数据库设计报告——学生管理系统

赵龙飞-18030100257

目录

1 团队成员

人数：	1
组长：	赵龙飞-18030100257

2 实验环境

后端	MicroSoft SQL Server 2019
前端	Python 3.8

3 需求分析

3.1 需求语义

今要建立关于系、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。

3.2 要完成的任务

3.2.1 模式设计

1. 画出 E-R 图。
2. 把 E-R 图转为关系模式。
3. 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定义自己定义。

3.2.2 后端设计

4. 创建一个视图，能显示每个学会的学会名，学生数 (实际不存在，也不能增加)。
5. 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。
6. 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。
7. 创建一个存储过程，使用游标完成如下功能：确定系表中人数字段的值与实际学生数是否相符。如果不相符，把人数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。

3.2.3 前端设计

8. 实现对基本表的增删改查操作，实现对 4-7 所创建对象的使用。

3.3 注意事项

- 在数据库的设计过程中注意运用规范化理论，可能需要分解或合并关系模式。
- 要求设定关系的完整性规则，如实体完整性（例如设置主码），参照完整性（例如设置外码和对应的主码），用户自定义完整性（例如性别只能为“男”或“女”）。

4 概念结构设计

4.1 实体和属性的划分

1. 根据实体和属性的两条准则和需求，系、专业、班级、学生、学会等都有与之对应的描述，故应该将他们作为实体。
2. 根据实体与属性的划分的基本原则：为了简化 E-R 图的处置，现实世界的事物能作为属性对待的，尽量作为属性对待。一个系的学生住在同一宿舍区，所以可以将宿舍区作为系的一个属性。

4.2 实体属性图和实体联系图

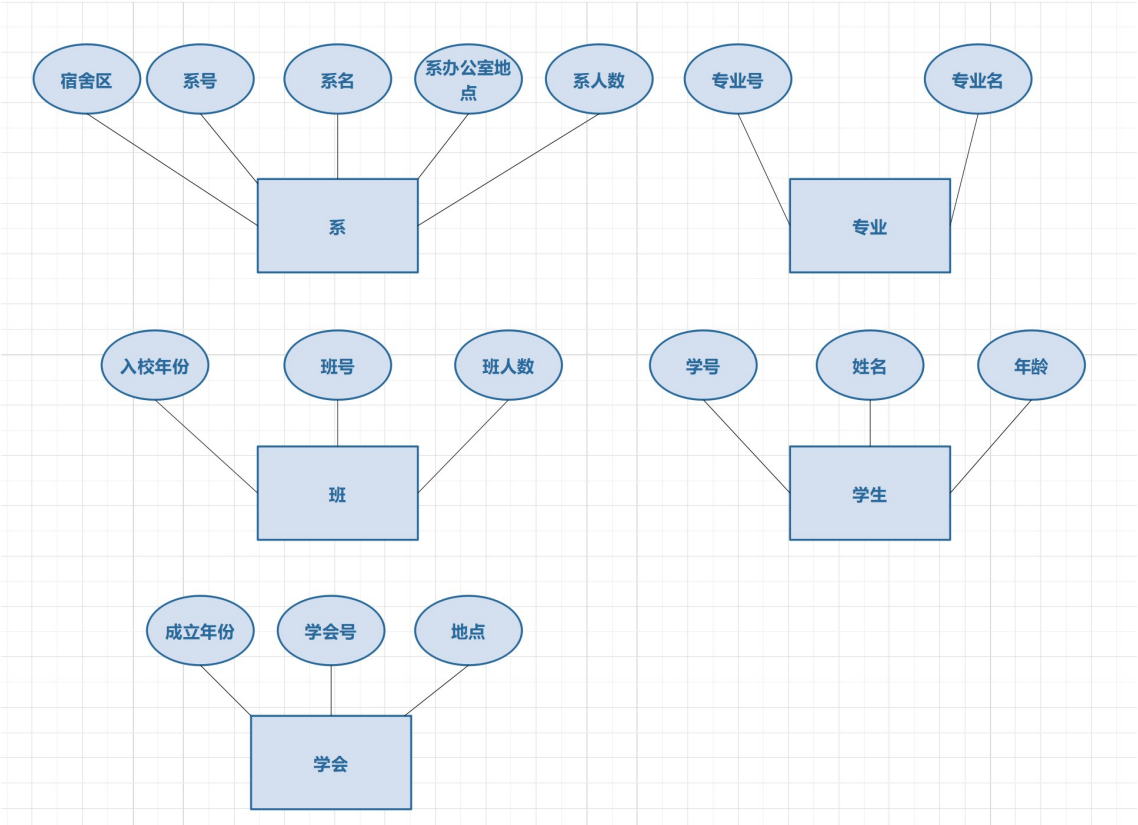

图 1: 实体属性图

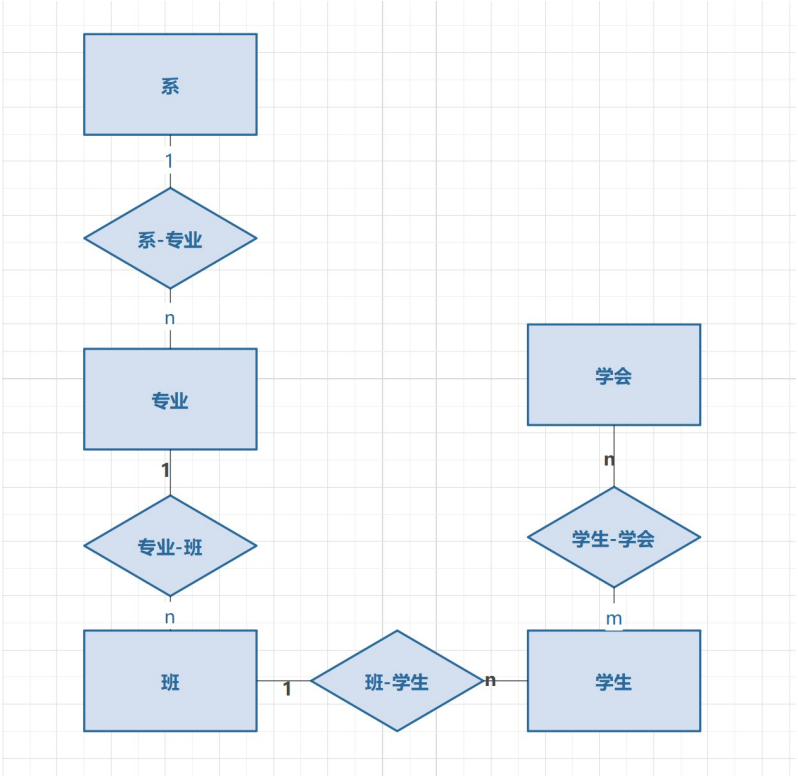

图 2: 实体联系图

4.3 合并后的 E-R 图

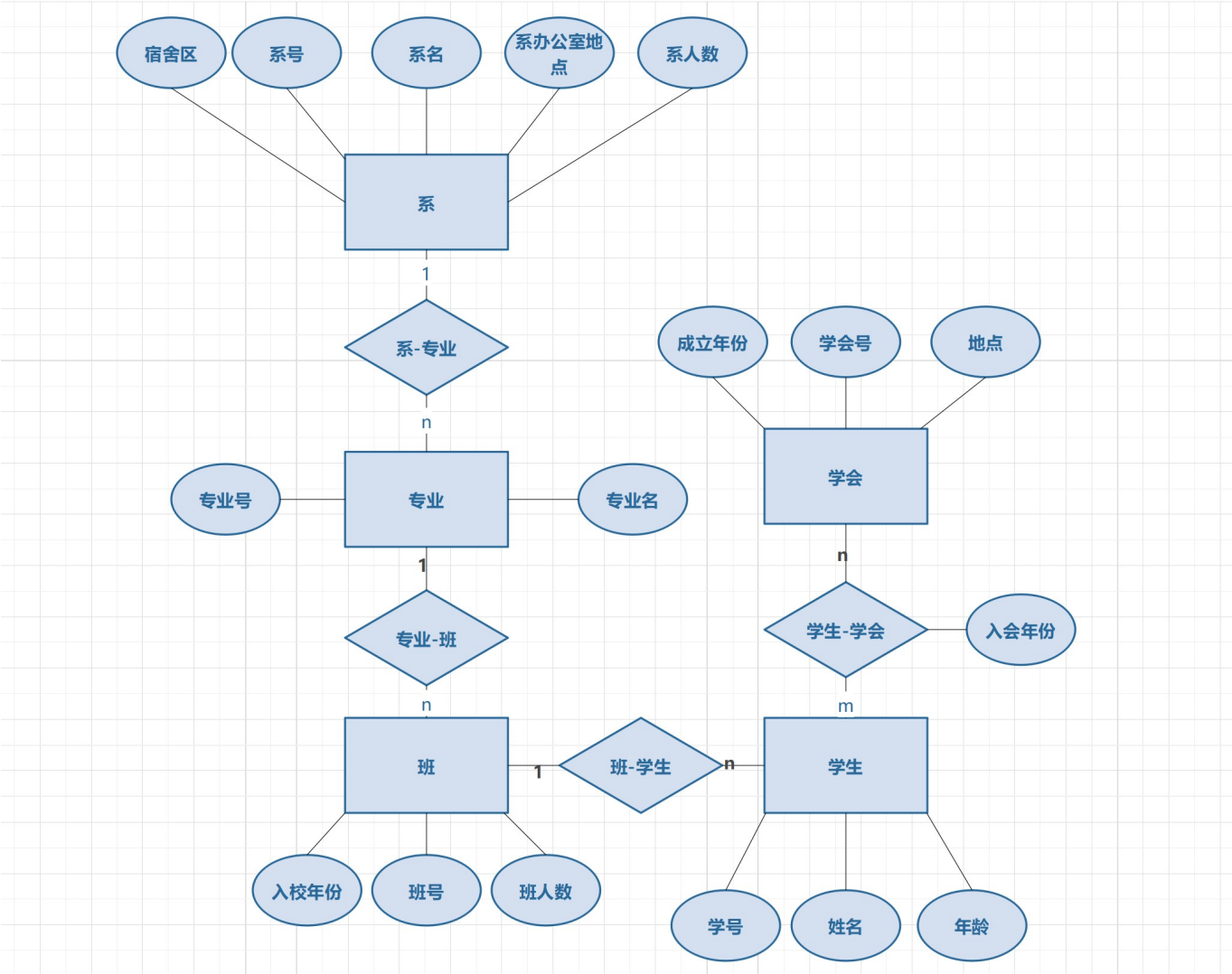

图 3: 合并后的 E-R 图

5 逻辑结构设计

将 E-R 转换为关系模式如下：

系：系号、系名、系办公室地点、人数。

专业：专业号、专业名、系号。

班级：班级号、专业号、入校年份、系号、人数。

学会：学会号、学会名、成立年份、地点。

学生：学号、姓名、年龄、系号、班号。

学生-学会：学号、学会号、入会年份。

- 下划线表示主码。
- 班级和学生中的系号字段是为了使得查找效率更高（折中）。

6 详细实现

6.1 建表

考虑了实体完整性、参照完整性和用户自定义完整性 (CHECK)。

```
CREATE TABLE D
(
    Dno VARCHAR(3) PRIMARY KEY,
    Dname VARCHAR(30) NOT NULL UNIQUE,
    Office VARCHAR(30),
    Dorm VARCHAR(30),
    Dsum int
)

CREATE TABLE M
(
    Mno VARCHAR(3) PRIMARY KEY,
    Mname VARCHAR(30) UNIQUE NOT NULL,
    Dno VARCHAR(3) FOREIGN KEY REFERENCES D(Dno) ON UPDATE CASCADE ON DELETE
        CASCADE
)

CREATE TABLE C
(
    Cno VARCHAR(7) PRIMARY KEY,
    Mno VARCHAR(3) FOREIGN KEY REFERENCES M(Mno) ON UPDATE CASCADE ON DELETE
        CASCADE,
    Cyear INT CHECK(Cyear > 2000),
    Dno VARCHAR(3) FOREIGN KEY REFERENCES D(Dno),
    Csum int
)

CREATE TABLE I
(
    Ino VARCHAR(7) PRIMARY KEY,
    Iname VARCHAR(30) NOT NULL UNIQUE,
    Iyear INT CHECK(Iyear > 2000),
    Iaddr VARCHAR(30)
)

CREATE TABLE S
(
    Sno VARCHAR(11) PRIMARY KEY,
    Sname VARCHAR(10) NOT NULL,
    Sage INT CHECK(Sage > 0),
    Dno VARCHAR(3) FOREIGN KEY REFERENCES D(Dno),
    Cno VARCHAR(7),
```

```
    CONSTRAINT SC_REF FOREIGN KEY(Cno) REFERENCES C(Cno) ON UPDATE CASCADE ON
        DELETE CASCADE
)

CREATE TABLE SI
(
    Sno VARCHAR(11) FOREIGN KEY REFERENCES S(Sno),
    Ino VARCHAR(7) FOREIGN KEY REFERENCES I(Ino),
    SIyear int CHECK(SIyear > 2000),
    PRIMARY KEY(Sno, Ino)
)
```

6.2 后端设计

6.2.1 学会-人数视图

分组小计来统计各个学会的人数，并使用左外连接避免了当某学会人数为 0 时，选不到此学会的情况。

```
CREATE VIEW I_VIEW
AS
SELECT A.Iname AS '学会名', CNT AS '人数'
FROM I AS A LEFT OUTER JOIN (
    SELECT Ino, COUNT(Sno) AS CNT FROM SI GROUP BY Ino
) AS B
ON A.Ino = B.Ino;
```

6.2.2 自动增减班级表和系表的人数字段触发器

创建了三个触发器，分别实现了对增、删、改学生表时候系表和班级表人数字段的自动调整。

```
CREATE TRIGGER AUTO_INC ON S
AFTER INSERT
AS
    BEGIN
    UPDATE C SET Csum = Csum + (SELECT COUNT(Sno)
    FROM inserted
    WHERE inserted.Cno = C.Cno) WHERE C.Cno IN (SELECT Cno
    FROM inserted)
    UPDATE D SET Dsum = Dsum + (SELECT COUNT(Sno)
    FROM inserted
    WHERE inserted.Dno = D.Dno) WHERE D.Dno IN (SELECT Dno
    FROM inserted)
END
go
CREATE TRIGGER AUTO_DEC ON S
AFTER DELETE
AS
    BEGIN
    UPDATE C SET Csum = Csum - (SELECT COUNT(Sno)
```

```
    FROM deleted
    WHERE deleted.Cno = C.Cno) WHERE C.Cno IN (SELECT Cno
    FROM deleted)
    UPDATE D SET Dsum = Dsum - (SELECT COUNT(Sno)
    FROM deleted
    WHERE deleted.Dno = D.Dno) WHERE D.Dno IN (SELECT Dno
    FROM deleted)
END
go

CREATE TRIGGER AUTO_UPDATE ON S
AFTER UPDATE
AS
    BEGIN
    UPDATE C SET Csum = (SELECT COUNT(Sno) SUM FROM S WHERE Cno = C.Cno)
    UPDATE D SET Dsum = (SELECT COUNT(Sno) SUM FROM S WHERE Dno = D.Dno)
END
```

6.2.3 修改班级号存储过程

1. 该存储过程有两个参数：旧班号和新班号，并且返回该班级的人数。
2. 因为在创建学生表的时候班级号外码使用了级联更新，所以存储过程中更新班级号之后，学生表中对应的元组的班级号也会随之修改。

```
CREATE PROC CHANGE_CNO(@old char(7),
@new char(7))
AS
SELECT Csum
FROM C
WHERE Cno = @old
UPDATE C SET Cno = @new WHERE Cno = @old
```

6.2.4 确定系表人数字段是否符合实际人数存储过程

1. 在该存储过程中，首先定义了一个游标用来遍历系表。
2. 并在此存储过程中创建了一个临时表（在存储过程最后删去），用来存储不相符的系号、系名、原人数和实际人数。
3. 定义四个局部变量来配合游标完成对系表的遍历。

```
CREATE PROC CHECK_Dsum
AS
DECLARE Dcur CURSOR SCROLL FOR
    SELECT Dno, Dsum
FROM D

OPEN Dcur
CREATE TABLE D_TMP
```

```
(
    Dno VARCHAR(3),
    Dname VARCHAR(30),
    Psum INT,
    Dsum INT
)

DECLARE @Dno CHAR(3)
DECLARE @Dsum INT
DECLARE @TMP INT
DECLARE @Dname VARCHAR(30)
FETCH NEXT FROM Dcur INTO @Dno, @Dsum
WHILE @@FETCH_STATUS = 0
    BEGIN
    (SELECT @TMP = COUNT(Sno)
    FROM S
    WHERE Dno = @Dno)
    IF @TMP = @Dsum
    BEGIN
        PRINT @Dno + '系人数正确'
    END
    ELSE
    BEGIN
        UPDATE D SET Dsum = @TMP WHERE Dno = @Dno
        SELECT @Dname=Dname
        FROM D
        WHERE Dno = @Dno
        INSERT INTO D_TMP
        VALUES(@Dno, @Dname, @Dsum, @TMP)
    END
    FETCH NEXT FROM Dcur INTO @Dno, @Dsum
END
CLOSE Dcur
DEALLOCATE Dcur
SELECT Dno '系号', Dname '系名', Psum '原人数', Dsum '实际人数'
FROM D_TMP
DROP TABLE D_TMP;
```

6.3 前端设计

6.3.1 连接 SQL Server

使用 pyodbc 这个 Module 来连接后端数据库，将 SQL 语句发送到后端，执行完将结果返回到前端。

```
import pyodbc

class Query:
    def __init__(self, host, user, pwd, db):
        self.host = host
        self.user = user
```

```
        self.pwd = pwd
        self.db = db

    def __GetConnect(self):
        if not self.db:
            raise (NameError, "没有设置数据库信息")
        s = "DRIVER={ODBC Driver 17 for SQL
             Server};SERVER=%s;DATABASE=%s;UID=%s;PWD=%s" % (
            self.host, self.db, self.user, self.pwd)
        self.conn = pyodbc.connect(s)
        cur = self.conn.cursor()
        if not cur:
            raise (NameError, "连接数据库失败")
        else:
            return cur

    # 有返回结果的SQL
    def ExecQuery(self, sql):
        cur = self.__GetConnect()
        cur.execute(sql)
        resList = cur.fetchall()
        self.conn.commit()
        self.conn.close()
        return resList

    # 无返回结果的SQL
    def ExecNonQuery(self, sql):
        cur = self.__GetConnect()
        cur.execute(sql)
        self.conn.commit()
        self.conn.close()
```

6.3.2 界面 UI

使用了 Python 3 下的 Tk 工具包来实现前端设计。

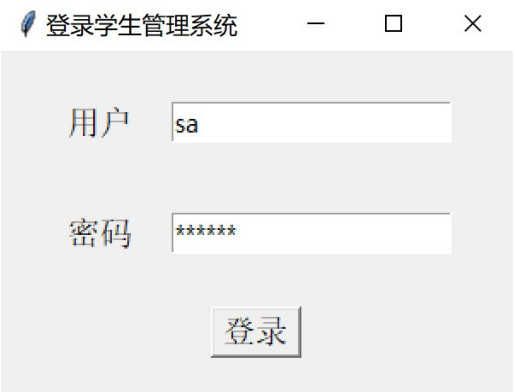

图 4: 登陆界面

图 5: 主页面

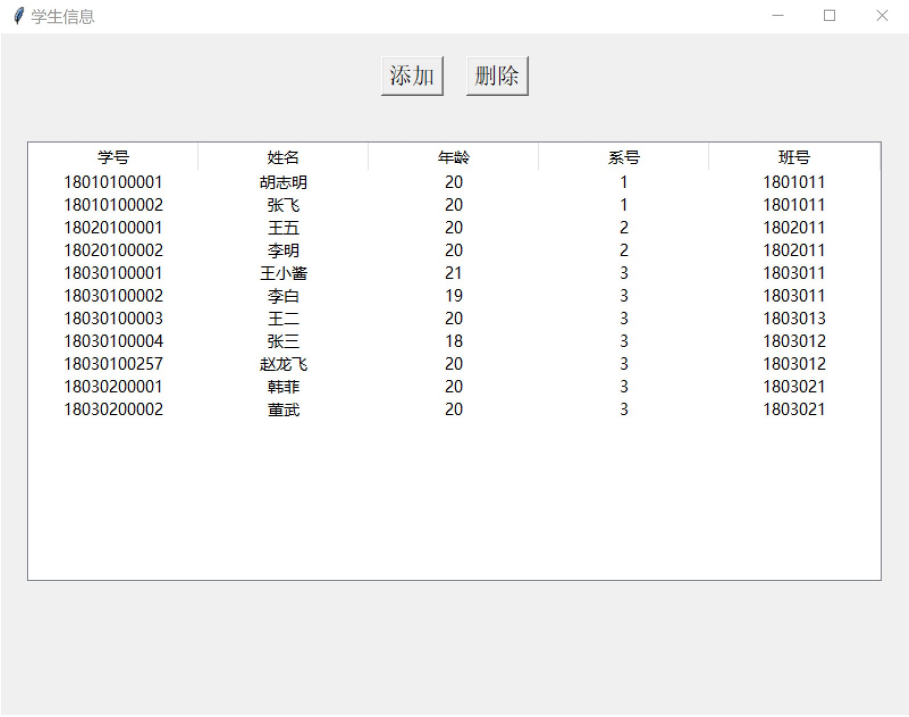

图 6: 子页面

7 结果演示

7.1 学会-人数视图

图 7: 学会-人数视图

7.2 增加学生

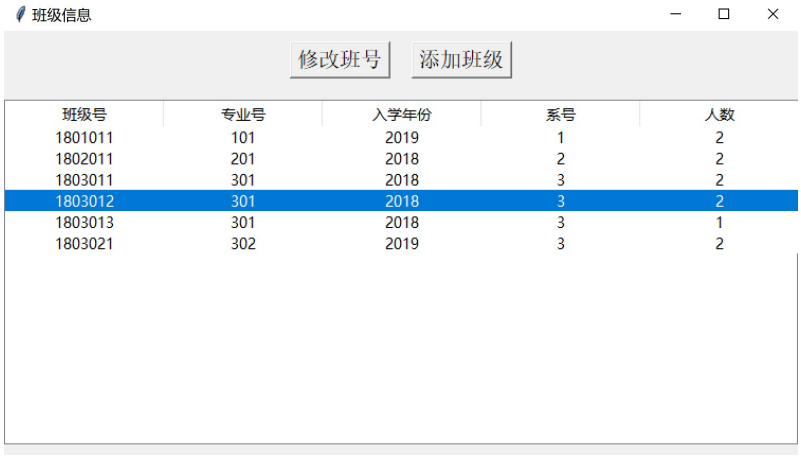

图 8: 插入学生前的班级表

图 9: 插入学生前的系表

图 10: 插入学生

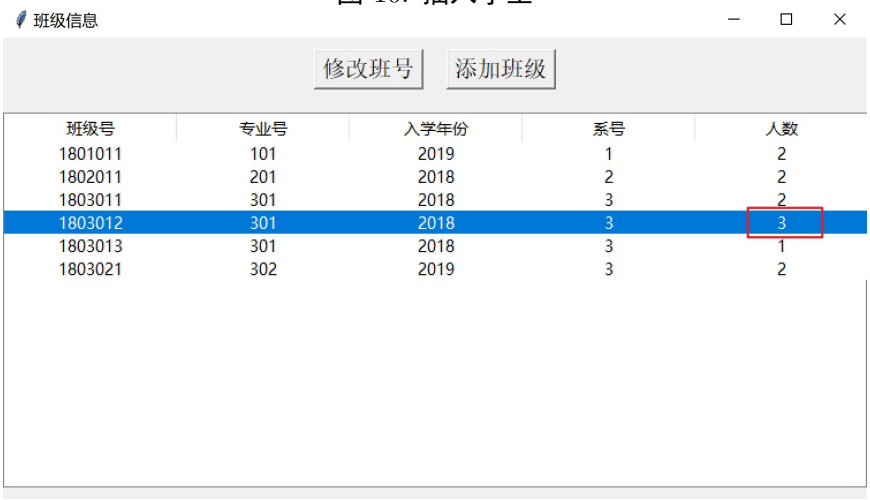

图 11: 插入学生后的班级表

图 12: 插入学生后的系表

可以看到当插入了一个新的学生之后，班级表和系表对应行的人数字段会自动加 1。

7.3 修改班级号

图 13: 修改班级号

图 14: 显示该班级人数

图 15: 修改后的班级表

图 16: 修改后的学生表

可以看到当修改了一个班的班级号后，显示了该班级的人数，而且对应学生表的字段也更新了。(级联更新)

7.4 检查系表人数字段

图 17: 检查前的系表

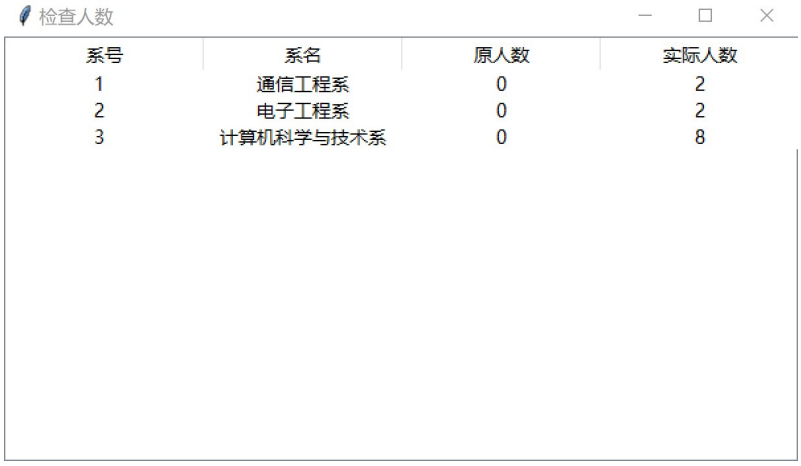

图 18: 检查系表人数

图 19: 检查后的系表

从图中可以看到，在执行检查系表人数时显示出了系表人数字段和当前不符的行，并在执行后将系表人数字段更新正确。

8 实验心得

在这次学生管理系统的设计过程中，我深刻感受到了学习数据库的重要性，对设计一个管理系统的具体步骤和方法有了进一步的认识，加深了我对数据库系统设计相关知识及 SQL Server 相关功能的理解。从建立基本表、视图到触发器和存储过程，通过本次实验这些方面都有了很大提升。除此之外，我还自学了前端设计，学会了如何从前端连接后端，并将 SQL 语句发送给后端去执行。对于前端 UI 界面设计和开发也有了长足的进步。我相信，这次的设计经验对于以后的学习和工作会有很大的帮助。