学生信息管理系统

**软件简介**

该学生信息管理系统旨在帮助用户（如学校管理员、教师或学生本人）有效地管理学生的课程信息。使用tkinter来构建交互界面，通过一个直观的图形用户界面（GUI）来操纵MySQL数据库，并提供了强大的操控系统。用户可以查看学生的课程列表、成绩、学分以及根据成绩计算的GPA（绩点平均分）。

**功能**

**课程信息展示**：在主界面中，用户可以看到一个课程列表，展示了课程名称、分数、学分和GPA。每个课程条目还包含一个课程代码，以便于识别。

**编辑课程信息**：用户可以通过双击课程条目来修改课程信息，如成绩或学分。通过点击“添加课程”按钮，用户可以为学生添加新的课程记录。用户可以选择一个课程条目并点击“删除课程”按钮来移除该课程记录。用户可以选择一个课程条目并点击“修改课程”按钮来编辑选定的课程信息。

**学院管理**：该软件将不同学生划分到不同学院中，方便操作和计算学院平均成绩等。

**修改学生信息**：除了课程信息外，用户还可以通过点击“修改学生信息”按钮来编辑学生的主要信息，如姓名或学号。

**排序**：用户可以根据不同的方式对学生进行排序

**自适应布局**：界面布局会根据窗口大小自动调整，确保用户在不同设备上都能获得良好的使用体验。

该系统提供了一个方便、直观的方式来管理学生的课程信息，能够使得教育管理工作更加高效和有序。

**软件功能操作及展示**

软件的启动需要预先编写**config.json**来确保启动时连接到正确的数据库（如有错误软件将报告错误并展示空的信息页面）

config.json配置如下:

{

    "user":"root",

    "password":"\*\*\*\*",

    "host":"localhost",

    "port":3306,

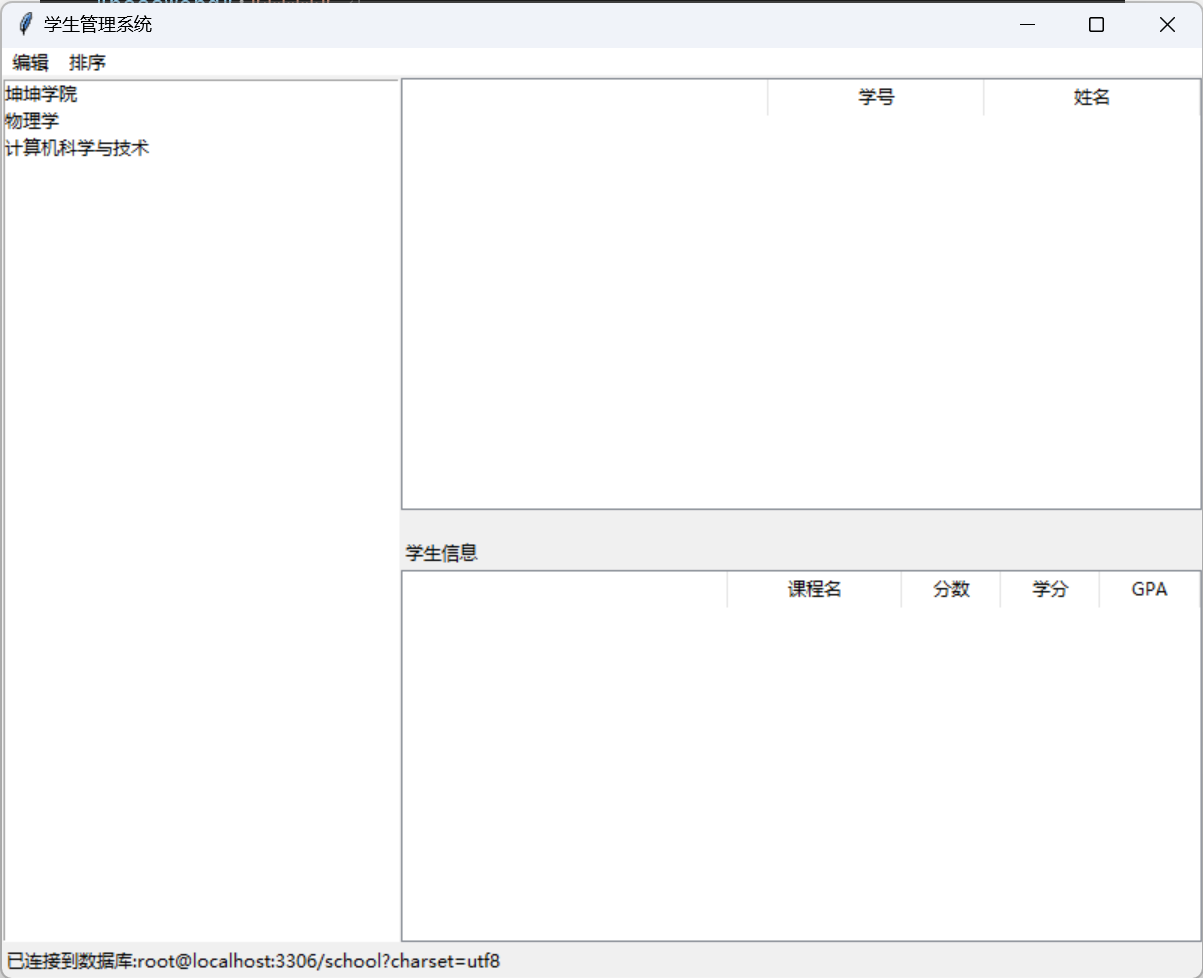
    "database":"school",

    "charset":"utf8"

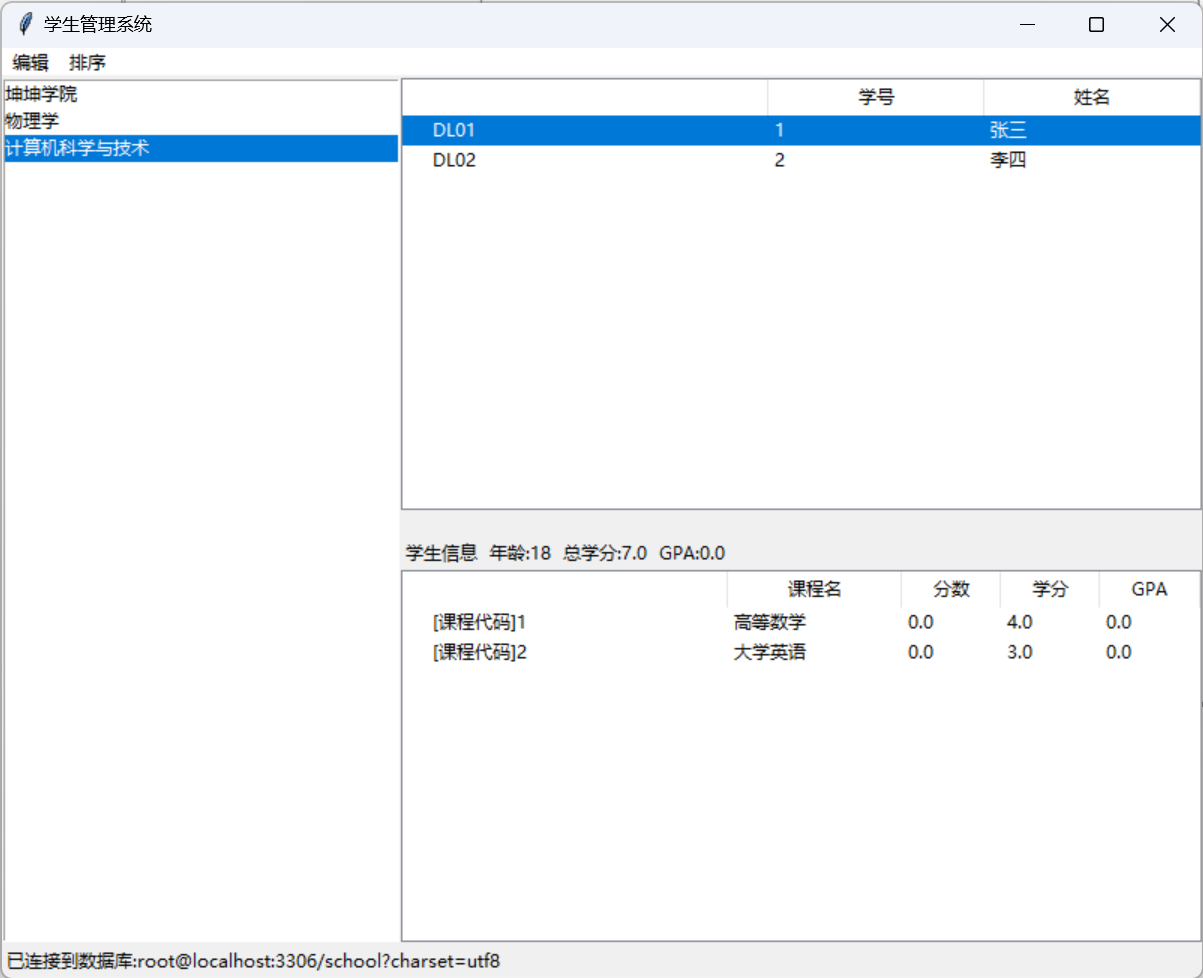
}

User是MySQL数据库的用户名，password是密码，host和port分别是服务器地址和端口，database是要连接的数据库的名称，charset是数据库的字符编码，推荐使用unicode（utf8）

软件启动成功后，界面如下：

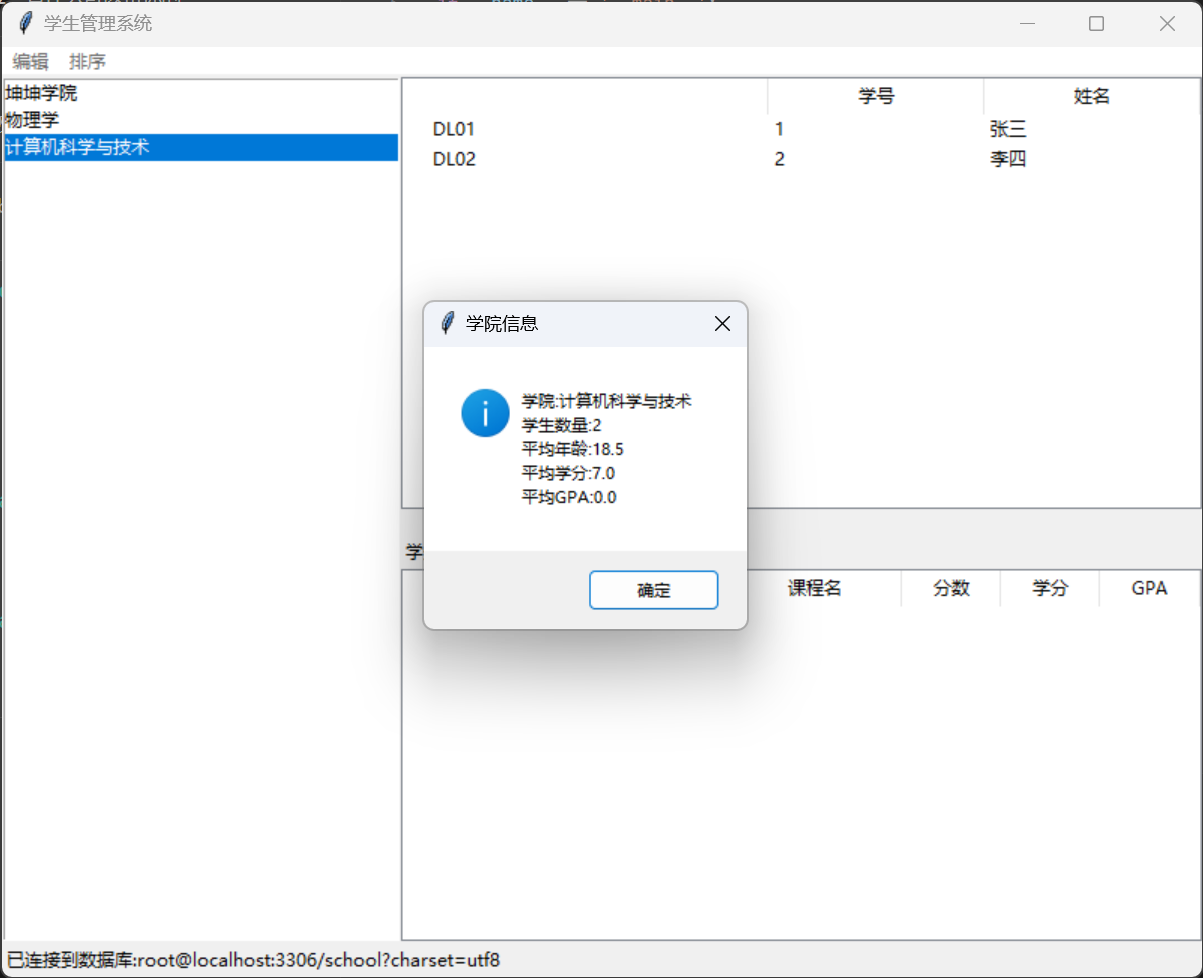


用户可以选中学院后选中学生条目来快速查看学生信息



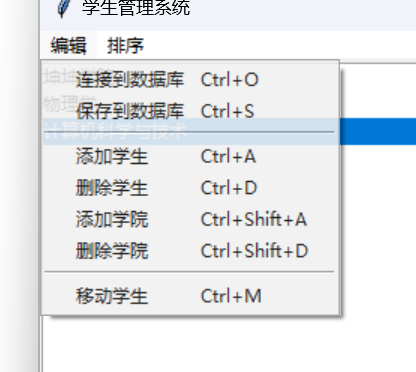
软件会计算出学生的总学分和GPA（各课程GPA的加权和）

双击学院条目则可以显示学院的统计信息

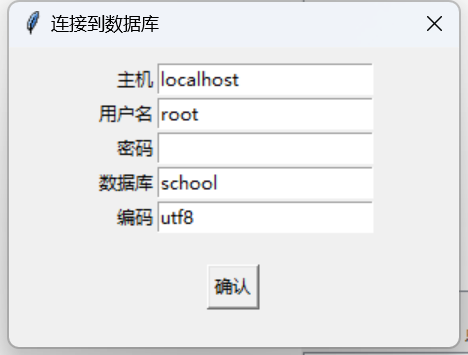


**编辑信息**

选择编辑菜单可以看到如下功能：

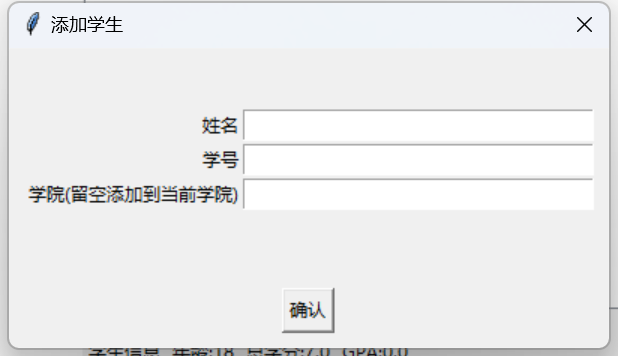


**连接到数据库**可以加载新数据库的数据



**保存到数据库**则可以将当前所有操作写入到数据库中

**添加学生**则可以添加新的学生记录



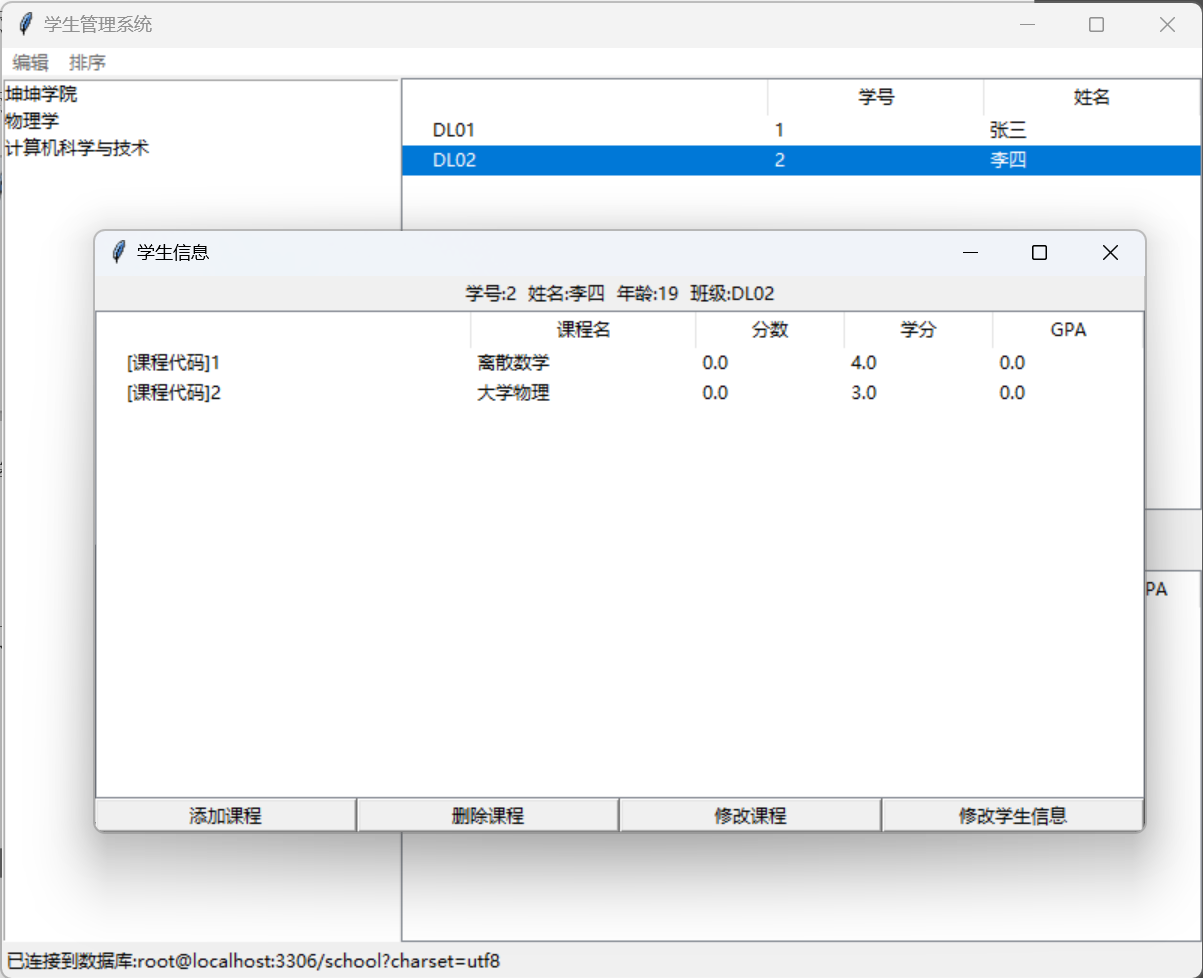
其中学院留空则添加到当前被选中的学院

**添加学院**则可以添加新的学院

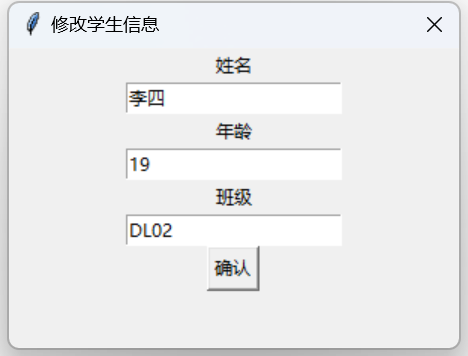
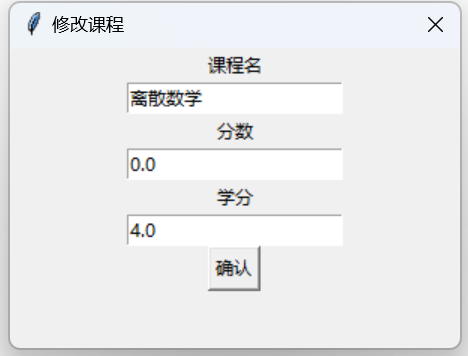
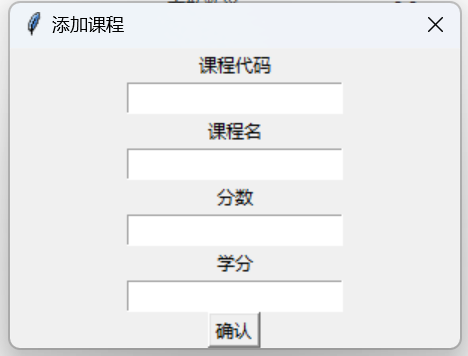


**删除学生**和**删除学院**则可以进行删除操作

当双击学生记录时，进入学生信息编辑窗口

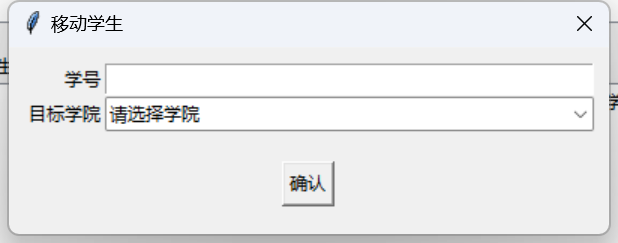


用户可以通过下方四个按钮进行学生信息记录的修改或者双击课程记录进行课程记录的修改

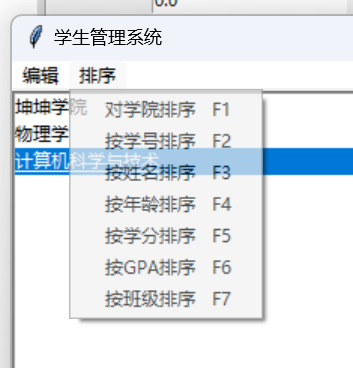


当修改窗口被关闭后，学生信息将被修改

**移动学生**则能将指定学生的记录迁移至其他学院

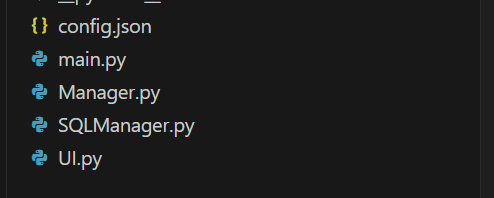


用户也可以通过各种排序来对学生记录进行排序



**代码解释**

本管理程序的代码文件如下：



其中main.py是程序主文件，UI.py是前端图形界面代码文件，Manager.py是后端信息管理代码文件，SQLManager.py是连接MySQL数据库的代码文件，config.json是程序配置

Main.py的结构非常简单：

import UI

import json

if \_\_name\_\_ == '\_\_main\_\_':

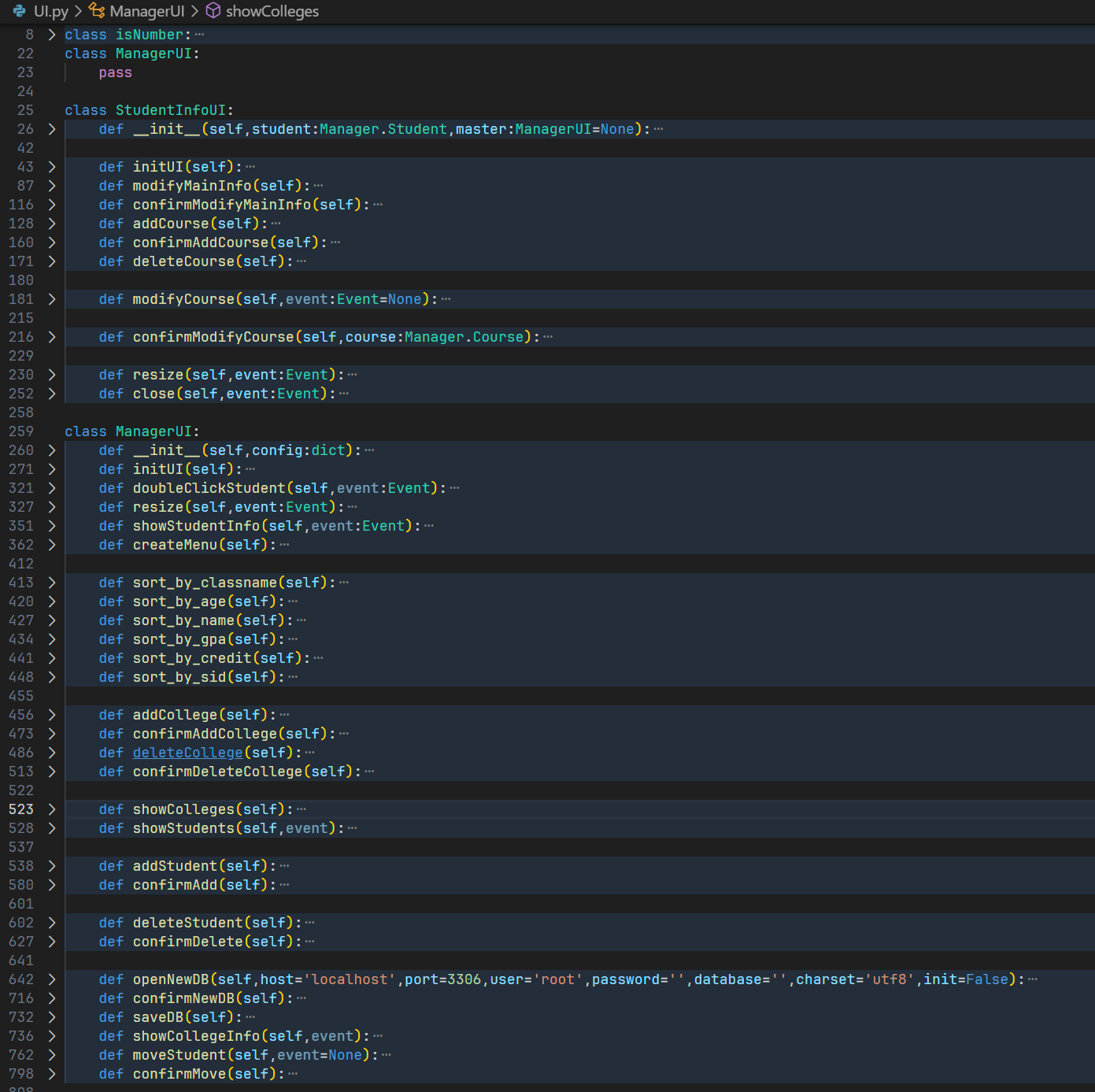
    with open('config.json','r') as f:

        config = json.load(f)

    UI.ManagerUI(config)

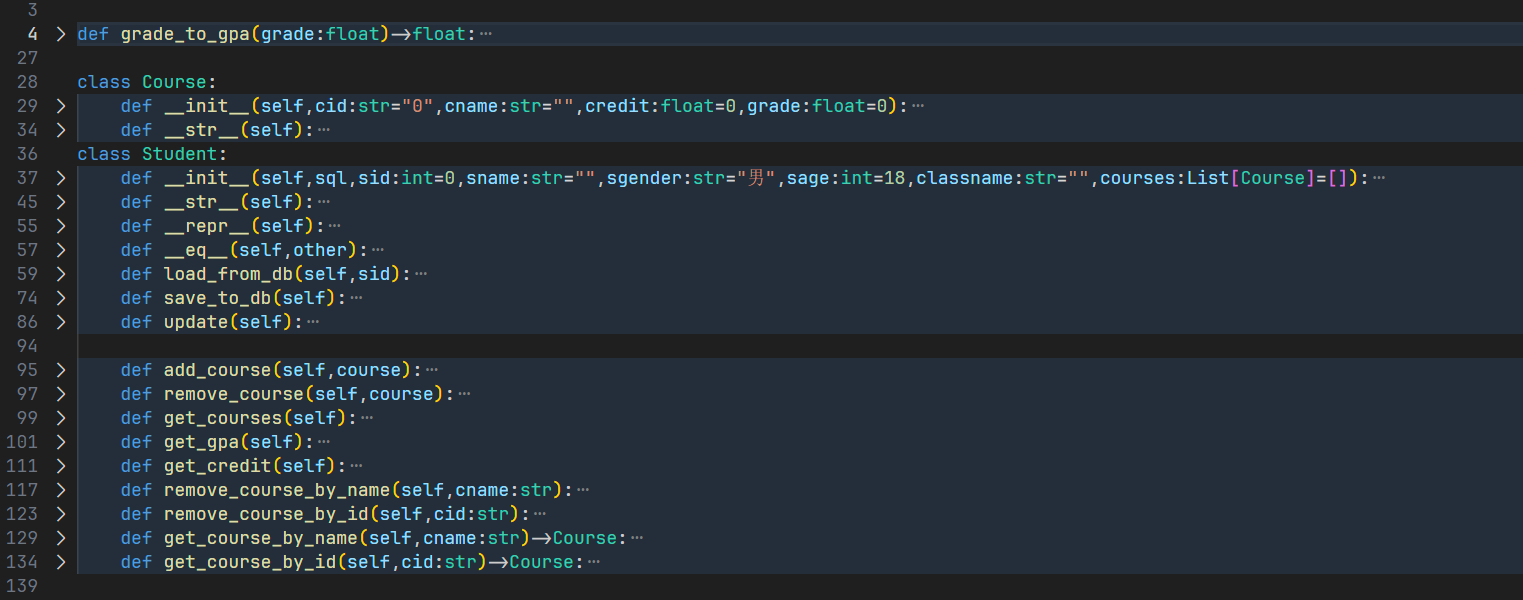
其主要用途是加载配置文件并创建图形界面

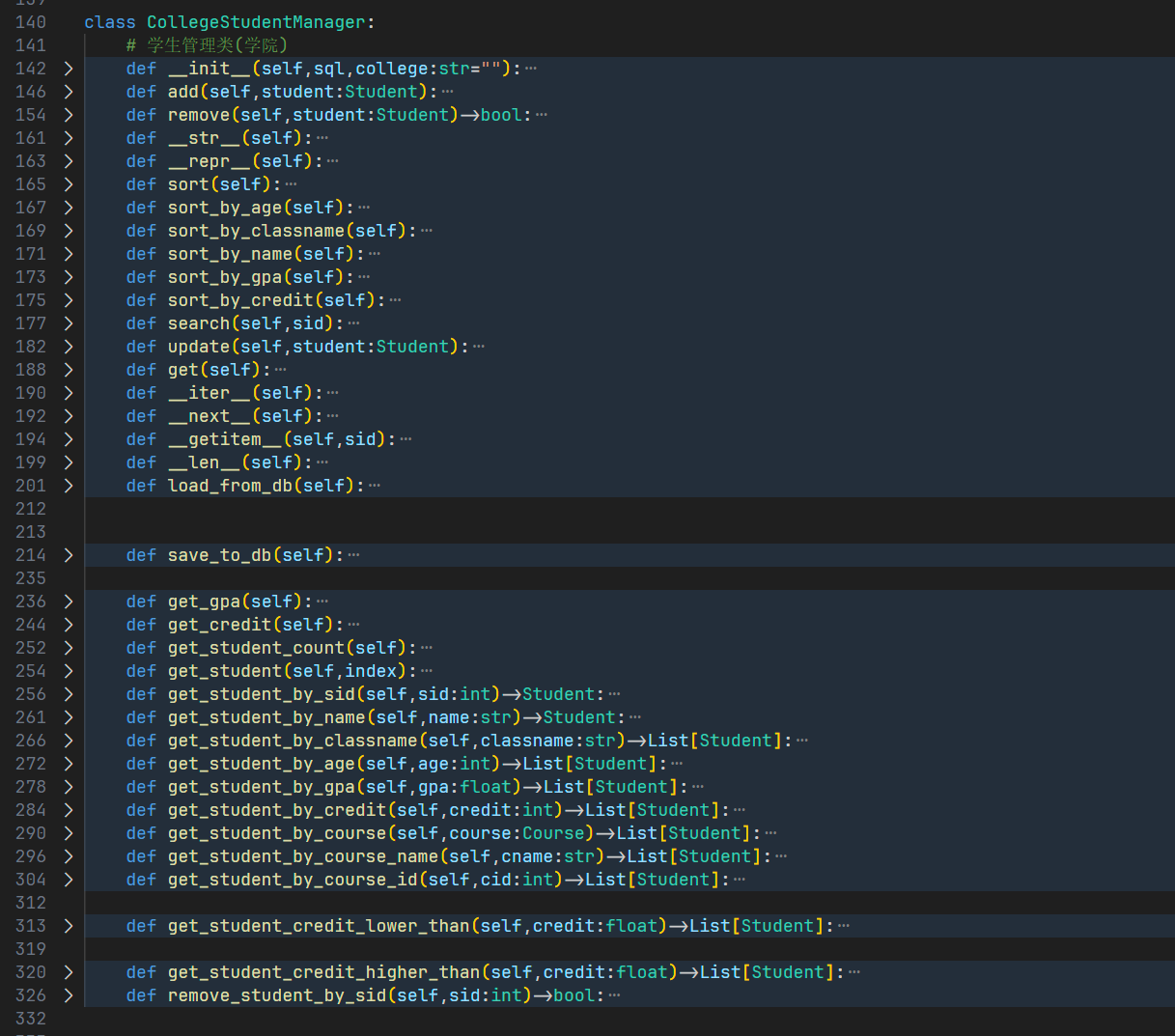
UI.py负责整个用户界面交互，因此代码较为复杂（多数为控件操作），简化的声明如下

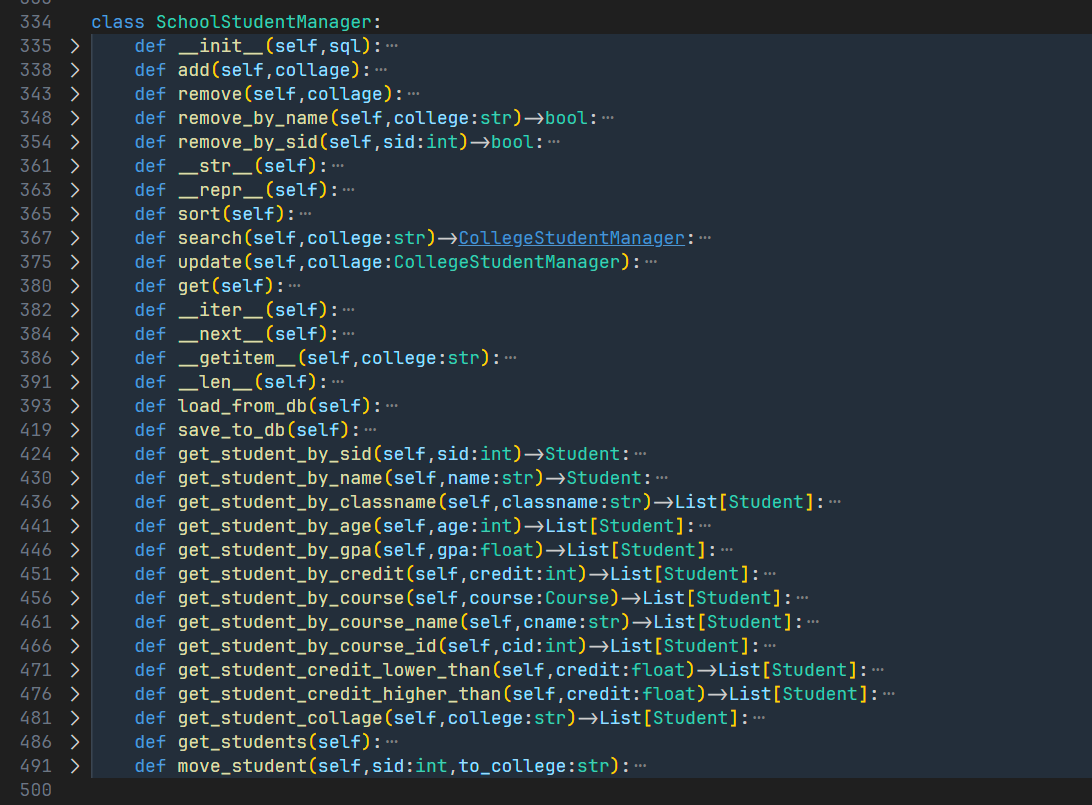


其中ManagerUI类负责整个主界面的交互，而StudentInfoUI类负责学生信息编辑的窗口的交互。其共同实现了对后端Manager.py中代码逻辑的操作。

Manager.py中代码实现如下：







其中SchoolStudentManager类实现了总的学生管理（分学院查找、按学生信息查找、修改等），CollegeStudentManager类实现了单个学院的学生管理（查找、修改、排序等），Student类则实现了学生信息的储存修改等，Course类则实现了课程信息的管理。包含关系为SchoolStudentManager>CollegeStudentManager>Student>Course

以上的类都重载了一些python内置的函数来实现更方便的代码调用和实现（如通过重载**\_\_getitem\_\_**方法来实现[]索引学院）

SQLManager.py的实现如下：

import pymysql as sql

class SQLManager:

    def \_\_init\_\_(self,host='localhost',port=3306,user='root',password='',database='',charset='utf8'):

        self.host = host

        self.port = port

        self.database = database

        self.user = user

        self.database = database

        self.charset = charset

        if database == '':

            raise ValueError('database must be not empty')

        try:

            self.conn = sql.connect(

                host=host,

                port=port,

                user=user,

                password=password,

                database=database,

                charset=charset

            )

        except Exception as e:

            print('Connect failed:',e)

            raise e

        self.cursor = self.conn.cursor()

    def \_\_del\_\_(self):

        if not hasattr(self,'cursor'):

            return

        self.cursor.close()

    def query(self,sql):

        self.cursor.execute(sql)

        return self.cursor.fetchall()

    def update(self,sql):

        self.cursor.execute(sql)

        self.conn.commit()

    def insert(self,sql):

        self.cursor.execute(sql)

        self.conn.commit()

    def delete(self,sql):

        self.cursor.execute(sql)

        self.conn.commit()

    def close(self):

        self.cursor.close()

        self.conn.close()

    def \_\_str\_\_(self):

        # 返回当前数据库的连接信息

        return f"{self.user}@{self.host}:{self.port}/{self.database}?charset={self.charset}"

这个代码主要实现了MySQL数据库的连接以及MySQL命令的提交和数据库修改

SQL语句的实现如下

对于学生信息的存储

    def load\_from\_db(self,sid):

        self.sid = sid

        sql = f"select \* from student where sid={sid}"

        result = self.sql.query(sql)

        if result:

            self.sname = result[0][1]

            self.sgender = result[0][2]

            self.sage = int(result[0][3])

            self.classname = result[0][4]

            # load courses

            sql = f"select \* from course where sid={sid}"

            result = self.sql.query(sql)

            for row in result:

                course = Course(row[1],row[2],float(row[3]),float(row[4]))

                self.courses.append(course)

    def save\_to\_db(self):

        # 检查学生是否存在，如果存在则更新，否则插入

        sql = f"select \* from student where sid={self.sid}"

        result = self.sql.query(sql)

        if result:

            self.update()

            return

        sql = f"insert into student values({self.sid},'{self.sname}','{self.sgender}',{self.sage},'{self.classname}')"

        self.sql.insert(sql)

        for course in self.courses:

            sql = f"insert into course values({self.sid},'{course.cid}','{course.cname}',{course.credit},{course.grade})"

            self.sql.insert(sql)

对于学院信息的存储

def load\_from\_db(self):

        if not self.sql:

            raise ValueError("SQLManager is not set")

        # 从数据库加载学生信息

        # 读取学院下的所有学生的id，然后加载学生信息

        sql = f"select sid from college\_students where classname='{self.college}'"

        result = self.sql.query(sql)

        for row in result:

            student = Student(self.sql)

            student.load\_from\_db(row[0])

            self.students.append(student)

    def save\_to\_db(self):

        if not self.sql:

            raise ValueError("SQLManager is not set")

        sql = f"delete from college where collegename='{self.college}'"

        self.sql.delete(sql)

        # 插入学院，然后在学院的子级插入所有学生id

        # 类似

        # collegename

        # - id1

        # - id2

        # 插入学院

        sql = f"insert into college values('{self.college}')"

        self.sql.insert(sql)

        sql = f"delete from college\_students where classname='{self.college}'"

        self.sql.delete(sql)

        # 在学院条目下插入所有学生id

        for student in self.students:

            sql = f"insert into college\_students values('{self.college}',{student.sid})"

            self.sql.insert(sql)

            student.save\_to\_db()

对于全体学生信息的存储，通过遍历学院进行存储

    def load\_from\_db(self):

        if not self.sql:

            raise ValueError("SQLManager is not set")

        # 先检索是否存在college表和student表，如果不存在则创建

        sql = "show tables"

        result = self.sql.query(sql)

        if ('college',) not in result:

            sql = "create table college(collegename varchar(255) primary key)"

            self.sql.update(sql)

        if ('college\_students',) not in result:

            sql = "create table college\_students(classname varchar(255),sid int)"

            self.sql.update(sql)

        if ('student',) not in result:

            sql = "create table student(sid int primary key,sname varchar(255),sgender varchar(255),sage int,classname varchar(255))"

            self.sql.update(sql)

        if ('course',) not in result:

            sql = "create table course(sid int,cid varchar(255),cname varchar(255),credit float,grade float)"

            self.sql.update(sql)

        # 从数据库加载学生信息

        # 读取学院下的所有学生的id，然后加载学生信息

        sql = f"select collegename from college"

        result = self.sql.query(sql)

        for row in result:

            college = CollegeStudentManager(self.sql,row[0])

            college.load\_from\_db()

            self.collages.append(college)

    def save\_to\_db(self):

        if not self.sql:

            raise ValueError("SQLManager is not set")

        for collage in self.collages:

            collage.save\_to\_db()

创建后的数据库结构如下：

