar(8),IN `s_secid` varchar(8),IN `s_semester` varchar(6
),IN `s_year` numeric(4,0),OUT `error_msg` varchar(100))
BEGIN
  #Routine body goes here...
  DECLARE `current_enrollment` INT;
  DECLARE `limit_capacity` INT;
  SELECT COUNT(*) INTO `current_enrollment` FROM `takes`
  WHERE `course_id` = `s_courseid` AND `sec_id` = `s_secid`
    AND `semester` = `s_semester` AND `year` = `s_year`;
  SELECT `capacity` INTO `limit_capacity` FROM `classroom`
  NATURAL JOIN `section`
  WHERE `course_id` = `s_courseid` AND `sec_id` = `s_secid`
    AND `semester` = `s_semester` AND `year` = `s_year`;
  IF (`current_enrollment` < `limit_capacity`) THEN
  BEGIN
    INSERT INTO `takes`
    VALUES (`s_id`, `s_courseid`, `s_secid`, `s_semester`, `s_year`, NULL);
    SET `error_msg` = 'Successful!';
  END;
  ELSE
    SET `error_msg` = CONCAT('Enrollment limit reached for course ', s_courseid, ' section ', s_secid);
  END IF;
END;
```

图 13: 保存存储过程

3. 在 Navicat 的查询窗口中通过 call 语句调用存储过程 register_student，为学号 10481 的同学注册 2009 年秋季学期开设的 105 课程和 2010 年秋季学期开设的 476 课程；

```sql
CALL `register_student`('10481', '105', '1', 'Fall', 2009, @msg1);
CALL `register_student`('10481', '476', '1', 'Fall', 2010, @msg2);
SELECT @msg1, @msg2;
```

结果为：



```sql
CALL `register_student`('10481', '105', '1', 'Fall', 2009, @msg1);
CALL `register_student`('10481', '476', '1', 'Fall', 2010, @msg2);
SELECT @msg1, @msg2;
```

@msg1	@msg2
Enrollment limit reached for course 105 section 1	Enrollment limit reached for course 476 section 1

图 14: 调用存储过程

4. 显然，教材 203 页 Figure 5.8 中的处理逻辑并不完善，没有考虑不存在指定学号的学生和该学生已经注册了该门课程等情况，这就可能会导致存储过程异常退出。请对存储过程 `register_student` 的处理逻辑进行完善，确保其能正确处理各种情况并返回正确的结果。

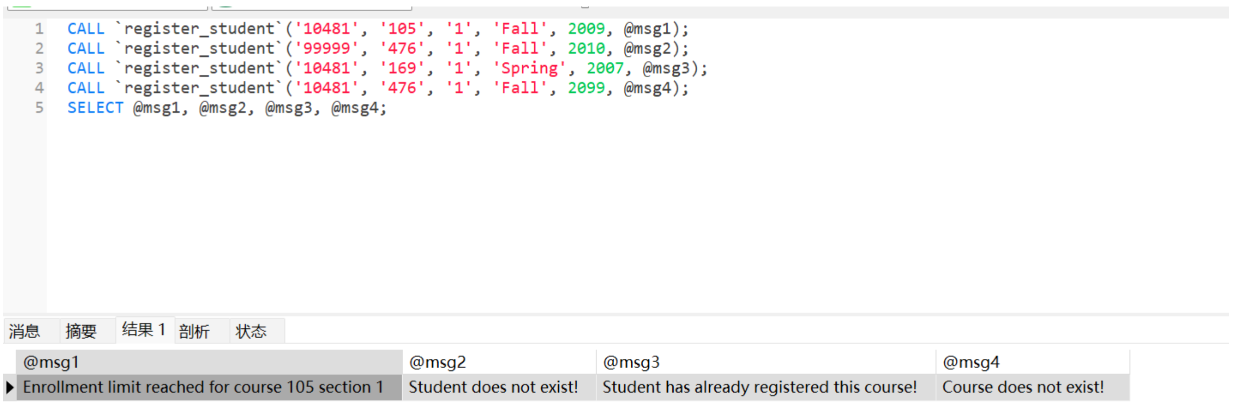
```sql
DROP PROCEDURE `register_student`;
DELIMITER $$
CREATE DEFINER = CURRENT_USER PROCEDURE `register_student`(
  IN `s_id` varchar(5),
  IN `s_courseid` varchar(8),
  IN `s_secid` varchar(8),
  IN `s_semester` varchar(6),
  IN `s_year` numeric(4, 0),
  OUT `error_msg` varchar(100)
)
BEGIN
  -- #Routine body goes here...
  DECLARE `current_enrollment` INT;
  DECLARE `limit_capacity` INT;
  DECLARE `student_exist` INT;
  DECLARE `course_exist` INT;
  DECLARE `student_registered` INT;
  SELECT COUNT(*) INTO `student_exist` FROM `student`
  WHERE `ID` = `s_id`;
  SELECT COUNT(*) INTO `course_exist` FROM `section`
  WHERE `course_id` = `s_courseid` AND `sec_id` = `s_secid`
    AND `semester` = `s_semester` AND `year` = `s_year`;
  SELECT COUNT(*) INTO `student_registered` FROM `takes`
  WHERE `ID` = `s_id` AND `course_id` = `s_courseid` AND `sec_id` = `s_secid`
    AND `semester` = `s_semester` AND `year` = `s_year`;
  IF (`student_exist` = 0) THEN
    SET `error_msg` = 'Student does not exist!';
  ELSEIF (`course_exist` = 0) THEN
    SET `error_msg` = 'Course does not exist!';
  ELSEIF (`student_registered` > 0) THEN
    SET `error_msg` = 'Student has already registered this course!';
  ELSE
    SELECT COUNT(*) INTO `current_enrollment` FROM `takes`
    WHERE `course_id` = `s_courseid` AND `sec_id` = `s_secid`
      AND `semester` = `s_semester` AND `year` = `s_year`;
    SELECT `capacity` INTO `limit_capacity` FROM `classroom`
    NATURAL JOIN `section`
    WHERE `course_id` = `s_courseid` AND `sec_id` = `s_secid`
      AND `semester` = `s_semester` AND `year` = `s_year`;
    IF (`current_enrollment` < `limit_capacity`) THEN
```

```
        BEGIN
          INSERT INTO `takes`
          VALUES (`s_id`, `s_courseid`, `s_secid`, `s_semester`, `s_year`, NULL);
          SET `error_msg` = 'Successful!';
        END;
        ELSE
          SET `error_msg` = CONCAT('Enrollment limit reached for course ', `
              s_courseid`, ' section ', `s_secid`);
        END IF;
      END IF;
    END$$
    DELIMITER ;
```

使用以下语句进行测试:

```
CALL `register_student`('10481', '105', '1', 'Fall', 2009, @msg1);
CALL `register_student`('99999', '476', '1', 'Fall', 2010, @msg2);
CALL `register_student`('10481', '169', '1', 'Spring', 2007, @msg3);
CALL `register_student`('10481', '476', '1', 'Fall', 2099, @msg4);
SELECT @msg1, @msg2, @msg3, @msg4;
```

结果如下:



```
CALL `register_student`('10481', '105', '1', 'Fall', 2009, @msg1);
CALL `register_student`('99999', '476', '1', 'Fall', 2010, @msg2);
CALL `register_student`('10481', '169', '1', 'Spring', 2007, @msg3);
CALL `register_student`('10481', '476', '1', 'Fall', 2099, @msg4);
SELECT @msg1, @msg2, @msg3, @msg4;
```

@msg1	@msg2	@msg3	@msg4
Enrollment limit reached for course 105 section 1	Student does not exist!	Student has already registered this course!	Course does not exist!

图 15: 测试结果

5. 编写一个名为 `section_setup` 的存储过程，完成下面的任务:

- 输入参数: `course_id`
- 为该课程在接下来的新学期开设一个新的 section;
- 该课程所属院系的学生如果之前没有修过该课程，则为其注册该课程;

- 为其它所需参数设置合理的值；

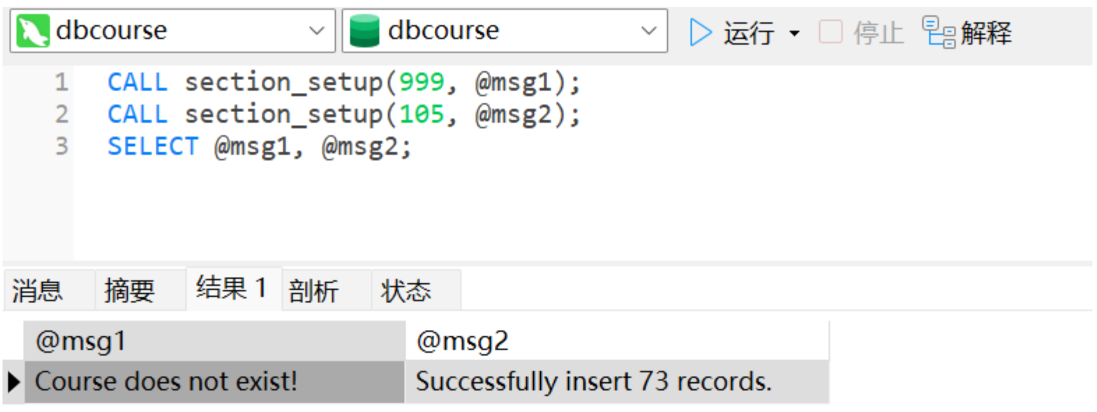
```
DELIMITER $$
CREATE DEFINER = CURRENT_USER PROCEDURE `section_setup`(
  IN `course_id` int,
  OUT `message` varchar(100)
)
BEGIN
  #Routine body goes here...
  DECLARE `@sec_cnt` INT;
  DECLARE `@dept_name` VARCHAR(20);
  DECLARE `@sec_id` INT;
  DECLARE `@building` VARCHAR(15);
  DECLARE `@room_number` INT;
  DECLARE `@semester` VARCHAR(6);
  DECLARE `@year` INT;
  DECLARE `@cnt` INT;
  SELECT COUNT(*) INTO `@sec_cnt` FROM `course`
  WHERE `course`.`course_id` = `course_id`;
  SELECT DISTINCT `dept_name` INTO `@dept_name` FROM `course`
  WHERE `course`.`course_id` = `course_id`;
  IF (`@sec_cnt` = 0) THEN
    SET `message` = 'Course does not exist!';
  ELSE
    BEGIN
      SELECT MAX(`sec_id`) + 1 INTO `@sec_id` FROM `section`
      WHERE `section`.`course_id` = `course_id`;
      SELECT DISTINCT `building` INTO `@building`
      FROM `department` WHERE `dept_name` = `@dept_name`;
      SELECT DISTINCT `room_number` INTO `@room_number`
      FROM `classroom` WHERE `building` = `@building`
      ORDER BY `room_number` DESC LIMIT 1;
      IF (MONTH(CURDATE()) < 2) THEN
        SET `@semester` = 'Spring';
        SET `@year` = YEAR(CURDATE());
      ELSEIF (MONTH(CURDATE()) < 8) THEN
        SET `@semester` = 'Fall';
        SET `@year` = YEAR(CURDATE());
      ELSE
        SET `@semester` = 'Spring';
        SET `@year` = YEAR(CURDATE()) + 1;
      END IF;
      SELECT COUNT(*) INTO `@cnt`
      FROM `student`
```

```
        WHERE `dept_name` = `@dept_name` AND `ID` NOT IN (
          SELECT `ID`
          FROM `takes`
          WHERE `takes`.`course_id` = `course_id`
        );
        INSERT INTO `section`
        VALUES (`course_id`, `@sec_id`, `@semester`, `@year`, `@building`, `
            @room_number`, 'A');
        INSERT INTO `takes`
        SELECT `ID`, `course_id`, `@sec_id`, `@semester`, `@year`, NULL
        FROM `student`
        WHERE `dept_name` = `@dept_name` AND `ID` NOT IN (
          SELECT `ID`
          FROM `takes`
          WHERE `takes`.`course_id` = `course_id`
        );
        SET `message` = CONCAT('Successfully insert ', `@cnt`, ' records.');
      END;
    END IF;
  END$$
  DELIMITER ;
```

使用以下语句进行测试：

```
CALL section_setup(999, @msg1);
CALL section_setup(105, @msg2);
SELECT @msg1, @msg2;
```

结果如下：



```
CALL section_setup(999, @msg1);
CALL section_setup(105, @msg2);
SELECT @msg1, @msg2;
```

```
@msg1                  @msg2
Course does not exist! Successfully insert 73 records.
```

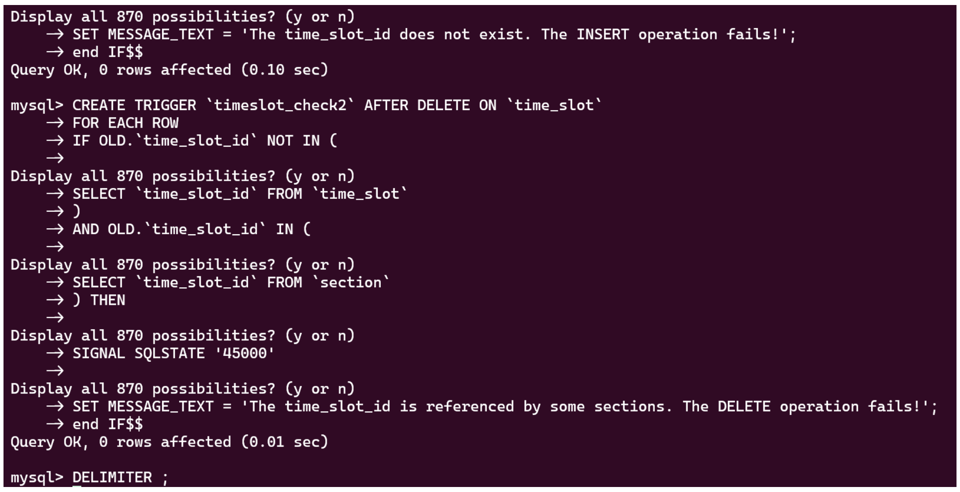
图 16: 测试结果

2.3 触发器

1. 参照教材 207 页 Figure 5.9，编写两个名为 timeslot_check1 和 timeslot_check2 的触发器；

```sql
DELIMITER $$
CREATE TRIGGER `timeslot_check1` AFTER INSERT ON `section`
FOR EACH ROW
  IF NEW.`time_slot_id` NOT IN (
    SELECT `time_slot_id` FROM `time_slot`
  ) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'The time_slot_id does not exist. The INSERT operation
       fails!';
  end IF$$
CREATE TRIGGER `timeslot_check2` AFTER DELETE ON `time_slot`
FOR EACH ROW
  IF OLD.`time_slot_id` NOT IN (
    SELECT `time_slot_id` FROM `time_slot`
  )
  AND OLD.`time_slot_id` IN (
    SELECT `time_slot_id` FROM `section`
  ) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'The time_slot_id is referenced by some sections. The
       DELETE operation fails!';
  end IF$$
DELIMITER ;
```

结果如下：



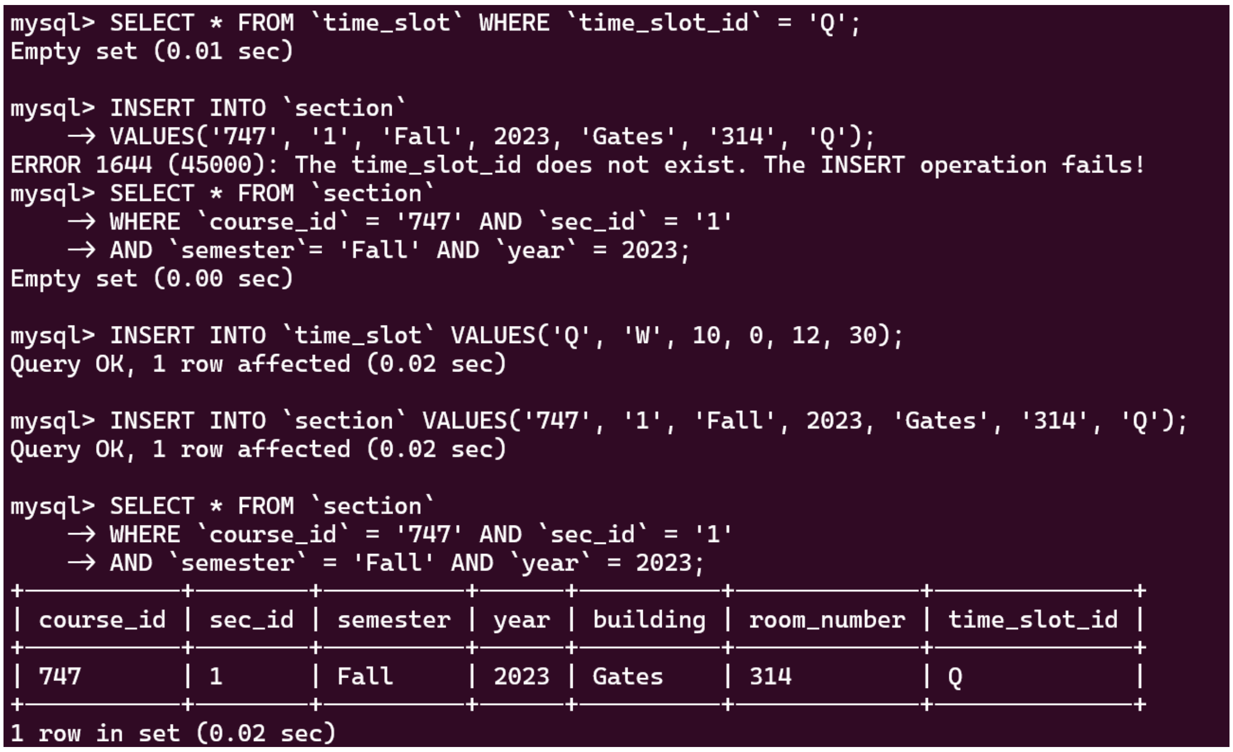
```
Display all 870 possibilities? (y or n)
    -> SET MESSAGE_TEXT = 'The time_slot_id does not exist. The INSERT operation fails!';
    -> end IF$$
Query OK, 0 rows affected (0.10 sec)

mysql> CREATE TRIGGER `timeslot_check2` AFTER DELETE ON `time_slot`
    -> FOR EACH ROW
    -> IF OLD.`time_slot_id` NOT IN (
    ->
Display all 870 possibilities? (y or n)
    -> SELECT `time_slot_id` FROM `time_slot`
    -> )
    -> AND OLD.`time_slot_id` IN (
    ->
Display all 870 possibilities? (y or n)
    -> SELECT `time_slot_id` FROM `section`
    -> ) THEN
    ->
Display all 870 possibilities? (y or n)
    -> SIGNAL SQLSTATE '45000'
    ->
Display all 870 possibilities? (y or n)
    -> SET MESSAGE_TEXT = 'The time_slot_id is referenced by some sections. The DELETE operation fails!';
    -> end IF$$
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
```

图 17: 创建触发器

2. 依次执行下列 SQL 语句并检查触发器 `timeslot_check1` 的执行效果；

```sql
SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
INSERT INTO `section`
VALUES('747', '1', 'Fall', 2023, 'Gates', '314', 'Q');
SELECT * FROM `section`
WHERE `course_id` = '747' AND `sec_id` = '1'
AND `semester`= 'Fall' AND `year` = 2023;
INSERT INTO `time_slot` VALUES('Q', 'W', 10, 0, 12, 30);
INSERT INTO `section` VALUES('747', '1', 'Fall', 2023, 'Gates', '314', 'Q');
SELECT * FROM `section`
WHERE `course_id` = '747' AND `sec_id` = '1'
AND `semester` = 'Fall' AND `year` = 2023;
```

结果如下：



```
mysql> SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
Empty set (0.01 sec)

mysql> INSERT INTO `section`
    -> VALUES('747', '1', 'Fall', 2023, 'Gates', '314', 'Q');
ERROR 1644 (45000): The time_slot_id does not exist. The INSERT operation fails!
mysql> SELECT * FROM `section`
    -> WHERE `course_id` = '747' AND `sec_id` = '1'
    -> AND `semester`= 'Fall' AND `year` = 2023;
Empty set (0.00 sec)

mysql> INSERT INTO `time_slot` VALUES('Q', 'W', 10, 0, 12, 30);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO `section` VALUES('747', '1', 'Fall', 2023, 'Gates', '314', 'Q');
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM `section`
    -> WHERE `course_id` = '747' AND `sec_id` = '1'
    -> AND `semester` = 'Fall' AND `year` = 2023;
+-----------+--------+----------+------+----------+-------------+--------------+
| course_id | sec_id | semester | year | building | room_number | time_slot_id |
+-----------+--------+----------+------+----------+-------------+--------------+
| 747       | 1      | Fall     | 2023 | Gates    | 314         | Q            |
+-----------+--------+----------+------+----------+-------------+--------------+
1 row in set (0.02 sec)
```

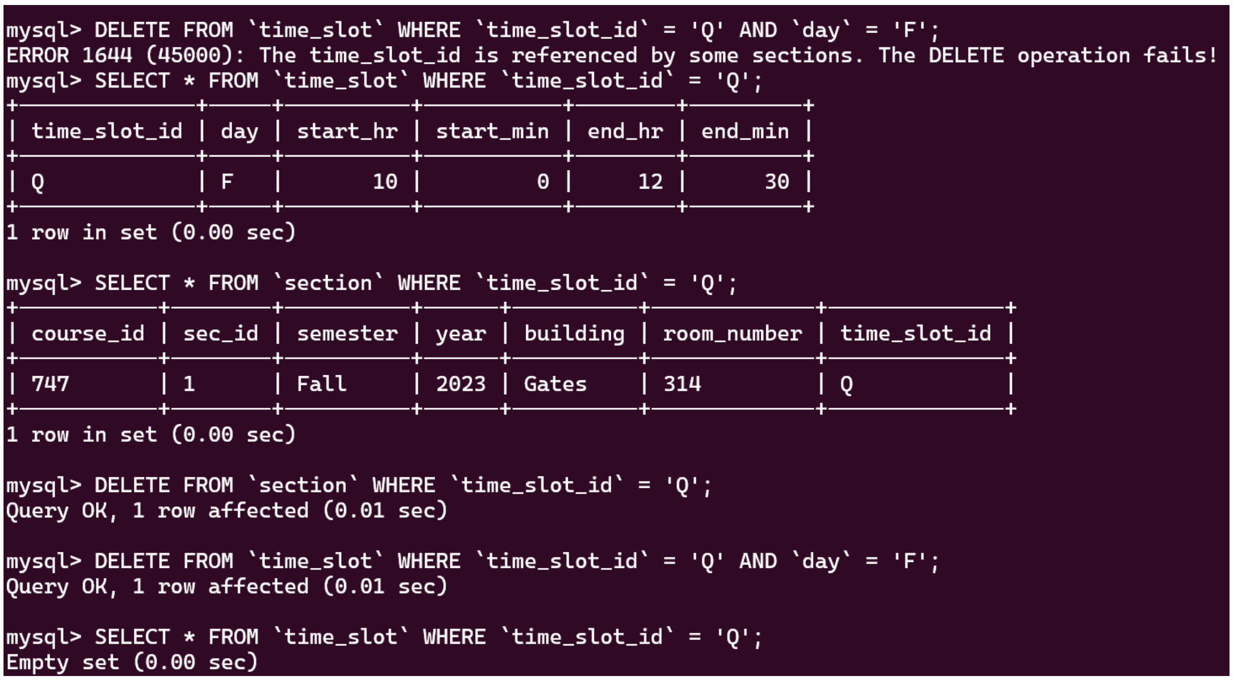
图 18: 测试结果

3. 依次执行下列 SQL 语句并检查触发器 `timeslot_check2` 的执行效果；

```sql
INSERT INTO `time_slot` VALUES( 'Q', 'F', 10, 0, 12, 30);
SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
DELETE FROM `time_slot` WHERE `time_slot_id` = 'Q' AND `day` = 'W';
```

```
SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
DELETE FROM `time_slot` WHERE `time_slot_id` = 'Q' AND `day` = 'F';
SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
SELECT * FROM `section` WHERE `time_slot_id` = 'Q';
DELETE FROM `section` WHERE `time_slot_id` = 'Q';
DELETE FROM `time_slot` WHERE `time_slot_id` = 'Q' AND `day` = 'F';
SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
```

结果如下：



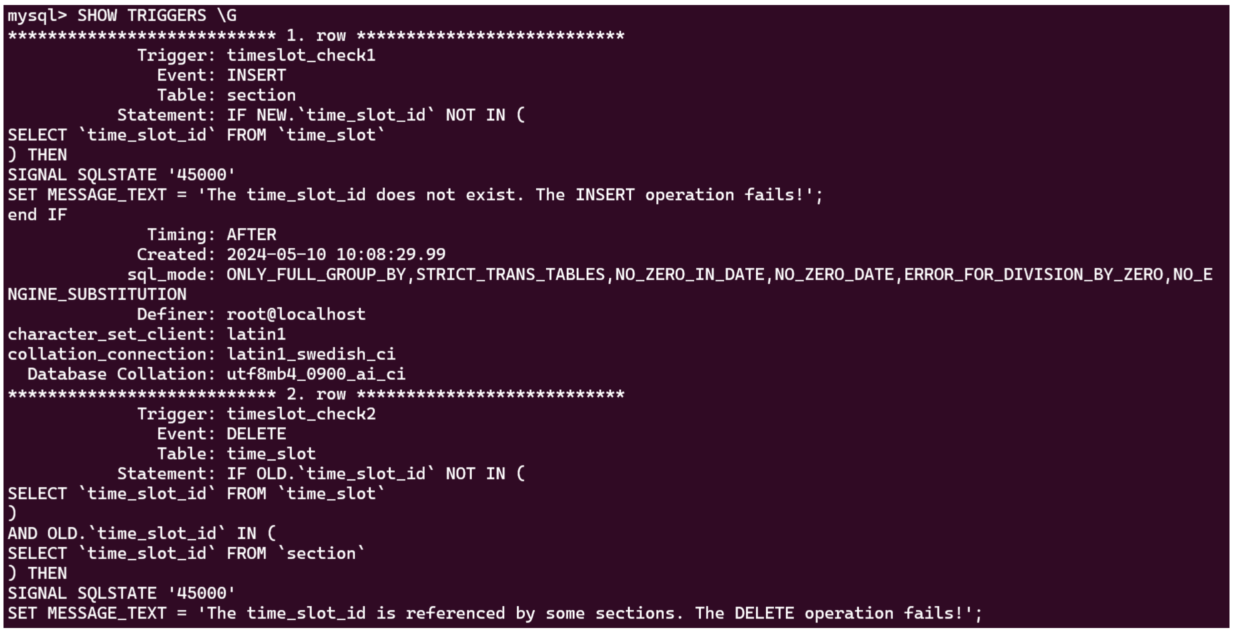
```
mysql> DELETE FROM `time_slot` WHERE `time_slot_id` = 'Q' AND `day` = 'F';
ERROR 1644 (45000): The time_slot_id is referenced by some sections. The DELETE operation fails!
mysql> SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
+--------------+-----+----------+-----------+--------+---------+
| time_slot_id | day | start_hr | start_min | end_hr | end_min |
+--------------+-----+----------+-----------+--------+---------+
| Q            | F   |       10 |         0 |     12 |      30 |
+--------------+-----+----------+-----------+--------+---------+
1 row in set (0.00 sec)

mysql> SELECT * FROM `section` WHERE `time_slot_id` = 'Q';
+-----------+--------+----------+------+----------+-------------+--------------+
| course_id | sec_id | semester | year | building | room_number | time_slot_id |
+-----------+--------+----------+------+----------+-------------+--------------+
| 747       | 1      | Fall     | 2023 | Gates    | 314         | Q            |
+-----------+--------+----------+------+----------+-------------+--------------+
1 row in set (0.00 sec)

mysql> DELETE FROM `section` WHERE `time_slot_id` = 'Q';
Query OK, 1 row affected (0.01 sec)

mysql> DELETE FROM `time_slot` WHERE `time_slot_id` = 'Q' AND `day` = 'F';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM `time_slot` WHERE `time_slot_id` = 'Q';
Empty set (0.00 sec)
```

图 19: 测试结果

4. 执行下列 SQL 语句，查看并删除所创建的触发器；

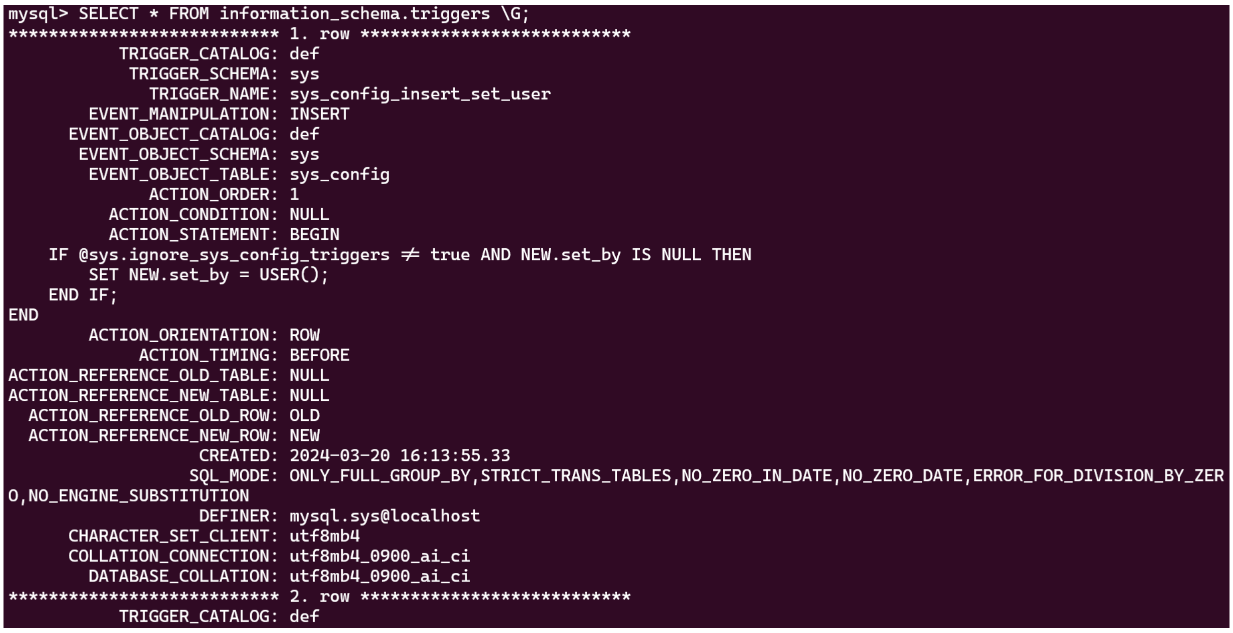
```
SHOW TRIGGERS \G
SELECT * FROM information_schema.triggers \G
DROP TRIGGER timeslot_check1;
DROP TRIGGER timeslot_check2;
```

结果如下：



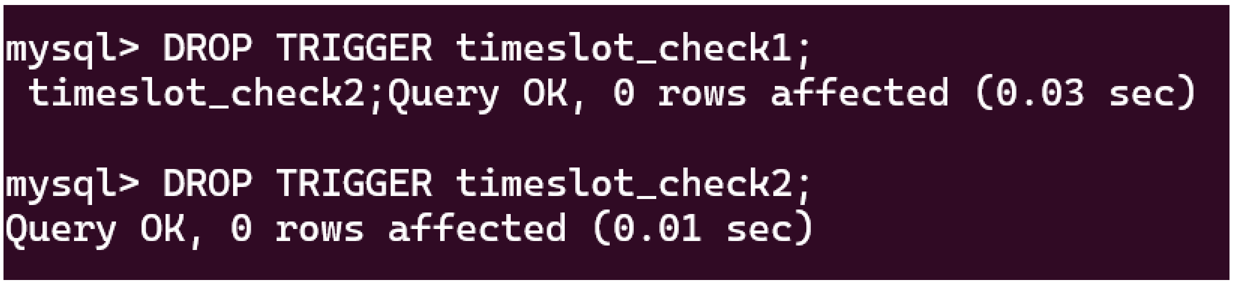
```
mysql> SHOW TRIGGERS \G
*************************** 1. row ***************************
             Trigger: timeslot_check1
               Event: INSERT
               Table: section
           Statement: IF NEW.`time_slot_id` NOT IN (
SELECT `time_slot_id` FROM `time_slot`
) THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'The time_slot_id does not exist. The INSERT operation fails!';
end IF
              Timing: AFTER
             Created: 2024-05-10 10:08:29.99
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_E
NGINE_SUBSTITUTION
             Definer: root@localhost
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: utf8mb4_0900_ai_ci
*************************** 2. row ***************************
             Trigger: timeslot_check2
               Event: DELETE
               Table: time_slot
           Statement: IF OLD.`time_slot_id` NOT IN (
SELECT `time_slot_id` FROM `time_slot`
)
AND OLD.`time_slot_id` IN (
SELECT `time_slot_id` FROM `section`
) THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'The time_slot_id is referenced by some sections. The DELETE operation fails!';
```

图 20: 查看触发器（1）



```
mysql> SELECT * FROM information_schema.triggers \G;
*************************** 1. row ***************************
           TRIGGER_CATALOG: def
            TRIGGER_SCHEMA: sys
              TRIGGER_NAME: sys_config_insert_set_user
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
       EVENT_OBJECT_SCHEMA: sys
        EVENT_OBJECT_TABLE: sys_config
              ACTION_ORDER: 1
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: BEGIN
    IF @sys.ignore_sys_config_triggers != true AND NEW.set_by IS NULL THEN
        SET NEW.set_by = USER();
    END IF;
END
        ACTION_ORIENTATION: ROW
             ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                   CREATED: 2024-03-20 16:13:55.33
                  SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZER
O,NO_ENGINE_SUBSTITUTION
                   DEFINER: mysql.sys@localhost
      CHARACTER_SET_CLIENT: utf8mb4
      COLLATION_CONNECTION: utf8mb4_0900_ai_ci
        DATABASE_COLLATION: utf8mb4_0900_ai_ci
*************************** 2. row ***************************
           TRIGGER_CATALOG: def
```

图 21: 查看触发器（2）

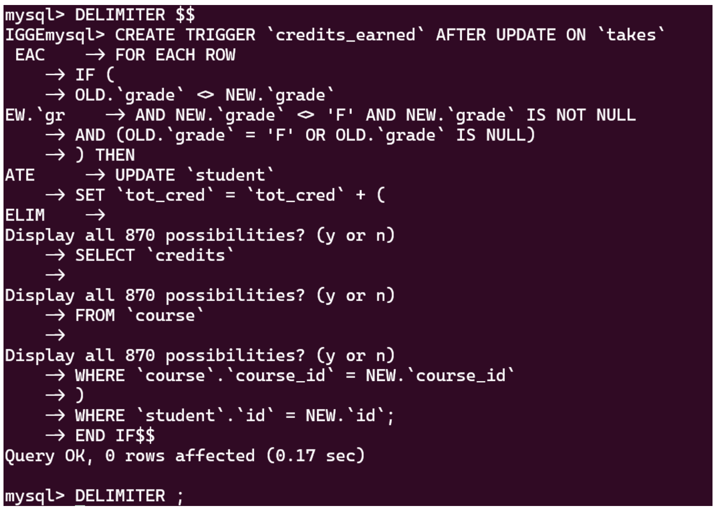


```
mysql> DROP TRIGGER timeslot_check1;
 timeslot_check2;Query OK, 0 rows affected (0.03 sec)

mysql> DROP TRIGGER timeslot_check2;
Query OK, 0 rows affected (0.01 sec)
```

图 22: 删除触发器

5. 参照教材 209 页 Figure 5.10，创建一个名为 credits_earned 的触发器用于维护学生所获得的学分，然后测试该触发器是否能正确执行；

```sql
CREATE TRIGGER `credits_earned` AFTER UPDATE OF `grade` ON `takes`
FOR EACH ROW
WHEN NEW.`grade` <> 'F' AND NEW.`grade` IS NOT NULL
AND (OLD.`grade` = 'F' OR OLD.`grade` IS NULL)
BEGIN ATOMIC
    UPDATE `student`
    SET `tot_cred` = `tot_cred` + (
    SELECT `credits`
        FROM `course`
        WHERE `course`.`course_id` = NEW.`course_id`
  )
    WHERE `student`.`id` = NEW.`id`;
END;
```



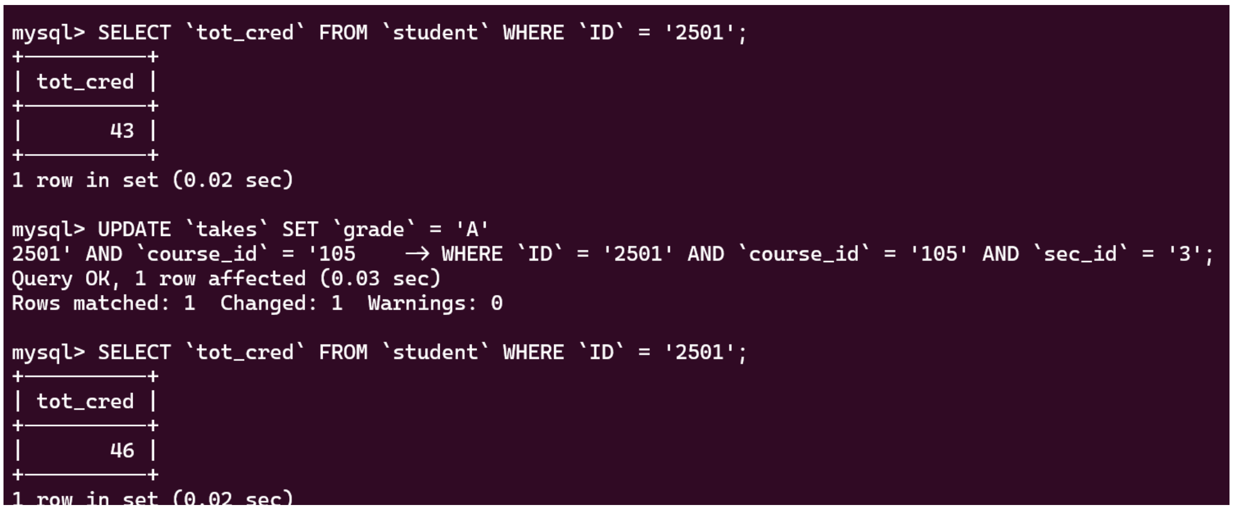
```
mysql> DELIMITER $$
IGGEmysql> CREATE TRIGGER `credits_earned` AFTER UPDATE ON `takes`
 EAC    -> FOR EACH ROW
    -> IF (
    -> OLD.`grade` <> NEW.`grade`
EW.`gr    -> AND NEW.`grade` <> 'F' AND NEW.`grade` IS NOT NULL
    -> AND (OLD.`grade` = 'F' OR OLD.`grade` IS NULL)
    -> ) THEN
ATE     -> UPDATE `student`
    -> SET `tot_cred` = `tot_cred` + (
ELIM    ->
Display all 870 possibilities? (y or n)
    -> SELECT `credits`
    ->
Display all 870 possibilities? (y or n)
    -> FROM `course`
    ->
Display all 870 possibilities? (y or n)
    -> WHERE `course`.`course_id` = NEW.`course_id`
    -> )
    -> WHERE `student`.`id` = NEW.`id`;
    -> END IF$$
Query OK, 0 rows affected (0.17 sec)

mysql> DELIMITER ;
```

图 23: 创建触发器

使用以下语句进行测试:

```sql
SELECT `tot_cred` FROM `student` WHERE `ID` = '2501';
UPDATE `takes` SET `grade` = 'A'
WHERE `ID` = '2501' AND `course_id` = '105' AND `sec_id` = '3';
SELECT `tot_cred` FROM `student` WHERE `ID` = '2501';
```

结果如下:



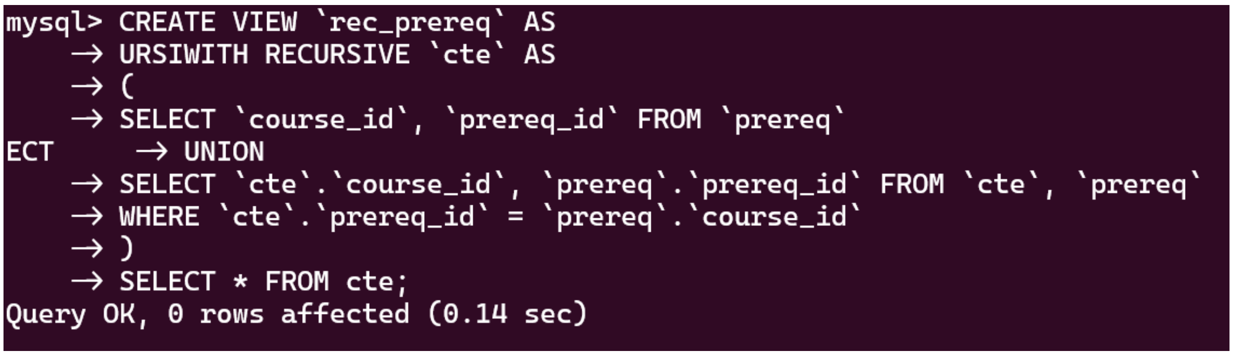
```
mysql> SELECT `tot_cred` FROM `student` WHERE `ID` = '2501';
+----------+
| tot_cred |
+----------+
|       43 |
+----------+
1 row in set (0.02 sec)

mysql> UPDATE `takes` SET `grade` = 'A'
2501' AND `course_id` = '105    -> WHERE `ID` = '2501' AND `course_id` = '105' AND `sec_id` = '3';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT `tot_cred` FROM `student` WHERE `ID` = '2501';
+----------+
| tot_cred |
+----------+
|       46 |
+----------+
1 row in set (0.02 sec)
```

图 24: 测试结果

2.4 SQL 查询进阶

1. 参照教材 217 页 Figure 5.16，使用递归查询语句创建一个名为 rec_prereq 的视图，找出所有课程的直接和间接前导课程；

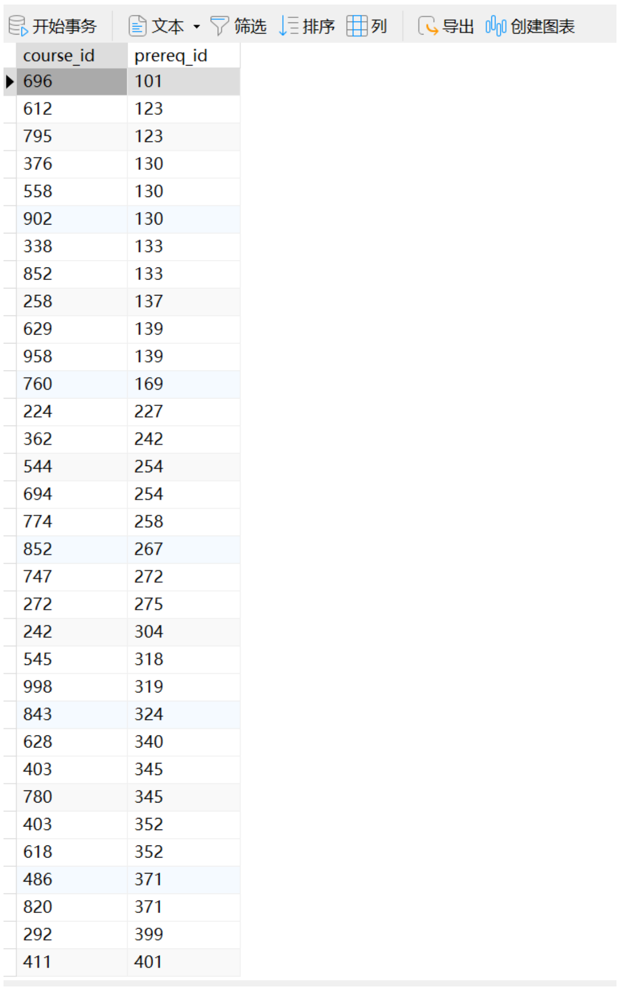
```sql
CREATE VIEW `rec_prereq` AS
WITH RECURSIVE `cte` AS
(
  SELECT `course_id`, `prereq_id` FROM `prereq`
  UNION
  SELECT `cte`.`course_id`, `prereq`.`prereq_id` FROM `cte`, `prereq`
  WHERE `cte`.`prereq_id` = `prereq`.`course_id`
)
SELECT * FROM `cte';
```



```
mysql> CREATE VIEW `rec_prereq` AS
    -> URSIWITH RECURSIVE `cte` AS
    -> (
    -> SELECT `course_id`, `prereq_id` FROM `prereq`
ECT     -> UNION
    -> SELECT `cte`.`course_id`, `prereq`.`prereq_id` FROM `cte`, `prereq`
    -> WHERE `cte`.`prereq_id` = `prereq`.`course_id`
    -> )
    -> SELECT * FROM cte;
Query OK, 0 rows affected (0.14 sec)
```

图 25: 创建视图

视图的内容如图所示:

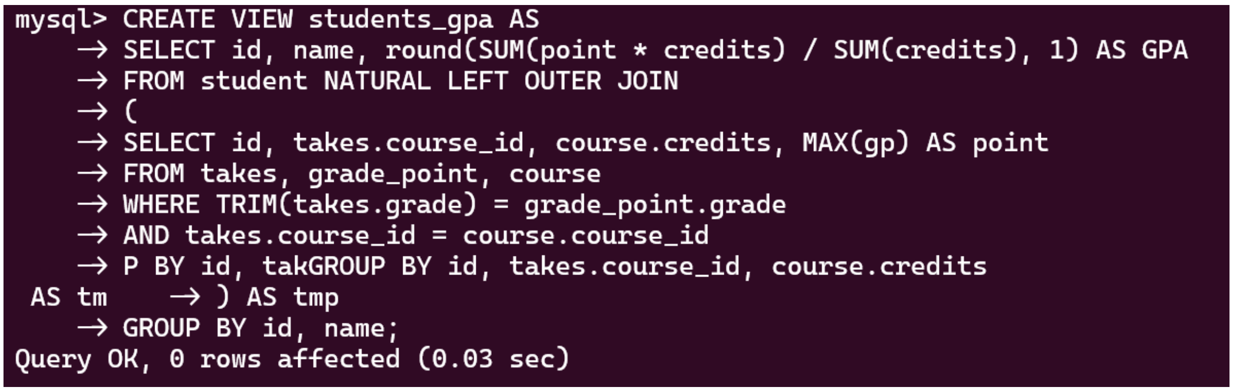


course_id	prereq_id
696	101
612	123
795	123
376	130
558	130
902	130
338	133
852	133
258	137
629	139
958	139
760	169
224	227
362	242
544	254
694	254
774	258
852	267
747	272
272	275
242	304
545	318
998	319
843	324
628	340
403	345
780	345
403	352
618	352
486	371
820	371
292	399
411	401

图 26: 视图内容

2. 执行下列 SQL 语句，创建一个名为 students_gpa 的视图；

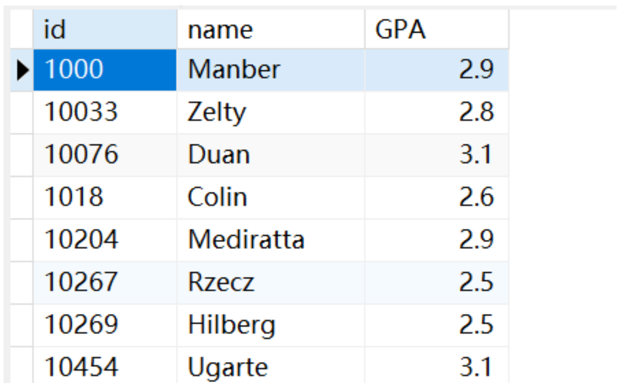
```sql
CREATE VIEW students_gpa AS
SELECT id, name, round(SUM(point * credits) / SUM(credits), 1) AS GPA
FROM student NATURAL LEFT OUTER JOIN
(
  SELECT id, takes.course_id, course.credits, MAX(gp) AS point
  FROM takes, grade_point, course
  WHERE TRIM(takes.grade) = grade_point.grade
  AND takes.course_id = course.course_id
  GROUP BY id, takes.course_id, course.credits
) AS tmp
GROUP BY id, name;
```



```
mysql> CREATE VIEW students_gpa AS
    -> SELECT id, name, round(SUM(point * credits) / SUM(credits), 1) AS GPA
    -> FROM student NATURAL LEFT OUTER JOIN
    -> (
    -> SELECT id, takes.course_id, course.credits, MAX(gp) AS point
    -> FROM takes, grade_point, course
    -> WHERE TRIM(takes.grade) = grade_point.grade
    -> AND takes.course_id = course.course_id
    -> P BY id, takGROUP BY id, takes.course_id, course.credits
 AS tm     -> ) AS tmp
    -> GROUP BY id, name;
Query OK, 0 rows affected (0.03 sec)
```

图 27: 创建视图

视图的内容如图所示：



id	name	GPA
1000	Manber	2.9
10033	Zelty	2.8
10076	Duan	3.1
1018	Colin	2.6
10204	Mediratta	2.9
10267	Rzecz	2.5
10269	Hilberg	2.5
10454	Ugarte	3.1

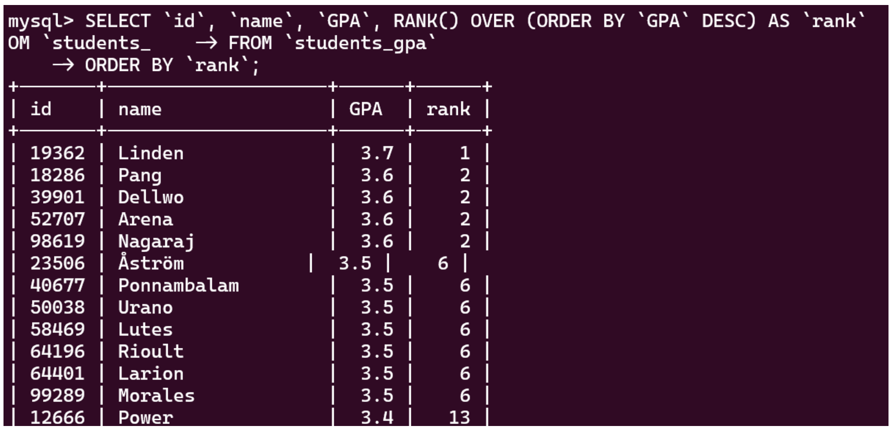
图 28: 视图内容

3. 编写 SQL 语句，按 GPA 的降序列出学生的 ID、姓名、GPA 和排名，结果按 GPA 排名的升序排列；

(a) 当名次出现并列时，下一个名次不连续；

```sql
SELECT `id`, `name`, `GPA`, RANK() OVER (ORDER BY `GPA` DESC) AS `rank`
FROM `students_gpa`
ORDER BY `rank`;
```

结果如下：



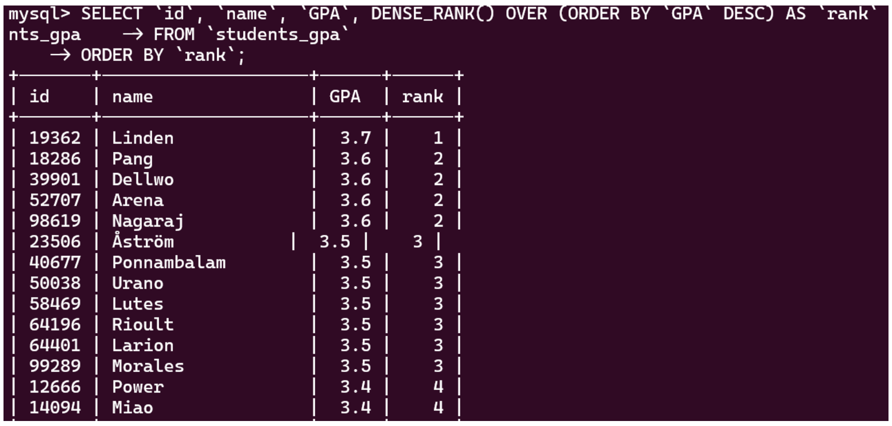
```
mysql> SELECT `id`, `name`, `GPA`, RANK() OVER (ORDER BY `GPA` DESC) AS `rank`
OM `students_     -> FROM `students_gpa`
    -> ORDER BY `rank`;
+-------+--------------------------+------+------+
| id    | name                     | GPA  | rank |
+-------+--------------------------+------+------+
| 19362 | Linden                   |  3.7 |    1 |
| 18286 | Pang                     |  3.6 |    2 |
| 39901 | Dellwo                   |  3.6 |    2 |
| 52707 | Arena                    |  3.6 |    2 |
| 98619 | Nagaraj                  |  3.6 |    2 |
| 23506 | Åström                | 3.5 |    6 |
| 40677 | Ponnambalam              |  3.5 |    6 |
| 50038 | Urano                    |  3.5 |    6 |
| 58469 | Lutes                    |  3.5 |    6 |
| 64196 | Rioult                   |  3.5 |    6 |
| 64401 | Larion                   |  3.5 |    6 |
| 99289 | Morales                  |  3.5 |    6 |
| 12666 | Power                    |  3.4 |   13 |
```

图 29: 查询结果

(b) 当名次出现并列时，下一个名次连续；

```sql
SELECT `id`, `name`, `GPA`, DENSE_RANK() OVER (ORDER BY `GPA` DESC) AS `
    rank`
FROM `students_gpa`
ORDER BY `rank`;
```

结果如下：



```
mysql> SELECT `id`, `name`, `GPA`, DENSE_RANK() OVER (ORDER BY `GPA` DESC) AS `rank`
nts_gpa     -> FROM `students_gpa`
    -> ORDER BY `rank`;
+-------+--------------------------+------+------+
| id    | name                     | GPA  | rank |
+-------+--------------------------+------+------+
| 19362 | Linden                   |  3.7 |    1 |
| 18286 | Pang                     |  3.6 |    2 |
| 39901 | Dellwo                   |  3.6 |    2 |
| 52707 | Arena                    |  3.6 |    2 |
| 98619 | Nagaraj                  |  3.6 |    2 |
| 23506 | Åström                | 3.5 |    3 |
| 40677 | Ponnambalam              |  3.5 |    3 |
| 50038 | Urano                    |  3.5 |    3 |
| 58469 | Lutes                    |  3.5 |    3 |
| 64196 | Rioult                   |  3.5 |    3 |
| 64401 | Larion                   |  3.5 |    3 |
| 99289 | Morales                  |  3.5 |    3 |
| 12666 | Power                    |  3.4 |    4 |
| 14094 | Miao                     |  3.4 |    4 |
```

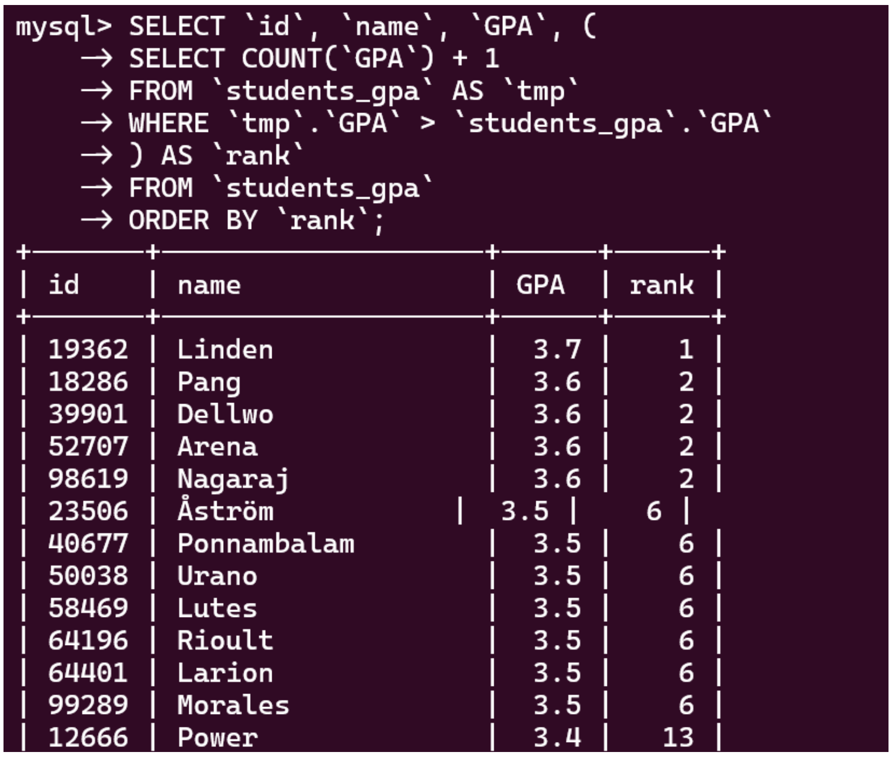
图 30: 查询结果

(c) 不使用 RANK 或 DENSE_RANK 函数;

i. 完成 (a) 的需求

```sql
SELECT `id`, `name`, `GPA`, (
  SELECT COUNT(`GPA`) + 1
  FROM `students_gpa` AS `tmp`
  WHERE `tmp`.`GPA` > `students_gpa`.`GPA`
) AS `rank`
FROM `students_gpa`
ORDER BY `rank`;
```

结果如下:



```
mysql> SELECT `id`, `name`, `GPA`, (
    -> SELECT COUNT(`GPA`) + 1
    -> FROM `students_gpa` AS `tmp`
    -> WHERE `tmp`.`GPA` > `students_gpa`.`GPA`
    -> ) AS `rank`
    -> FROM `students_gpa`
    -> ORDER BY `rank`;
+-------+---------------------+------+------+
| id    | name                | GPA  | rank |
+-------+---------------------+------+------+
| 19362 | Linden              |  3.7 |    1 |
| 18286 | Pang                |  3.6 |    2 |
| 39901 | Dellwo              |  3.6 |    2 |
| 52707 | Arena               |  3.6 |    2 |
| 98619 | Nagaraj             |  3.6 |    2 |
| 23506 | Åström            |  3.5 |    6 |
| 40677 | Ponnambalam         |  3.5 |    6 |
| 50038 | Urano               |  3.5 |    6 |
| 58469 | Lutes               |  3.5 |    6 |
| 64196 | Rioult              |  3.5 |    6 |
| 64401 | Larion              |  3.5 |    6 |
| 99289 | Morales             |  3.5 |    6 |
| 12666 | Power               |  3.4 |   13 |
```

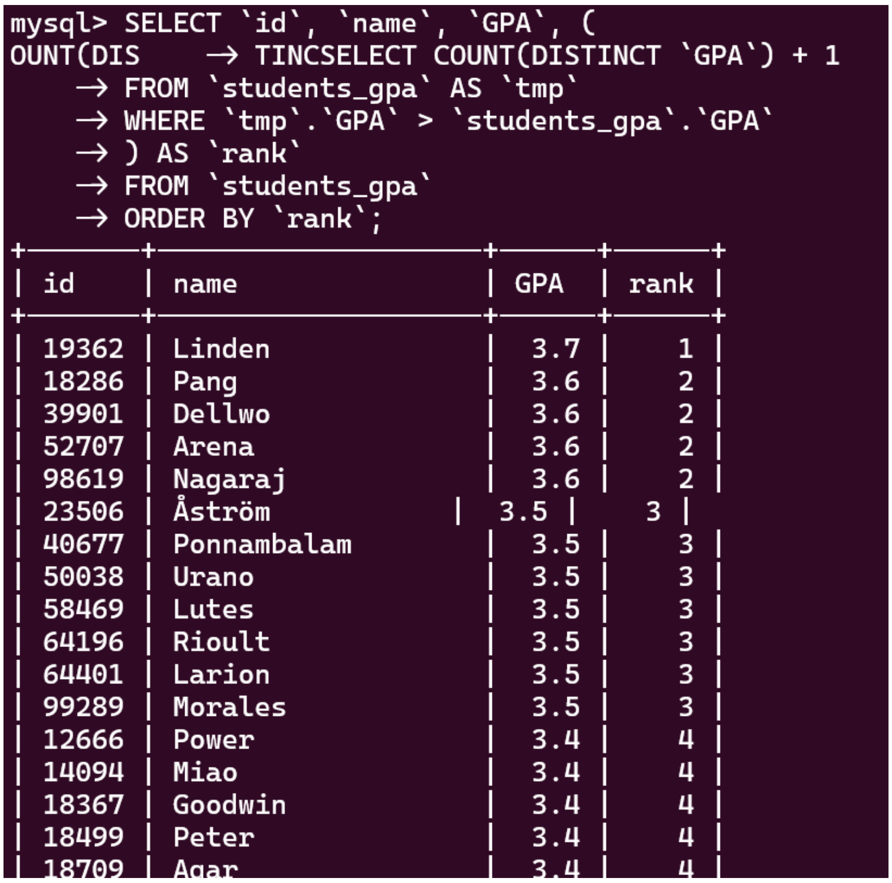
图 31: 查询结果

ii. 完成 (b) 的需求

```sql
SELECT `id`, `name`, `GPA`, (
  SELECT COUNT(DISTINCT `GPA`) + 1
  FROM `students_gpa` AS `tmp`
  WHERE `tmp`.`GPA` > `students_gpa`.`GPA`
) AS `rank`
FROM `students_gpa`
```

```sql
ORDER BY `rank`;
```

结果如下：

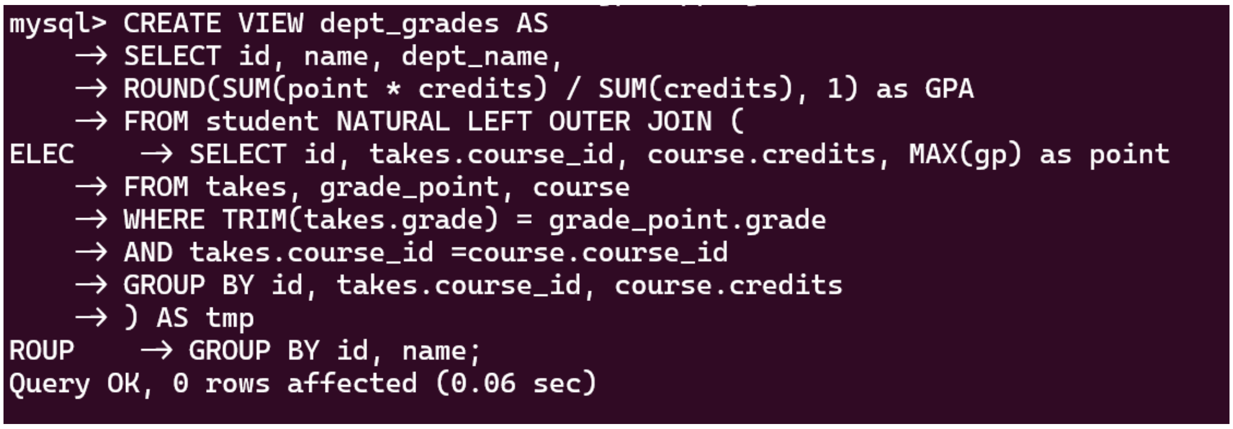


```
mysql> SELECT `id`, `name`, `GPA`, (
OUNT(DIS     -> TINCSELECT COUNT(DISTINCT `GPA`) + 1
    -> FROM `students_gpa` AS `tmp`
    -> WHERE `tmp`.`GPA` > `students_gpa`.`GPA`
    -> ) AS `rank`
    -> FROM `students_gpa`
    -> ORDER BY `rank`;
+-------+---------------------+------+------+
| id    | name                | GPA  | rank |
+-------+---------------------+------+------+
| 19362 | Linden              |  3.7 |    1 |
| 18286 | Pang                |  3.6 |    2 |
| 39901 | Dellwo              |  3.6 |    2 |
| 52707 | Arena               |  3.6 |    2 |
| 98619 | Nagaraj             |  3.6 |    2 |
| 23506 | Åström            |  3.5 |    3 |
| 40677 | Ponnambalam         |  3.5 |    3 |
| 50038 | Urano               |  3.5 |    3 |
| 58469 | Lutes               |  3.5 |    3 |
| 64196 | Rioult              |  3.5 |    3 |
| 64401 | Larion              |  3.5 |    3 |
| 99289 | Morales             |  3.5 |    3 |
| 12666 | Power               |  3.4 |    4 |
| 14094 | Miao                |  3.4 |    4 |
| 18367 | Goodwin             |  3.4 |    4 |
| 18499 | Peter               |  3.4 |    4 |
| 18709 | Agar                |  3.4 |    4 |
```

图 32: 查询结果

4. 执行下列 SQL 语句，创建一个名为 dept_grades 的视图；

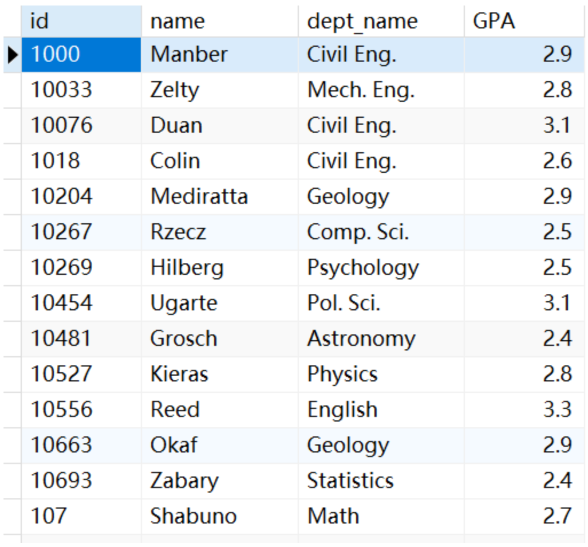
```sql
CREATE VIEW dept_grades AS
SELECT id, name, dept_name,
ROUND(SUM(point * credits) / SUM(credits), 1) as GPA
FROM student NATURAL LEFT OUTER JOIN (
  SELECT id, takes.course_id, course.credits, MAX(gp) as point
  FROM takes, grade_point, course
  WHERE TRIM(takes.grade) = grade_point.grade
  AND takes.course_id =course.course_id
  GROUP BY id, takes.course_id, course.credits
) AS tmp
GROUP BY id, name;
```



```
mysql> CREATE VIEW dept_grades AS
    -> SELECT id, name, dept_name,
    -> ROUND(SUM(point * credits) / SUM(credits), 1) as GPA
    -> FROM student NATURAL LEFT OUTER JOIN (
ELEC    -> SELECT id, takes.course_id, course.credits, MAX(gp) as point
    -> FROM takes, grade_point, course
    -> WHERE TRIM(takes.grade) = grade_point.grade
    -> AND takes.course_id =course.course_id
    -> GROUP BY id, takes.course_id, course.credits
    -> ) AS tmp
ROUP    -> GROUP BY id, name;
Query OK, 0 rows affected (0.06 sec)
```

图 33: 创建视图

视图的内容如图所示：



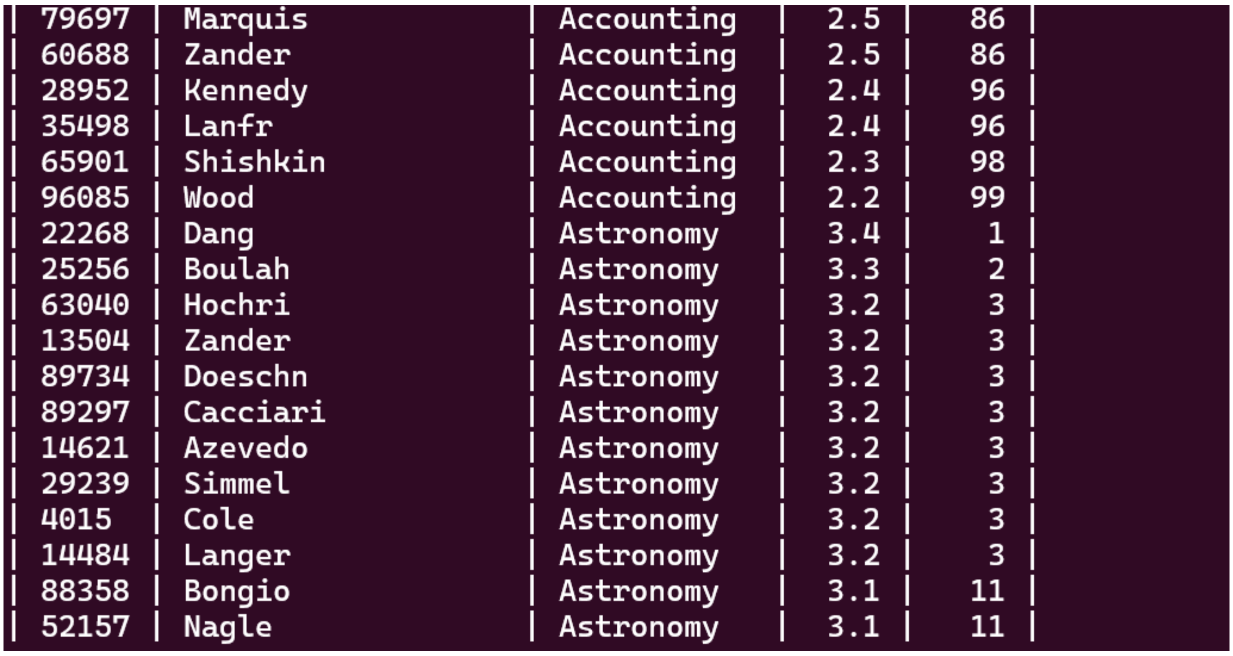
id	name	dept_name	GPA
1000	Manber	Civil Eng.	2.9
10033	Zelty	Mech. Eng.	2.8
10076	Duan	Civil Eng.	3.1
1018	Colin	Civil Eng.	2.6
10204	Mediratta	Geology	2.9
10267	Rzecz	Comp. Sci.	2.5
10269	Hilberg	Psychology	2.5
10454	Ugarte	Pol. Sci.	3.1
10481	Grosch	Astronomy	2.4
10527	Kieras	Physics	2.8
10556	Reed	English	3.3
10663	Okaf	Geology	2.9
10693	Zabary	Statistics	2.4
107	Shabuno	Math	2.7

图 34: 视图内容

5. 编写 SQL 语句，按院系查询学生的 GPA 排名，列出学生的 ID、姓名、院系、GPA 和排名，结果按院系名称和 GPA 排名的升序排列；

```sql
SELECT `id`, `name`, `dept_name`, `GPA`,
RANK() OVER (PARTITION BY `dept_name` ORDER BY `GPA` DESC) AS `rank`
FROM `dept_grades`
ORDER BY `dept_name`, `rank`;
```

结果如下：



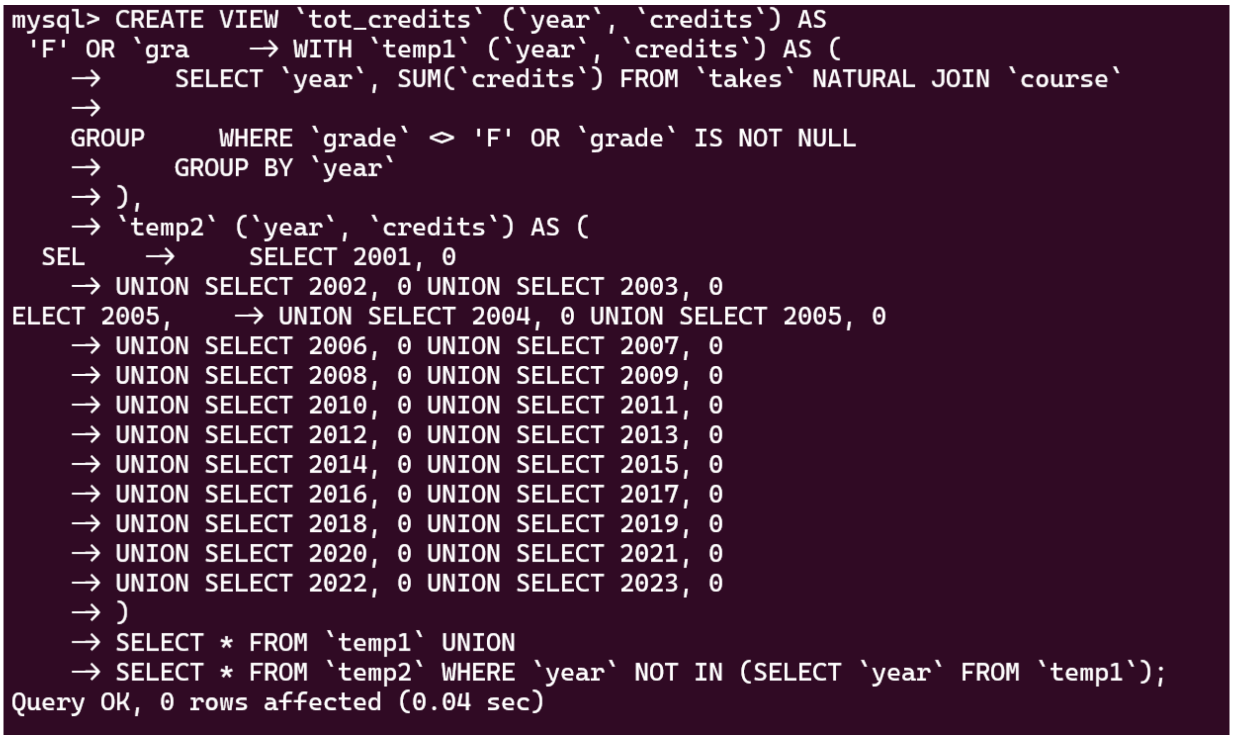
```
| 79697 | Marquis                     | Accounting   |  2.5 |   86 |
| 60688 | Zander                      | Accounting   |  2.5 |   86 |
| 28952 | Kennedy                     | Accounting   |  2.4 |   96 |
| 35498 | Lanfr                       | Accounting   |  2.4 |   96 |
| 65901 | Shishkin                    | Accounting   |  2.3 |   98 |
| 96085 | Wood                        | Accounting   |  2.2 |   99 |
| 22268 | Dang                        | Astronomy    |  3.4 |    1 |
| 25256 | Boulah                      | Astronomy    |  3.3 |    2 |
| 63040 | Hochri                      | Astronomy    |  3.2 |    3 |
| 13504 | Zander                      | Astronomy    |  3.2 |    3 |
| 89734 | Doeschn                     | Astronomy    |  3.2 |    3 |
| 89297 | Cacciari                    | Astronomy    |  3.2 |    3 |
| 14621 | Azevedo                     | Astronomy    |  3.2 |    3 |
| 29239 | Simmel                      | Astronomy    |  3.2 |    3 |
| 4015  | Cole                        | Astronomy    |  3.2 |    3 |
| 14484 | Langer                      | Astronomy    |  3.2 |    3 |
| 88358 | Bongio                      | Astronomy    |  3.1 |   11 |
| 52157 | Nagle                       | Astronomy    |  3.1 |   11 |
```

图 35: 查询结果

6. 创建一个名为 `tot_credits(year, credits)` 的视图，用于统计 2001 年 2023 年期间，每年所有学生所获得的总学分数；

```sql
CREATE VIEW `tot_credits` (`year`, `credits`) AS
WITH `temp1` (`year`, `credits`) AS (
    SELECT `year`, SUM(`credits`) FROM `takes` NATURAL JOIN `course`
    WHERE `grade` <> 'F' OR `grade` IS NOT NULL
    GROUP BY `year`
),
`temp2` (`year`, `credits`) AS (
    SELECT 2001, 0
  UNION SELECT 2002, 0 UNION SELECT 2003, 0
  UNION SELECT 2004, 0 UNION SELECT 2005, 0
  UNION SELECT 2006, 0 UNION SELECT 2007, 0
  UNION SELECT 2008, 0 UNION SELECT 2009, 0
  UNION SELECT 2010, 0 UNION SELECT 2011, 0
  UNION SELECT 2012, 0 UNION SELECT 2013, 0
  UNION SELECT 2014, 0 UNION SELECT 2015, 0
  UNION SELECT 2016, 0 UNION SELECT 2017, 0
  UNION SELECT 2018, 0 UNION SELECT 2019, 0
  UNION SELECT 2020, 0 UNION SELECT 2021, 0
  UNION SELECT 2022, 0 UNION SELECT 2023, 0
)
```

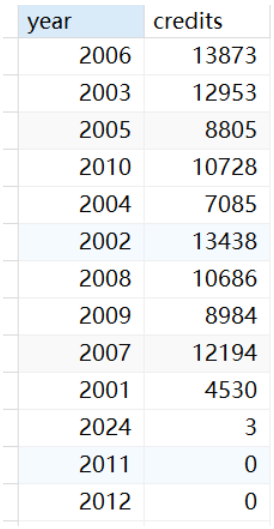
```sql
SELECT * FROM `temp1` UNION
SELECT * FROM `temp2` WHERE `year` NOT IN (SELECT `year` FROM `temp1`);
```



```
mysql> CREATE VIEW `tot_credits` (`year`, `credits`) AS
 'F' OR `gra     -> WITH `temp1` (`year`, `credits`) AS (
    ->     SELECT `year`, SUM(`credits`) FROM `takes` NATURAL JOIN `course`
    ->
    GROUP     WHERE `grade` <> 'F' OR `grade` IS NOT NULL
    ->     GROUP BY `year`
    -> ),
    -> `temp2` (`year`, `credits`) AS (
  SEL    ->     SELECT 2001, 0
    -> UNION SELECT 2002, 0 UNION SELECT 2003, 0
ELECT 2005,    -> UNION SELECT 2004, 0 UNION SELECT 2005, 0
    -> UNION SELECT 2006, 0 UNION SELECT 2007, 0
    -> UNION SELECT 2008, 0 UNION SELECT 2009, 0
    -> UNION SELECT 2010, 0 UNION SELECT 2011, 0
    -> UNION SELECT 2012, 0 UNION SELECT 2013, 0
    -> UNION SELECT 2014, 0 UNION SELECT 2015, 0
    -> UNION SELECT 2016, 0 UNION SELECT 2017, 0
    -> UNION SELECT 2018, 0 UNION SELECT 2019, 0
    -> UNION SELECT 2020, 0 UNION SELECT 2021, 0
    -> UNION SELECT 2022, 0 UNION SELECT 2023, 0
    -> )
    -> SELECT * FROM `temp1` UNION
    -> SELECT * FROM `temp2` WHERE `year` NOT IN (SELECT `year` FROM `temp1`);
Query OK, 0 rows affected (0.04 sec)
```

图 36: 创建视图

视图的内容如图所示:



year	credits
2006	13873
2003	12953
2005	8805
2010	10728
2004	7085
2002	13438
2008	10686
2009	8984
2007	12194
2001	4530
2024	3
2011	0
2012	0

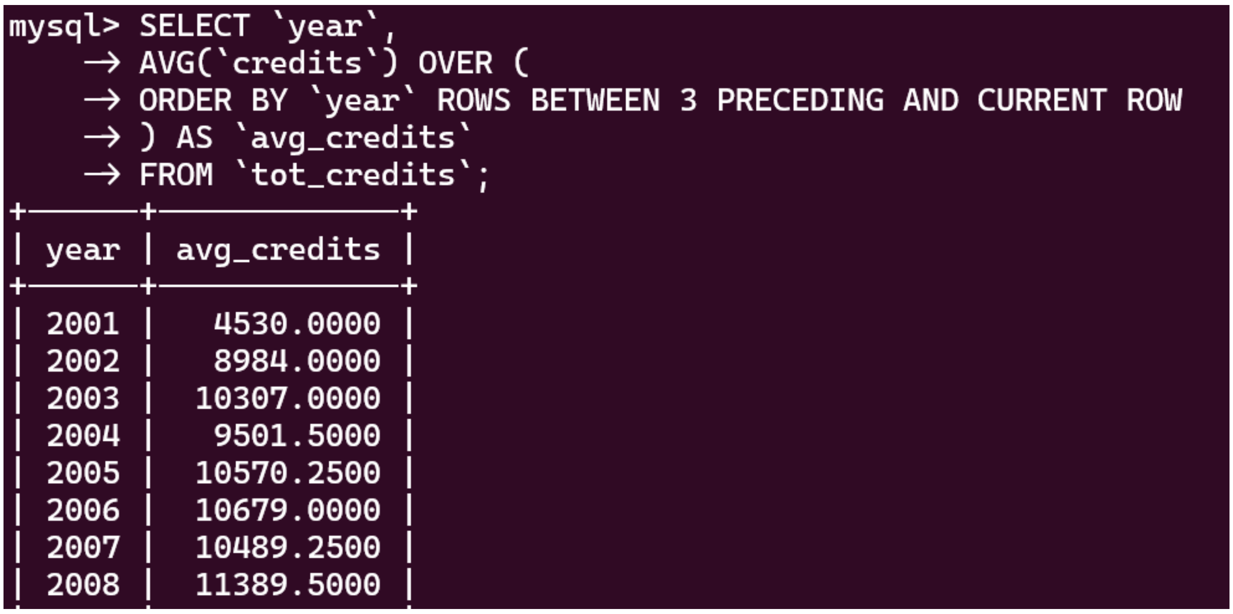
图 37: 视图内容

7. 编写 SQL 语句，使用窗口函数查询不同条件下每年所有学生所获得的总学分数的平均值；

(a) 前三年至今总学分数的平均值；

```sql
SELECT `year`,
AVG(`credits`) OVER (
  ORDER BY `year` ROWS BETWEEN 3 PRECEDING AND CURRENT ROW
) AS `avg_credits`
FROM `tot_credits`;
```

结果如下：



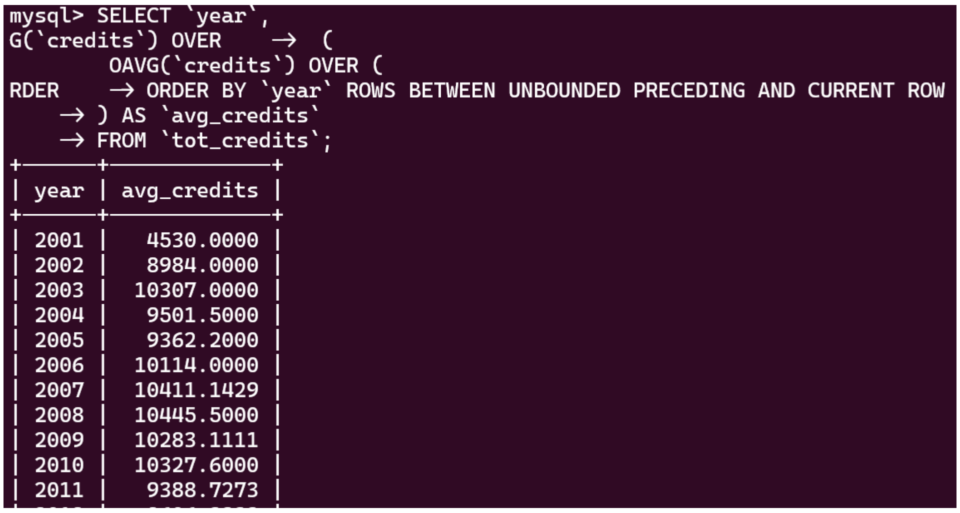
```
mysql> SELECT `year`,
    -> AVG(`credits`) OVER (
    -> ORDER BY `year` ROWS BETWEEN 3 PRECEDING AND CURRENT ROW
    -> ) AS `avg_credits`
    -> FROM `tot_credits`;
+------+-------------+
| year | avg_credits |
+------+-------------+
| 2001 |   4530.0000 |
| 2002 |   8984.0000 |
| 2003 |  10307.0000 |
| 2004 |   9501.5000 |
| 2005 |  10570.2500 |
| 2006 |  10679.0000 |
| 2007 |  10489.2500 |
| 2008 |  11389.5000 |
```

图 38: 查询结果

(b) 从 2001 年起至今总学分数的平均值；

```sql
SELECT `year`,
AVG(`credits`) OVER (
  ORDER BY `year` ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
) AS `avg_credits`
FROM `tot_credits`;
```

结果如下：



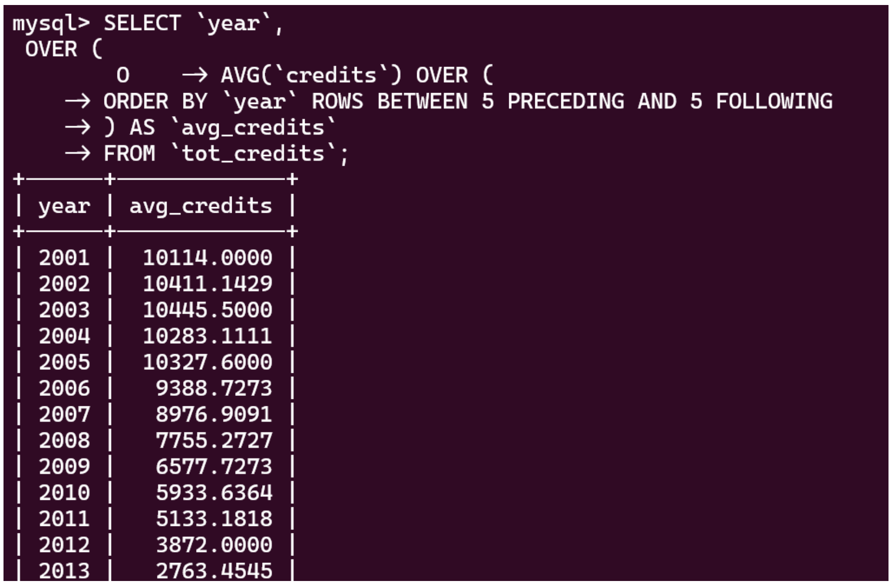
```
mysql> SELECT `year`,
G(`credits`) OVER    -> (
        OAVG(`credits`) OVER (
RDER    -> ORDER BY `year` ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    -> ) AS `avg_credits`
    -> FROM `tot_credits`;
+------+-------------+
| year | avg_credits |
+------+-------------+
| 2001 |   4530.0000 |
| 2002 |   8984.0000 |
| 2003 |  10307.0000 |
| 2004 |   9501.5000 |
| 2005 |   9362.2000 |
| 2006 |  10114.0000 |
| 2007 |  10411.1429 |
| 2008 |  10445.5000 |
| 2009 |  10283.1111 |
| 2010 |  10327.6000 |
| 2011 |   9388.7273 |
```

图 39: 查询结果

(c) 前后五年总学分数的平均值;

```sql
SELECT `year`,
AVG(`credits`) OVER (
  ORDER BY `year` ROWS BETWEEN 5 PRECEDING AND 5 FOLLOWING
) AS `avg_credits`
FROM `tot_credits`;
```

结果如下:

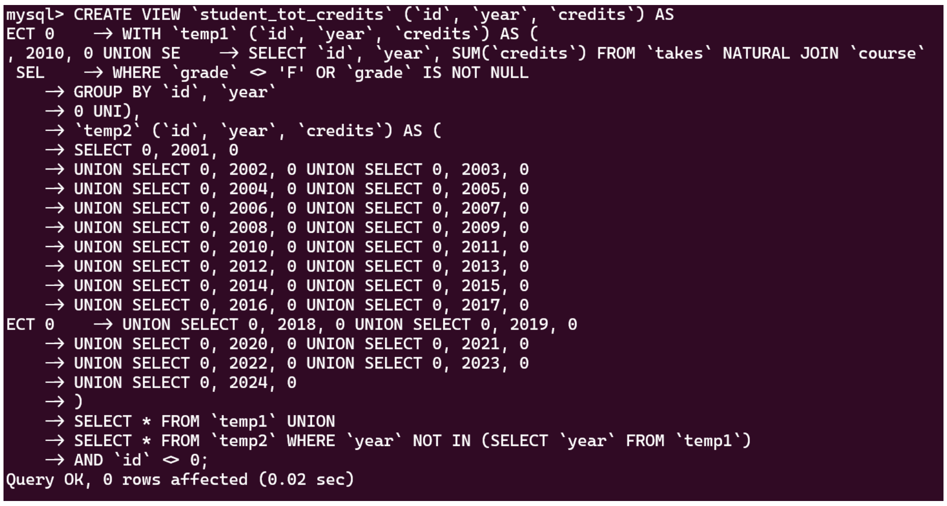


```
mysql> SELECT `year`,
 OVER (
       0     -> AVG(`credits`) OVER (
    -> ORDER BY `year` ROWS BETWEEN 5 PRECEDING AND 5 FOLLOWING
    -> ) AS `avg_credits`
    -> FROM `tot_credits`;
+------+-------------+
| year | avg_credits |
+------+-------------+
| 2001 |  10114.0000 |
| 2002 |  10411.1429 |
| 2003 |  10445.5000 |
| 2004 |  10283.1111 |
| 2005 |  10327.6000 |
| 2006 |   9388.7273 |
| 2007 |   8976.9091 |
| 2008 |   7755.2727 |
| 2009 |   6577.7273 |
| 2010 |   5933.6364 |
| 2011 |   5133.1818 |
| 2012 |   3872.0000 |
| 2013 |   2763.4545 |
```

图 40: 查询结果

8. 创建一个名为 student_tot_credits(id, year, credits) 的视图，用于统计每个学生每年所获得的总学分；

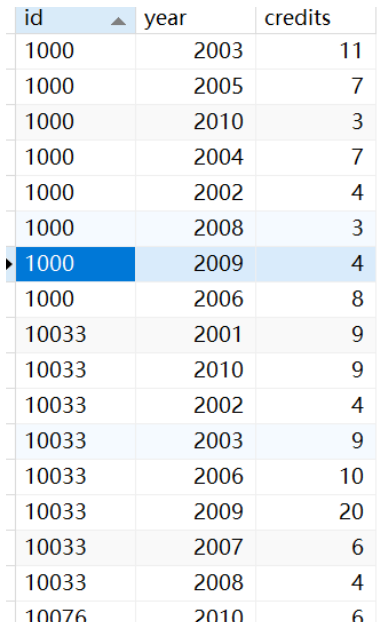
```sql
CREATE VIEW `student_tot_credits` (`id`, `year`, `credits`) AS
WITH `temp1` (`id`, `year`, `credits`) AS (
  SELECT `id`, `year`, SUM(`credits`) FROM `takes` NATURAL JOIN `course`
  WHERE `grade` <> 'F' OR `grade` IS NOT NULL
  GROUP BY `id`, `year`
),
`temp2` (`id`, `year`, `credits`) AS (
  SELECT 0, 2001, 0
  UNION SELECT 0, 2002, 0 UNION SELECT 0, 2003, 0
  UNION SELECT 0, 2004, 0 UNION SELECT 0, 2005, 0
  UNION SELECT 0, 2006, 0 UNION SELECT 0, 2007, 0
  UNION SELECT 0, 2008, 0 UNION SELECT 0, 2009, 0
  UNION SELECT 0, 2010, 0 UNION SELECT 0, 2011, 0
  UNION SELECT 0, 2012, 0 UNION SELECT 0, 2013, 0
  UNION SELECT 0, 2014, 0 UNION SELECT 0, 2015, 0
  UNION SELECT 0, 2016, 0 UNION SELECT 0, 2017, 0
  UNION SELECT 0, 2018, 0 UNION SELECT 0, 2019, 0
  UNION SELECT 0, 2020, 0 UNION SELECT 0, 2021, 0
  UNION SELECT 0, 2022, 0 UNION SELECT 0, 2023, 0
  UNION SELECT 0, 2024, 0
)
SELECT * FROM `temp1` UNION
SELECT * FROM `temp2` WHERE `year` NOT IN (SELECT `year` FROM `temp1`)
AND `id` <> 0;
```



```
mysql> CREATE VIEW `student_tot_credits` (`id`, `year`, `credits`) AS
ECT 0    -> WITH `temp1` (`id`, `year`, `credits`) AS (
, 2010, 0 UNION SE    -> SELECT `id`, `year`, SUM(`credits`) FROM `takes` NATURAL JOIN `course`
 SEL    -> WHERE `grade` <> 'F' OR `grade` IS NOT NULL
    -> GROUP BY `id`, `year`
    -> 0 UNI),
    -> `temp2` (`id`, `year`, `credits`) AS (
    -> SELECT 0, 2001, 0
    -> UNION SELECT 0, 2002, 0 UNION SELECT 0, 2003, 0
    -> UNION SELECT 0, 2004, 0 UNION SELECT 0, 2005, 0
    -> UNION SELECT 0, 2006, 0 UNION SELECT 0, 2007, 0
    -> UNION SELECT 0, 2008, 0 UNION SELECT 0, 2009, 0
    -> UNION SELECT 0, 2010, 0 UNION SELECT 0, 2011, 0
    -> UNION SELECT 0, 2012, 0 UNION SELECT 0, 2013, 0
    -> UNION SELECT 0, 2014, 0 UNION SELECT 0, 2015, 0
    -> UNION SELECT 0, 2016, 0 UNION SELECT 0, 2017, 0
ECT 0    -> UNION SELECT 0, 2018, 0 UNION SELECT 0, 2019, 0
    -> UNION SELECT 0, 2020, 0 UNION SELECT 0, 2021, 0
    -> UNION SELECT 0, 2022, 0 UNION SELECT 0, 2023, 0
    -> UNION SELECT 0, 2024, 0
    -> )
    -> SELECT * FROM `temp1` UNION
    -> SELECT * FROM `temp2` WHERE `year` NOT IN (SELECT `year` FROM `temp1`)
    -> AND `id` <> 0;
Query OK, 0 rows affected (0.02 sec)
```

图 41: 创建视图

视图的内容如图所示：



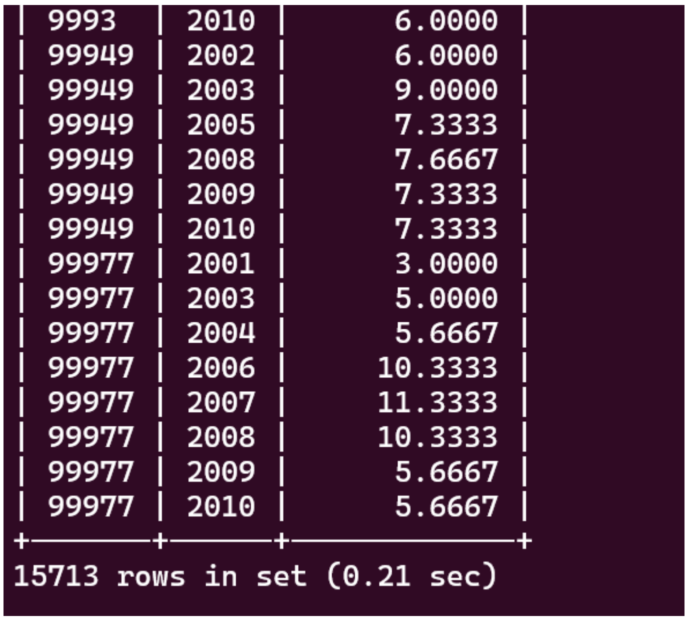
id	year	credits
1000	2003	11
1000	2005	7
1000	2010	3
1000	2004	7
1000	2002	4
1000	2008	3
1000	2009	4
1000	2006	8
10033	2001	9
10033	2010	9
10033	2002	4
10033	2003	9
10033	2006	10
10033	2009	20
10033	2007	6
10033	2008	4
10076	2010	6

图 42: 视图内容

9. 编写 SQL 语句查询每个学生连续三年所获得的平均学分数；

```sql
SELECT `id`, `year`,
AVG(`credits`) OVER (
  PARTITION BY `id` ORDER BY `year` ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
) AS `avg_credits`
FROM `student_tot_credits`;
```

结果如下：



```
| 9993  | 2010 |       6.0000 |
| 99949 | 2002 |       6.0000 |
| 99949 | 2003 |       9.0000 |
| 99949 | 2005 |       7.3333 |
| 99949 | 2008 |       7.6667 |
| 99949 | 2009 |       7.3333 |
| 99949 | 2010 |       7.3333 |
| 99977 | 2001 |       3.0000 |
| 99977 | 2003 |       5.0000 |
| 99977 | 2004 |       5.6667 |
| 99977 | 2006 |      10.3333 |
| 99977 | 2007 |      11.3333 |
| 99977 | 2008 |      10.3333 |
| 99977 | 2009 |       5.6667 |
| 99977 | 2010 |       5.6667 |
+-------+------+--------------+
15713 rows in set (0.21 sec)
```

图 43: 查询结果

3 存在的问题及解决方案

1. 在使用 `INSERT INTO SELECT ...` 语句时频繁报错，原因是 `SELECT` 的表不能是使用 `WITH ...` 语句创建的表。解决方案：在此类情况时避免使用 `WITH ...` 语句；
2. 创建存储过程时自定义变量与列名重合导致错误。解决方案：在创建存储过程时避免使用与列名重合的自定义变量名。

4 实验小结

本次实验主要学习了 MySQL 数据库的高级应用，包括存储过程、触发器、视图和窗口函数等。通过实验，我学会了如何创建存储过程、触发器和视图，以及如何使用窗口函数进行高级查询。这些内容对于提高数据库的查询效率和数据处理能力非常有帮助。同时，实验中还遇到了一些问题，如自定义变量与列名重合导致错误等，通过查阅文档和调试代码，我成功解决了这些问题。总的来说，本次实验收获颇丰，对 MySQL 数据库的高级应用有了更深入的了解。