



西安电子科技大学

XIDIAN UNIVERSITY

数据库系统 课程设计报告

姓名：赵宇盛

班级：1903013 计算机科学与技术专业

任课教师

2021 年 12 月 10 日

目录

0 简介	1
1 需求分析	2
1.1 需求语义	2
1.2 设计要求	2
1.2.1 模式设计	2
1.2.2 后端设计	2
1.2.3 前端设计	2
1.3 实体分析	2
2 设计分析	4
2.1 ER图	4
2.2 关系模式	4
2.3 根据关系模式创建数据库	5
2.3.1 系表 (Department)	5
2.3.2 专业表 (Major)	5
2.3.3 班表 (Class)	6
2.3.4 学生表 (Student)	6
2.3.5 学会表 (Conference)	7
2.3.6 参会表 (Attend)	7
3 后端设计	8
3.1 创建视图	8
3.2 创建触发器	10
3.3 创建存储过程完成班号替换	13
3.4 使用游标完成存储过程	15
4 前端设计	17
4.1 基本表的增删改查功能	18
4.2 对后端设计功能的使用	22
5 效果实现	25
6 心得体会	30

0 简介

应数据库系统课程要求，我们要求设计并实现一个数据库应用系统。

本课程设计报告应用工具栈中后端为mysql，前端为python，GUI界面由python的tkinter库实现，完成满足课设需求的学生管理系统。

本课设从前期设计准备、代码书写以及后期文档报告撰写都由笔者个人独立完成。

1 需求分析

该课设要求我们应用数据库系统相关知识设计并实现一个学生管理系统。

1.1 需求语义

该学生管理系统要建立关于系、学生、班级、学会等诸信息的一个关系数据库。

一个系有若干专业，每个专业每年只招一个班，每个班有若干学生。一个系的学生住在同一宿舍区。每个学生可参加若干学会，每个学会有若干学生。学生参加某学会有一个入会年份。描述各个实体的属性（加下划线者为实体标识符）如下：

学生：学号、姓名、年龄、系名、班号、宿舍区。

班级：班号、专业名、入校年份、系名、人数。

系：系号、系名、系办公室地点、人数。

学会：学会号、学会名、成立年份、地点。

1.2 设计要求

1.2.1 模式设计

1. 画出E-R图，将其转为关系模式，
2. 根据关系模式创建数据库。表名和属性名用英文，属性的数据类型根据上面的描述自己定义。

1.2.2 后端设计

1. 创建视图显示学会名和学生数。
2. 创建触发器，自动增减班级表和系表的人数字段的值。
3. 创建函数，实现更改旧班号为新班号，并返还此班人数的功能。
4. 创建存储过程，使用游标检查系表人数与实际学生人数是否相符，不符就更改系表该字段的值为实际数。

1.2.3 前端设计

1. 实现对基本表的增删改查功能。
2. 实现对后端设计四条要求所创建对象的使用

1.3 实体分析

通过设计要求可以设计出各种实体以及它们之间的关系图，这些实体包含各种具体信息，则实体有：班级实体，专业实体，院系实体，学生实体、学会实体。

一个系有多个专业，一个专业只能属于一个系，因此系与专业之间具有一对多关系。

一个专业只有一个班级，一个班级只能属于一个专业，因此专业与班级之间具有一对一关系。

一个班级有多个学生，一个学生只能属于一个班级，因此班级与学生之间具有一对多关系。

一个学生可以加入多个学会，一个学会可以被多个学生加入，因此学生与学会之间是多对多的联系。学生“入会”学会，所以用入会年份来表示学生和学会之间联系的属性。

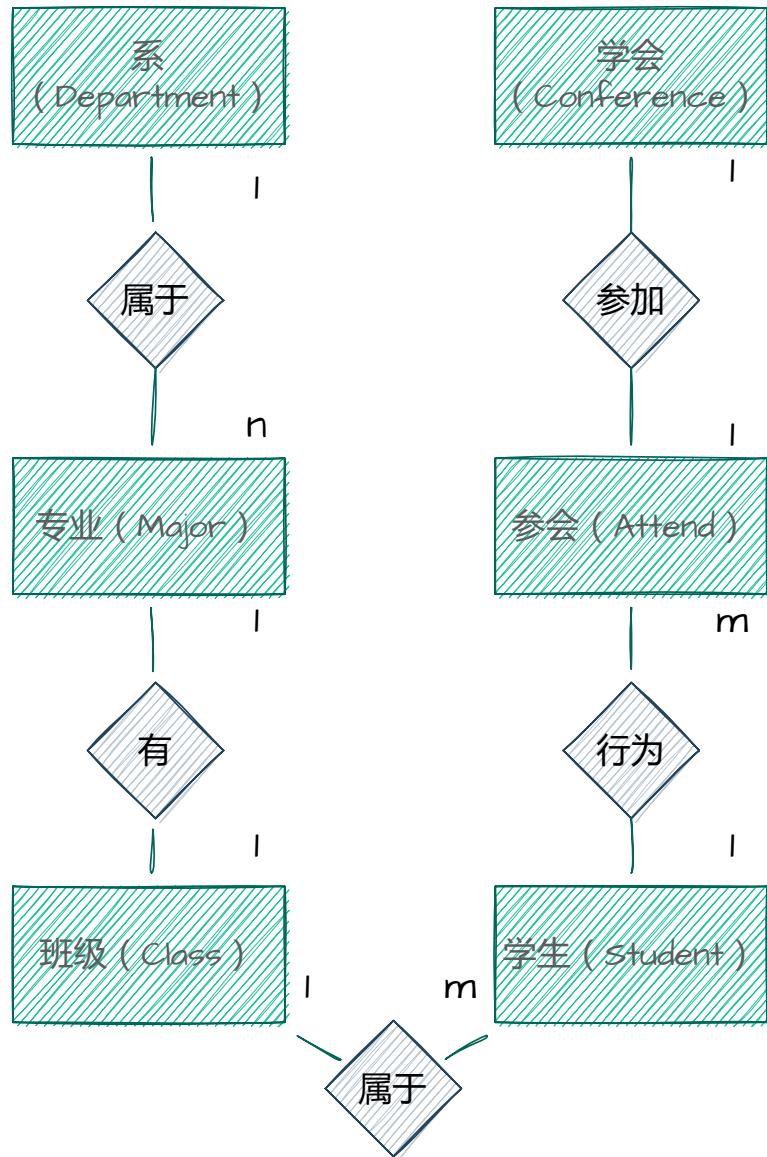


图 1: 实体之间的联系

2 设计分析

2.1 ER图

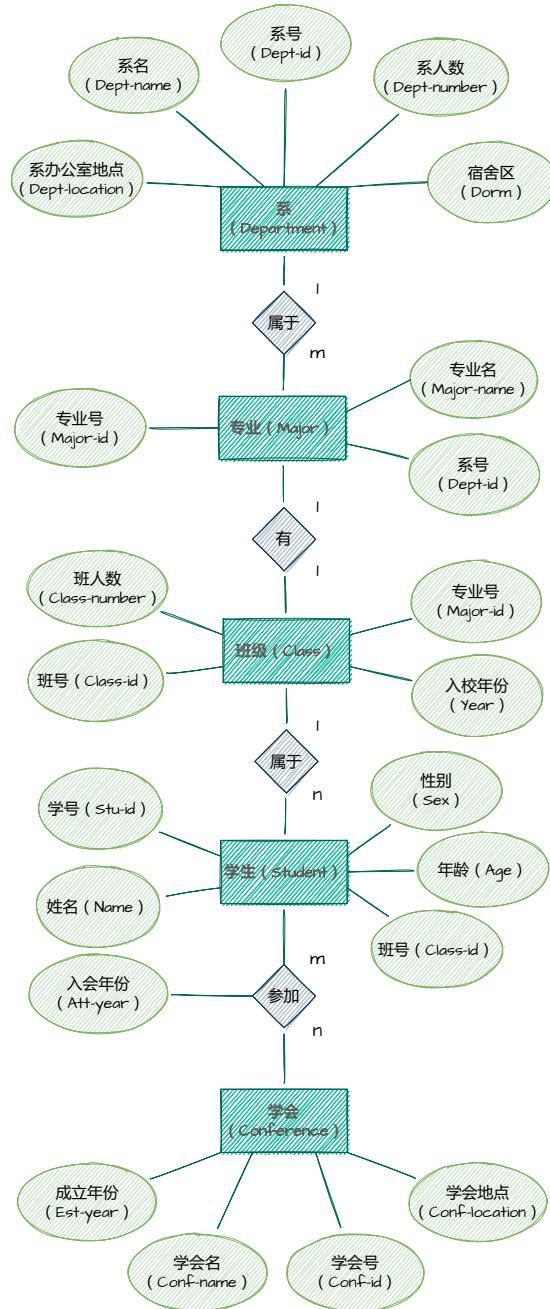


图 2: ER图

2.2 关系模式

由以上ER图，我们得到关系模式如下：

1. 系(系号, 系名, 系办公室地点, 系人数, 宿舍区)

2. 专业(专业号, 专业名, 系号)
3. 班级(班号, 专业号, 入校年份, 班人数)
4. 学生(学号, 姓名, 年龄, 性别, 班号)
5. 参会(学号, 学会号, 入会年份)
6. 学会(学会号, 学会名, 成立年份, 地点)

2.3 根据关系模式创建数据库

2.3.1 系表 (Department)

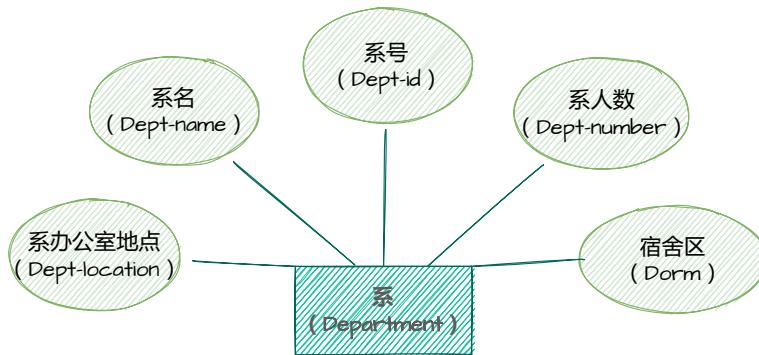


图 3: 系实体属性图

属性	属性名	数据类型	完整性及其他规定
系号	Dno	INT	主码, 值大于0
系名	Dname	CHAR(20)	唯一 (UNIQUE)
系人数	Dnum	INT	值大于等于0
系办公室地点	Dloc	CHAR(50)	NOT NULL
宿舍区	Dorm	INT	值大于0

范式分析:

Dno为主码, 在关系模式Department(Dno,Dname,Dnum, Dloc, Dorm) 中所有非主属性对码均完全依赖, 同时也没有对码的传递函数依赖。另外, 也没有主属性对码的部分和传递函数依赖。因此, 该关系模式范式为BCNF。

2.3.2 专业表 (Major)

属性	属性名	数据类型	完整性及其他规定
专业号	Mno	INT	主码, 值大于0
专业名	Mname	CHAR(20)	唯一 (UNIQUE)
系号	Dno	INT	外码, 参照完整性

范式分析:

Mno为主码, 关系模式Major(Mno, Mname, Dno)中所有非主属性对码均完全依赖, 同时也没有对码的传递函数依赖。另外, 也没有主属性对码的部分和传递函数依赖。因此, 该关系模式范式为BCNF。

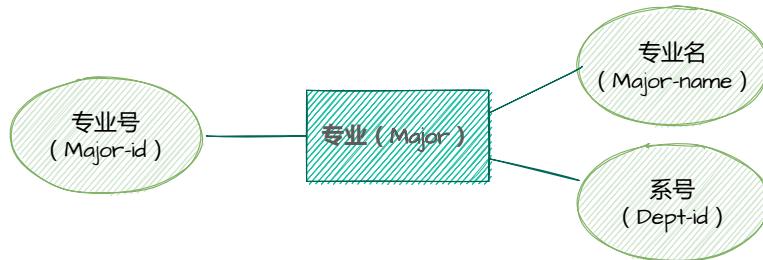


图 4: 专业实体属性图

2.3.3 班表 (Class)

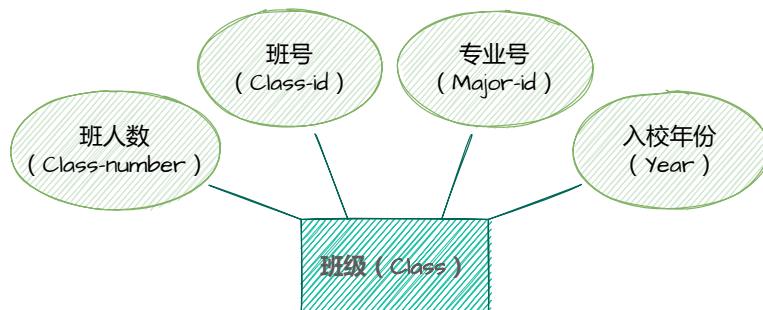


图 5: 班级实体属性图

属性	属性名	数据类型	完整性及其他规定
班号	Cno	INT	主码, 值大于0
专业号	Mno	INT	外码, 参照完整性
班人数	Cnum	INT	值大于等于0
入校年份	Cyear	YEAR	自定义完整性: 大于1949, 小于等于2021

范式分析:

Cno为主码, 关系模式Class(Cno,Mno,Cnum,Cyear)中所有非主属性对码均完全依赖, 同时也没有对码的传递函数依赖。另外, 也没有主属性对码的部分和传递函数依赖。因此, 该关系模式范式为BCNF。

2.3.4 学生表 (Student)

属性	属性名	数据类型	完整性及其他规定
学号	Sno	INT	主码, 值大于0
姓名	Sname	CHAR(20)	可以重名
性别	Sex	CHAR(8)	自定义完整性: 男或女
年龄	Age	INT	大于等于7, 小于等于100
班号	Cno	INT	外码, 参照完整性

范式分析:

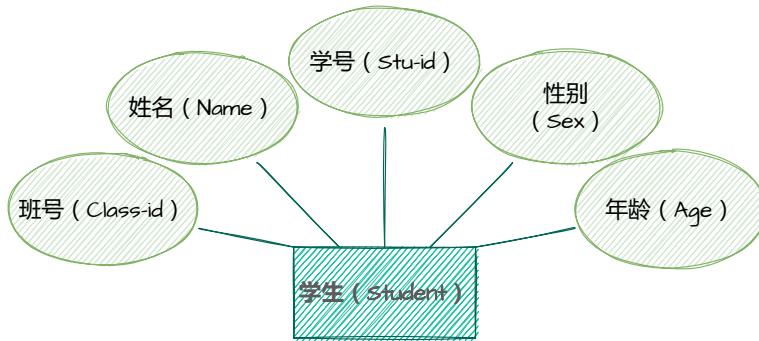


图 6: 学生实体属性图

Sno为主码，关系模式Student(Sno,Sname,Sex,Age,Cno)中所有非主属性对码均完全依赖，同时也没有对码的传递函数依赖。另外，也没有主属性对码的部分和传递函数依赖。因此，该关系模式范式为BCNF。

2.3.5 学会表 (Conference)

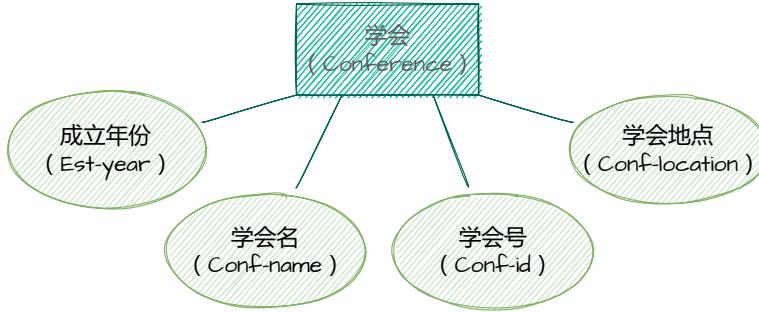


图 7: 学会实体属性图

属性	属性名	数据类型	完整性及其他规定
学会号	Confno	INT	主码，值大于0
学会名	Confname	CHAR(20)	唯一 (UNIQUE)
成立年份	Eyear	YEAR	大于1949，小于2021
学会地点	Confloc	CHAR(50)	NOT NULL

范式分析：

Confno为主码，关系模式Conference(Confno,Confname, Eyear,Confloc)中所有非主属性对码均完全依赖，同时也没有对码的传递函数依赖。另外，也没有主属性对码的部分和传递函数依赖。因此，该关系模式范式为BCNF。

2.3.6 参会表 (Attend)

属性	属性名	数据类型	完整性及其他规定
学号	Sno	INT	主码，值大于0
学会号	Confno	INT	主码，值大于0
入会时间	Atime	DATE	该时间应在该学生入学年之后

范式分析：

(Sno,Confno)为主码，关系模式Attend(Sno,Confno,Atime)中所有非主属性对码均完全依赖，同时也没有对码的传递函数依赖。另外，也没有主属性对码的部分和传递函数依赖。因此，该关系模式范式为BCNF。

3 后端设计

3.1 创建视图

首先我们可以创建关于系、专业、班级、学生实体的相对应视图，视图内属性同1.1需求语义。

1. 系视图

```

1 -- departmentView 系视图
2 CREATE VIEW departmentView(Dno,Dname,Dloc,Dnum) AS
3 SELECT Department.Dno,Department.Dname,Department.Dloc,Department.Dnum
4 FROM Department;

```

结果如下：

MySQL localhost:33060+ ssl stu dbms SQL > select * from departmentview;				
Dno	Dname	Dloc	Dnum	
1	通信工程系	主楼二区通院办公室	667	
2	电子工程系	办公楼二层西侧	682	
3	计算机系	主楼IV213-218	865	
4	机电系	主楼III区	514	
20	人工智能系	南校区	311	

图 8: 系视图结果

2. 班级视图

```

1 -- classView 班级视图
2 CREATE VIEW ClassView(Cno,Dname,Mname,Cyear,Cnum) AS
3 SELECT Class.Cno,Department.Dname,Major.Mname,Class.Cyear,Class.Cnum
4 FROM Department, Major, Class
5 WHERE Class.Mno = Major.Mno AND Major.Dno = Department.Dno;

```

结果如下：

MySQL localhost:33060+ ssl stu dbms SQL > select * from classview;				
Cno	Dname	Mname	Cyear	Cnum
1501012	通信工程系	信息工程	2015	75
1601011	通信工程系	通信工程	2016	80
1702015	电子工程系	电子工程	2017	150
1820011	人工智能系	人工智能	2018	25
1903013	计算机系	计算机科学与技术	2019	120
1904006	机电系	自动化	2019	40
1905045	计算机系	软件工程	2019	120

图 9: 班级视图结果

3. 学生视图

```

1 -- studentView 学生视图
2 CREATE VIEW studentView(Sno,Sname,Age,Dname,Cno,Dorm) AS
3 SELECT Student.Sno,Student.Sname,Student.Age,Department.Dname,Class.
   Cno,Department.Dorm
4 FROM Student, Department, Class, Major
5 WHERE Student.Cno = Class.Cno AND Class.Mno = Major.Mno
6   AND Major.Dno = Department.Dno;

```

结果如下：

MySQL localhost:33060+ ssl stu_dbms SQL > select * from studentview;					
Sno	Sname	Age	Dname	Cno	Dorm
16010001	李华	23	通信工程系	1601011	12
17020010	丽丽	22	电子工程系	1702015	12
18200145	黑夜男爵	21	人工智能系	1820011	13
19030001	赵宇盛	20	计算机系	1903013	11
19030002	小明	20	计算机系	1903013	11
19030003	Amy	20	计算机系	1903013	11
19050001	老王	20	计算机系	1905045	11

7 rows in set (0.0016 sec)

图 10: 学生视图结果

4. 创建学会视图

任务：创建视图显示学会名和学生数（实际不存在，也不能增加）

```

1 -- 学会视图
2 -- 创建视图显示学会号、学会名和学生数
3 CREATE VIEW conferenceView(Confno,Confname,Confnum) AS
4 SELECT Conference.Confno,Conference.Confname,COUNT(Attend.Sno)
5 FROM Conference INNER JOIN Attend
6 ON Conference.Confno = Attend.Confno
7 GROUP BY Conference.Confno
8 ORDER BY Confno;

```

如上SQL语句所示，我们通过“INNER JOIN”内部联结学会表和参会表；通过聚集函数COUNT来统计学生数，学生数实际不存在，也不能增加。该视图采用分组视图，由GROUP BY关键词创建；最后得到视图根据学会号高低排序。其结果如下：

```

MySQL localhost:33060+ ssl stu_dbms SQL > select * from conference;
+-----+-----+-----+-----+
| Confno | Confname      | Esyear | Confloc      |
+-----+-----+-----+-----+
| 1 | 数学学会      | 1990   | 数学中心      |
| 2 | 计算机学会      | 1985   | 计算机楼      |
| 3 | 校园动植物学会 | 2014   | 百草园        |
| 4 | 反内卷学会      | 2019   | 书院3楼      |
| 5 | 内卷卷内学会    | 2019   | 书院3楼对面  |
+-----+-----+-----+-----+
5 rows in set (0.0008 sec)

MySQL localhost:33060+ ssl stu_dbms SQL > select * from Attend;
+-----+-----+-----+
| Sno   | Confno | ATime   |
+-----+-----+-----+
| 16010001 | 2 | 2018-07-04 |
| 16010001 | 3 | 2019-11-04 |
| 19030001 | 1 | 2019-12-01 |
| 19030001 | 2 | 2019-11-01 |
| 19030001 | 5 | 2021-12-01 |
| 19030003 | 1 | 2020-05-08 |
| 19030003 | 2 | 2020-05-06 |
| 19030003 | 4 | 2020-05-10 |
+-----+-----+-----+
8 rows in set (0.0003 sec)

MySQL localhost:33060+ ssl stu_dbms SQL > select * from conferenceview;
+-----+-----+-----+
| Confno | Confname      | Confnum |
+-----+-----+-----+
| 1 | 数学学会      | 2 |
| 2 | 计算机学会      | 3 |
| 3 | 校园动植物学会 | 1 |
| 4 | 反内卷学会      | 1 |
| 5 | 内卷卷内学会    | 1 |
+-----+-----+-----+
5 rows in set (0.0006 sec)

MySQL localhost:33060+ ssl stu_dbms SQL >

```

图 11: 学会视图结果

3.2 创建触发器

任务：创建一个触发器，能根据每个班的学生变动情况自动增减班级表和系表的人数字段的值。

“学生的变动情况”大致有三种情况：学生增多、学生减少、学生更新。这对应着表的INSERT、DELETE以及UPDATE操作。我们以下设计对应三种活动各自相应是触发器。

1. INSERT触发器

SQL代码如下

```

1 CREATE TRIGGER stuInsert
2 AFTER INSERT ON Student
3 FOR EACH ROW
4 BEGIN
5     —— 更新班级表
6     UPDATE Class
7     SET Class.Cnum = Class.Cnum + 1
8     WHERE Class.Cno = NEW.Cno;
9
10    —— 更新系表
11    UPDATE Department

```

```

12   SET Department.Dnum = Department.Dnum + 1 WHERE Department.Dno = (
13     SELECT Major.Dno FROM Major WHERE Major.Mno = (
14       SELECT Class.Mno FROM Class WHERE Class.Cno = NEW.Cno));
15 END;

```

在mysql中触发器实现的较其他DBMS更为初级，mysql中的触发器无法在视图中操作，只能在表中使用，这使得上述sql语句较为冗长。除此以外，需要注意的是mysql中设置了NEW和OLD两个虚表来存储触发器响应前后INSERT和DELETE下相关行的副本。

2. DELETE触发器

```

1 -- DELETE Trigger
2 CREATE TRIGGER stuDelete
3 BEFORE DELETE ON Student
4 FOR EACH ROW
5 BEGIN
6   —— 更新班级表
7   UPDATE Class
8   SET Class.Cnum = Class.Cnum - 1
9   WHERE Class.Cno = OLD.Cno;
10
11  ——更新系表
12  UPDATE department
13  SET Department.Dnum = Department.Dnum - 1
14  WHERE Department.Dno = (
15    SELECT Major.Dno FROM Major WHERE Major.Mno = (
16      SELECT Class.Mno FROM Class WHERE Class.Cno = OLD.Cno));
17 END;

```

3. UPDATE触发器

```

1 -- UPDATE Trigger
2 CREATE TRIGGER stuUpdate
3 BEFORE UPDATE ON Student
4 FOR EACH ROW
5 BEGIN
6   —— 更新班级表
7   UPDATE Class SET Class.Cnum = Class.Cnum + 1
8   WHERE Class.Cno = NEW.Cno;
9   UPDATE Class SET Class.Cnum = Class.Cnum - 1
10  WHERE Class.Cno = OLD.Cno;
11
12  —— 更新系表

```

```

13   UPDATE Department
14   SET Department.Dnum = Department.Dnum + 1
15   WHERE Department.Dno = (
16       SELECT Major.Dno FROM Major WHERE Major.Mno = (
17           SELECT Class.Mno FROM Class WHERE Class.Cno = NEW.Cno));
18
19   UPDATE department
20   SET Department.Dnum = Department.Dnum - 1
21   WHERE Department.Dno = (
22       SELECT Major.Dno FROM Major WHERE Major.Mno = (
23           SELECT Class.Mno FROM Class WHERE Class.Cno = OLD.Cno));
24 END;

```

其中删除和插入触发器后端测试结果如下：

我往学生表插入一条人工智能系学生的信息，班级表和系表的人数自动发生了变化（加1）。

```

MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from class;
+-----+-----+-----+-----+
| Cno | Mno | Cnum | Cyear |
+-----+-----+-----+-----+
| 1501012 | 2 | 75 | 2015 |
| 1601011 | 1 | 80 | 2016 |
| 1702015 | 3 | 150 | 2017 |
| 1820011 | 9 | 26 | 2018 |
| 1903013 | 5 | 121 | 2019 |
| 1904006 | 8 | 39 | 2019 |
| 1905045 | 6 | 120 | 2019 |
+-----+-----+-----+-----+
7 rows in set (0.0006 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from department;
+-----+-----+-----+-----+
| Dno | Dname | Dnum | Dloc | Dorm |
+-----+-----+-----+-----+
| 1 | 通信工程系 | 667 | 主楼二区通院办公室 | 12 |
| 2 | 电子工程系 | 682 | 办公楼二层西侧 | 12 |
| 3 | 计算机系 | 241 | 主楼IV213-218 | 11 |
| 4 | 机电系 | 513 | 主楼III区 | 11 |
| 20 | 人工智能系 | 312 | 南校区 | 13 |
+-----+-----+-----+-----+
5 rows in set (0.0005 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > insert into Student
-> values (18200777,'Emily','女',21,1820011);
Query OK, 1 row affected (0.0187 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from class;
+-----+-----+-----+-----+
| Cno | Mno | Cnum | Cyear |
+-----+-----+-----+-----+
| 1501012 | 2 | 75 | 2015 |
| 1601011 | 1 | 80 | 2016 |
| 1702015 | 3 | 150 | 2017 |
| 1820011 | 9 | 27 | 2018 |
| 1903013 | 5 | 121 | 2019 |
| 1904006 | 8 | 39 | 2019 |
| 1905045 | 6 | 120 | 2019 |
+-----+-----+-----+-----+
7 rows in set (0.0082 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from department;
+-----+-----+-----+-----+
| Dno | Dname | Dnum | Dloc | Dorm |
+-----+-----+-----+-----+
| 1 | 通信工程系 | 667 | 主楼二区通院办公室 | 12 |
| 2 | 电子工程系 | 682 | 办公楼二层西侧 | 12 |
| 3 | 计算机系 | 241 | 主楼IV213-218 | 11 |
| 4 | 机电系 | 513 | 主楼III区 | 11 |
| 20 | 人工智能系 | 313 | 南校区 | 13 |
+-----+-----+-----+-----+
5 rows in set (0.0089 sec)

```

图 12: 插入触发器后端结果

班级人数从26变为27，系人数从312变为313。

我从学生表中删除一条计算机系学生的信息，班级表和系表的人数自动发生了变化（减1）。班级人数从121变为120，系人数从241变为240。

```
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from class where cno=1903013;
+-----+-----+-----+
| Cno | Mno | Cnum | Cyear |
+-----+-----+-----+
| 1903013 | 5 | 121 | 2019 |
+-----+-----+-----+
1 row in set (0.0004 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from department where dno=3;
+-----+-----+-----+
| Dno | Dname | Dnum | Dloc | Dorm |
+-----+-----+-----+
| 3 | 计算机系 | 241 | 主楼IV213-218 | 11 |
+-----+-----+-----+
1 row in set (0.0004 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > delete from student where sno=19030003;
Query OK, 1 row affected (0.0123 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from department where dno=3;
+-----+-----+-----+
| Dno | Dname | Dnum | Dloc | Dorm |
+-----+-----+-----+
| 3 | 计算机系 | 240 | 主楼IV213-218 | 11 |
+-----+-----+-----+
1 row in set (0.0009 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from class where cno=1903013;
+-----+-----+-----+
| Cno | Mno | Cnum | Cyear |
+-----+-----+-----+
| 1903013 | 5 | 120 | 2019 |
+-----+-----+-----+
1 row in set (0.0089 sec)
```

图 13: 删除触发器后端结果

功能完备，触发器设计任务完成。

3.3 创建存储过程完成班号替换

任务：创建一个函数（或存储过程），实现如下功能：给定一个班的旧班号和新班号，把所有相关表中此班的旧班号改为新班号，并返回此班的人数。

在实验中我采用了创建存储过程实现该功能。SQL代码如下：

```
1 DELIMITER //
2 DROP PROCEDURE IF EXISTS Alter_Cno;
3 CREATE PROCEDURE Alter_Cno(
4     IN New_Cno INT ,
5     IN Old_Cno INT ,
6     OUT num INT )
7 BEGIN
8     —— 声明变量
9     DECLARE Classnum INT DEFAULT 0;
10    —— 返回值赋值
11    SELECT Cnum FROM Class WHERE Class.Cno = Old_Cno INTO Classnum;
12    —— 外码参照失效
13    SET FOREIGN_KEY_CHECKS = 0;
14    —— 新班号UPDATE
```

```
15 UPDATE Student SET Student.Cno = New_Cno WHERE Student.Cno = Old_Cno ;
16 UPDATE Class SET Class.Cno = New_Cno WHERE Class.Cno = Old_Cno ;
17 — 外码参照恢复
18 SET FOREIGN_KEY_CHECKS = 1;
19 — 返回值
20 SELECT Classnum INTO num;
21 END //
22 DELIMITER ;
```

首先我们了解到含有“班号”属性(Cno)的表只有班级表和学生表，因此我们创建的函数主要就是对这两个表进行更新。由任务可知，该函数需要传入两个参数：旧班号和新班号，这里命名为Old_Cno和New_Cno，函数在班级表中查询到该班号对应旧班号并改为新班号，同时需要返回该行Cnum(即班级人数)的值。

这里面包含两个问题：

1. 学生表的班号属性使作为外码的，相对于班级表的主码完成其参照完整性。因此在UPDATE前要使相关表的外码参照失效，UPDATE完成后恢复回来。

```
1 SET FOREIGN_KEY_CHECKS = 0;
2 .....
3 SET FOREIGN_KEY_CHECKS = 1;
```

2. 在UPDATE学生表的同时会“误触”上一个任务所写的触发器功能。遗憾的是mysql无法使用“Alter table ... disable ...”的语句来临时禁用触发器。由于UPDATE在增减学生数量没有起到作用，故此处直接删除掉UPDATE触发器。(更委婉的做法：设置一个全局变量，通过该变量和条件判断语句来保证存储过程运行时，触发器不被触发。)

通过CALL语句来调用存储过程以验证功能是否完成。在测试中，我们把原班号1904006的班级换上新班号1904031，同时返回该班人数，为39。结果如下：

```

MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from class;
+-----+-----+-----+-----+
| Cno | Mno | Cnum | Cyear |
+-----+-----+-----+-----+
| 1501012 | 2 | 75 | 2015 |
| 1601011 | 1 | 80 | 2016 |
| 1702015 | 3 | 150 | 2017 |
| 1820011 | 9 | 27 | 2018 |
| 1903013 | 5 | 120 | 2019 |
| 1904006 | 8 | 39 | 2019 |
| 1905045 | 6 | 120 | 2019 |
+-----+-----+-----+-----+
7 rows in set (0.0004 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > CALL Alter_Cno(1904031,1904006,@retNum );
Query OK, 1 row affected (0.0078 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from class;
+-----+-----+-----+-----+
| Cno | Mno | Cnum | Cyear |
+-----+-----+-----+-----+
| 1501012 | 2 | 75 | 2015 |
| 1601011 | 1 | 80 | 2016 |
| 1702015 | 3 | 150 | 2017 |
| 1820011 | 9 | 27 | 2018 |
| 1903013 | 5 | 120 | 2019 |
| 1904031 | 8 | 39 | 2019 |
| 1905045 | 6 | 120 | 2019 |
+-----+-----+-----+-----+
7 rows in set (0.0010 sec)
MySQL [localhost:33060+ ssl stu_dbms SQL] > select @retNum;
+-----+
| @retNum |
+-----+
| 39 |
+-----+
1 row in set (0.0003 sec)

```

图 14: 替换班号存储过程后端结果

3.4 使用游标完成存储过程

任务：创建一个存储过程，使用游标完成如下功能：确定系表中人数组字段的值与实际学生数是否相符。如果不相符，把人数组字段的值改为实际数，并返回此系的系号、系名、原人数、实际人数。

首先，我们要创建一张（系号，系名，系真实人数）的视图，其中由于系和专业是一对多的关系，而专业和班级是一对一关系，所以系对班级也是一对多关系，所以 系真实人数是系内各专业人数之和，换言之，是系内各班人数之和。这里我们用到了聚集函数SUM()以及内部联结（INNER JOIN）。

```

1 CREATE VIEW Drealnum(Dno,Dname,Drnum) AS
2 SELECT Department.Dno,Department.Dname,SUM(classView.Cnum)
3 FROM Department INNER JOIN classView
4 ON Department.Dname = classView.Dname
5 GROUP BY Department.Dno
6 ORDER BY Dno;

```

结果如下：

```

MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from Drealnum;
+-----+-----+-----+
| Dno | Dname | Drnum |
+-----+-----+-----+
| 1 | 通信工程系 | 155 |
| 2 | 电子工程系 | 150 |
| 3 | 计算机系 | 242 |
| 4 | 机电系 | 39 |
| 20 | 人工智能系 | 26 |
+-----+-----+-----+
5 rows in set (0.3665 sec)

```

图 15: 真实系人数视图结果

至此，该任务创建的存储过程主干部分可以这样描述：定义一个游标，不断滚动此游标，查找系表中对应系号的系人数是否和上面视图中系真实人数是否相等，若不等，则修改系表人数，同时返回（系号，系名，系原人数，系真实人数）。

```
1 DROP PROCEDURE IF EXISTS CheckDnum;
2 CREATE PROCEDURE CheckDnum(
3     IN aDno INT,
4     OUT oDno INT,
5     OUT oDname CHAR(20),
6     OUT oDoldnum INT,      — 原人数
7     OUT oDnewnum INT      — 实际人数
8 )
9 BEGIN
10    DECLARE done BOOLEAN DEFAULT 0; 定义结束标识      #
11    DECLARE tDno INT DEFAULT 0;
12    DECLARE tDname CHAR(20) DEFAULT "";
13    DECLARE tDnum INT DEFAULT 0;
14    DECLARE tDrnum INT DEFAULT 0;
15    — 定义游标
16    DECLARE Checknum CURSOR
17        FOR SELECT Department .Dno ,Department .Dname ,Department .Dnum
18        FROM Department ;
19
20    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
21    — 打开游标
22    OPEN Checknum;
23    — 循环遍历，从第一行到最后一行
24    REPEAT
25        FETCH Checknum INTO tDno ,tDname ,tDnum;
26        — 循环结束
27        UNTIL ((done = 1) OR (tDno = aDno))
28        END REPEAT;
29        — 关闭游标
30    CLOSE Checknum;
31    SELECT Drnum
32        FROM Drealnum
33        WHERE Drealnum .Dno = aDno
34        INTO tDrnum;
35    UPDATE Department SET Dnum = tDrnum WHERE Dno = tDno;
36    SELECT tDno ,tDname ,tDnum ,tDrnum INTO oDno ,oDname ,oDoldnum ,oDnewnum;
37 END;
38
```

```

39 | — 调用该存储过程
40 | call checkdnum(3 ,@no ,@name ,@old ,@new ) ;  ##Dno=3 计算机系
41 |
42 | — 打印返回数据
43 | SELECT  @no ,@name ,@old ,@new ;

```

该存储过程后端测试结果如下：

The screenshot shows the MySQL Shell interface. It starts with a query to select all data from the 'department' table. Then, it executes a stored procedure 'call CheckDnum(3, @dno, @dname, @oldnum, @newnum)'. After the procedure is run, it performs another select query to retrieve the updated values for the row where Dno is 3. Finally, it runs a third select query to show the current state of the 'department' table.

```

MySQL Shell
MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from department;
+---+-----+-----+-----+
| Dno | Dname | Dnum | Dloc      | Dorm |
+---+-----+-----+-----+
| 1  | 通信工程系 | 667 | 主楼二区通院办公室 | 12 |
| 2  | 电子工程系 | 682 | 办公楼二层西侧 | 12 |
| 3  | 计算机系   | 864 | 主楼IV213-218    | 11 |
| 4  | 机电系     | 513 | 主楼III区        | 11 |
| 20 | 人工智能系 | 312 | 南校区          | 13 |
+---+-----+-----+-----+
5 rows in set (0.0004 sec)

MySQL [localhost:33060+ ssl stu_dbms SQL] > call CheckDnum(3, @dno, @dname, @oldnum, @newnum);
Query OK, 1 row affected (0.0177 sec)

MySQL [localhost:33060+ ssl stu_dbms SQL] > select @dno, @dname, @oldnum, @newnum ;
+-----+-----+-----+-----+
| @dno | @dname | @oldnum | @newnum |
+-----+-----+-----+-----+
| 3    | 计算机系 | 864    | 241    |
+-----+-----+-----+-----+
1 row in set (0.0003 sec)

MySQL [localhost:33060+ ssl stu_dbms SQL] > select * from department;
+---+-----+-----+-----+
| Dno | Dname | Dnum | Dloc      | Dorm |
+---+-----+-----+-----+
| 1  | 通信工程系 | 667 | 主楼二区通院办公室 | 12 |
| 2  | 电子工程系 | 682 | 办公楼二层西侧 | 12 |
| 3  | 计算机系   | 241 | 主楼IV213-218    | 11 |
| 4  | 机电系     | 513 | 主楼III区        | 11 |
| 20 | 人工智能系 | 312 | 南校区          | 13 |
+---+-----+-----+-----+
5 rows in set (0.0004 sec)

MySQL [localhost:33060+ ssl stu_dbms SQL] >

```

图 16: 后端使用游标的存储过程结果

如图所示，计算机系原人数为864，调用该存储过程后，系人数修正为241，同时也测试了该存储过程的返回值，表现正常。

4 前端设计

前端设计采用了python的tkinter库，同时使用了pymysql库来连接mysql数据库。我们需要：

```

1 import tkinter as tk
2 from tkinter.messagebox import *
3 import pickle
4 import pymysql

```

以下前端实现的功能都封装成函数进行调用。事实上由于python有“嵌套函数”的特性，因此我们直接将一张前端页面作为一个自定义函数，该页面上的功能为改页面下的子函数。

在实现增删改查的功能中，注意到前后端交互需要哪些数据：

```
1 ## 前后端交互的数据
2 deleteNo = tk.IntVar()
3 selectNo = tk.IntVar()
4 insertNo = tk.IntVar()
5 insertName = tk.StringVar()
6 insertSex = tk.StringVar()
7 insertAge = tk.IntVar()
8 insertCno = tk.IntVar()
9 updateNo = tk.IntVar()
10 updateName = tk.StringVar()
11 updateSex = tk.StringVar()
12 updateAge = tk.IntVar()
```

4.1 基本表的增删改查功能

1. 插入学生信息

```
1 def insertPage():
2     window_insert = tk.Toplevel(window_main_page)
3     window_insert.geometry('400x500')
4     window_insert.title('插入学生信息')
5
6     def insertData():
7         iSno = insertNo.get()
8         iSname = insertName.get()
9         iSex = insertSex.get()
10        iAge = insertAge.get()
11        iCno = insertCno.get()
12
13     try:
14         db = pymysql.connect(host='localhost', user='root',
15                               password='zys@010717', database="stu_dbms")
16         cur = db.cursor()
17         sql = """
18             INSERT INTO Student VALUES ("{iSno}","{iSname}","{iSex}","{iAge}",
19                                         "{iCno}");
20         """.format(iSno=iSno, iSname=iSname, iSex=iSex, iAge=iAge,
21                    iCno=iCno)
22         cur.execute(sql)
23         db.commit()
24     except Exception as e:
```

```
22         showerror(message="操作非法，插入失败！")
23         print(e)
24         db.rollback()
25         return
26     finally:
27         cur.close()
28         db.close()
29     tk.Label(window_insert, text="插入后学生表如下").pack()
30     showStudent(window_insert, 10)
31     showinfo(message='插入成功！')
32
33     tk.Label(window_insert, text="插入信息填写如下: ").pack(side='top')
34     tk.Label(window_insert, text="学号: ").pack()
35     tk.Entry(window_insert, textvariable=insertNo).pack()
36     tk.Label(window_insert, text="姓名: ").pack()
37     tk.Entry(window_insert, textvariable=insertName).pack()
38     tk.Label(window_insert, text="性别: ").pack()
39     tk.Entry(window_insert, textvariable=insertSex).pack()
40     tk.Label(window_insert, text="年龄: ").pack()
41     tk.Entry(window_insert, textvariable=insertAge).pack()
42     tk.Label(window_insert, text="班号: ").pack()
43     tk.Entry(window_insert, textvariable=insertCno).pack()
44     tk.Button(window_insert, text="确认插入", command=insertData).pack()
```

2. 删除学生信息

```
1 def deletePage():
2     window_delete = tk.Toplevel(window_main_page)
3     window_delete.geometry('400x400')
4     window_delete.title('删除学生信息')
5
6     def deleteData():
7         dSno = deleteNo.get()
8         try:
9             db = pymysql.connect(host='localhost', user='root',
10                     password='zys@010717', database="stu_dbms")
11             cur = db.cursor()
12             sql = """DELETE FROM Student WHERE Sno = %d"""
13             format(dSno=dSno)
14             cur.execute(sql)
15             db.commit()
16         except Exception as e:
```

```

15         showerror(message="操作非法，删除失败！")
16         db.rollback()
17     return
18 finally:
19     cur.close()
20     db.close()
21 tk.Label(window_delete, text="删除后学生表如下").pack()
22 showStudent(window_delete, 10)
23 showinfo(message='删除成功！')
24
25 tk.Label(window_delete, text="需要删除的学生学号为: ").pack()
26 tk.Entry(window_delete, textvariable=deleteNo).pack()
27 tk.Button(window_delete, text="确认删除", command=deleteData).pack()

```

3. 查询学生信息

```

1 def selectData():
2     selNo = selectNo.get()
3     sql = """SELECT * FROM Student WHERE Sno = '{selNo}' """.format(
4         selNo=selNo);
5     res = sql_information(sql)
6     window_select = tk.Toplevel(window_main_page)
7     window_select.geometry('400x80')
8     window_select.resizable(0, 0)
9     window_select.title('查询结果')
10    head_string = ('学号', '姓名', '性别', '年龄', '班号')
11    for i in range(len(res[0])):
12        studentlist = tk.Listbox(window_select, width=10, height=3, bd
13            =4, relief='flat')
14        studentlist.pack(side='left', fill='both')
15        studentlist.insert('end', head_string[i])
16        for each in res:
17            studentlist.insert('end', each[i])

```

4. 更改学生信息

由于学号是学生表主，并且班号是外码，在后端设计中有班号替换的存储过程的设计，所以这部分只更改学生信息中的姓名、性别、年龄部分。

```

1 def updatePage():
2     window_update = tk.Toplevel(window_main_page)
3     window_update.geometry('400x300')
4     window_update.title('更新学生信息')

```

```
5
6     def updateData():
7         uno = updateNo.get()
8         uname = updateName.get()
9         usex = updateSex.get()
10        uage = updateAge.get()
11        try:
12            db = pymysql.connect(host='localhost', user='root',
13                                  password='zys@010717', database="stu_dbms")
14            cur = db.cursor()
15            sql = """
16                UPDATE Student SET Sname = "{uname}" , Sex = "{usex}" , Age =
17                    "{uage}"
18                WHERE Sno = "{uno}" """.format(uno=uno, uname=uname, usex=
19                    usex, uage=uage)
20            cur.execute(sql)
21            db.commit()
22        except Exception as e:
23            showerror(message="操作非法，更新失败！")
24            db.rollback()
25            return
26        finally:
27            cur.close()
28            db.close()
29            tk.Label(window_update, text="更新后学生表如下").pack()
30            showStudent(window_update, 10)
31            showinfo(message='更新成功！')
32
33            tk.Label(window_update, text="需要更新的学生学号为: ").pack(side='top')
34            tk.Entry(window_update, textvariable=updateNo).pack()
35            tk.Label(window_update, text="姓名: ").pack()
36            tk.Entry(window_update, textvariable=updateName).pack()
37            tk.Label(window_update, text="性别: ").pack()
38            tk.Entry(window_update, textvariable=updateSex).pack()
39            tk.Label(window_update, text="年龄: ").pack()
40            tk.Entry(window_update, textvariable=updateAge).pack()
41            tk.Button(window_update, text="确认更新", command=updateData).pack()
```



图 17: 对学生表做增删改查的功能界面

4.2 对后端设计功能的使用

1. 学会视图由于后端sql已存在学会视图的虚表，故在前端只需要读出此表。

```
1 ## 学会视图
2 sql = """SELECT * FROM ConferenceView"""
3 res = sql_information(sql)
4 head_string = ('学会号', '学会名', '学会人数')
5 for i in range(len(res[0])):
6     conflist = tk.Listbox(grid1, height=5, bd=4, relief='flat')
7     conflist.pack(side='left', fill='both')
8     conflist.insert('end', head_string[i])
9     for each in res:
10         conflist.insert('end', each[i])
```

前端展示结果如下：



图 18: 前端学会视图

- ## 2. 人数更新（触发器）

由于人数自动更新，在学生表发生插入或删除操作时，班级表和系表就反映出来了，所以前端观察班级表和系表的变化即可。

3. 班号替换

```

1 oldcno = tk.IntVar()
2 newcno = tk.IntVar()
3 def exchangeClassno():
4     ono = oldcno.get()
5     nno = newcno.get()
6     returnum = 0
7     try:
8         db = pymysql.connect(host='localhost', user='root', password='
9             zys@010717', database="stu_dbms")
10        cur = db.cursor()
11        cur.callproc('Alter_Cno', args=(nno, ono, returnum))
12        db.commit()
13        res1 = cur.fetchall()
14
15    except Exception as e:
16        print(e)
17        showerror(message="操作非法，替换失败！")
18        db.rollback()
19        return
20
21    finally:
22        cur.close()
23        db.close()
24        showinfo(message="替换成功")

```

在python中调用mysql的存储过程：创建游标cursor，调用callproc方法，该方法包含两参数（存储过程名、需要传入的形参）

前端展示结果如下：



图 19: 登录界面

4. 系人数矫正

上文中后端展现出该存储过程的最终效果，前端部分只需要可视化其班级表和系表即可。

调用后端写好的存储过程“CheckDnum”，用该存储过程完成系号修正。

```
1 def exchangeDno():
2     dno = execdno.get()
3     rdno = 0
4     rdname = ""
5     roldnum = 0
6     rnewnum = 0
7     try:
8         db = pymysql.connect(host='localhost', user='root', password='',
9                               zys@010717', database="stu_dbms")
10        cur = db.cursor()
11        resarg = cur.callproc('CheckDnum', args=(dno, rdno, rdname,
12                                              roldnum, rnewnum))
13        db.commit()
14        res1 = cur.fetchall()
15        print(res1)
16        print(resarg)
17
18        pvalue = cur.fetchall()
19    except Exception as e:
20        print(e)
21        showerror(message="操作非法，矫正失败！")
22        db.rollback()
23        return
24    finally:
25        cur.close()
26        db.close()
27        showinfo(message="矫正成功")
28
29 tk.Label(grid4, text="系号为").pack()
30 tk.Entry(grid4, textvariable=execdno).pack()
31 tk.Button(grid4, text='系号修正', font=(‘华文黑
32           体’, 10, "bold"), height=2, width=7, command=exchangeDno).pack()
```

前端界面如下：



图 20: 系号修正功能

5 效果实现

1. 前端登录界面



图 21: 登录界面

2. 登陆后，进入管理员界面

由于数据库内表的增删改查操作非常类似，以下前端以学生表为例。



图 22: 管理员操作界面

由前端操作界面，我们可以对学生表做增删改查操作。

3. 增加学生信息



图 23: 成功插入学生信息

4. 更新学生信息

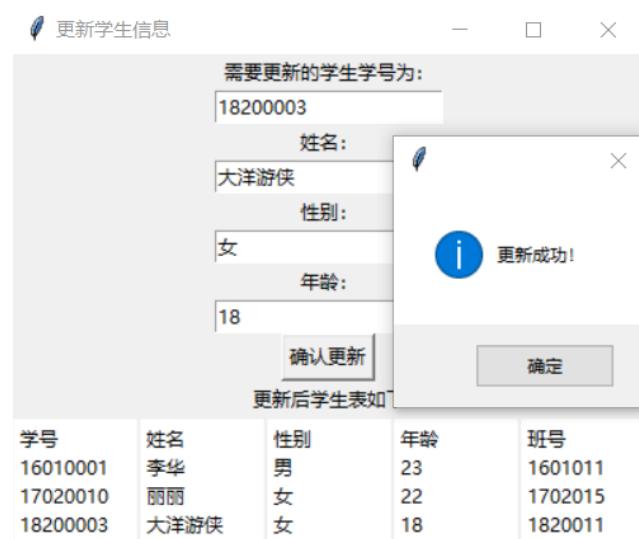


图 24: 成功更新学生信息

5. 删除学生信息



图 25: 成功删除学生信息

6. 查询学生信息

在插入学生信息后，查询一下



图 26: 成功查询新增学生信息

在更新该学生信息后，查询到更新后的信息



图 27: 成功查询学生更新信息

在删除该学生信息后，查询无果。



图 28: 删除学生信息后，查询无果

在操作页面按“其他功能按钮”，进入后端自主设计功能验证页面。



图 29: 其他功能页面

如页面所示，前端设计了可以自主查看现有视图和表。

1. 前端验证触发器

我插入一条班级为1601011的通信工程系的学生信息，班表和系表如下图所示



图 30: 验证触发器

见划红线部分，人数自动增加了。

Student Record Search Result				
学号	姓名	性别	年龄	班号
16010001	李华	男	23	1601011
16010050	阿旺	男	25	1601011
17020010	丽丽	女	22	1702015
18200145	黑夜男爵	男	21	1820011
18200255	明日香	女	21	1820011
18200777	Emily	女	21	1820011
19030001	赵宇盛	男	20	1903013
19030002	小明	男	20	1903013
19030006	马走日	男	20	1903013
19030500	大壮	男	20	1903013
19040001	凌波丽	女	20	1904006
19050001	老王	男	20	1905045

图 31: 学生表增加了该名同学信息

删除触发器也类似，此处不再展示。

2. 前端验证班号替换调用该功能前后查看班级表，见划红线部分，班号发生改变，如下图所示

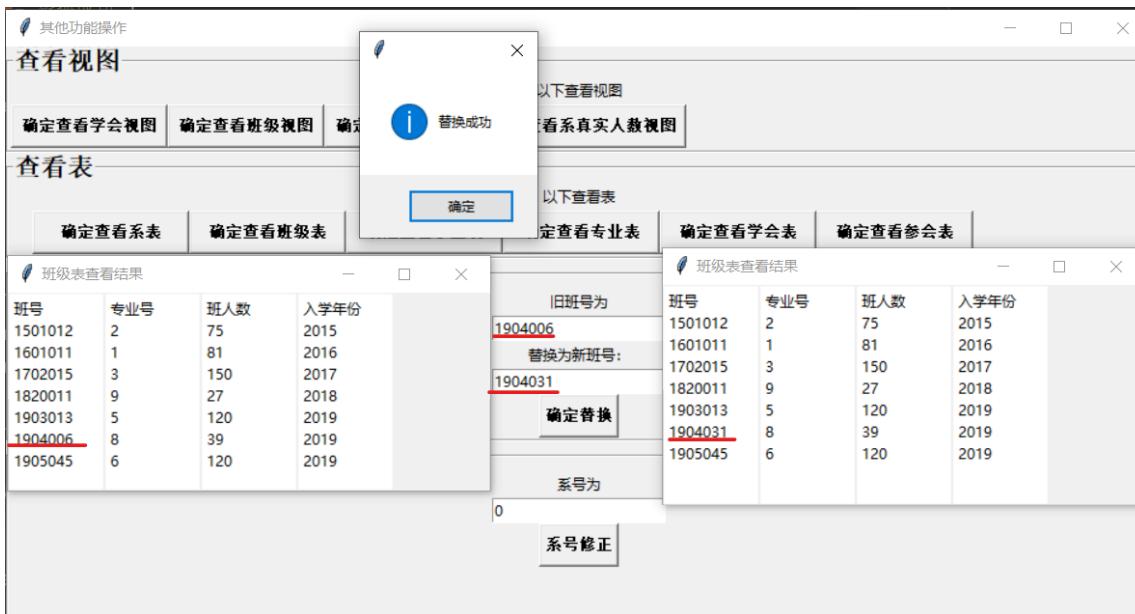


图 32: 验证班号替换

3. 前端验证系人数矫正测试矫正机电系系人数。原人数513，真实人数为39，矫正为39。如下图所示

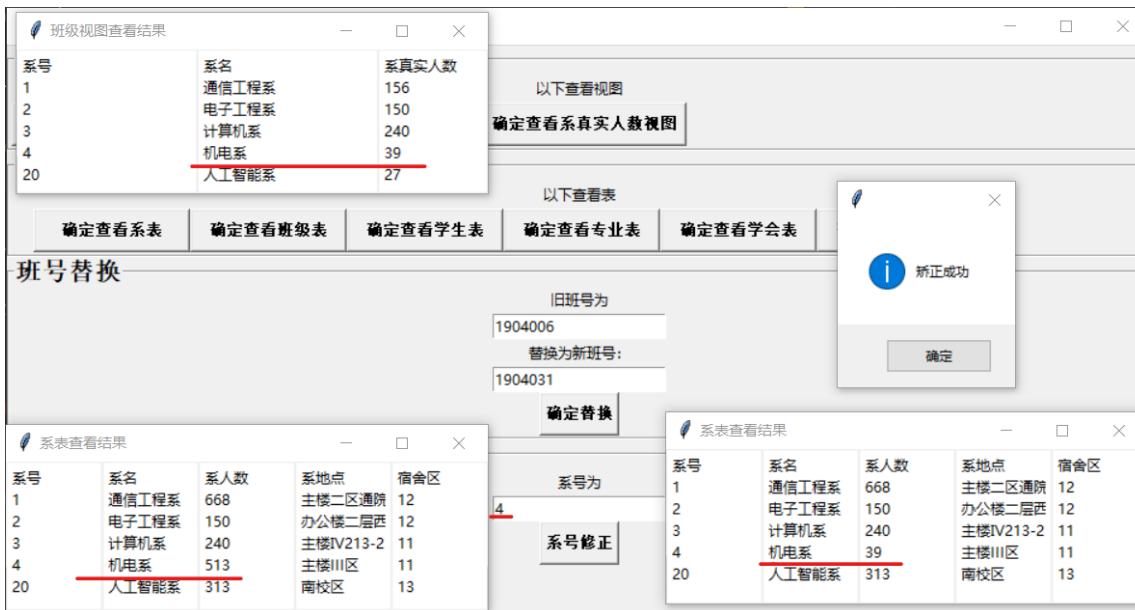


图 33: 验证系人数矫正

6 心得体会

- 在本课设中后端采用的mysql，很遗憾的是无法使用sql脚本，所以作业初始时后端一直采用命令行书写sql语句；后期使用Vscode配置MySQL相关插件，完成后端代码书写。
- 前端部分采用了python自带的GUI库tinker，该库比较老旧，也由于笔者本人水平有限，时间紧急，所以前端界面比较简单。由于表的增删改查操作是类似的，为了减少工作量，在前端部分只提

供了学生表的增删改查功能，除此之外前端界面还可以主动查看表和视图，完成后端触发器、存储过程等功能的验证。

3. 前端代码最初书写时，没有抽象一定的设计模式，没有采用面向对象的设计方式，因此在实现前端时很深刻的感觉到一些代码在频繁的被重复书写，所以呈现的源码比较冗长，也会出现一些小bug，总体功能还是能用的。
4. 本课设有学生（笔者）独自完成，在本课设中一定也存在许多不足和缺陷；虽然工作量很大，任务繁重，但是能够完成这样一个简单的数据库管理系统，也让我收获颇丰。