

# 数据库系统 课程实验2

## 数据库安全性/完整性定义与检查

---

计科210X 甘晴void 202108010XXX

### 目录

数据库系统 课程实验2数据库安全性/完整性定义与检查

实验目的

实验环境

实验内容

#### 2.1数据库安全性

##### 2.1.1 自主存取控制实验

1) 实验内容与要求

2) 实验重难点

3) 实验过程

①预先准备

②创建两个部门经理用户并授予创建用户的权限

③给予部门经理用户其他权限

★激活角色

④使用部门经理创建员工用户

⑤验证员工用户权限

⑥收回员工查看的权限，并验证

##### 2.1.2 审计实验

1) 实验内容与要求

2) 实验重难点

3) 实验过程

①首先查看审计配置情况：

②开启日志

③验证审计

④关闭日志

#### 2.2数据库完整性

##### 2.2.1 实体完整性实验

1) 实验内容与要求

2) 实验重难点

3) 实验过程

①创建新数据库HNUT2

②创建表时定义完整性并验证

③创建表之后定义并验证

#### ④删除实体完整性并验证

### 2.2.2 参照完整性实验

#### 1) 实验内容与要求

#### 2) 实验重难点

#### 3) 实验过程

##### ①创建表时定义参照完整性+验证

##### ②创建表后定义参照完整性

##### ③定义参照完整性的违约处理

##### ④删除参照完整性

### 2.2.3 用户自定义完整性实验

#### 1) 实验内容与要求

#### 2) 实验重难点

#### 3) 实验过程

##### ①建立完整性约束

##### ②验证完整性约束

##### ③删除完整性约束并验证

### 2.3 触发器实验

#### 1) 实验内容与要求

#### 2) 实验重难点

#### 3) 示例

##### ①创建触发器（以BEFORE为例）

##### ★mysql关于触发器的语法区别

##### ②验证触发器（以BEFORE为例）

##### ③验证触发器执行顺序

##### ④删除触发器

实验感悟

## 实验目的

1) 熟悉通过 SQL 对数据库进行安全性控制的方法，其中包括自主存取控制实验和审计实验。

2) 熟悉通过 SQL 对数据库进行完整性控制的方法，其中包括实体完整性、参照完整性、用户自定义完整性。

3) 熟悉并掌握数据库触发器的设计和使用方法。

## 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

命令行: 由Microsoft商店提供的windows终端(version=1.18.2822.0)启动的windows powershell命令行（这个主要是在安全性实验中登录FINANCE用户和U1用户时使用，Navicat自带命令行默认以root登录，不方便这样操作）

## 实验内容

### 2.1数据库安全性

#### 2.1.1 自主存取控制实验

##### 1) 实验内容与要求

定义用户、角色，分配权限给用户、角色，回收权限，以相应的用户名登录数据库验证权限分配是否正确。选择一个应用场景，使用自主存取控制机制设计权限分配。可以采用两种方案：

方案一：采用 **SYSTEM** 超级用户登录数据库，完成所有权限分配工作，然后用相应用户名登陆数据库以验证权限分配正确性；

方案二：采用 **SYSTEM** 用户登陆数据库创建两个部门经理用户，并分配相应的权限，然后分别用两个经理用户名登陆数据库，创建相应部门的 **USER, ROLE**，并分配相应权限。验证权限分配之前，请备份好数据库；针对不同用户所具有的权限，分别设计相应的 **SQL**语句加以验证。

**【注意】**这两种方式是参考文档提供给我们的，不一定需要照抄方案。

我准备采用自己的方案来进行操作。

## 2) 实验重难点

定义角色，分配权限和回收权限，实现权限的再分配与回收。

## 3) 实验过程

我们实验题目提供的示例。过程如下

### ① 预先准备

创建数据库 TEST\_COMPANY，包含 salary 工资表、employee 员工表。

根据题目提供的参考示例

Salary ( <u>name</u> , salarise)	
name 姓名	Salarise 工资
刘星	3000
刘晨	3000
张三	3000
李勇	5000
李四	3000
王明	3000

Employee (name, <u>number</u> , dept)		
Name 姓名	Number 员工号	Dept 职位
刘晨	1	Zhigong (职工)
刘星	2	Zhigong
张三	3	Zhigong
李四	4	Zhigong
王明	5	Zhigong
李勇	6	Daiban (代班)

构建好数据库和相应的数据集

文件	编辑	查看	窗口	帮助
开始事务	文本	筛选	排序	列
导入	导出	数据生成	创建图表	
NAME	NUMBER	DEPT		
刘晨	1	zhigong		
刘星	2	zhigong		
张三	3	zhigong		
李四	4	zhigong		
王明	5	zhigong		
李勇	6	daiban		

文件	编辑	查看	窗口	帮助
开始事务	文本	筛选	排序	列
导入	导出	数据生成	创建图表	
NAME	SALARIES			
刘星	3000			
刘晨	3000			
张三	3000			
李勇	5000			
李四	3000			
王明	3000			

整体思路：

使用 **SYSTEM** 超级用户登录数据库创建两个部门经理用户，分配相应权限，然后使用经理用户名登录数据库创建相应部门的 **USER**、**ROLE**，分配相应权限。

- 创建用户财务部经理**FINANCE**和人事部经理**HR**
- 用户**FINANCE**管理工资表，拥有查询、删除、修改工资的权限
- 用户**HR**管理员工表，拥有删除员工、添加员工、更新员工信息的权限

②创建两个部门经理用户并授予创建用户的权限

先创建两个用户并给与他们创建用户的权限。

```
mysql> CREATE USER 'FINANCE'@'%' IDENTIFIED BY '111111';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT CREATE USER ON *.* TO 'FINANCE'@'%';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER 'HR'@'%' IDENTIFIED BY '222222';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT CREATE USER ON *.* TO 'HR'@'%';
Query OK, 0 rows affected (0.00 sec)
```

此时这两个用户只有创建用户的权限，如果它们想查看表，甚至访问数据库都会报错。

```
mysql> USE TEST_COMPANY
ERROR 1044 (42000): Access denied for user 'FINANCE'@'%' to
database 'test_company'
```

### ③给予部门经理用户其他权限

I 给予两位部门经理用户更新相应表的权限。创建两个角色，然后给这两个角色分别赋予其对应的权限，并将角色权限给用户，同时加上 **with grant option** 选项，方便其用户赋予员工查询权限。然后将这两个角色赋予部门经理。

```
CREATE ROLE FIN_ROLE;
GRANT SELECT,DELETE,UPDATE,INSERT ON TABLE SALARY TO FIN_ROLE WITH
GRANT OPTION;
GRANT FIN_ROLE TO FINANCE WITH ADMIN OPTION;

CREATE ROLE HR_ROLE;
GRANT SELECT,DELETE,UPDATE,INSERT ON TABLE EMPLOYEE TO HR_ROLE;
GRANT HR_ROLE TO HR WITH ADMIN OPTION;
```

II 这样子仍然会出现问题，两位部门进入数据库还会受到阻拦，即使使用navicat可视化查看其权限，仍然显示不具备该权限。

经杨老师提醒，问题在于GRANT XXX ON TABLE时，具体的TABLE并没有指定该表所在的数据库，因此是失败的。我们需要在授权的同时指定其表所在的数据库，如TEST\_COMPANY.SALARY。

III 结果仍然发现ROLE上被赋予的权限并没有到达FINANCE上，说明这可能不是主要的问题。

★经过刘lq同学提醒，这个主要问题在于mysql与课本内容的一个区别：激活角色。

### ★激活角色

一个角色被创造出来并被赋予用户之后，事实上用户并没有立刻获得这个“角色”的权限，因为该角色并没有被激活。只有角色被激活，该用户才能够成功“扮演”这个角色。其实这也是为了提升安全性。

使用

```
SELECT CURRENT_ROLE();
```

可以查看当前已经被激活的角色。

有两种激活角色的方法：

```
#非永久激活
set default role all to FINANCE@'%';

#永久激活
SET global activate_all_roles_on_login=ON;
#由于mysql默认禁止了角色自动激活，我们修改这个系统变量可以将其调整为允许角色自动激活，从此一劳永逸解决这个问题。
```

我们可以采取第一种方式

```
mysql> select current_role();
+-----+
| current_role() |
+-----+
| NONE           |
+-----+
1 row in set (0.03 sec)

mysql> set default role all to FINANCE@'%';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

然后退出该命令行，以FINANCE的身份进入mysql数据库（这里我使用win提供的shell终端进行操作）

```
(base) PS C:\Users\y\Desktop> mysql -u FINANCE -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25
Server version: 8.0.33 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 state.

mysql> select current_role();
+-----+
| current_role() |
+-----+
| 'FIN_ROLE'@'%' |
+-----+
1 row in set (0.00 sec)

mysql> |
```

现在可以发现该部门经理FINANCE获得了该角色。

IV 接下来重新对该角色进行授权，授权会自动转交到FINANCE上。

```
GRANT FIN_ROLE TO FINANCE WITH ADMIN OPTION;
```

现在该部门经理再请求进入数据库，就不会受到阻拦了。

```
mysql> use test_company
Database changed
mysql>
```

V 但是这还是有点烦，不能一劳永逸解决问题，所以采用修改系统变量永久激活角色自动激活的功能。

```
SET global activate_all_roles_on_login=ON;
```

之后应该就不会再遇到这种问题了。



#### ④使用部门经理创建员工用户

在 FINANCE 财务部经理下创建员工用户，给与查询权限：

```
CREATE USER U1 IDENTIFIED BY "111111";  
GRANT SELECT ON TABLE SALARY TO U1;
```

这创建了一个员工U1，并有SELECT权限。

接下来进行验证。

#### ⑤验证员工用户权限

验证如下：

重新开启另一个终端，使用用户U1登录

```
use test_company;  
select * from salary;  
INSERT INTO SALARY VALUES('甘晴',3000);
```

运行截图如下：

```
Windows PowerShell
安装最新的 PowerShell, 了解新功能和改进! https://aka.ms/PSWindows

加载个人及系统配置文件用了 971 毫秒。
(base) PS C:\Users\y\Desktop> mysql -u U1 -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 27
Server version: 8.0.33 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> use test_company;
Database changed
mysql> select * from salary;
+-----+-----+
| NAME | SALARIES |
+-----+-----+
| 刘星 |      3000 |
| 刘晨 |      3000 |
| 张三 |      3000 |
| 李勇 |      5000 |
| 李四 |      3000 |
| 王明 |      3000 |
+-----+-----+
6 rows in set (0.01 sec)

mysql> INSERT INTO SALARY VALUES('甘晴',3000);
ERROR 1142 (42000): INSERT command denied to user 'U1'@'localhost' for table 'salary'
```

可以看到用户 U1 仅有查询权限，不具有修改权限。完全符合预期。

## ⑥收回员工查看的权限，并验证

使用FINANCE登录sql并收回员工U1的权限

```
REVOKE SELECT ON SALARY FROM U1;
```

收回权限后，使用U1登录sql试图查询工资，被拒绝。

如下：

```
Windows PowerShell X Windows PowerShell X + v
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 27
Server version: 8.0.33 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> use test_company;
Database changed
mysql> select * from salary;
+-----+-----+
| NAME | SALARIES |
+-----+-----+
| 刘星 | 3000 |
| 刘晨 | 3000 |
| 张三 | 3000 |
| 李勇 | 5000 |
| 李四 | 3000 |
| 王明 | 3000 |
+-----+-----+
6 rows in set (0.01 sec)

mysql> INSERT INTO SALARY VALUES('甘晴',3000);
ERROR 1142 (42000): INSERT command denied to user 'U1'@'localhost' for table 'salary'
mysql> ^C
mysql> select * from salary;
ERROR 1142 (42000): SELECT command denied to user 'U1'@'localhost' for table 'salary'
mysql> |
```

## 2.1.2 审计实验

### 1) 实验内容与要求

掌握数据库审计的设置和管理方法，以便监控数据库操作，维护数据库安全。打开数据库审计开关，以具有审计权限的用户登录数据库，设置审计权限，然后以普通用户登录数据库，执行相应的 SQL 语句，验证审计设置是否有效，最后再以具有审计权限的用户登录数据库，查看是否存在相应的审计信息。

### 2) 实验重难点

合理的设置各种审计信息，一方面，为了保护系统重要的敏感数据，需要系统地设置各种审计信息，不能留有漏洞，以便随时监督系统使用情况，一旦出现问题，也便于追查;另一方面，审计信息设置过多，会严重影响数据库的使用性能，因此需要合理设置。

### 3) 实验过程

MySQL不支持语句级审计，只能对所有的SQL使用进行日志记录。

①首先查看审计配置情况：

```
show variables like '%general_log%';
```

结果如下：



```
mysql> show variables like '%general_log%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| general_log   | OFF   |
| general_log_file | LAPTOP-S8GDLRKI.log |
+-----+-----+
2 rows in set (0.03 sec)

mysql>
```

可以看到当前日志的状态（默认是关闭的），以及日志的名称与保存地址（这里没有显示路径，花了点时间查找）

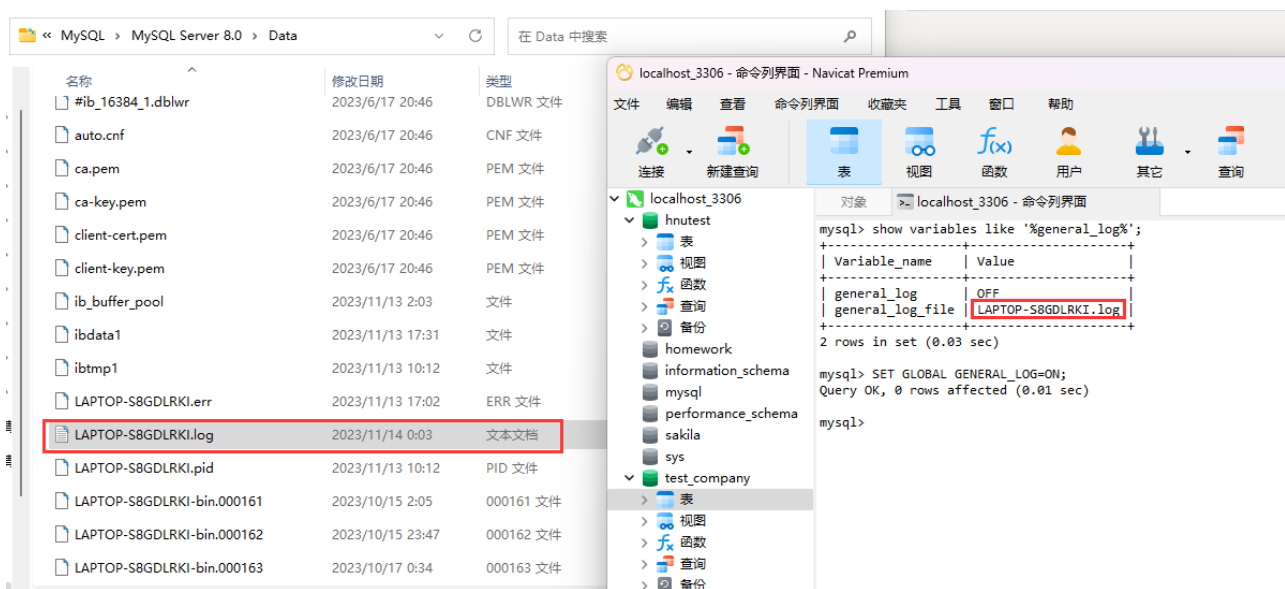
在首次未打开审计之前，文件夹内是没有该log文件的。

路径在E:\MySQL\MySQL Server 8.0\Data\LAPTOP-S8GDLRKI.log

②开启日志

```
SET GLOBAL GENERAL_LOG=ON;
```

使用该命令打开审计之后，文件夹内可以找到该log文件。



### ③验证审计

以没有审计权限的用户进行操作，以上一个实验的部门经理FINANCE为例。先验证该用户并没有有审计的权限，然后进行SELECT操作。

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

加载个人及系统配置文件用了 1193 毫秒。
(base) PS C:\Users\y> mysql -u FINANCE -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 32
Server version: 8.0.33 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> SET GLOBAL GENERAL_LOG=ON;
ERROR 1227 (42000): Access denied; you need (at least one of) the SUPER or SYSTEM_VARIABLES_ADMIN privilege(s) for this
operation
mysql> USE TEST_COMPANY;
Database changed
mysql> SELECT * FROM SALARY;
+-----+-----+
| NAME | SALARIES |
+-----+-----+
| 刘星 | 3000 |
| 刘晨 | 3000 |
| 张三 | 3000 |
| 李勇 | 5000 |
| 李四 | 3000 |
| 王明 | 3000 |
+-----+-----+
6 rows in set (0.00 sec)

mysql>
```

验证该用户没有审计权限

然后我们打开log文件查看。

```
LAPTOP-S8GDLRKL.log
文件 编辑 查看

E:\MySQL\MySQL Server 8.0\bin\mysqld.exe, Version: 8.0.33 (MySQL Community Server - GPL). started with:
TCP Port: 3306, Named Pipe: MySQL
Time          Id Command      Argument
2023-11-13T16:13:46.319043Z 31 Connect FINANCE@localhost on using SSL/TLS
2023-11-13T16:13:46.320243Z 31 Query select @@version_comment limit 1
2023-11-13T16:13:52.162625Z 31 Query SET GLOBAL GENERAL_LOG=ON
2023-11-13T16:14:47.859593Z 32 Connect FINANCE@localhost on using SSL/TLS
2023-11-13T16:14:47.860191Z 32 Query select @@version_comment limit 1
2023-11-13T16:14:54.692066Z 32 Query SET GLOBAL GENERAL_LOG=ON
2023-11-13T16:15:15.243079Z 32 Query SELECT DATABASE()
2023-11-13T16:15:15.243407Z 32 Init DB test_company
2023-11-13T16:15:25.565424Z 32 Query SELECT * FROM SALARY
```

发现该用户的操作确实被记录下来了。

需要指出的是，不同数据库对审计的支持都存在不同。这里只是把审计当作了一个概念，即记录下用户操作的这样一个动作。因此，在这里我们实际上是用日志这个具体的文件去实现了审计这个概念。实际上审计还有其他插件可以支持。

但可以料想得到的是，如果所有操作都被记录下来，那么这个日志文件会变得越来越大，这也是需要考虑的，究竟哪些步骤需要被记录，谁的操作需要被记录，都值得思考，而不是记录下所有的操作。

#### ④关闭日志

为了防止占太多空间，我选择暂时关闭日志。

```
SET GLOBAL GENERAL_LOG=OFF;
```

## 2.2数据库完整性

以下实验将不用TEST\_COMPANY表。

### 2.2.1 实体完整性实验

#### 1) 实验内容要求

定义实体完整性，删除实体完整性。能够写出两种方式定义实体完整性的 SQL 语句：创建表时定义实体完整性、创建表后定义实体完整性。设计SQL 语句验证完整性约束是否起作用。

## 2) 实验重难点

创建表时定义实体完整性，有多个候选码时实体完整性的定义。

## 3) 实验过程

### ①创建新数据库HNUT2

```
CREATE DATABASE HNUT2;
```

### ②创建表时定义完整性并验证

创建一个学生表，属性包括：学号、姓名、性别、年龄、所在班级，其中学号、

(姓名，所在班级)为候选码，此处设计后者作为候选码，均不允许为空值，性别只能是男或女，年龄设置小于 25。

Student (sno, name, sex, age, class\_num)，其中 (name, class\_num)


做为主码。

```
CREATE TABLE student(  
    Sno CHAR(9) UNIQUE NOT NULL,  
    Sname CHAR(20) ,  
    Ssex CHAR(2) CHECK(Ssex = '男' OR Ssex = '女'),  
    Ssage SMALLINT CHECK(Ssage < 25),  
    Sclass CHAR(10),  
    PRIMARY KEY(Sname,Sclass)  
);
```

验证完整性约束是否生效：插入一条学生记录，设置年龄为 26：会因为年龄不符合小于 25 的限制而插入失败。

```
INSERT INTO STUDENT VALUES('202108010','甘晴void','男',26,'CS2');
```

因为不符合完整性约束，插入的数据被拒绝。



The screenshot shows a MySQL command-line interface window titled 'localhost\_3306 - 命令列界面'. The user is logged in as 'course @hnutest (localhost\_3306)'. The command entered is `mysql> INSERT INTO STUDENT VALUES('202108010','甘晴void','男',26,'CS2');`. The response from the database is `3819 - Check constraint 'student_chk_2' is violated.`, followed by a prompt `mysql> |`.

### ③创建表之后定义并验证

```
CREATE TABLE Course(  
    Cno CHAR(9),  
    Cname CHAR(20),  
    Cpno CHAR(9),  
    Ctype CHAR(5),  
    Cdept CHAR(20),  
    Chours SMALLINT,  
    Ccredit SMALLINT  
);  
ALTER TABLE Course  
ADD CONSTRAINT C1 PRIMARY KEY(Cno);  
ALTER TABLE Course  
ADD CONSTRAINT C2 CHECK(Cno like "GE%");
```

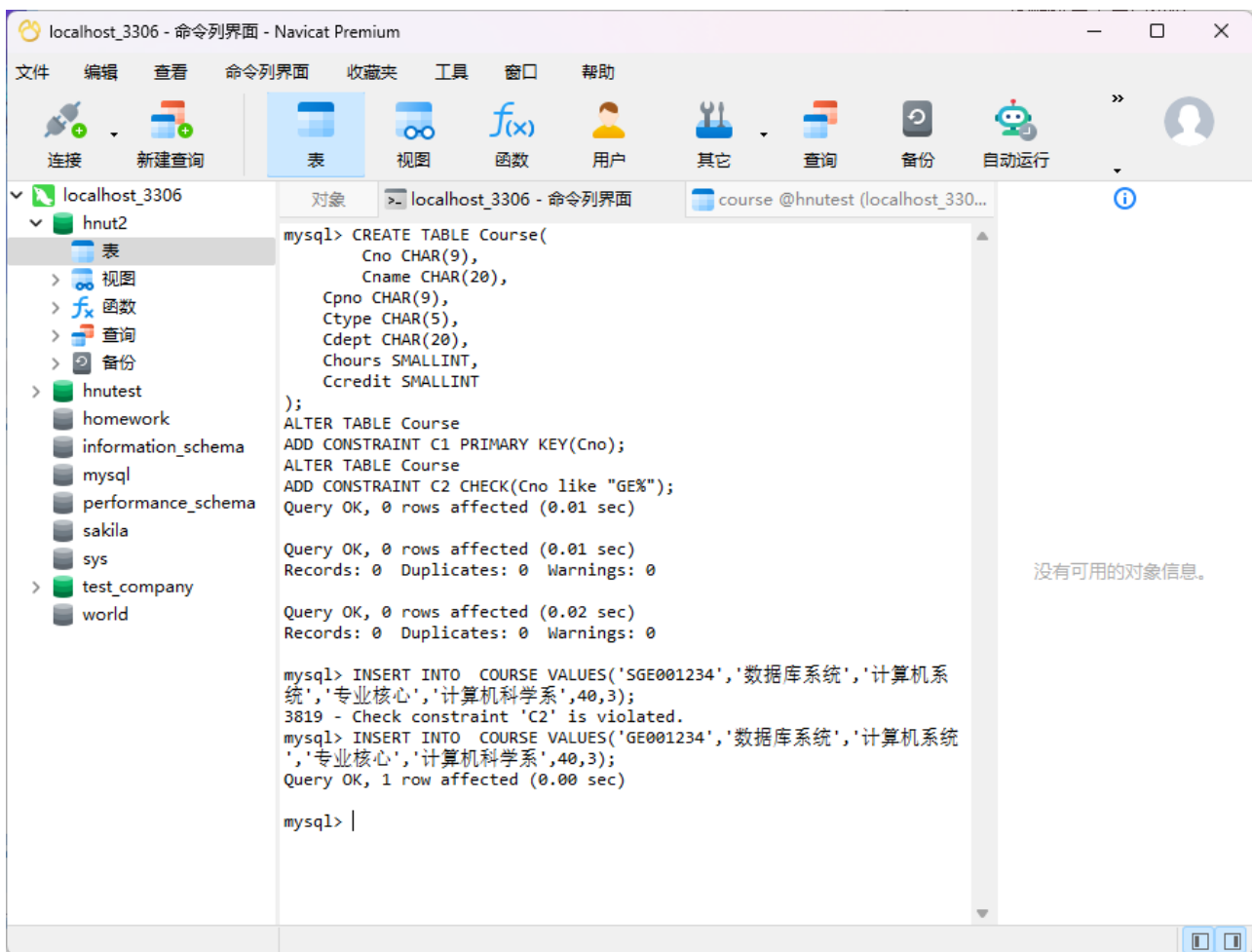
创建表之后定义完整性，限制课程号只能是类似"GE%"形式的。

验证完整性约束是否生效：插入2条课程记录,期中一条不合法，一条合法。

```
INSERT INTO COURSE VALUES('SGE001234','数据库系统','计算机系统','专业核  
心','计算机科学系',40,3);  
  
INSERT INTO COURSE VALUES('GE001234','数据库系统','计算机系统','专业核  
心','计算机科学系',40,3);
```

验证截图如下，符合预期。





#### ④删除实体完整性并验证

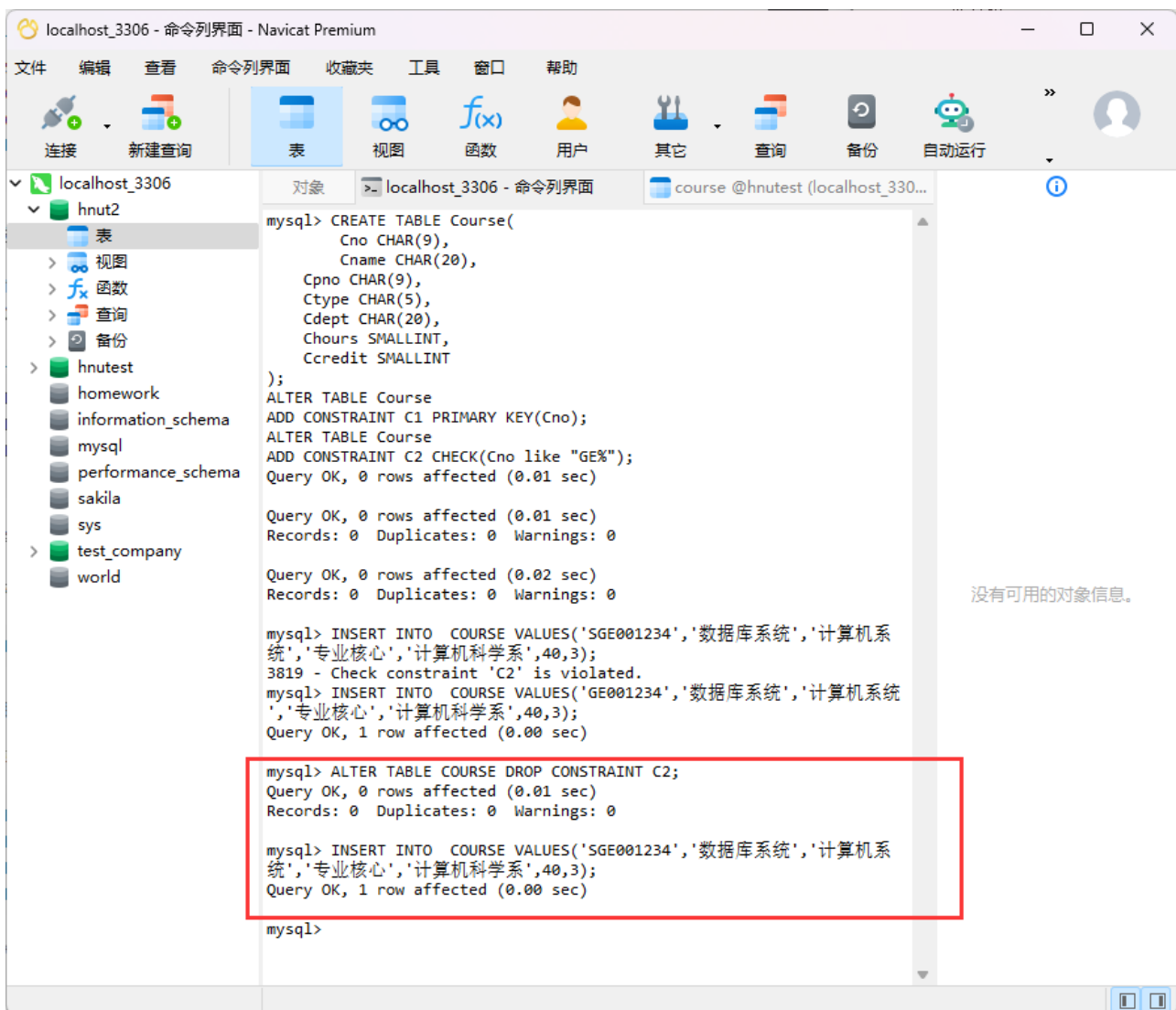
如果我们此时把限制课程号只能是类似"GE%"形式的这个实体完整性C2给删去。使用如下指令

```
ALTER TABLE COURSE DROP CONSTRAINT C2;
```

再次插入这条本来非法的记录。

```
INSERT INTO COURSE VALUES('SGE001234','数据库系统','计算机系统','专业核心','计算机科学系',40,3);
```

截图如下：



## 2.2.2 参照完整性实验

### 1) 实验内容与要求

定义参照完整性，定义参照完整性的违约处理，删除参照完整性。写出两种方式定义参照完整性的 SQL 语句：创建表时定义参照完整性、创建表后定义参照完整性。

### 2) 实验重难点

创建表时定义参照完整性，参照完整性的违约处理定义。

### 3) 实验过程

#### ①创建表时定义参照完整性+验证

创建一个班级表，属性包括姓名，学号，成绩，其中学号作为主码，姓名作为student 表的外码。

```
CREATE TABLE Class(  
    Sname CHAR(10),  
    Sno CHAR(9) PRIMARY KEY,  
    Sgrade SMALLINT,  
    FOREIGN KEY(Sname) REFERENCES STUDENT(SNAME)  
);
```

验证：

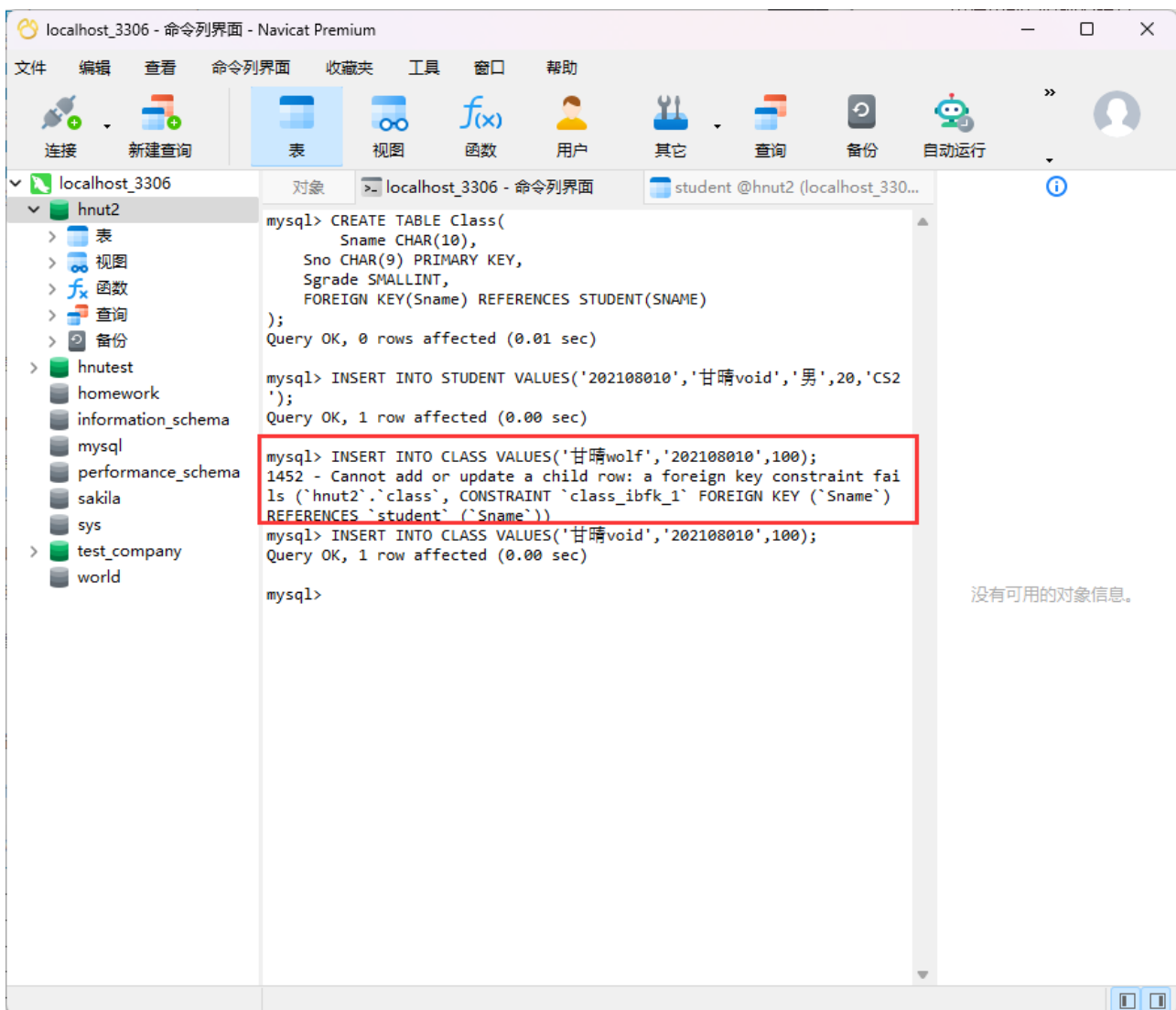
先往STUDENT表中成功加入一个学生。

```
INSERT INTO STUDENT VALUES('202108010', '甘晴void', '男', 20, 'CS2');
```

然后分别试图向Class表中加入一个不合法的学生和一个合法的学生。

```
INSERT INTO CLASS VALUES('甘晴wolf', '202108010', 100);  
INSERT INTO CLASS VALUES('甘晴void', '202108010', 100);
```

截图如下：



可见不合法的记录因为不满足外键要求而被拒绝。

## ②创建表后定义参照完整性

```
CREATE TABLE class(
    Sname CHAR(10),
    Sno CHAR(9) PRIMARY KEY,
    Sgrade SMALLINT
);
ALTER TABLE CLASS
ADD CONSTRAINT C3 FOREIGN KEY(Sname) REFERENCES STUDENT(SNAME);
```

验证同上，结果符合预期。

### ③定义参照完整性的违约处理

删除 **student** 表中内容时级联删除 **class** 表中相应内容。

一般的违约处理都是拒绝执行。教材指出，若想让系统采取其他策略，必须在创建参照表的时候显式地加以说明。

```
CREATE TABLE class(  
    Sname CHAR(10),  
    Sno CHAR(9) PRIMARY KEY,  
    Sgrade SMALLINT,  
    FOREIGN KEY(Sname) REFERENCES STUDENT(SNAME)  
        ON UPDATE CASCADE  
        ON DELETE CASCADE  
);
```

验证外键在delete上的级联是否生效：

在被参考表STUDENT表插入一条合法学生数据，然后子在参考表插入该学生的成绩信息。

```
INSERT INTO STUDENT VALUES('202108010','甘晴void','男',20,'CS2');  
INSERT INTO CLASS VALUES('甘晴void','202108010',100);
```

对象 localhost\_3306 - 命令列界面

```
mysql> INSERT INTO STUDENT VALUES('202108010','甘晴void','男',20,'CS2');
INSERT INTO CLASS VALUES('甘晴void','202108010',100);
Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

mysql>
```

hnut2 数据库  
localhost\_3306

class @hnut2 (localhost\_3306) - 表

Sname	Sno	Sgrade
甘晴void	202108010	100

student @hnut2 (localhost\_3306) - 表

Sno	Sname	Ssex	Ssage	Sclass
202108010	甘晴void	男	20	CS2

接着在STUDENT表删除该学生信息，查看CLASS表是否随之发生变化。

```
DELETE FROM STUDENT WHERE SNAME='甘晴void';
```

SQL 语句进行验证违约处理是否生效。

The screenshot shows a MySQL database management tool interface. At the top, there's a navigation bar with icons for '表' (Table), '视图' (View), '函数' (Function), '用户' (User), '其它' (Other), '查询' (Query), '备份' (Backup), and '自动运行' (Auto Run). Below this, the '对象' (Object) pane shows 'localhost\_3306 - 命令列界面' (Command Line Interface). The main SQL editor shows the following commands:

```
mysql> INSERT INTO STUDENT VALUES('202108010','甘晴void','男',20,'CS2');
INSERT INTO CLASS VALUES('甘晴void','202108010',100);
Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

mysql>
```

On the right, the 'hnut2 数据库' (hnut2 Database) is shown, connected to 'localhost\_3306'.

Below the SQL editor, two table views are displayed:

**class @hnut2 (localhost\_3306) - 表**

Sname	Sno	Sgrade
甘晴void	202108010	100

**student @hnut2 (localhost\_3306) - 表**

Sno	Sname	Ssex	Ssage	Sclass
202108010	甘晴void	男	20	CS2

#### ④删除参照完整性

删除参照完整性

```
ALTER TABLE Course DROP CONSTRAINT C3;
```

## 2.2.3 用户自定义完整性实验

### 1) 实验内容与要求

针对具体应用语义，选择 NULL/NOT NULL、DEFAULT、UNIQUE、CHECK 等，定义属性上的约束条件。

## 2) 实验重难点

NULL/NOT NULL, DEFAULT,CHECK。

## 3) 实验过程

### ①建立完整性约束

创建 **source** 课程记录表，属性包括学号，姓名，成绩，性别，所在小组，其中学号不能为空且唯一，姓名不能为空，成绩可以为空（期末考之后才会有成绩），性别不能为空，组号限制在 1-10，其中学号为主码。自行设计创建表之后定义完整性约束。

#在定义表时进行完整性约束

```
CREATE TABLE SOURCE(  
    SNO CHAR(9) UNIQUE NOT NULL,  
    SNAME CHAR(10) NOT NULL,  
    SGRADE SMALLINT,  
    SSEX CHAR(2) NOT NULL CHECK(SSEX = '男' OR SSEX = '女'),  
    SGROUP SMALLINT CHECK(SGROUP>=1 AND SGROUP<=10),  
    PRIMARY KEY(SNO));
```

#在定义表后进行完整性约束补充

#（列级完整性约束非check子句无法表后定义）

```
CREATE TABLE SOURCE(  
    SNO CHAR(9) UNIQUE NOT NULL,  
    SNAME CHAR(10) NOT NULL,  
    SGRADE SMALLINT,  
    SSEX CHAR(2) NOT NULL,  
    SGROUP SMALLINT,  
    PRIMARY KEY(SNO));  
  
ALTER TABLE SOURCE ADD CONSTRAINT C1 CHECK(SSEX = '男' OR SSEX =  
'女');  
ALTER TABLE SOURCE ADD CONSTRAINT C2 CHECK(SGROUP>=1 AND  
SGROUP<=10);
```

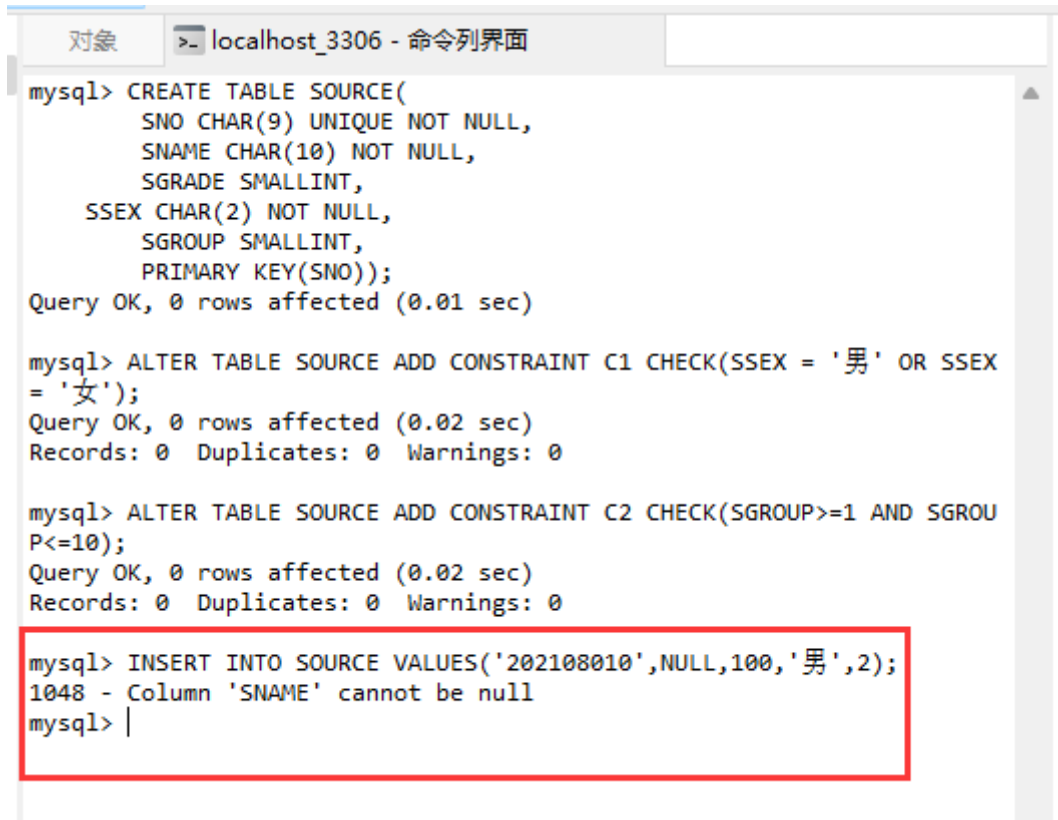


## ②验证完整性约束

设计 SQL 语句验证完整性约束：插入时姓名为空会因为违反了完整性约束而无法插入。

```
INSERT INTO SOURCE VALUES('202108010',NULL,100,'男',2);
```

运行截图如下：



```
mysql> CREATE TABLE SOURCE(  
    SNO CHAR(9) UNIQUE NOT NULL,  
    SNAME CHAR(10) NOT NULL,  
    SGRADE SMALLINT,  
    SSEX CHAR(2) NOT NULL,  
    SGROUP SMALLINT,  
    PRIMARY KEY(SNO));  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> ALTER TABLE SOURCE ADD CONSTRAINT C1 CHECK(SSEX = '男' OR SSEX  
= '女');  
Query OK, 0 rows affected (0.02 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql> ALTER TABLE SOURCE ADD CONSTRAINT C2 CHECK(SGROUP>=1 AND SGROU  
P<=10);  
Query OK, 0 rows affected (0.02 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql> INSERT INTO SOURCE VALUES('202108010',NULL,100,'男',2);  
1048 - Column 'SNAME' cannot be null  
mysql> |
```

## ③删除完整性约束并验证

试图插入一组不合法的请求（不符合完整性约束C2）

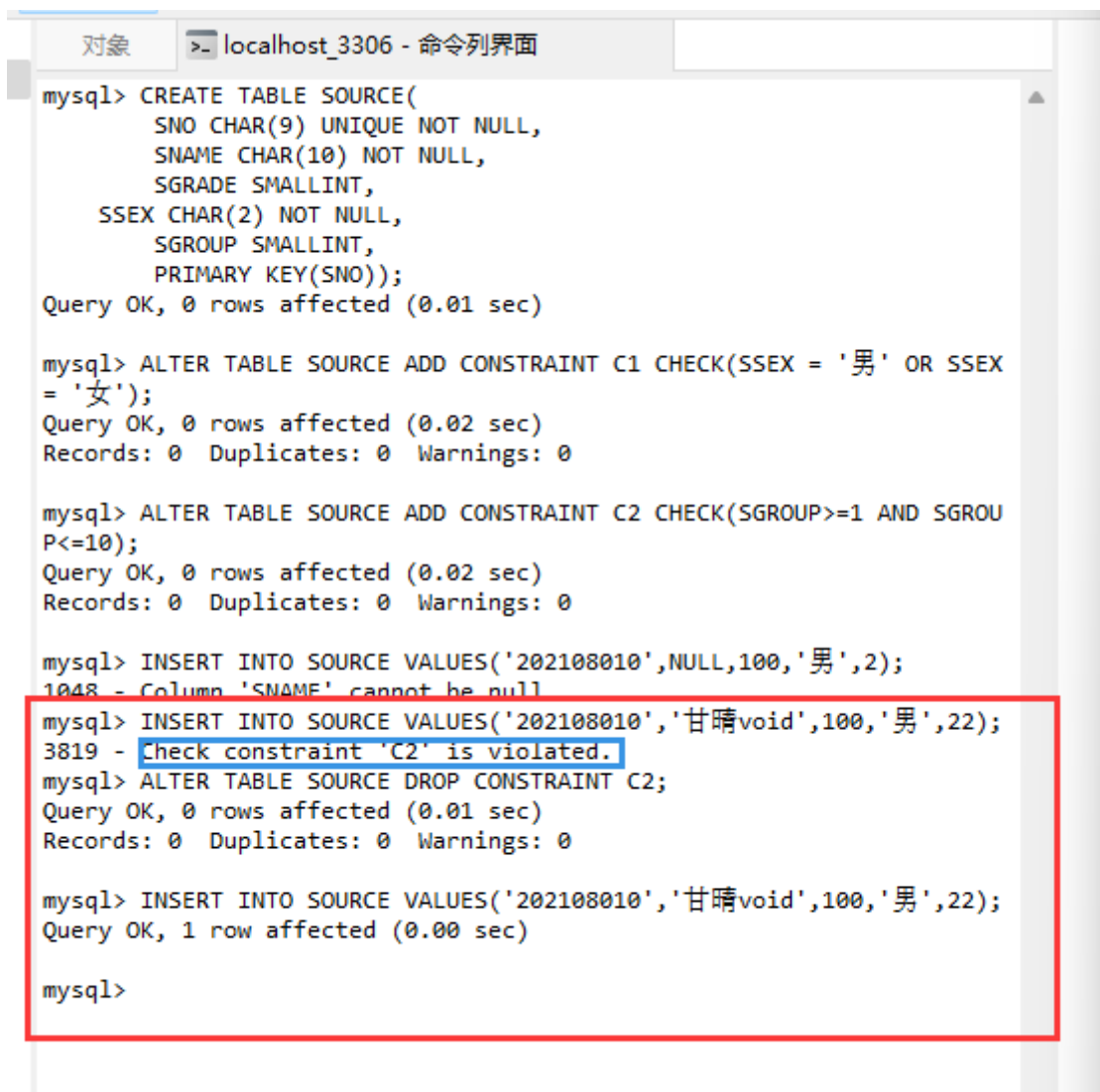
```
INSERT INTO SOURCE VALUES('202108010','甘晴void',100,'男',22);
```

删除完整性约束C2

```
ALTER TABLE SOURCE DROP CONSTRAINT C2;
```

再次试图插入该请求

运行截图如下：



```
mysql> CREATE TABLE SOURCE(
    SNO CHAR(9) UNIQUE NOT NULL,
    SNAME CHAR(10) NOT NULL,
    SGRADE SMALLINT,
    SSEX CHAR(2) NOT NULL,
    SGROUP SMALLINT,
    PRIMARY KEY(SNO));
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE SOURCE ADD CONSTRAINT C1 CHECK(SSEX = '男' OR SSEX
= '女');
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE SOURCE ADD CONSTRAINT C2 CHECK(SGROUP>=1 AND SGROU
P<=10);
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> INSERT INTO SOURCE VALUES('202108010',NULL,100,'男',2);
1048 - Column 'SNAME' cannot be null

mysql> INSERT INTO SOURCE VALUES('202108010','甘晴void',100,'男',22);
3819 - Check constraint 'C2' is violated.
mysql> ALTER TABLE SOURCE DROP CONSTRAINT C2;
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> INSERT INTO SOURCE VALUES('202108010','甘晴void',100,'男',22);
Query OK, 1 row affected (0.00 sec)

mysql>
```

在删除该完整性约束之后，该请求能被正常插入了。

## 2.3 触发器实验

### 1) 实验内容与要求

掌握数据库触发器的设计与使用方法，定义 BEFORE 触发器和 AFTER 触发器，能够理解不同类型触发器的作用和执行原理，验证触发器的有效性。

### 2) 实验重难点

利用触发器实现较为复杂的用户自定义完整性。

### 3) 示例

#### ①创建触发器（以BEFORE为例）

以 before 触发器为例，定义一个 teacher\_salary 教师工资表，属性包括 name、job、salary，均不可为空，其中 name 作为主码。

```
CREATE TABLE TEACHER_SALARY(  
    NAME CHAR(10) PRIMARY KEY,  
    JOB CHAR(20) NOT NULL,  
    SALARY SMALLINT NOT NULL);
```

Before 触发器定义如下：教授工资不得低于 5000，若低于则自动更改为 5000。

```
CREATE TRIGGER TS_T  
BEFORE INSERT ON TEACHER_SALARY  
FOR EACH ROW  
BEGIN  
    IF NEW.JOB = '教授' AND NEW.SALARY < 5000 THEN  
        SET NEW.SALARY = 5000;  
    END IF;  
END;
```

#### ★mysql关于触发器的语法区别

特别注意，mysql关于触发器的语法与教材的不太一致，主要体现在如下：

- 不需要使用REFERENCING NEW AS NEWTURPLE，而是直接使用NEW来指代新表
- mysql触发器支持UPDATE,INSERT,DELETE的触发，但不支持它们的逻辑结合，例如UPDATE OR INSERT这样是不可以的，这个跟教材上有一定区别。
- 赋值语句使用SET而不是:=符号
- 删除触发器时直接DROP TRIGGER <触发器名>，不需要添加ON <表名>

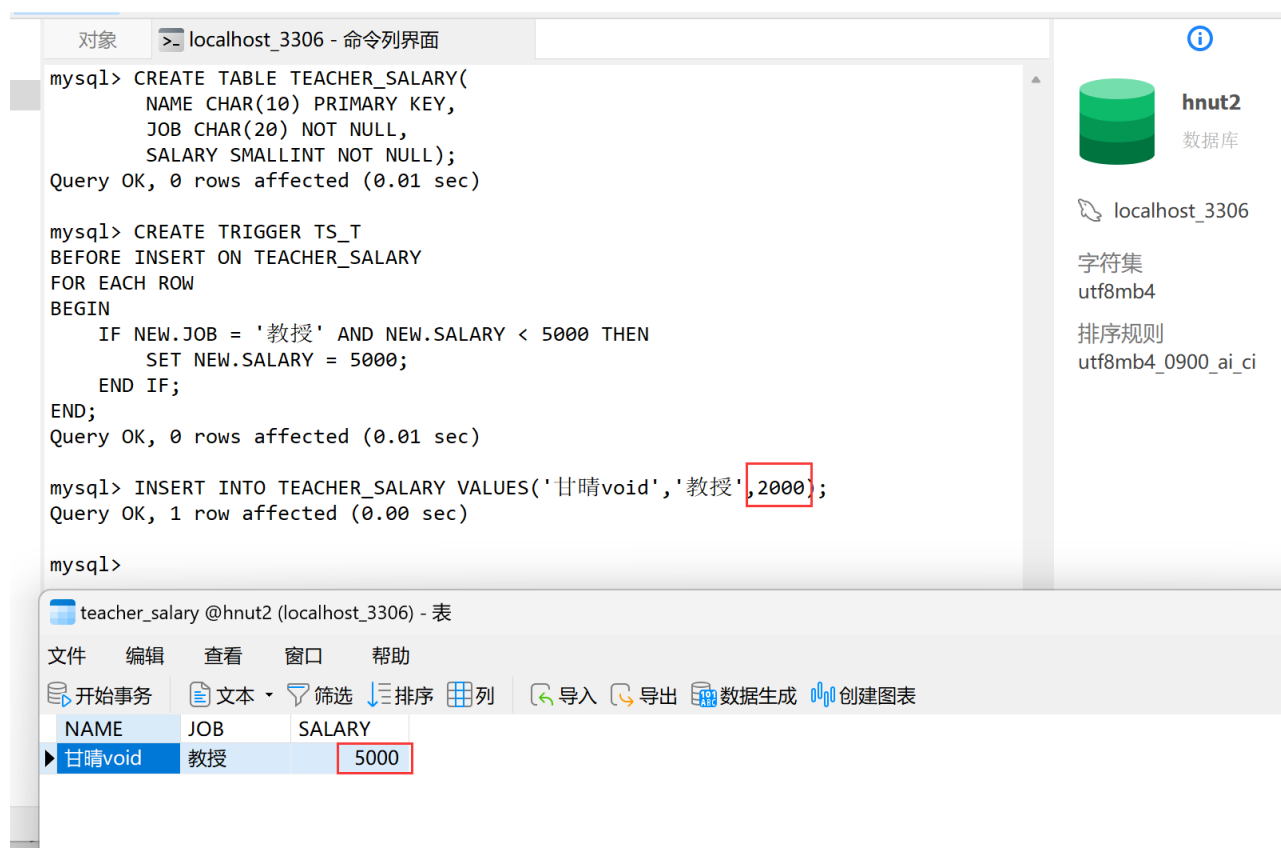
#### ②验证触发器（以BEFORE为例）

验证如下：

```
#插入一条数据  
INSERT INTO TEACHER_SALARY VALUES('甘晴void','教授',2000);
```

这条数据的salary是2000，这是不合规的，触发器会自动将它修改为5000。

运行截图如下：



### ③验证触发器执行顺序

触发器的执行是由触发事件激活，执行顺序如下：

- 执行该表上的 **before** 触发器，
- 激活触发器上 **sql** 语句，
- 执行表上 **after** 触发器，

对于表上多个 **before** 触发器，保持谁先创建谁执行原则，也有的是按照触发器名字的排序执行（这个应该就是不同数据库决定的了）

验证触发器的执行顺序可以这样进行：

- 我们分别创建**BEFORE**和**AFTER**触发器
- 每个触发器触发的时候就在另一个表中存一条记录
- 我们新建一个表**TS\_U**来存触发器的这种记录

具体过程如下：

#先删除上面那个触发器，

#新建一个表TS\_U

```
CREATE TABLE TS_U(  
    NAME CHAR(10),  
    JOB CHAR(20),  
    SALARY SMALLINT,  
    TAG CHAR(10)    #这个TAG我是用来标记记录的产生顺序  
);
```

#创建BEFORE触发器

```
CREATE TRIGGER TS_T1  
BEFORE INSERT ON TEACHER_SALARY  
FOR EACH ROW  
BEGIN  
    IF (NEW.JOB = '教授') AND (NEW.SALARY<5000) THEN  
        INSERT INTO TS_U VALUES(NEW.NAME,NEW.JOB,NEW.SALARY,'BEFORE');  
    END IF;  
END;
```

#创建AFTER触发器

```
CREATE TRIGGER TS_T2  
AFTER INSERT ON TEACHER_SALARY  
FOR EACH ROW  
BEGIN  
    IF (NEW.JOB = '教授') AND (NEW.SALARY<5000) THEN  
        INSERT INTO TS_U VALUES(NEW.NAME,NEW.JOB,NEW.SALARY,'AFTER');  
    END IF;  
END;
```

#插入一条数据

```
INSERT INTO TEACHER_SALARY VALUES('甘晴wolf','教授',3000);
```

验证截图如下：

对象localhost\_3306 - 命令行界面

```
mysql> #新建一个表TS_U
CREATE TABLE TS_U(
  NAME CHAR(10),
  JOB CHAR(20),
  SALARY SMALLINT,
  TAG CHAR(10)      #这个TAG我是用来标记记录的产生顺序
);
Query OK, 0 rows affected (0.01 sec)

mysql> #创建BEFORE触发器
CREATE TRIGGER TS_T1
BEFORE INSERT ON TEACHER_SALARY
FOR EACH ROW
BEGIN
  IF (NEW.JOB = '教授') AND (NEW.SALARY<5000) THEN
    INSERT INTO TS_U VALUES(NEW.NAME,NEW.JOB,NEW.SALARY,'BEFORE');
  END IF;
END;
Query OK, 0 rows affected (0.00 sec)

mysql> #创建AFTER触发器
CREATE TRIGGER TS_T2
AFTER INSERT ON TEACHER_SALARY
FOR EACH ROW
BEGIN
  IF (NEW.JOB = '教授') AND (NEW.SALARY<5000) THEN
    INSERT INTO TS_U VALUES(NEW.NAME,NEW.JOB,NEW.SALARY,'AFTER');
  END IF;
END;
Query OK, 0 rows affected (0.00 sec)

mysql> #插入一条数据
INSERT INTO TEACHER_SALARY VALUES('甘晴wolf','教授',3000);
Query OK, 1 row affected (0.00 sec)

mysql>
```

ts\_u @hnut2 (localhost\_3306) - 表

文件 编辑 查看 窗口 帮助

开始事务 文本 筛选 排序 列 导入 导出 数据生

NAME	JOB	SALARY	TAG
甘晴wolf	教授	3000	BEFORE
甘晴wolf	教授	3000	AFTER

我们排除顺序的影响，交换BEFORE触发器和AFTER触发器的定义顺序，再试一次。结果如下：

对象localhost\_3306 - 命令行界面

```
mysql> #新建一个表TS_U
CREATE TABLE TS_U(
  NAME CHAR(10),
  JOB CHAR(20),
  SALARY SMALLINT,
  TAG CHAR(10)      #这个TAG我是用来标记记录的产生顺序
);
Query OK, 0 rows affected (0.01 sec)

mysql> #创建AFTER触发器
CREATE TRIGGER TS_T2
AFTER INSERT ON TEACHER_SALARY
FOR EACH ROW
BEGIN
  IF (NEW.JOB = '教授') AND (NEW.SALARY<5000) THEN
    INSERT INTO TS_U VALUES(NEW.NAME,NEW.JOB,NEW.SALARY,'AFTER');
  END IF;
END;
Query OK, 0 rows affected (0.00 sec)

mysql> #创建BEFORE触发器
CREATE TRIGGER TS_T1
BEFORE INSERT ON TEACHER_SALARY
FOR EACH ROW
BEGIN
  IF (NEW.JOB = '教授') AND (NEW.SALARY<5000) THEN
    INSERT INTO TS_U VALUES(NEW.NAME,NEW.JOB,NEW.SALARY,'BEFORE');
  END IF;
END;
Query OK, 0 rows affected (0.00 sec)

mysql> #插入一条数据
INSERT INTO TEACHER_SALARY VALUES('甘晴wolf','教授',3000);
Query OK, 1 row affected (0.00 sec)

mysql>
```

ts\_u @hnut2 (localhost\_3306) - 表

文件 编辑 查看 窗口 帮助

开始事务 文本 筛选 排序 列 导入 导出 数据生成 创建图表

NAME	JOB	SALARY	TAG
甘晴wolf	教授	3000	BEFORE
甘晴wolf	教授	3000	AFTER

可以发现仍然没有变化，说明的确是BEFORE触发器在AFTER触发器之前执行。

#### ④删除触发器

注意与书上的语法区别，不需要指明所在表。

```
DROP TRIGGER TS_T1;
```

### 实验感悟

实践和理论还是有很大的差别的，特别是数据库的实验。先不说书上的一般范式和mysql数据库自己的“方言”的区别，仅仅是把一样的行为搬到数据库中就有可能产生好多好多的问题，比如mysql触发器的定义就不一样，从格式到作用范围都有差异。再比如说mysql的激活角色，这是很难想到的，这也是教材和mysql实际的一个很大的区别。如果不经同学的提醒，我可能一直会卡在这里。作为一个初学者，跟着书本上做，就是做不出预期的效果，然后就会以为这是bug，心态非常崩，而且找不到错误的原因，会非常沮丧和气恼。从这里我有一个感悟就是，做实验的时候还是不能仅仅看书，得先研究mysql提供的手册，遇到问题别先怀疑是bug，先查手册。

可惜的是这一周的实验和作业太多了，还有期中考试等，以至于我没有足够的实践对mysql手册进行足够深入的了解。希望以后在项目实战（OceanBase数据库项目）中对数据库系统进行更多的研究。