

华中科技大学

数据库系统概论实验报告

姓 名：李 翔
学 院：网络空间安全学院
专 业：信息安全
班 级：2103 班
学 号：U202112149
指导教师：余添龙

分数	
教师签名	

2024 年 5 月 4 日

目 录

1 课程任务概述	1
2 数据库定义与基本操作	2
2.1 任务要求.....	2
2.2 完成过程.....	2
2.3 任务总结.....	11
3 SQL 的复杂操作	12
3.1 任务要求.....	12
3.2 完成过程.....	12
3.3 任务总结	20
4 SQL 的高级实验	21
4.1 任务要求.....	21
4.2 完成过程.....	21
4.3 任务总结.....	28
5 数据库设计	30
5.1 任务要求.....	30
5.2 完成过程.....	30
5.3 任务总结.....	39
6 课程总结	40
6.1 主要工作.....	40
6.2 心得体会.....	40
6.3 有待改进和完善的工作.....	41
附录.....	42

1 课程任务概述

本课程的实验任务要求主要包括以下几个方面：

1、数据库定义与基本操作

- 掌握 DBMS 的数据定义功能和 SQL 语言的数据定义语句。
- 熟练掌握 SQL 的 CREATE、ALTER、DROP、SELECT 语句。
- 完成数据库的创建、基本表操作、创建和删除索引、查看和修改表的定义。

2、SQL 的复杂操作

- 熟练掌握 SQL 的连接查询、嵌套查询、表名前缀、别名前缀、不相关子查询和相关子查询。
- 掌握不同查询之间的等价替换方法及限制。
- 熟练掌握 SQL 的语句 INSERT、UPDATE、DELETE。

3、SQL 的高级实验

- 掌握视图的定义与操作、触发器的定义、存储过程的定义、用户授权和权限收回、用户定义完整性 ss 的方法。

4、数据库设计

- 使用 SQL 语句设计并实现学生管理系统。
- 系统功能包括学生信息管理、课程信息维护、学生成绩管理、统计学生成绩、按系对学生成绩进行排名、显示学生的基本信息和选课信息。

通过这些任务，让我们可以深入掌握数据库系统的基本操作和高级功能，并能够应用这些知识开发一个功能较为完善的学生管理系统。

2 数据库定义与基本操作

2.1 任务要求

- (1) 掌握 DBMS 的数据定义功能
- (2) 掌握 SQL 语言的数据定义语句
- (3) 掌握 DBMS 的数据单表查询功能
- (4) 掌握 SQL 语言的数据单表查询语句
- (5) 熟练掌握 SQL 的数据定义语句 CREATE、ALTER、DROP、Select
- (6) 记录实验结果，认真完成实验报告

2.2 完成过程

2.2.1 安装 openGauss 数据库

- (1) 使用 Docker 安装 openGauss 数据库管理系统 DBMS

如下图 2.1，使用 `docker pull enmotech/opengauss:3.0.0` 拉取 opengauss 镜像。

```
C:\Users\11946\Desktop>docker pull enmotech/opengauss:3.0.0
3.0.0: Pulling from enmotech/opengauss
22c5ef60a68e: Pull complete
28692a7bf40a: Pull complete
53d8b65999fe: Pull complete
2feb7fb751a7: Pull complete
21970696c490: Pull complete
Digest: sha256:b8a21dfa15d57476528051f1b8b138dac45cfbad3ef97cba2b340433476147f8
Status: Downloaded newer image for enmotech/opengauss:3.0.0
docker.io/enmotech/opengauss:3.0.0

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview enmotech/opengauss:3.0.0
```

图 2.1 拉取镜像

- (2) 从镜像启动容器并进入容器

如下图 2.2，首先使用 `docker run` 命令从镜像启动一个容器实例，设置密码和端口映射，然后用 `docker ps` 查看容器的 ID 号，并使用 `docker exec` 进入容器。

```
C:\Users\11946\Desktop>docker run --name opengauss --privileged=true -d -e GS_PASSWORD=Lixiang@123
-p 5432:5432 enmotech/opengauss:3.0.0
b3dd9384de753f39f1f6f49c89e1071b0866dab311299a1b317029ebda0ba74d

C:\Users\11946\Desktop>docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
PORTS          NAMES
b3dd9384de75   enmotech/opengauss:3.0.0           "entrypoint.sh gauss..." 10 seconds ago Up 9 seconds
0.0.0.0:5432->5432/tcp   opengauss

C:\Users\11946\Desktop>docker exec -it b3dd /bin/bash
root@b3dd9384de75:/# uname -a
Linux b3dd9384de75 5.15.133.1-microsoft-standard-WSL2 #1 SMP Thu Oct 5 21:02:42 UTC 2023 x86_64 x86_64 GNU/Linux
```

图 2.2 启动并进入容器

- (3) 创建数据库 CSEDB_U202112149

首先再 openGauss 的本地客户端中完成 omm 登录，其次我选择在本地完成该实验，所以直接使用 `gsql` 进行本地连接，然后使用 `CREATE DATABASE CSEDB_U202112149`;创建数据库，如下图 2.3。

```

root@b3dd9384de75:/# su - omm
omm@b3dd9384de75:~$ gsql
gsql ((OpenGauss 3.0.0 build 02c14696) compiled at 2022-04-01 18:12:34 commit 0 last mr )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

omm=# CREATE DATABASE CSEDB_U202112149 创建数据库
omm=# ;
CREATE DATABASE

```

图 2.3 命令行创建 CSEDB_U202112149 数据库

(3) 使用 Navicat Premium 16.3.9 连接 openGauss 本地数据库

打开 Navicat 后点击 连接->华为云->华为云 云数据库 GaussDB 主备版... 进入数据库连接编辑界面，如下图 2.4，进行相关配置，端口映射默认 5432，初始数据库 postgres，用户名默认为 gaussdb，密码为启动 Docker 容器时我自行设置的 Lixiang@123。使用 Navicat 的原因在于能更加方便可视化的查看与插入数据，但对于一些查询和创建操作，我依然选择使用 sql 语言的方式。

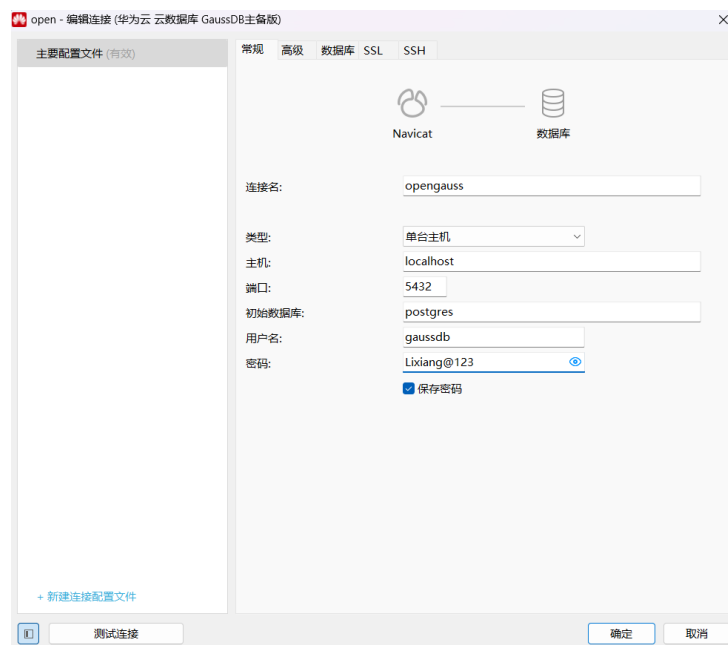


图 2.4 使用 Navicat 连接数据库

如下图 2.5，成功连接上本地 openGauss 数据库，并且有我们创建的 CSEDB_U202112149 数据库。

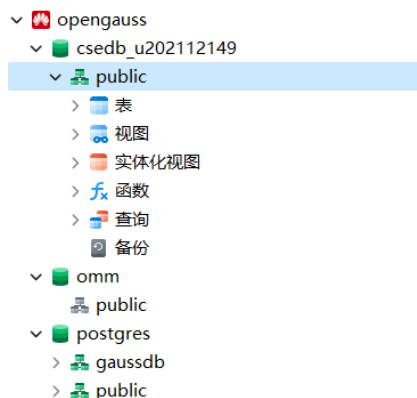


图 2.5 查看创建的 CSEDB_U202112149 数据库

2.2.2 基本表操作

(1) 切换到 CSEDB_U202112149 数据库

对于 PostgreSQL, 查看所有数据库可以使用 \l, 切换数据库使用 \c [数据库名], 如下图 2.6, 切换到我们创建的 CSEDB_U202112149 数据库并为后续的建表做准备。

```
omm=# \l 显示所有数据库
          List of databases
  Name      | Owner  | Encoding | Collate | Ctype  | Access privileges
-----+-----+-----+-----+-----+-----
csedb_u202112149 | omm    | UTF8     | C       | C      |
omm         | omm    | UTF8     | C       | C      |
postgres    | omm    | UTF8     | C       | C      |
template0   | omm    | UTF8     | C       | C      | =c/omm, omm=Ctc/omm
template1   | omm    | UTF8     | C       | C      | =c/omm, omm=Ctc/omm
(5 rows)

omm=# \c csedb_u202112149 切换到数据库
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "csedb_u202112149" as user "omm".
csedb_u202112149=#
```

图 2.6 查看所有数据库并切换

(2) 创建、删除表

如下图 2.7, 使用 CREATE TABLE 命令创建 Student 和 SC 表, 然后使用 \dt 查看数据库中的所有表:

```
csedb_u202112149=# \dt
          List of relations
 Schema | Name   | Type  | Owner  | Storage
-----+-----+-----+-----+-----
 public | sc     | table | omm    | {orientation=row,compression=no}
 public | student | table | omm    | {orientation=row,compression=no}
(2 rows)
```

图 2.7 查看创建的 2 个表

如图 2.8, 可以使用 \d tablename 显示指定表的结构、列信息、索引、外键等。

```
csedb_u202112149=# \d sc
          Table "public.sc"
  Column |      Type      | Modifiers
-----+-----+-----
 sno     | character(5)    | not null
 cno     | character(3)    | not null
 grade   | integer         |
Indexes:
    "sc_pkey" PRIMARY KEY, btree (sno, cno) TABLESPACE pg_default

csedb_u202112149=# \d student
          Table "public.student"
  Column      |      Type      | Modifiers
-----+-----+-----
 sno          | character(9)    | not null
 sname        | character(20)   |
 ssex         | character(2)    |
 sage         | smallint        |
 sdept        | character(20)   |
 scholarship  | character(2)    |
Indexes:
    "student_pkey" PRIMARY KEY, btree (sno) TABLESPACE pg_default
    "student_sname_key" UNIQUE CONSTRAINT, btree (sname) TABLESPACE pg_default
```

索引信息

图 2.8 查看两个表的信息

如下图 2.9, 使用 **DROP TABLE [tablename]**命令删除表 sc, 并使用/dt 查看所有的表, 发现 sc 表已经被删除:

```
csedb_u202112149=# DROP TABLE sc;
DROP TABLE
csedb_u202112149=# \dt
```

Schema	Name	Type	Owner	Storage
public	student	table	omm	{orientation=row,compression=no}

(1 row)

图 2.9 删除表 sc

(3) 查看、修改表的定义

使用下面两条 sql 语句分别增加 **TIMESTAMP** 类型的 Scome 字段和修该 Sage 的数据类型为 integer, 修改后如图 2.10。

ALTER TABLE Student ADD Scome TIMESTAMP;
ALTER TABLE Student ALTER COLUMN Sage TYPE INT;

csedb_u202112149=# \d student 修改后				csedb_u202112149=# \d student 修改前			
Table "public.student"				Table "public.student"			
Column	Type		Modifiers	Column	Type		Modifiers
sno	character(9)		not null	sno	character(9)		not null
sname	character(20)			sname	character(20)		
ssex	character(2)			ssex	character(2)		
sage	integer			sage	smallint		
sdept	character(20)			sdept	character(20)		
scholarship	character(2)			scholarship	character(2)		
scome	timestamp without time zone			Indexes:			

图 2.10 查看、修改表 Student 的定义

(4) 创建和删除索引

使用 **CREATE UNIQUE INDEX Stusno ON Student(Sno);**创建 Student 表中 Sno 列的唯一索引, 创建结果如下图 2.11。

```
Indexes: 新建索引后
"student_pkey" PRIMARY KEY, btree (sno) TABLESPACE pg_default
"student_sname_key" UNIQUE CONSTRAINT, btree (sname) TABLESPACE pg_default
"stusno" UNIQUE, btree (sno) TABLESPACE pg_default 新创建索引Stusno
Indexes: 新建索引前
"student_pkey" PRIMARY KEY, btree (sno) TABLESPACE pg_default
"student_sname_key" UNIQUE CONSTRAINT, btree (sname) TABLESPACE pg_default
```

图 2.11 创建唯一索引 Stusno

使用 **DROP INDEX Stusno** 可以删除刚创建的索引, 结果如下图 2.12。

```
csedb_u202112149=# DROP INDEX Stusno;
DROP INDEX
csedb_u202112149=# \d student
```

Column	Type	Modifiers
sno	character(9)	not null
sname	character(20)	
ssex	character(2)	
sage	integer	
sdept	character(20)	
scholarship	character(2)	
scome	timestamp without time zone	

Indexes:
"student_pkey" PRIMARY KEY, btree (sno) TABLESPACE pg_default
"student_sname_key" UNIQUE CONSTRAINT, btree (sname) TABLESPACE pg_default

图 2.12 删除索引 Stusno

2.2.3 删除数据库

如下图 2.13 通过 DROP DATABASE csedb_u202112149 删除该数据库，并在命令前后使用 \l 来显示所有的数据库从而确定其已被删除。

```
omm=# \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
csedb_u202112149	omm	UTF8	C	C	待删除数据库
omm	omm	UTF8	C	C	
postgres	omm	UTF8	C	C	
template0	omm	UTF8	C	C	=c/omm +
					omm=CTc/omm
template1	omm	UTF8	C	C	=c/omm +
					omm=CTc/omm

(5 rows)

```
omm=# DROP DATABASE csedb_u202112149;
DROP DATABASE
omm=# \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
omm	omm	UTF8	C	C	
postgres	omm	UTF8	C	C	
template0	omm	UTF8	C	C	=c/omm +
					omm=CTc/omm
template1	omm	UTF8	C	C	=c/omm +
					omm=CTc/omm

(4 rows)

图 2.13 删除数据库 csedb_u202112149

2.2.4 创建示例数据库 S_T_U202112149

操作同上图 2.3 故不再给出，使用 CREATE DATABASE [数据库名]创建即可。

2.2.5 在示例数据库中创建学生表 Student、课程表 Course 和选修表 SC

(1) 创建三个基本表学生表 Student、课程表 Course 和选修表 SC

```
create table Student
```

```
(Sno CHAR(9) PRIMARY KEY, Sname CHAR(20) UNIQUE, Ssex CHAR(2),
Sage SMALLINT, Sdept CHAR(20), Scholarship char(2) );
```

/*表 Student 的主码为 Sno，属性列 Sname 取唯一值*/

```
create table Course
```

```
(Cno CHAR(4) PRIMARY KEY, Cname CHAR(40), Cpno CHAR(4), Ccredit
SMALLINT, FOREIGN KEY (Cpno) REFERENCES Course(Cno) );
```

/*表 Course 的主码为 Cno，属性列 Cpno(先修课)为外码，被参照表为 Course，被参照列是 Cno*/

```
create table SC
```

```
(Sno CHAR(9), Cno CHAR(4), Grade SMALLINT, primary key (Sno, Cno),
FOREIGN KEY (Sno) REFERENCES Student(Sno), FOREIGN KEY (Cno)
REFERENCES Course(Cno) );
```


/*表 SC 的主码为(Sno, Cno), Sno 和 Cno 均为外码, 被参照表分别为 Student 和 Course, 被参照列分别为 Student.Sno 和 Course.Cno*/

使用上述 3 个创建表的 sql 语句进行表的创建, 结果如图 2.14:

```
csedb_u202112149=# \d
```

List of relations				
Schema	Name	Type	Owner	Storage
public	course	table	omm	{orientation=row,compression=no}
public	sc	table	omm	{orientation=row,compression=no}
public	student	table	omm	{orientation=row,compression=no}

(3 rows)

图 2.14 3 个基本表的创建结果

(2) 向三个表中添加示例数据

在此我选择通过可视化的方式为 3 个表添加数据项。其中学生表 Student、课程表 Course、选修 SC 表分别对应图片 2.15、2.16、2.17。

sno	sname	ssex	sage	sdept	scholarship
200215121	李勇	男	20	CS	否
200215122	刘晨	女	19	CS	否
200215123	王敏	女	18	MA	否
200215125	张立	男	19	IS	否

图 2.15 学生表 Student 添加数据

cno	cname	cpno	ccredit
1	数据库	5	4
2	数学	(Null)	2
3	信息系统	5	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	(Null)	2
7	PASCAL语言	6	4

图 2.16 课程表 Course 添加数据

sno	cno	grade
200215121	1	92
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80

图 2.17 选修 SC 添加数据

2.2.6 对学生关系 Student、课程关系 Course 和选修关系 SC 进行查询

由于 SQL 语句执行结果图片和练习任务一一对应, 故此处不再特别提及图片编号。

(1) 基本练习

1. SELECT 语句的基本用法

例如: 查询全体学生的详细记录。

SQL 语句: SELECT Sno,Sname,Ssex,Sage,Sdept FROM Student

sno	sname	ssex	sage	sdept
200215121	李勇	男	20	CS
200215122	刘晨	女	19	CS
200215123	王敏	女	18	MA
200215125	张立	男	19	IS

图 2.18 查询结果

2. 使用 WHERE 子句进行有条件的查询

例如：查询选修 2 号课程且成绩在 90 分以上的所有学生的学号、姓名

SQL 语句：SELECT student.sno,student.sname FROM student,scWHERE student.sno=sc.sno AND sc.cno='2' AND sc.grade>90

sno	sname
(N/A)	(N/A)

图 2.19 查询结果

3. 使用 IN, NOT IN, BETWEEN 等谓词查询

例如：查询信息系（IS）、数学系（MA）和计算机科学系（CS）学生的姓名和性别。

SQL 语句：SELECT Sname,Ssex FROM Student WHERE Sdept IN ('IS','MA','CS')

sname	ssex
李勇	男
刘晨	女
王敏	女
张立	男

图 2.20 查询结果

例如：查询年龄在 20~23 岁（包括 20 岁和 23 岁）之间的学生的姓名、系别和年龄。

SQL 语句：SELECT Sname,Sdept,Sage FROM Student WHERE Sage BETWEEN 20 AND 23

sname	sdept	sage
李勇	CS	20

图 2.21 查询结果

4. 利用 LIKE 子句实现模糊查询

例如：查询所有姓刘学生的姓名、学号和性别。

SQL 语句：SELECT Sname,Sno,Ssex FROM Student WHERE Sname LIKE '刘%'

sname	sno	ssex
刘晨	200215122	女

图 2.22 查询结果

5. 利用 ORDER 子句为结果排序

例如：查询选修了 3 号课程的学生的学号及其成绩，查询结果按分数降序排列

SQL 语句: SELECT Sno,Grade FROM SC WHERE Cno= '3' ORDER BY Grade DESC

sno	grade
200215121	88
200215122	80

图 2.23 查询结果

6. 用 SQL Server 的统计函数进行统计计算

例如: 计算 1 号课程的学生平均成绩。

SQL 语句: SELECT AVG(Grade) FROM SC WHERE Cno= '1'

avg
92.0000000000000000

图 2.24 查询结果

7. 用 GROUP BY 子句实现分组查询的方法

例如: 查询选修了 3 门以上课程的学生学号。

SQL 语句: SELECT Sno FROM SC GROUP BY Sno HAVING COUNT(*) >3

sno
(N/A)

图 2.25 查询结果

(2) 扩展练习 (要求写出并执行 SQL 语句来完成以下各种操作, 记录查询结果)

由于 SQL 语句执行结果图片和练习任务一一对应, 故此处不再特别提及图片编号。

1. 查询全体学生的学号、姓名和年龄

SQL 语句: SELECT sno,sname,sage FROM student

sno	sname	sage
200215121	李勇	20
200215122	刘晨	19
200215123	王敏	18
200215125	张立	19

图 2.26 查询结果

2. 查询所有计算机系学生的详细记录

SQL 语句: SELECT * FROM student WHERE sdept='CS'

sno	sname	ssex	sage	sdept	scholarship
200215121	李勇	男	20	CS	否
200215122	刘晨	女	19	CS	否

图 2.27 查询结果

3. 考试成绩为优秀 (90 分及以上) 或不及格的学生的学号、课程号及成绩

SQL 语句: SELECT sno,cno,grade FROM sc WHERE grade>=90 OR grade<60

sno	cno	grade
200215121	1	92
200215122	2	90

图 2.28 查询结果

4. 查询年龄不在 19~20 岁之间的学生姓名、性别和年龄

SQL 语句: SELECT sname,ssex,sage FROM student WHERE sage NOT BETWEEN 19 AND 20

sname	ssex	sage
王敏	女	18

图 2.29 查询结果

5. 查询数学系（MA）、信息系（IS）的学生的姓名和所在系

SQL 语句：SELECT sname,sdept FROM student WHERE sdept='MA' OR sdept='IS'

sname	sdept
王敏	MA
张立	IS

图 2.30 查询结果

6. 查询名称中包含“数据”的所有课程的课程号、课程名及其学分

SQL 语句：SELECT cno,cname,ccredit FROM course WHERE cname LIKE '%数据%'

cno	cname	ccredit
1	数据库	4
5	数据结构	4
6	数据处理	2

图 2.31 查询结果

7. 找出所有没有选修课成绩的学生学号和课程号

SQL 语句：SELECT sno,cno FROM sc WHERE grade IS NULL

sno	cno
(N/A)	(N/A)

图 2.32 查询结果

8. 查询学生 200215121 选修课的最高分、最低分以及平均成绩

SQL 语句：SELECT max(grade),min(grade),avg(grade) FROM sc WHERE sno='200215121'

max	min	avg
92	85	88.33333333333333

图 2.33 查询结果

9. 查询选修了 2 号课程的学生的学号及其成绩，查询结果按成绩升序排列

SQL 语句：SELECT sno,grade FROM sc WHERE cno='2' ORDER BY grade

sno	grade
200215121	85
200215122	90

图 2.34 查询结果

10. 查询每个系名及其学生的平均年龄。

SQL 语句：SELECT sdept,avg(sage) FROM student GROUP BY sdept

sdept	avg
MA	18.000000000000000
CS	19.500000000000000
IS	19.000000000000000

图 2.35 查询结果

（思考：如何查询学生平均年龄在 19 岁以下（含 19 岁）的系别及其学生的平均年龄？）

思考题 SQL 查询语句：SELECT sdept,AVG(sage) FROM student GROUP BY sdept **HAVING AVG(sage) <= 19**

HAVING 用于对 GROUP BY 的结果进行过滤。

sdept	avg
MA	18.0000000000000000
IS	19.0000000000000000

图 2.36 查询结果

2.3 任务总结

(1) MacOS 系统下 openGauss 容器无法正常启动

每次一启动容器过几秒就会异常 exit，从而无法进入容器执行任何命令，更新 Docker、换镜像之后均无效果。

解决方案：更换带有 Windows 操作系统的电脑。

(2) VMCourse 课程平台申请容器启动异常

当我尝试 gsql 连接远程数据库的时候，发现始终无法连接申请的容器（OpenVPN 已经连接），检查网站上的申请镜像的状态，发现容器过一会就 exited，提示我点开机，点了开机之后还是出现此异常。

解决方案：询问老师后，采用本地数据库的方式完成实验。

(3) 字符编码问题

设置唯一索引的时候莫名提示 invalid byte sequence for encoding “UTF8”

```
csedb_u202112149=# CREATE UNIQUE INDEX Stusno ON Student(Sno);  
ERROR:  invalid byte sequence for encoding "UTF8": 0xef 0x3b
```

图 2.37 UTF8 字符编码异常

解决方案：设置 client_encoding 为 UTF8

```
csedb_u202112149=# SET client_encoding TO 'UTF8';  
SET
```

图 2.38 设置 client_encoding 为 UTF8

在这一部分实验中，我对 openGauss 数据库环境进行了一定的配置，遇到了很多容器异常方面的问题，最终只能更换操作系统并使用本地数据库进行实验。

然后就是学习并使用了一些基本的 SQL 语句。首先，我学习了一些常用的数据定义 SQL 语句，包括建立、删除、修改表等基本操作，成功在数据库中建立了 3 个基本表。然后，在这三个表的基础之上，我学习了如何使用 SQL 语句进行数据查询，完成了一些基础和扩展练习。

3 SQL 的复杂操作

3.1 任务要求

- (1) 熟练掌握 SQL 的连接查询语句
- (2) 熟练掌握 SQL 的嵌套查询语句
- (3) 掌握表名前缀、别名前缀的用法
- (4) 掌握不相关子查询和相关子查询的区别和用法
- (5) 掌握不同查询之间的等价替换方法（一题多解）及限制
- (6) 熟练掌握 SQL 的数据更新语句 INSERT、UPDATE、DELETE
- (7) 记录实验结果，认真完成实验报告

3.2 完成过程

对学生关系 Student、课程关系 Course 和选修关系 SC 进行多表查询。由于 SQL 语句执行结果图片和练习任务一一对应，故此处不再特别提及图片编号。

3.2.1 基本练习

- (1) 等值连接查询与自然连接查询

1. 查询每个学生及其选修课的情况

SQL 语句(一般等值连接): `SELECT student.*,sc.* FROM student,sc WHERE student.sno=sc.sno /* 一般等值连接 */`

sno	sname	ssex	sage	sdept	scholarship	sno(1)	cno	grade
200215121	李勇	男	20	CS	否	200215121	1	92
200215121	李勇	男	20	CS	否	200215121	2	85
200215121	李勇	男	20	CS	否	200215121	3	88
200215122	刘晨	女	19	CS	否	200215122	2	90
200215122	刘晨	女	19	CS	否	200215122	3	80

图 3.1 查询结果

2. 查询每个学生及其选修课的情况（去掉重复列）

SQL 语句(自然连接): `SELECT student.sno, sname, ssex, sage, cno, grade FROM student,sc WHERE student.sno=sc.sno /*自然连接-特殊的等值连接*/`

sno	sname	ssex	sage	cno	grade
200215121	李勇	男	20	1	92
200215121	李勇	男	20	2	85
200215121	李勇	男	20	3	88
200215122	刘晨	女	19	2	90
200215122	刘晨	女	19	3	80

图 3.2 查询结果

- (2) 自身连接查询

1. 查询每一门课的间接先修课

SQL 语句: `SELECT FIRST.cno,SECOND.cjno FROM course FIRST, course SECOND WHERE FIRST.cjno=SECOND.cno`

cno	cpno
1	7
3	7
4	(Null)
5	6
7	(Null)

图 3.3 查询结果

(3) 外连接查询

1. 查询每个学生及其选修课的情况（要求输出所有学生--含未选修课程的学生
的情况）

SQL 语句: `SELECT student.sno,sname,ssex,sage,sdept,cno,grade FROM student
LEFT OUTER JOIN sc ON(student.sno=sc.sno) /* 和 LEFT JOIN 一样效果 */`

sno	sname	ssex	sage	sdept	cno	grade
200215121	李勇	男	20	CS	1	92
200215121	李勇	男	20	CS	2	85
200215121	李勇	男	20	CS	3	88
200215122	刘晨	女	19	CS	2	90
200215122	刘晨	女	19	CS	3	80
200215125	张立	男	19	IS	(Null)	(Null)
200215123	王敏	女	18	MA	(Null)	(Null)

图 3.4 查询结果

(4) 复合条件连接查询

1. 查询选修了 2 号课程而且成绩在 90 以上的所有学生的学号和姓名

SQL 语句: `SELECT student.sno,sname FROM student,sc WHERE student.sno =
sc.sno AND sc.grade >= 90 AND sc.cno = '2'`

sno	sname
200215122	刘晨

图 3.5 查询结果

2. 查询每个学生的学号、姓名、选修的课程名及成绩

SQL 语句: `SELECT student.sno,sname,cname,grade FROM student,course,sc
WHERE student.sno=sc.sno AND sc.cno=course.cno`

sno	sname	cname	grade
200215121	李勇	数据库	92
200215121	李勇	数学	85
200215121	李勇	信息系统	88
200215122	刘晨	数学	90
200215122	刘晨	信息系统	80

图 3.6 查询结果

(5) 嵌套查询（带有 IN 谓词的子查询）

1. 查询与“刘晨”在同一个系学习的学生的学号、姓名和所在系

SQL 语句 1(IN): `SELECT sno,sname,sdept FROM student WHERE sdept IN
(SELECT sdept FROM student WHERE sname='刘晨')`

SQL 语句 2(=替换 IN): `SELECT sno,sname,sdept FROM student WHERE sdept
= (SELECT sdept FROM student WHERE sname='刘晨')`

SQL 语句 3(自身连接): SELECT s1.sno,s1.sname,s1.sdept FROM student s1,student s2 WHERE s1.sdept=s2.sdept AND s2.sname='刘晨'

SQL 语句 4(EXISTS): SELECT sno,sname,sdept FROM student s1 WHERE EXISTS (SELECT * FROM student s2 WHERE s1.sdept=s2.sdept AND s2.sname='刘晨')

sno	sname	sdept
200215121	李勇	CS
200215122	刘晨	CS

图 3.7 查询结果

2. 查询选修了课程名为“信息系统”的学生号和姓名

SQL 语句 1(IN): SELECT sno,sname FROM student WHERE sno IN (SELECT sno FROM sc WHERE cno IN (SELECT cno FROM course WHERE cname='信息系统'))

SQL 语句 2(连接查询): SELECT student.sno,sname FROM student,sc,course WHERE student.sno=sc.sno AND sc.cno=course.cno AND course.cname='信息系统'

sno	sname
200215121	李勇
200215122	刘晨

图 3.8 查询结果

(6) 嵌套查询（带有比较运算符的子查询）

1. 找出每个学生超过他所选修课程平均成绩的课程号

SQL 语句: SELECT sno,cno FROM sc x WHERE grade >= (SELECT AVG(grade) FROM sc y WHERE x.sno=y.sno)

sno	cno
200215121	1
200215122	2

图 3.9 查询结果

(7) 嵌套查询（带有 ANY 或 ALL 谓词的子查询）

1. 查询其他系中比计算机系某个学生年龄小的学生的姓名和年龄

SQL 语句 1: SELECT sname,sage FROM student WHERE sage<ANY(SELECT sage FROM student WHERE sdept='CS') AND sdept<>'CS'; // ANY

SQL 语句 2: SELECT sname,sage FROM student WHERE sage<(SELECT MAX(sage) FROM student WHERE sdept='CS') AND sdept<>'CS'; //聚集函数

sname	sage
王敏	18
张立	19

图 3.10 查询结果

2. 查询其他系中比计算机系所有学生年龄都小的学生的姓名和年龄

SQL 语句 1: `SELECT sname,sage FROM student WHERE sage<ALL(SELECT sage FROM student WHERE sdept='CS') AND sdept<>'CS';` // ALL

SQL 语句 2: `SELECT sname,sage FROM student WHERE sage<(SELECT MIN(sage) FROM student WHERE sdept='CS') AND sdept<>'CS';` //聚集函数

sname	sage
王敏	18

图 3.11 查询结果

- (8) 嵌套查询（带有 EXISTS 谓词的子查询）

1. 查询所有选修了 1 号课程的学生姓名

SQL 语句: `SELECT sname FROM student WHERE EXISTS(SELECT * FROM sc WHERE sno=student.sno AND cno='1')`

sname
李勇

图 3.12 查询结果

2. 查询所有未选修 1 号课程的学生姓名

SQL 语句: `SELECT sname FROM student WHERE NOT EXISTS(SELECT * FROM sc WHERE sno=student.sno AND cno='1')`

sname
刘晨
王敏
张立

图 3.13 查询结果

3. 查询选修了全部课程的学生姓名

SQL 语句: `SELECT sname FROM student WHERE NOT EXISTS (SELECT * FROM course WHERE NOT EXISTS (SELECT * FROM sc WHERE sno=student.sno AND cno=course.cno))`

理解：对每个学生按照 course 表在 sc 表里查询没有成绩的课程，如果存在没有成绩的课程那么外层的返回值就是 False，该学生不选择，如果不存在没有成绩的课程（所有课程都被选择），内层的查询为空，外层返回值为 True，选择该学生。

sname
(N/A)

图 3.14 查询结果

4. 查询至少选修了学生 200215122 选修的全部课程的学生号码

SQL 语句: `SELECT DISTINCT sno FROM sc scx WHERE NOT EXISTS (SELECT * FROM sc scy WHERE scy.sno='200215122' AND NOT EXISTS (SELECT * FROM sc scz WHERE scz.sno=scx.sno AND scz.cno=scy.cno))`

理解：最内层的 not exists 用于判断当前最外层的 sno 是否有和 sno='200215122'相同的课程，如果有则返回 False，没有相同则返回 True，

如果对于所有 200215122 选择的课程，该学生都选择了，则内层全为 False，外层 not exists 则为 True，从而选择该学生；如果该学生选择了部分或都没选，则内层有部分的 True，not exists 则返回 False，从而不选。

sno
200215121
200215122

图 3.15 查询结果

3、4 表明使用带有 EXISTS 谓词的子查询实现全称量词或蕴涵逻辑运算功能

(9) 集合查询

1. 查询计算机系的学生以及年龄不大于 19 岁的学生

SQL 语句 1(并集): SELECT * FROM student WHERE sdept='CS' UNION
SELECT * FROM student WHERE sage<=19

SQL 语句 2(OR): SELECT * FROM student WHERE sdept='CS' OR sage<=19

sno	sname	ssex	sage	sdept	scholarship
200215121	李勇	男	20	CS	否
200215122	刘晨	女	19	CS	否
200215125	张立	男	19	IS	否
200215123	王敏	女	18	MA	否

图 3.16 查询结果

2. 查询既选修了课程 1 又选修了课程 2 的学生（交集运算）

SQL 语句 1(交集): SELECT sno FROM sc WHERE cno='1' INTERSECT
SELECT sno FROM sc WHERE cno='2'

SQL 语句 2(嵌套查询): SELECT sno FROM sc WHERE cno='1' AND sno IN
(SELECT sno FROM sc WHERE cno='2')

sno
200215121

图 3.17 查询结果

3. 思考(例 2): 能不能改用多重条件查询?

错误的 SQL 语句: SELECT sno FROM sc WHERE cno='1' AND cno='2'

这样的查询不会返回任何结果，因为一个字段（cno）在同一行数据中无法同时具有两个不同的值，可以使用 JOIN 来实现本查询。

SQL 语句 (JOIN): SELECT s1.sno FROM sc s1 JOIN sc s2 ON s1.sno=s2.sno
WHERE s1.cno='1' AND s2.cno='2'

4. 查询计算机系的学生与年龄不大于 19 岁的学生的差集

SQL 语句 1: SELECT * FROM student WHERE sdept='CS' EXCEPT SELECT
* FROM student WHERE sage<=19 /*差集运算*/

SQL 语句 2: SELECT * FROM student WHERE sdept='CS' AND sage>19 /*多
重条件查询*/

sno	sname	ssex	sage	sdept	scholarship
200215121	李勇	男	20	CS	否

图 3.18 查询结果

(10) update 语句用于对表进行更新

1. 将信息系所有学生的年龄增加 1 岁

SQL 语句: UPDATE student SET sage=sage+1 WHERE Sdept='IS'

查询	消息
UPDATE student SET sage=sage+1 WHERE Sdept='IS'	Affected rows: 1

图 3.19 更新结果

(11) delete 语句用于对表进行删除

1. 删除学号为 95019 的学生记录

SQL 语句: DELETE FROM Student WHERE Sno='95019'

查询	消息
DELETE FROM Student WHERE Sno='95019'	Affected rows: 0

图 3.20 删除结果

(12) insert 语句用于对表进行插入

1. 插入一条选课记录('95020', '1')

SQL 语句: INSERT INTO sc(sno,cno) VALUES ('95020','1')

查询	消息
INSERT INTO sc(sno,cno) VALUES ('95020','1')	ERROR: insert or update on table "sc" violates foreign key constraint "sc_sno_fkey" DETAIL: Key (sno)=(95020) is not present in table "student".

图 3.21 插入结果

由上图 3.21 可知, 因为 sc 有约束 sno 必须要在 student 表中出现才行, 故添加失败。

3.2.2 扩展练习 (要求写出并执行 SQL 语句完成以下各种操作, 记录查询结果)

(1) 查询每门课程及其被选情况 (输出所有课程中每门课的课程号、课程名称、选修该课程的学生学号及成绩--如果没有学生选择该课, 则相应的学生学号及成绩为空值)。

SQL 语句: SELECT course.cno,cname,sno,grade FROM course LEFT JOIN sc ON (course.cno=sc.cno)

	cno	cname	sno	grade
▶	1	数据库	200215121	92
	2	数学	200215121	85
	3	信息系统	200215121	88
	2	数学	200215122	90
	3	信息系统	200215122	80
	7	PASCAL语	(Null)	(Null)
	6	数据处理	(Null)	(Null)
	5	数据结构	(Null)	(Null)
	4	操作系统	(Null)	(Null)

图 3.22 查询结果

(2) 查询与“张立”同岁的学生的学号、姓名和年龄。(要求使用至少 3 种方法求解)

SQL 语句 1: SELECT sno,sname,sage FROM student WHERE sage=(SELECT sage FROM student WHERE sname='张立') // 嵌套查询

SQL 语句 2: SELECT s1.sno,s1.sname,s1.sage FROM student s1,student s2
WHERE s2.sname='张立' AND s1.sage=s2.sage // 自身连接查询

SQL 语句 3: SELECT sno,sname,sage FROM student s1 WHERE EXISTS
(SELECT * FROM student s2 WHERE s2.sname='张立' AND s2.sage=s1.sage)

sno	sname	sage
200215122	刘晨	19
200215125	张立	19

图 3.23 查询结果

(3) 查询选修了 3 号课程而且成绩为良好（80~89 分）的所有学生的学号和姓名。

SQL 语句: SELECT student.sno,sname FROM student,sc WHERE
student.sno=sc.sno AND cno='3' AND grade BETWEEN 80 AND 89

sno	sname
200215121	李勇
200215122	刘晨

图 3.24 查询结果

(4) 查询学生 200215122 选修的课程号、课程名。（思考：如何查询学生 200215122 选修的课程号、课程名及成绩？）

SQL 语句: SELECT sc.cno,cname FROM sc,course WHERE sc.cno=course.cno
AND sc.sno='200215122'

思考题：添加成绩的话直接在 SELECT 处加上 grade 就行。

cno	cname
2	数学
3	信息系统

图 3.25 查询结果

(5) 找出每个学生低于他所选课程平均成绩 5 分的课程号。（输出学号和课程号）

SQL 语句: SELECT sno,cno FROM sc x WHERE grade+5<=(SELECT
AVG(grade) FROM sc WHERE sc.sno=x.sno)

由于没有低于 5 分以上的，但有恰好等于 5 分的故将小于号改为小于等于。

sno	cno
200215122	3

图 3.26 查询结果

(6) 查询比所有男生年龄都小的女生的学号、姓名和年龄。

SQL 语句: SELECT sno,sname,sage FROM student WHERE ssex='女' AND
sage<(SELECT MIN(sage) FROM student WHERE ssex='男')

sno	sname	sage
200215123	王敏	18

图 3.27 查询结果

(7) 查询所有选修了 2 号课程的学生姓名及所在系。

SQL 语句: SELECT student.sname,sdept FROM student,sc WHERE
student.sno=sc.sno AND cno='2'

sname	sdept
李勇	CS
刘晨	CS

图 3.28 查询结果

(8) 使用 update 语句把平均成绩为良的学生的年龄增加 2 岁，并查询出来。

SQL 语句 1(update): UPDATE student set sage=sage+2 WHERE sno IN (SELECT sno FROM sc GROUP BY sno HAVING AVG(grade) BETWEEN 80 AND 89)

SQL 语句 2(select): SELECT sno,sname,sage FROM student WHERE sno IN (SELECT sno FROM sc GROUP BY sno HAVING AVG(grade) BETWEEN 80 AND 89)

sno	sname	sage
200215122	刘晨	19
200215121	李勇	20

图 3.29 修改前查询结果

sno	sname	sage
200215122	刘晨	21
200215121	李勇	22

图 3.30 修改后查询结果

(9) 使用 insert 语句增加两门课程：C 语言和人工智能，并查询出来。

SQL 语句 (insert) : INSERT INTO course VALUES('8', 'C 语言', NULL,NULL),('9','人工智能', NULL,NULL)

SQL 语句(select): SELECT * FROM course

cno	cname	cpno	ccredit
1	数据库	5	4
2	数学	(Null)	2
3	信息系统	5	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	(Null)	2
7	PASCAL语言	6	4

增加后

cno	cname	cpno	ccredit
1	数据库	5	4
2	数学	(Null)	2
3	信息系统	5	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	(Null)	2
7	PASCAL语言	6	4
8	C语言	(Null)	(Null)
9	人工智能	(Null)	(Null)

图 3.31 增加两门课程

(10) 使用 delete 语句把人工智能课程删除，并查询出来。

SQL 语句(delete): delete from course where cname='人工智能'

SQL 语句(select): SELECT * FROM course

cno	cname	cpno	ccredit
1	数据库	5	4
2	数学	(Null)	2
3	信息系统	5	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	(Null)	2
7	PASCAL语言	6	4
8	C语言	(Null)	(Null)
9	人工智能	(Null)	(Null)

删除后

cno	cname	cpno	ccredit
1	数据库	5	4
2	数学	(Null)	2
3	信息系统	5	4
4	操作系统	6	3
5	数据结构	7	4
6	数据处理	(Null)	2
7	PASCAL语言	6	4
8	C语言	(Null)	(Null)

图 3.32 删除人工智能课程

3.3 任务总结

在这一部分实验中，我深入学习探索了较为复杂的 SQL 查询语句，特别是多表查询和嵌套查询的复杂用法。通过实践和实验，我不仅加深了对 SQL 语言的理解，而且提高了解决实际数据库问题的能力。以下是我在实验过程中遇到的一些主要挑战、采取的解决策略，以及我从中获得的一些启发。

(1) 复杂嵌套查询的理解和编写困难

遇到复杂的查询任务，经常逻辑混乱导致无法编写出正确的 SQL 语句。

解决方案：为了克服嵌套查询的复杂性，我采取了分步构建和逐层测试的策略。这不仅帮助我更好地理解每个子查询的作用，还使得错误更容易被发现和修正。我也学会了使用图形化的数据库工具来可视化查询结果，这极大地帮助了我的理解和调试过程。

(2) 多表查询中的二义性错误

多表查询中，对于两个表中都出现的列，如果不指明该列具体来自哪个表会造成二义性的错误。



图 3.33 报错信息

解决方案：用前缀指明该字段来自于哪个表。

(3) 处理未选课程的输出问题

不确定如何输出那些没有学生选择的课程。

解决方案：通过示例和相关资料我了解到可以使用左连接（LEFT JOIN）解决这个问题。左连接允许从主表输出所有记录，即使在关联表中没有匹配的记录，也能保持主表记录的完整性，显示为空值。

(4) 多方式查询实践

任务要求使用三种不同的查询方式来得到结果，这在一定程度上挑战了我的 SQL 应用能力。

解决方案：我参考了一些基础实例，通过实现嵌套查询、连接查询和使用 EXISTS 谓词，成功构建了所需的查询语句。

(5) 优化查询性能

在实验中，我还学习到连接查询通常比嵌套查询性能更优。这一点在实际应用中非常关键，特别是处理大数据集时，优化查询能显著提高效率。

(6) 逻辑与 SQL 语法错误

在使用 UPDATE 语句修改符合特定成绩条件学生的年龄时遇到了问题。最初尝试直接在 WHERE 子句中使用平均函数，却因 SQL 语法限制失败。

解决方案：使用 GROUP BY 和 HAVING 子句组合来筛选符合条件的学生，然后对这些学生的年龄进行更新。

4 SQL 的高级实验

4.1 任务要求

- (1) 掌握视图的定义与操作
- (2) 掌握对触发器的定义
- (3) 掌握对存储过程的定义
- (4) 掌握如何对用户进行授权和收回权限
- (5) 掌握用户定义完整性的方法
- (6) 记录实验结果，认真完成实验报告

4.2 完成过程

由于图片和实验任务一一对应，故在每个任务中不再引用图片的编号，而是直接给出 SQL 语句执行结果的图片。

4.2.1 视图的定义与操作

- (1) 创建 CS 系的视图 CS_View

SQL 语句: CREATE VIEW CS_View AS SELECT * FROM student WHERE sdept='CS'

sno	sname	ssex	sage	sdept	scholarship
200215122	刘晨	女	19	CS	否
200215121	李勇	男	20	CS	否

图 4.1 CS_View 视图

- (2) 在视图 CS_View 上查询 CS 系选修了 1 号课程的学生

SQL 语句: SELECT sc.sno,sc.cno FROM "CS_View",sc WHERE cno='1' AND "CS_View".sno=sc.sno

sno	cno
200215121	1

图 4.2 查询结果

- (3) 创建 IS 系成绩大于 80 的学生的视图 IS_View

SQL 语句: CREATE VIEW IS_View AS SELECT * FROM student WHERE student.sdept = 'IS' :: BPCHAR AND (student.sno IN (SELECT sc.sno FROM sc WHERE sc.grade > 80))

sno	sname	ssex	sage	sdept	scholarship
(N/A)	(N/A)	(N/A)	(N/A)	(N/A)	(N/A)

图 4.3 IS_View 视图

- (4) 在视图 IS_View 查询 IS 系成绩大于 80 的学生

由于 IS_View 视图本身的属性就是 IS 系成绩大于 80 的学生，所以直接查询整个视图 SELECT * FROM “IS_View”，由于结果同图 4.3 在此不在给出。

- (5) 删除视图 IS_View

DROP VIEW “IS_View” CASCADE /* 级联删除视图 */

4.2.2 对用户进行授权和收回权限

(1) 利用可视化窗口创建 2 个不同的用户 U1 和 U2,利用系统管理员给 U1 授予 Student 表的查询和更新的权限，给 U2 对 SC 表授予插入的权限。然后用 U1 登录，分别 1) 查询学生表的信息；2) 把所有学生的年龄增加 1 岁，然后查询；3) 删除 IS 系的学生；4) 查询 CS 系的选课信息。用 U2 登录，分别 1) 在 SC 表中插入 1 条记录（‘200215122’，‘1’，75）；2) 查询 SC 表的信息，3) 查询视图 CS_View 的信息。

1. U1 用户创建、权限授予、表操作

在 Navicat 中点击新建角色，键入相关信息如图 4.4 即可。

图 4.4 U1 用户创建

为该角色授予 s_t_u202112149 数据库中 student 表的查询和更新权限，效果如图 4.5

数据库	模式	名	Connect	Create	Delete	Execute	Insert	References	Select	Temporary	Trigger	Truncate	Update	Usage
s_t_u202112149	public	student	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

图 4.5 U1 表权限授予

① 查询学生表的信息，结果如下图 4.6:

SQL 语句: SELECT * FROM student

sno	sname	ssex	sage	sdept	scholarship
200215125	张立	男	19	IS	否
200215122	刘晨	女	19	CS	否
200215121	李勇	男	20	CS	否
200215123	王敏	女	18	MA	否

图 4.6 查询结果

② 把所有学生的年龄增加 1 岁，然后查询，结果如图 4.7:

SQL 语句: UPDATE student SET sage=sage+1

sno	sname	ssex	sage	sdept	scholarship
200215125	张立	男	20	IS	否
200215122	刘晨	女	20	CS	否
200215121	李勇	男	21	CS	否
200215123	王敏	女	19	MA	否

图 4.7 查询结果

- ③ 删除 IS 系的学生，结果如图 4.8，没有 student 关系的权限：

SQL 语句：DELETE FROM student WHERE sdept='IS'

查询	消息
DELETE FROM student WHERE sdept='IS'	ERROR: permission denied for relation student DETAIL: N/A

图 4.8 报错信息

- ④ 查询 CS 系的选课信息，结果如图 4.9，没有 sc 关系的权限：

SQL 语句：SELECT student.sno,sname,cno,grade FROM student,sc WHERE sc.sno=student.sno AND sdept='CS'

查询	消息
SELECT student.sno,sname,cno,grade FROM student,sc WHERE sc.sno=student.sno AND sdept='CS'	ERROR: permission denied for relation sc DETAIL: N/A

图 4.9 报错信息

2. U2 用户创建、权限授予、表操作

与 U1 用户相同，在 Navicat 新建角色中创建，不同的是授予权限的部分，见下图 4.10:

数据库	模式	名	Connect	Create	Delete	Execute	Insert	References	Select	Temporary	Trigger	Truncate	Update	Usage
s_t_u20211214	public	sc	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 4.10 U2 表权限授予

- ① 在 SC 表中插入 1 条记录('200215122', '1', 75)，如图 4.11，成功插入：

SQL 语句：INSERT INTO sc VALUES('200215122','1',75)

查询	消息
INSERT INTO sc VALUES('200215122','1',75)	Affected rows: 1

图 4.11 插入结果

- ② 查询 SC 表的信息，如图 4.12，查询失败，没有 sc 关系的权限：

SQL 语句：SELECT * FROM sc

查询	消息
SELECT * FROM sc	ERROR: permission denied for relation sc DETAIL: N/A

图 4.12 报错信息

- ③ 查询视图 CS_View 的信息，如图 4.13，查询失败，没有 CS_VIEW 关系的权限：

SQL 语句：SELECT * FROM "CS_VIEW"

查询	消息
SELECT * FROM "CS_VIEW"	ERROR: permission denied for relation CS_VIEW DETAIL: N/A

图 4.13 报错信息

(2) 用系统管理员登录，收回 U1 的所有权限

在图形界面处，删除 U1 对表 student 的所有权限即可，如图 XX

对象: * U1 (opengauss) - 角色														
保存 添加权限 删除权限														
常规 成员属于 成员 权限 注释 SQL 预览														
数据库: s_t_u202112149														
数据库	模式	名	Connect	Create	Delete	Execute	Insert	References	Select	Temporary	Trigger	Truncate	Update	Usage
			<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 4.14 删除 U1 所有权限

(3) 用 U1 登录，查询学生表的信息

SQL 语句：SELECT * FROM student

如下图 4.15，发现此时没有 student 关系的权限，查询失败。

查询 SELECT * FROM student	消息 ERROR: permission denied for relation student DETAIL: N/A
-----------------------------	--

图 4.15 报错信息

(4) 用系统管理员登录

如下图 4.16, 只有管理员 opengauss 连接处于激活状态, U1 和 U2 连接均关闭。

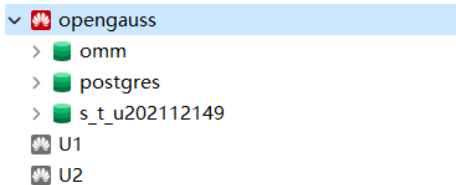


图 4.16 用系统管理员登录

4.2.3 触发器的定义与操作

(1) 对 SC 表建立一个更新触发器, 当更新了 SC 表的成绩时, 如果更新后的成绩大于等于 95, 则检查该成绩的学生是否有奖学金, 如果奖学金是“否”, 则修改为“是”。如果修改后的成绩小于 95, 则检查该学生的其他成绩是不是有大于 95 的, 如果都没有, 且修改前的成绩是大于 95 时, 则把其奖学金修改为“否”。然后进行成绩修改, 并进行验证是否触发器正确执行。1) 首先把某个学生成绩修改为 98, 查询其奖学金。2) 再把刚才的成绩修改为 80, 再查询其奖学金。

1. 定义触发器函数 update_scholarship()

创建触发器函数的 SQL 语句如图 4.17:

```

1 CREATE OR REPLACE FUNCTION update_scholarship()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     -- 如果更新后的成绩达到或超过95分
5     IF NEW.grade >= 95 THEN
6         -- 检查是否已有奖学金, 如果没有则更新
7         IF EXISTS (SELECT 1 FROM Student WHERE Sno = NEW.Sno AND Scholarship = '否') THEN
8             UPDATE Student SET Scholarship = '是' WHERE Sno = NEW.Sno;
9         END IF;
10    -- 如果更新前的成绩是95分以上, 且更新后的成绩低于95分
11    ELIF OLD.grade >= 95 AND NEW.grade < 95 THEN
12        -- 检查该学生是否还有其他成绩保持在95分以上
13        IF NOT EXISTS (SELECT 1 FROM SC WHERE Sno = NEW.Sno AND Grade >= 95) THEN
14            UPDATE Student SET Scholarship = '否' WHERE Sno = NEW.Sno;
15        END IF;
16    END IF;
17    RETURN NEW;
18 END;
19 $$ LANGUAGE plpgsql;

```

图 4.17 创建触发器函数的 SQL 语句

2. 定义触发器 sc_trigger

可以使用 SQL 语句进行触发器的配置, SQL 语句: CREATE TRIGGER "sc_trigger" AFTER UPDATE OF "grade" ON "sc" FOR EACH ROW EXECUTE PROCEDURE "update_scholarship"();

还有一种方式可以直接使用数据库可视化软件 Navicat 直接在工具里点击触发器一栏, 新建触发器, 进行类型、表名、触发、事件和函数的配置, 调用我们之前定义的函数 update_scholarship(), 具体的配置如下图 4.18:

☒ 启用

触发器类型: TABLE

表名: SC

给每个: ROW

触发: AFTER

事件: ☐ 插入 ☒ 更新 ☐ 删除 ☐ 截断

更新字段: ☐ sno ☐ cno ☒ grade

当:

函数名: public update_scholarship

参数:

图 4.18 sc_trigger 配置

3. 验证触发器是否正确执行

- ① 首先把某个学生成绩修改为 98，查询其奖学金，见图 4.19:

SQL 语句: UPDATE sc SET grade=98 WHERE sno='200215121' AND cno='1';

sno	sname	ssex	sage	sdept	scholarship
200215125	张立	男	20	IS	否
200215122	刘晨	女	20	CS	否
200215123	王敏	女	19	MA	否
200215121	李勇	男	21	CS	否

sno	cno	grade
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80
200215122	1	75
200215121	1	98

更新成绩

触发修改

图 4.19 触发器触发

- ② 再把刚才的成绩修改为 80，再查询其奖学金，见图 4.20:

SQL 语句: UPDATE sc SET grade=80 WHERE sno='200215121' AND cno='1';

sno	sname	ssex	sage	sdept	scholarship
200215125	张立	男	20	IS	否
200215122	刘晨	女	20	CS	否
200215123	王敏	女	19	MA	否
200215121	李勇	男	21	CS	是

sno	cno	grade
200215121	2	85
200215121	3	88
200215122	2	90
200215122	3	80
200215122	1	75
200215121	1	80

更新成绩

触发修改

图 4.20 触发器触发

(2) 删除刚定义的触发器

直接执行 SQL 语句: `DROP TRIGGER sc_trigger` 即可, 或者直接在可视化图形界面中点击删除触发器即可。

4.2.4 存储过程与函数的定义

(1) 定义一个存储过程计算 CS 系的课程的平均成绩和最高成绩, 在查询分析器或查询编辑器中执行存储过程, 查看结果。

SQL 定义语句如下图 4.21, 将查询结果存放入了一个临时表:

```
1 CREATE OR REPLACE PROCEDURE "public"."get_cs_avg_max"()
2 AS DECLARE
3 BEGIN
4     -- 先清理旧的临时表(如果存在)
5     DROP TABLE IF EXISTS temp_cs_results;
6     -- 创建一个新的临时表来存储结果
7     CREATE TEMP TABLE temp_cs_results (sno VARCHAR, avg_grade FLOAT, max_grade INT);
8
9     -- 将SELECT查询的结果插入到临时表中
10    INSERT INTO temp_cs_results
11    SELECT sc.sno, AVG(sc.grade) AS avg_grade, MAX(sc.grade) AS max_grade
12    FROM sc
13    JOIN student ON student.sno = sc.sno
14    WHERE student.sdept = 'CS'
15    GROUP BY sc.sno;
16
17    -- 如果需要查看结果, 可以在调用过程后查询这个临时表
18    RAISE NOTICE 'Results stored in temporary table temp_cs_results.';
19 END
20
```

图 4.21 过程定义

过程执行与表查看语句, 结果见图 4.22:

1. `CALL get_cs_avg_max();`
2. `SELECT * FROM temp_cs_results;`

sno	avg_grade	max_grade
200215121	84.33333333333333	88
200215122	81.66666666666667	90

图 4.22 过程调用结果

(2) 定义一个带学号为参数的查看某个学号的所有课程的成绩, 查询结果要包含学生姓名, 进行验证。

SQL 定义语句如下图 4.23, 同样将查询结果存放入了一个临时表:

```
1 CREATE OR REPLACE PROCEDURE "public"."get_student_grade"(_sno VARCHAR)
2 AS BEGIN
3     -- 先清理旧的临时表(如果存在)
4     DROP TABLE IF EXISTS temp_student_grades;
5     -- 创建一个新的临时表来存储结果
6     CREATE TEMP TABLE temp_student_grades (student_name VARCHAR, course_id VARCHAR, grade INT);
7
8     -- 将查询结果插入到临时表中
9     INSERT INTO temp_student_grades
10    SELECT student.sno, student.sname, sc.cno, sc.grade
11    FROM student
12    JOIN sc ON student.sno = sc.sno
13    WHERE student.sno = _sno;
14
15    -- 输出到通知中以确认操作
16    RAISE NOTICE 'Results for student sno % stored in temporary table temp_student_grades.', _sno;
17 END
```

图 4.23 过程定义

过程执行与表查看语句，结果见图 4.24：

1. CALL get_student_grade('200215121');
2. SELECT * FROM temp_student_grades;

student_id	student_name	course_id	grade
200215121	李勇	1	80
200215121	李勇	2	85
200215121	李勇	3	88

图 4.24 过程调用结果

(3) 把上一题改成函数。再进行验证。

如下图 4.25，由于函数与过程不同可以有返回值，所以可以直接将查询的结果作为返回值返回，定义的格式和过程有差别，且返回值类型要严格匹配。

```

1 CREATE OR REPLACE FUNCTION "public"."get_student_grade"("_sno" varchar)
2 RETURNS TABLE("student_id" bpchar, "student_name" bpchar, "course_id" bpchar, "grade" int2)
3 LANGUAGE plpgsql VOLATILE
4 COST 100
5 ROWS 1000
6 AS $BODY$ BEGIN
7     RETURN QUERY
8     SELECT student.sno, student.sname, sc.cno, sc.grade
9     FROM student
10    JOIN sc ON student.sno = sc.sno
11   WHERE student.sno = _sno;
12 END; $BODY$

```

图 4.25 函数定义

函数调用语句：SELECT get_student_grade('200215121'); 执行结果见图 4.26

get_student_grade
(200215121,"李勇", "1", 80)
(200215121,"李勇", "2", 85)
(200215121,"李勇", "3", 88)

图 4.26 函数调用结果

4.2.5 用户定义完整性约束

(1) 在 SC 表上定义一个完整性约束，要求成绩再 0-100 之间。定义约束前，先把某个学生的成绩修改成 120，进行查询，再修改回来。定义约束后，再把该学生成绩修改为 120，然后进行查询。

SQL 语句：ALTER TABLE sc ADD CONSTRAINT grade_check CHECK (grade >= 0 AND grade <= 100);

sno	cno	grade
200215121	2	120
200215121	3	88
200215122	2	90
200215122	3	80
200215122	1	75
200215121	1	80

图 4.27 定义完整性约束之前

sno	cno	grade
200215121	3	88
200215122	2	120
200215122	3	80
200215122	1	75
200215121	1	80
200215121	2	85

ERROR: new row for relation "sc" violates check constraint "grade_check"
DETAIL: N/A

图 4.28 定义完整性约束之后

4.3 任务总结

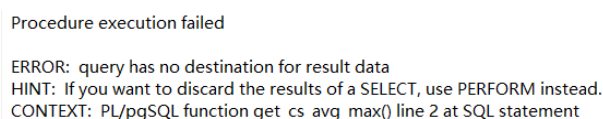
(1) 名称大小写与双引号问题

当使用命令行创建视图和表的时候大小写默认都会转成小写，所以这时候没有什么问题，但当我使用 Navicat 进行可视化的创建表和视图的时候，并且名称有大小写之分，这时候直接对表和视图进行操作的话，会报错找不到。

解决方案：这是因为不同工具对大小写敏感的处理不同（Navicat 默认区分大小写），如果大小写敏感的情况下使用双引号括起标识符，表明其区分大小写，而且每次操作涉及该表的时候都需要使用双引号。

(2) query has no destination for result data

在 PostgreSQL 中，执行存储过程时遇到一个错误，显示为：“query has no destination for result data”，如图 4.29。这个错误发生在执行了一个 SELECT 查询，但没有为这个查询提供一个合适的输出目标。由于 SELECT 查询生成结果，而这个结果在函数中没有被处理或返回，所以数据库系统不知道该如何处理这些数据。



```
Procedure execution failed
ERROR: query has no destination for result data
HINT: If you want to discard the results of a SELECT, use PERFORM instead.
CONTEXT: PL/pgSQL function get_cs_avg_max() line 2 at SQL statement
```

图 4.29 报错信息

解决方案：根据错误信息的提示：“If you want to discard the results of a SELECT, use PERFORM instead.”，不需要处理 SELECT 查询的结果，可以使用 PERFORM 语句代替 SELECT。PERFORM 在 PL/pgSQL 中用于执行一个操作但不需要返回结果的查询。因为过程不同于函数，所以我将查询的结果存入了一个临时的 table 中用于让用户查询。

(3) 尝试修改函数报错

在尝试修改一个已存在的 PostgreSQL 函数时遇到错误：“cannot change return type of existing function”，如图 4.30。这个错误提示表明尝试改变一个函数的返回类型，但 PostgreSQL 不允许直接修改现有函数的返回类型。在 PostgreSQL 中，一旦函数被创建，其返回类型就被固定，不能通过简单的 ALTER FUNCTION 命令更改。

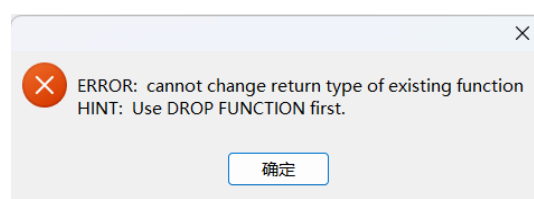


图 4.30 报错信息

解决方案：根据错误信息中的提示：“Use DROP FUNCTION first.”，如果需要修改函数的返回类型，首先需要删除当前的函数，可以通过使用 DROP FUNCTION 语句完成。删除函数后，可以重新定义函数，包括新的返回类型。

(4) 函数返回类型不匹配错误

在执行 PostgreSQL 函数 `get_student_grade` 时出现错误：“structure of query does not match function result type”。如图 4.31，根据报错显示，函数试图返回一个 `character(9)` 类型的数据，但该函数的定义期望的是 `character varying` 类型。这种类型不匹配导致函数无法正确执行并返回结果。

```
Procedure execution failed

ERROR: structure of query does not match function result type
DETAIL: Returned type character(9) does not match expected type character varying in column 1.
CONTEXT: PL/pgSQL function get_student_grade(character varying) line 2 at RETURN QUERY
referenced column: get_student_grade
```

图 4.31 报错信息

解决方案：需要确保函数中的 `RETURN QUERY` 语句返回的数据类型与函数定义的返回类型完全匹配。根据 `student` 关系中的数据类型定义，将字符类型的修改为 `bpchar`，整数类型的修改为 `int2`(对应 `smallint`)。

(5) 违反检查约束错误

在执行用户完整性约束定义操作时遇到错误：“check constraint 'grade' is violated by some row”，如图 4.32。这表明在进行插入或更新操作时，某些行违反了名为 `'grade'` 的检查约束。检查约束是数据库中用来确保列中数据满足特定条件的一种规则。如果违反了这些条件，数据库将拒绝相关的数据修改请求。

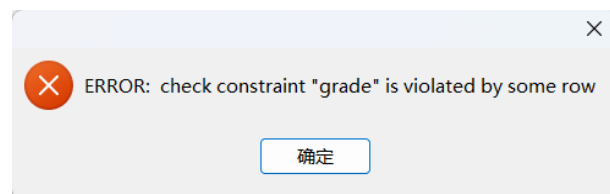


图 4.32 报错信息

解决方案：当时没有把 120 这条违反了该检查约束的数据修改回去，修改回去后，该检查约束添加成功。这提醒了我只有当前表中的数据符合该检查的时候，这个检查约束才能生效。

在该部分实验中，我实践了对数据库的一些高级操作，包括视图、授权、触发器、存储过程和函数、完整性约束等，对数据库有了进一步的认知。

5 数据库设计

5.1 任务要求

熟练掌握使用 SQL 语句设计数据库的方法，实现前述实验的学生管理系统。

系统功能要求：

- (1) 新生入学信息增加，学生信息修改。
- (2) 课程信息维护（增加新课程，修改课程信息，删除没有选课的课程信息）。
- (3) 录入学生成绩，修改学生成绩。
- (4) 按系统统计学生的平均成绩、最好成绩、最差成绩、优秀率、不及格人数。
- (5) 按系对学生成绩进行排名，同时显示出学生、课程和成绩信息。
- (6) 输入学号，显示该学生的基本信息和选课信息。

5.2 完成过程

在该实验部分，我主要采用 Python 语言开发，使用 openGauss 作为数据库，配合 Python 库 tkinter（提供简单的方式来构建桌面应用程序的界面）和 psycopg2（提供和 PostgreSQL 数据库交互的能力）。开发时使用的 IDE 为 PyCharm2023，其提供了可视化的数据库控制界面，支持 SQL 语法的检查和试运行测试，非常的方便。相关源代码我已上传至 [Github](#)。

5.2.1 项目结构

下面给出项目的结构，如图 5.1：

```
pythonProject [DB_EXP4]
├── postgresSQL
│   ├── __init__.py           # 标识postgresSQL为一个Python包
│   ├── DB_Info.py           # 包含查询数据库信息的函数
│   ├── Manage_Course_Info.py # 处理课程信息管理的函数
│   ├── Manage_Grade_Info.py # 处理成绩信息管理的函数
│   └── Manage_Student_Info.py # 处理学生信息管理的函数
├── resources
│   ├── icon.ico             # 图标文件
│   └── login_cover.jpg       # 登录封面图片
├── .gitignore               # 配置Git忽略哪些文件和文件夹
├── connection_pool.py       # 数据库连接池的实现（在此修改数据库连接信息）
├── main.py                  # 主程序入口文件
├── manage.py                # 管理应用程序的脚本
├── output.log               # 输出日志文件
├── README.md                # 项目说明文件
└── images                   # 项目说明文件的配图
```

图 5.1 项目结构树图

下面对主要的文件做一些解释说明：

- (1) `main.py` 为程序的主入口，提供了用户登录界面，效果展示如图 5.2。



图 5.2 用户登录界面效果图

- (2) `manage.py` 为登录后的系统管理界面，主要定义了相关的 UI 组件，登陆后默认显示连接数据库的相关信息，例如版本、用户、数据库和表信息，效果展示如图 5.3 所示。



图 5.3 系统管理界面效果图

- (3) `connection_pool.py` 文件的作用是管理与 PostgreSQL 数据库的连接，通过使用连接池来优化数据库连接的管理，这是我在思考频繁执行 SQL 语句导致的连接和断开会不会带来的很大的性能开销时想到的解决办法。其主要提供了获取、释放和关闭数据库连接的功能，从而优化了资源管理和应用性能，简化了数据库连接的管理过程。其本质就是维护一组数据库连接，这些连接在程序的生命周期内不会被关闭。当应用程序需要与数据库通信时，它可以从连接池中获取一个现有的连接，而不需要重新建立一个新的连接。当不再需要使用连接时，这个连接会被归还到连接池中，而不是被关闭。

(4) **postgreSQL** 文件夹包含了与数据库交互的所有模块。每个模块负责处理不同类型的数据操作，如课程信息、成绩信息和学生信息等。

- **DB_Info.py**: 包含查询数据库基本信息的函数，如数据库版本、当前用户、已连接的数据库名称等。
- **Manage_Student_Info.py**: 处理学生信息管理的相关函数，如插入、更新、删除和根据学号查询学生基本信息和选课信息等。
- **Manage_Course_Info.py**: 处理课程信息管理的相关函数，如插入、更新、删除和删除没有选课记录的课程信息等。
- **Manage_Grade_Info.py**: 处理成绩信息管理的相关函数，如插入、更新、删除、按系统统计学生成绩信息（包括平均成绩、最好成绩、最差成绩、优秀率、不及格人数）和按系对学生成绩进行排名（显示出学生、课程和成绩信息）等。

综上所述，这种划分方式将不同的功能模块化，提供了清晰的职责分离和高代码重用性，优化了系统性能，具备良好的可扩展性和可维护性。这种结构不仅简化了开发和维护过程，还确保了系统面对未来扩展需求时的灵活性和稳定性。

5.2.2 代码编写细节

其实使用高级语言编写嵌入式的 SQL 流程相对来说比较的固定。

- (1) 通过 **get_connection** 向连接池申请一条连接。
- (2) 创建游标，利用游标执行相应的 SQL 语句。
- (3) 执行成功则 **commit** 提交事务，否则 **rollback** 回退。

我编写过程中主要遇到两种模式，一种是直接将所有需要插入的字段及其值包含在 SQL 语句当中，另一种是根据条件动态生成需要更新的字段和对应的值。

例如插入操作，如图 5.4:

```
cursor.execute( query: """
    INSERT INTO Course (cno, cname, cpno, ccredit)
    VALUES (%s, %s, %s, %s)
""", vars: (cno if cno else None, cname if cname else None, cpno if cpno else None, ccredit if ccredit else None))
```

图 5.4 全部包含类型的 SQL 语句

例如更新操作，需要动态的生成需要更新的部分，通过条件选择形成 **query** 和 **values** 然后放入 **execute** 执行，如图 5.5:

```
update_query = f"UPDATE Course SET {' '.join(update_fields)} WHERE cno = %s"
update_values.append(cno)

conn = get_connection()
if conn:
    try:
        cursor = conn.cursor()
        cursor.execute(update_query, update_values)
```

图 5.5 动态生成的 SQL 语句

至于 SQL 语句的编写，本实验要实现的功能基本与前述实验类似，故在此不再赘述其编写方法，下面主要是针对每个任务进行效果的演示。

在编写代码的时候，SQL 语句执行前我就已经根据表的约束对字段的类型和长度进行非常完备的限制和检查，然后弹窗提示用户，比如学号必须为 9 位数字、姓名为中文、成绩的取值限制等等。而且针对相应的操作对应的某些字段不能为空。针对插入操作，除键之外（主键和外键，当然还包括 Unique 修饰的学生姓名字段）不为空，其余都能为空；针对更新操作，如果留空则默认保持原来的值；针对删除操作，只给主键就能删除一条记录，也能同时填入别的字段，删除时用 AND 合并条件进行删除。

对于语句执行前不能进行的判断，比如删除学生表记录时，如果他在成绩表中有成绩则会造成冲突；又或者插入时主键和已存在的记录相同造成的冲突。这些报错均由数据库报错返回，也会通过 Error 弹窗的形式提示用户。

综上，该程序具有良好的用户交互性以及友好的使用方式，部分如图 5.6：

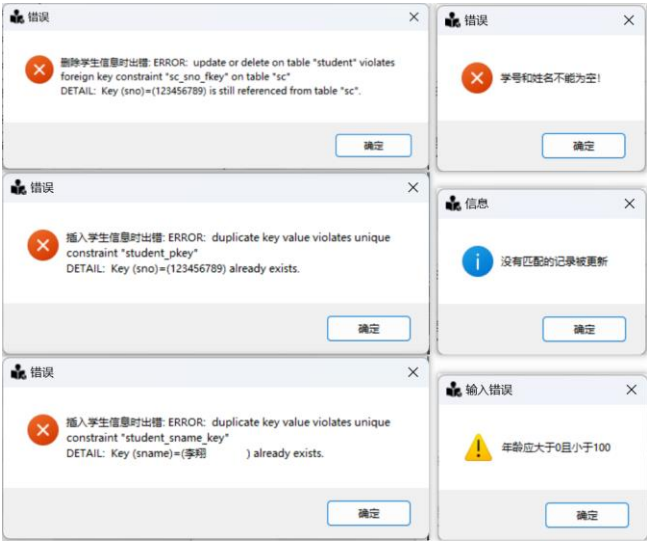


图 5.6 完备的检查与用户提示弹窗

5.2.3 学生信息维护

(1) 学生信息插入

插入学号为 123456789 姓名为李翔（本人姓名），年龄为 20 的男性 CS 系同学，有奖学金，结果如图 5.7 所示。

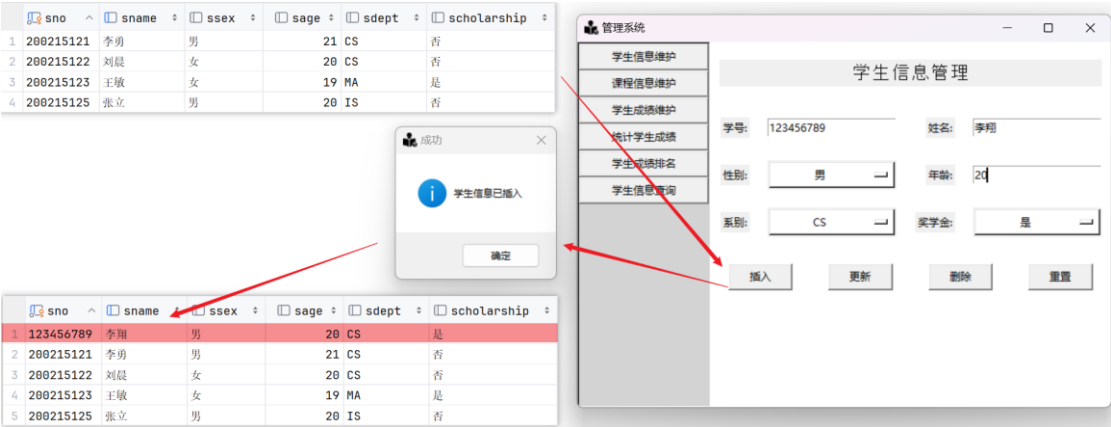


图 5.7 学生信息插入

(2) 学生信息更新

更新学号为上面插入的一条记录的系别 IS，奖学金为否，结果如图 5.8 所示。

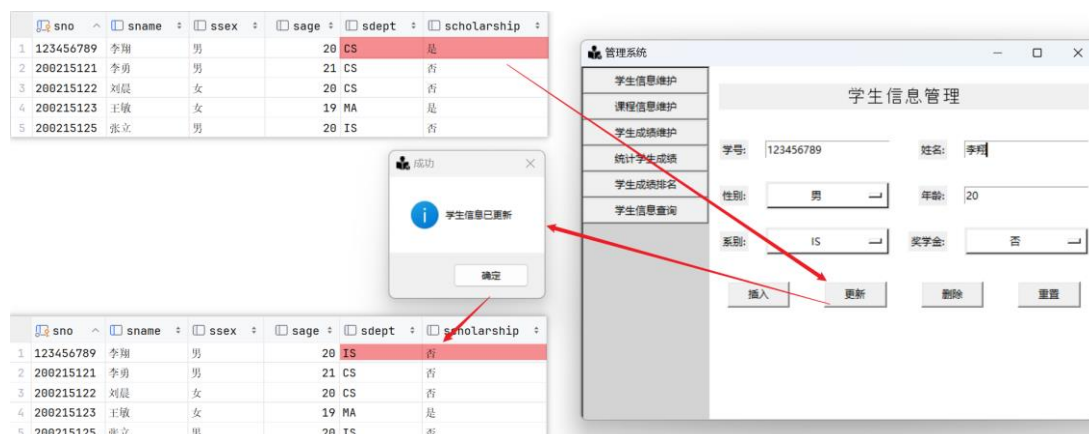


图 5.8 学生信息更新

(3) 学生信息删除

删除刚刚插入和更新的信息，仅通过学号就能删除，也可以提供更加严格的条件进行匹配删除，结果如图 5.9 所示。

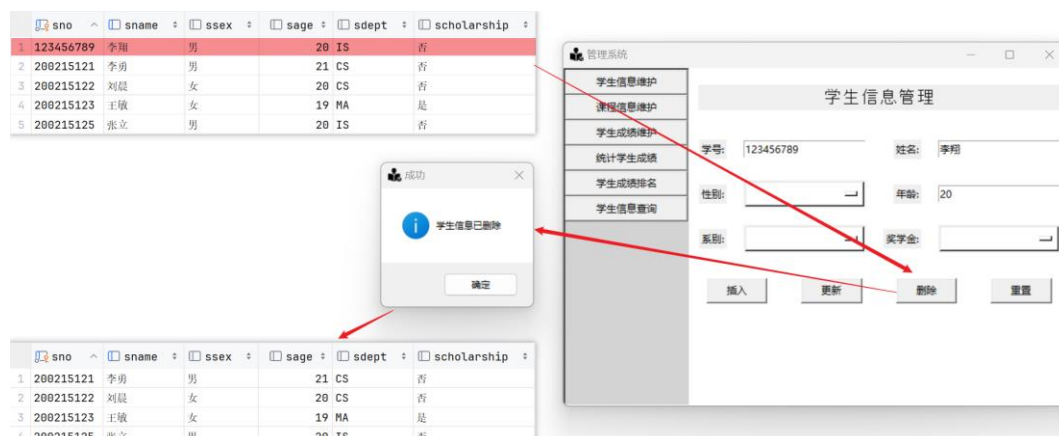


图 5.9 学生信息删除

5.2.4 课程信息维护

(1) 课程信息插入

插入课程号为 8，Python 语言的 4 学分课程，先修课暂时留空，如图 5.10。

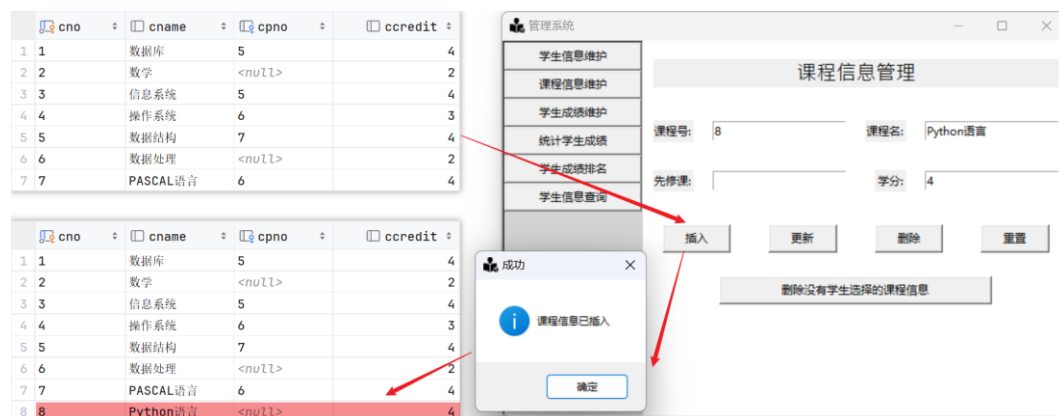


图 5.10 课程信息插入

(2) 课程信息更新

将上面插入的课程先修课更改为数据处理，即 cno=6 的课程，如图 5.11。

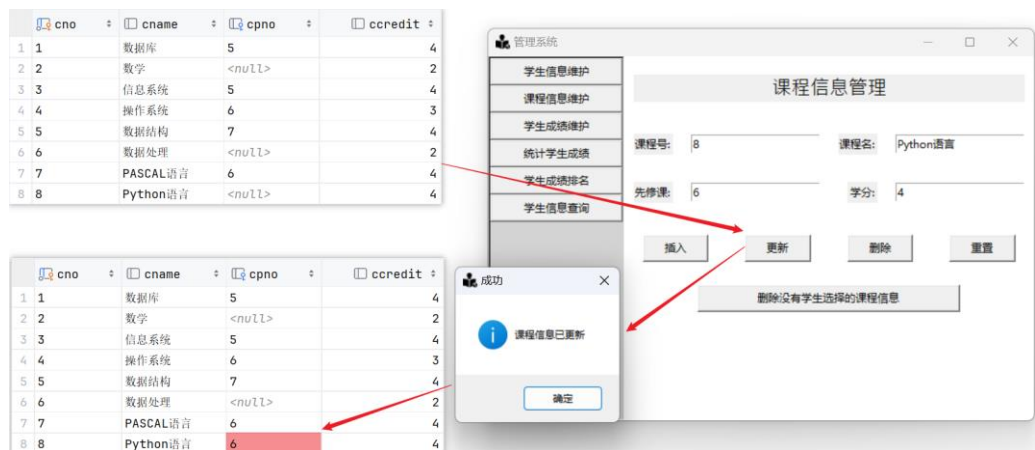


图 5.11 课程信息更新

(3) 课程信息删除

将上面插入和更新的 8 号课程 Python 语言删除，如图 5.12。

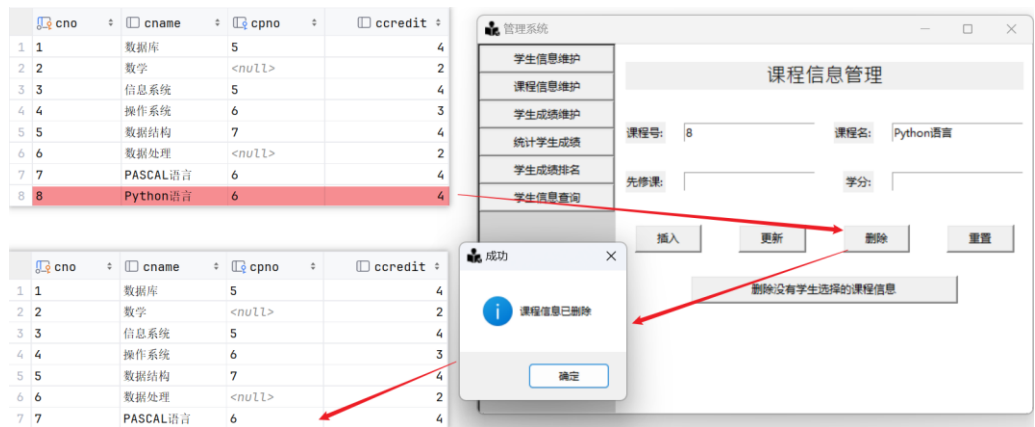


图 5.12 课程信息删除

(4) 删除没有学生选择的课程信息

先插入一些数据，构造出只有 JAVA 语言课没有学生选择，如图 5.13 所示。

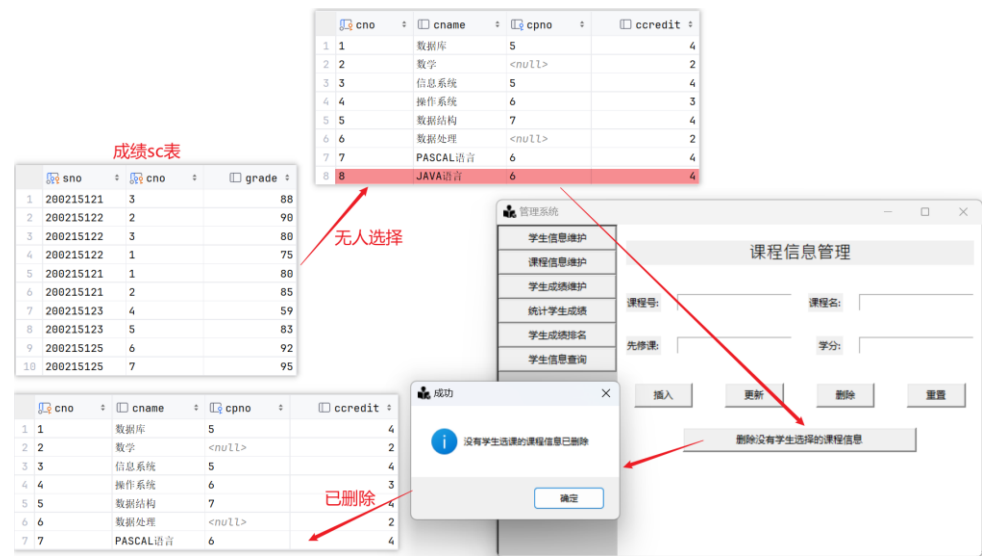


图 5.13 删除没有学生选择的课程信息

5.2.5 学生成绩维护

(1) 学生成绩插入

插入学号为 200215125 且选修课程号为 2，成绩暂时留空的记录，如图 5.14。

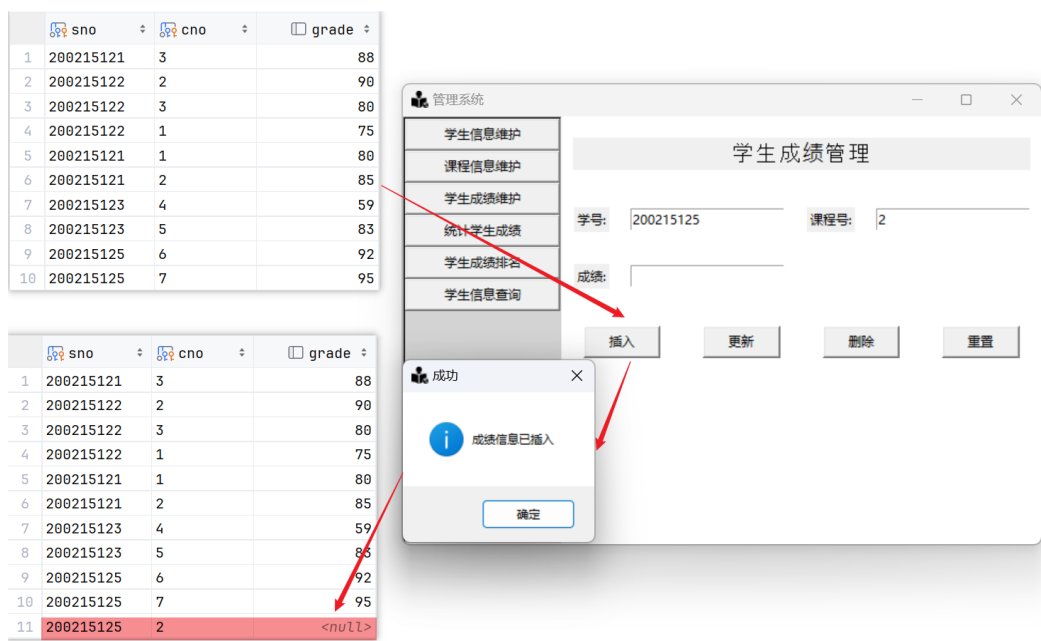


图 5.14 学生成绩插入

(2) 学生成绩更新

将上面刚插入的记录的 grade 字段更新成 100，结果如图 5.15 所示。

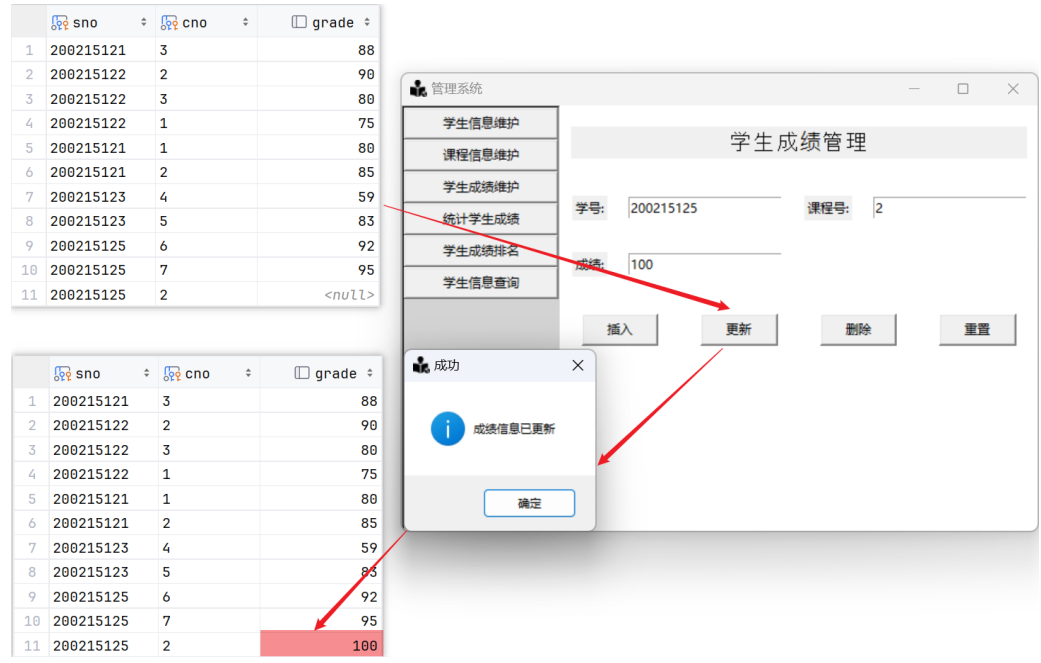


图 5.15 学生成绩更新

(3) 学生成绩删除

删除上面插入和更新的记录，结果如图 5.16 所示。

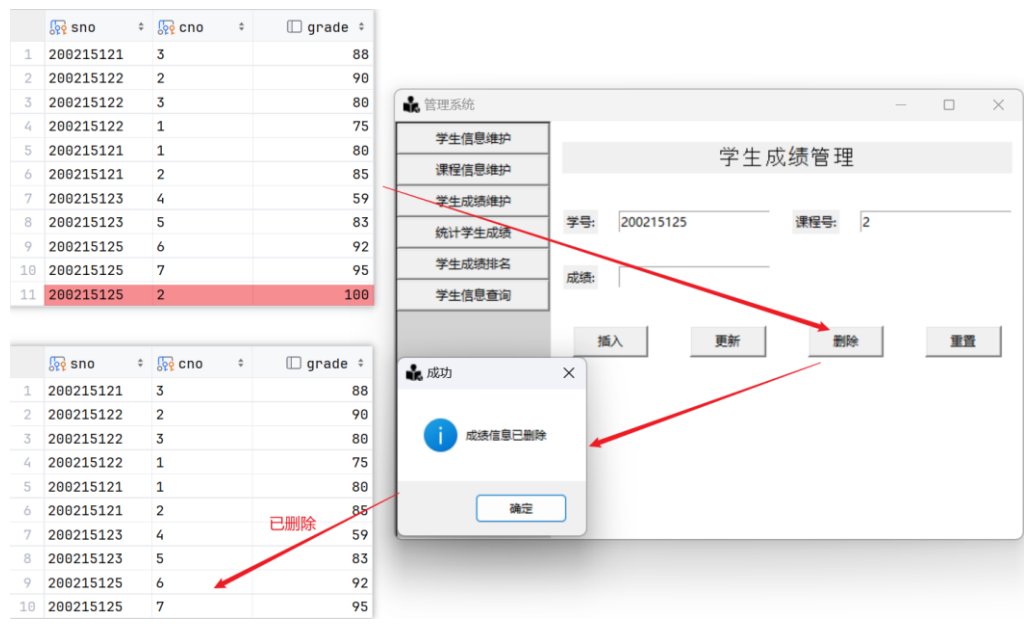


图 5.16 学生成绩删除

5.2.6 统计学生成绩

点击按系统统计学生信息的按钮，得到统计结果，包括系别、平均成绩、最好成绩、最差成绩、优秀率和不及格人数等项，结果如图 5.17 所示。

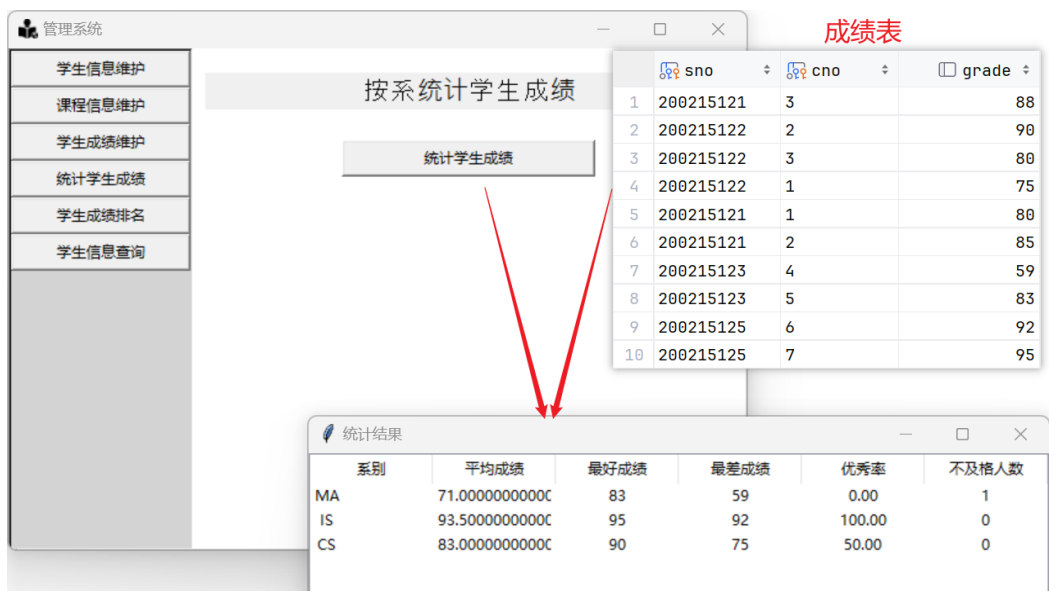


图 5.17 按系统统计学生信息

5.2.7 学生成绩排名

点击按系对学生排名按钮，得到统计结果，系别、学号、姓名、课程号、课程名以及成绩这些字段信息，具体的结果如图 5.18 所示。

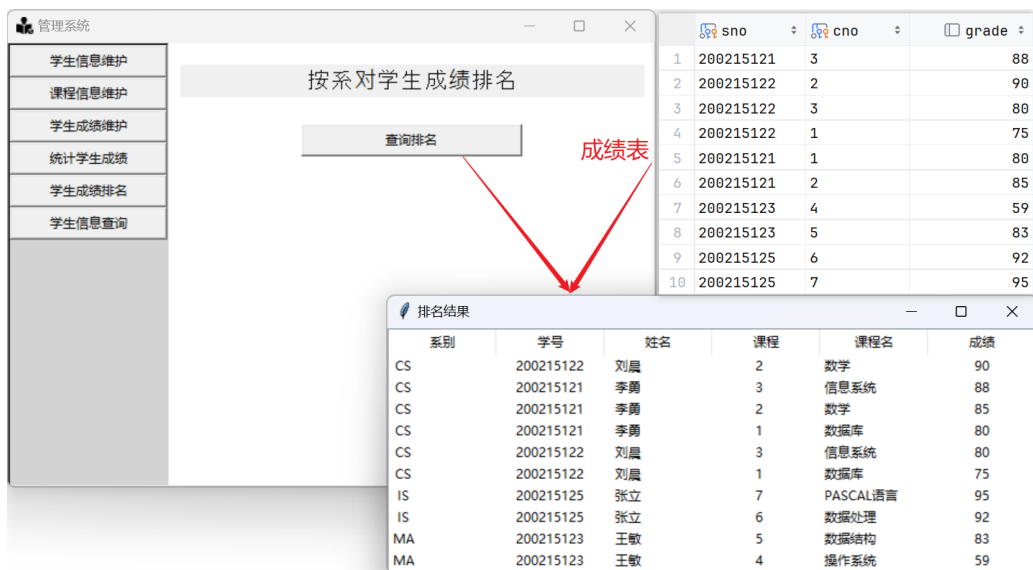


图 5.18 按系对学生排名

5.2.8 学生信息查询

输入学号 200215121 查询该学生的相关信息，包括学号、姓名、性别、年龄、系别、选修课程名、课程号以及成绩，结果如图 5.19 所示

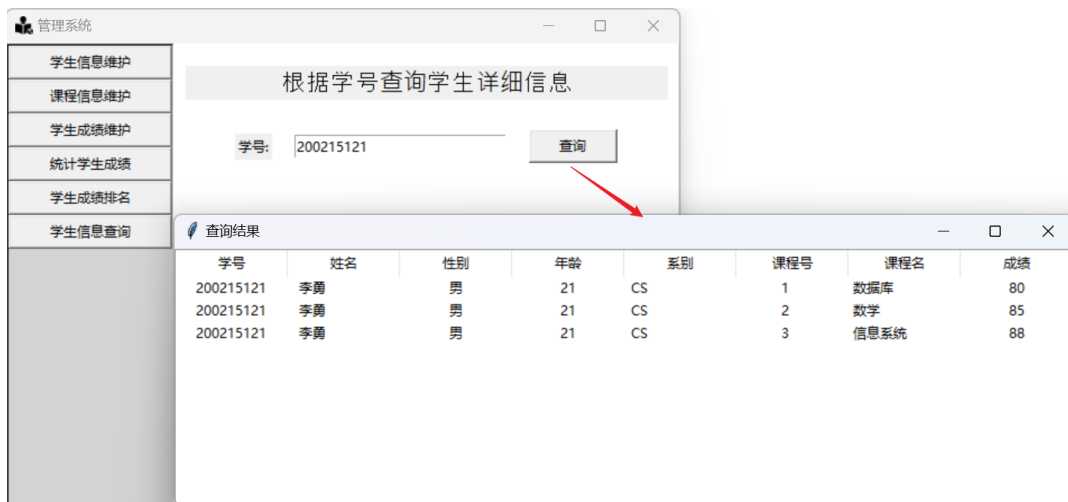


图 5.19 学生信息查询

5.2.8 日志输出

本应用程序同时提供了详尽的日志记录供用户审计，效果见图 5.20。

```
2024-06-07 13:20:29,722 ERROR Failed to delete courses without enrollments: ERROR: update or delete on table "course" violates foreign key constraint
DETAIL: Key (cno)=(5 ) is still referenced from table "course".

2024-06-07 13:20:39,296 INFO Successfully deleted course info: cno=8, cname=Python\u8bed\u8a00, cpno=, ccredit=
2024-06-07 13:34:04,582 INFO Successfully deleted courses without enrollments
2024-06-07 13:42:09,921 INFO Successfully retrieved grade statistics grouped by department
```

图 5.20 日志记录

5.3 任务总结

在本次实验中，我利用 Python 语言结合 openGauss 数据库，使用 tkinter 构建用户界面，并通过 psycopg2 库实现与数据库的交互，成功开发了一个学生管理系统。项目结构清晰，功能模块化，主要实现了学生信息的增删改查、课程信息维护、学生成绩录入与修改、学生成绩统计与排名等功能。通过连接池管理数据库连接，优化了系统性能，并提供了详尽的日志记录供用户审计。开发过程中，我对输入参数进行了严格的校验和提示，确保系统的可靠性和用户体验。最终的系统不仅简化了开发和维护过程，还具备良好的可扩展性和稳定性，能够高效地处理各种学生管理相关的数据库操作。

在整个项目代码编写的过程中我认为主要的难度来源于项目代码逻辑结构的划分，我觉得自己在这个方面的思考时间是最长的，UI 虽然格式调整的比较麻烦，但是写好一个熟悉一下其语言定义的特性，剩下的也比较好写。

我也搜索参考了一下网上的代码，大多数都是杂糅成一个文件，代码结构混乱，使用的大多也是 mysql 数据库，而我是 openGauss，所以我基本自己重头写了一下这个项目，而且提供了非常详细的函数注释，但最重要的改进还是对模块进行划分形成一个结构清晰的项目。

而且在这个过程中我也考虑到了每次执行语句新建连接，执行完毕之后再拆除连接，这个过程也是没有必要的，于是我引入了连接池，能降低一点额外的性能开销。

在代码编写过程中我自己都发现不少的小 bug，自己遇到一个就修复一个，当然肯定还有没发现的 bug 需要修复。

因为这是我第一次结合数据库开发一个应用程序，我使用的是 IDE 是 PyCharm 能够很方便的连接数据库，提供可视化界面，对 SQL 也有很好的支持。

6 课程总结

6.1 主要工作

本实验主要做了以下几个工作：

- (1) **数据库设计与实现：**使用 SQL 语句设计了学生管理系统的数据库，包括创建学生表、课程表和成绩表。通过定义表结构、添加约束条件、创建索引等方式，确保了数据库的完整性和高效性。
- (2) **基本操作：**掌握了基本的 SQL 操作，如插入、更新、删除和查询。通过实际操作，熟悉了 SQL 语句的语法和使用方法，并学会了使用 Navicat 等工具进行数据库管理。
- (3) **复杂操作：**进行了复杂的 SQL 查询练习，包括多表连接、嵌套查询、集合查询等。通过这些练习，我深入理解了 SQL 的强大功能和灵活性，提高了处理复杂数据需求的能力。
- (4) **高级功能：**学习并实现了视图、存储过程、函数、触发器等高级数据库功能。通过创建和管理这些对象，增强了数据库的功能性和安全性，进一步优化了系统性能。
- (5) **应用程序开发：**使用 Python 语言开发了一个学生管理系统，利用 tkinter 库构建用户界面，通过 psycopg2 库与 openGauss 数据库进行交互。项目实现了学生信息管理、课程信息维护、成绩管理、统计与排名等功能。

6.2 心得体会

经过本实验我有如下的一些心得体会：

- (1) **理论与实践结合：**通过实践操作，将课堂上学到的理论知识应用到实际项目中，加深了对数据库原理和技术的理解。同时，实践过程中遇到的问题和挑战也促使我进一步学习和探索，提升了自己的问题解决能力。
- (2) **代码结构与优化：**在项目开发过程中，我意识到良好的代码结构和模块化设计的重要性。通过将不同功能模块化，实现了清晰的职责分离，提高了代码的可读性和可维护性。此外，使用连接池优化数据库连接管理，有效提升了系统性能。
- (3) **工具使用：**熟练使用了 PyCharm 和 Navicat 等开发工具，这些工具提供了丰富的功能和友好的用户界面，大大提高了开发效率和质量。通过这些工具，我能够更直观地管理数据库，编写和调试 SQL 语句。
- (4) **用户交互与体验：**在开发过程中，我比较注重用户输入的校验和提示，确保系统的可靠性和用户体验。通过弹窗提示和错误处理，增强了系统的友好性，使用户在操作过程中能够得到及时的反馈和指导。

6.3 有待改进和完善的工作

当然，我甚至这个过程中有很多做的还是不够，有以下几条：

- (1) 进一步优化代码：虽然当前的代码结构已经比较清晰，但仍有改进的空间。例如，引入更先进的设计模式和框架，进一步提高代码的可维护性和扩展性。
- (2) 完善错误处理机制：当前的错误处理机制主要依赖于简单的提示和日志记录，未来可以考虑引入更全面的错误处理策略和异常管理机制，提升系统的鲁棒性。
- (3) 功能扩展：现有系统实现了基本的学生管理功能，但还有许多可以扩展的功能。例如，增加更多统计分析功能、实现更加复杂的业务逻辑、提升系统的可用性和安全性等。
- (4) 用户界面优化：当前的用户界面虽然基本满足需求，但美观性和易用性方面还有改进空间。可以引入更多现代化的 UI 设计和交互方式，提升用户体验。

综上所述，通过这次课程实践，我不仅掌握了数据库系统的基本操作和高级功能，还提升了综合应用这些知识解决实际问题的能力。未来，我将继续学习和探索，不断提升自己的技术水平和项目开发能力。

附录

- [1] 朱良根,雷振甲,张玉清.数据库安全技术研究[J].计算机应用研究,2004,(09):127-129+138.
- [2] 程力.计算机网络数据库的安全管理技术分析[J].网络安全技术与应用,2022,(12):46-48.
- [3] 秦健,韩斌,崔芸.分布式数据库技术在大数据中的应用[J].电脑知识与技术,2022,18(30):54-56+70.
- [4] 王志辉.分布式数据库技术在大数据中的应用[J].信息系统工程,2019(12):21-22.
- [5] 孟小峰.数据库技术发展趋势[J].软件学报,2004,15(12):1822-1836.
- [6] domer. (2023). Python 学生管理系统（Tkinter GUI）使用 MySQL 数据库，期末作业可用！. 知乎专栏. <https://zhuanlan.zhihu.com/p/639034884>