

中国科学技术大学计算机学院
《数据库系统实验报告》



实验题目：HOMO-MIX学生宿舍管理系统

学生姓名：李田所

学生学号：PB21114514

完成时间：2024年6月22日

需求分析

引言

近年来，随着校园信息化管理的实施，大到学籍和教务管理，小到宿舍维修管理，都可以通过数据库系统高效实现。在本次数据库课程大实验中，我计划实现一个简单的宿舍管理系统“HOMO-MIX”，以实现基本的人员信息管理、维修申报、访客登记等功能。走完整个数据库开发流程，将会帮助我理解书本知识，并提高实践能力。

用户角色分析

“HOMO-MIX”数据库系统中的主要用户角色包括学生公寓管理部门、公寓管理员、学生、维修人员。对于每种角色，他们对于数据库系统的需求都是不同的：

- **学生公寓管理部门**：录入和修改楼栋和房间信息；查看已有楼栋和房间信息；删除楼栋和房间信息。录入新入职公寓管理员的信息；删除离职公寓管理员的信息；查看已有的公寓管理员信息；修改错误的公寓管理员信息。
- **公寓管理员**：录入新入住学生的信息；删除退宿学生的信息；查看已有的学生信息；修改错误的学生信息。填写和修改访客记录；查看已有访客记录；删除访客记录。
- **学生**：发起或取消宿舍维修申报，查看申报进度。
- **维修人员**：查看宿舍维修申报记录，修改维修状态。

功能需求

- **宿管信息管理**：考虑到公寓管理员的入职和离职、个人信息补录等事件，需要能够增、删、改、查公寓管理员信息。
- **学生信息管理**：考虑到学生的入住和退宿、个人信息补录等事件，需要能够增、删、改、查学生信息，同时需要更新公寓房间的状态。
- **房间维修管理**：学生能够发起、查看和取消宿舍维修申报，已完成维修的申报不能取消。维修人员可以查看和修改维修记录。需要能够增、改、查维修申报记录。
- **访客登记管理**：宿管的工作职责之一是填写访客记录、修改或删除错误的访客记录，需要能够增、删、改、查访客记录信息。
- **公寓楼栋和房间信息管理**：需要能够增、删、改、查公寓楼栋和房间信息。这种信息通常不会发生变化，且误操作代价较大，有需求时可以交给数据库开发人员进行迭代升级。所以我不再做出这一功能的操作界面，只需要保证数据库的可拓展性，以便于更新迭代。

非功能需求

- 考虑到用户的实际使用体验，需要设计用户友好的图形化界面，注意界面的简洁和美观。还需保证操作简便，学习成本低。
- 保证数据库系统可以扩展，例如加上“学生假期离校返校记录”等功能。

总体设计

系统模块结构

1. 前端部分

前端部分主要通过调用python的tkinter包来完成，分成GUI.py等6个文件。

运行主文件GUI.py进入主界面，可以通过点击图形化界面的按钮，运行其余5个文件，以进入不同的界面。这样能够增加不同用户界面的独立性，增强代码的可读性，在代码编写过程中也更加方便。

剩余5个文件分别负责：

- dk_GUI1.py和dk_GUI2.py：面向宿管用户的界面，前者可进行学生信息管理，后者可进行访客信息管理；
- stu_GUI.py：面向学生用户的界面，可进行宿舍维修申报；
- rp_GUI.py：面向维修工用户的界面，可处理学生的宿舍维修申报；
- super_GUI.py：面向物业管理人员的界面，可管理在职宿管的身份信息。

2. 后端部分

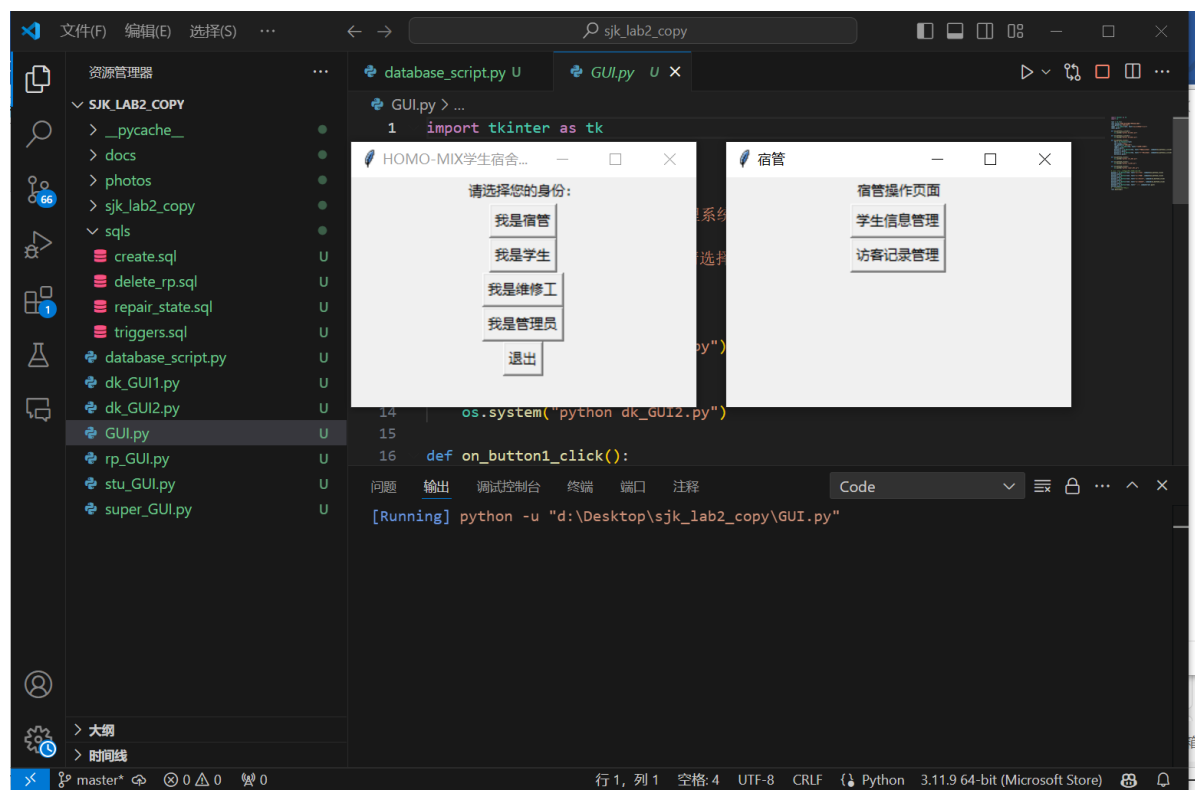
后端部分由4个sql文件和database_script.py组成。

4个sql文件分别负责：

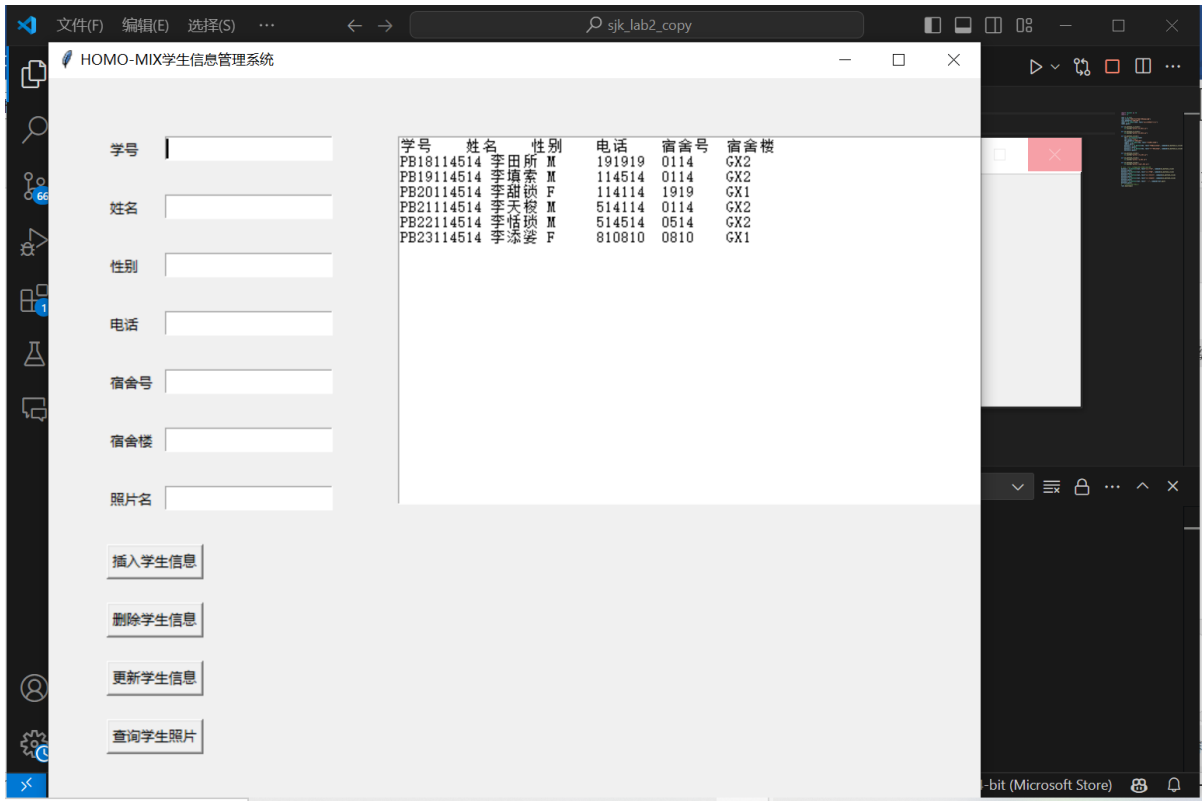
- create.sql：表的建立和初始数据的导入；
- delete_rp.sql：一个sql过程，用来删除未维修的维修记录；
- triggers.sql：一个sql触发器，处理学生信息的删除；
- repair_state.sql：一个sql函数，用来返回维修状态符号对应的名称。

系统工作流程

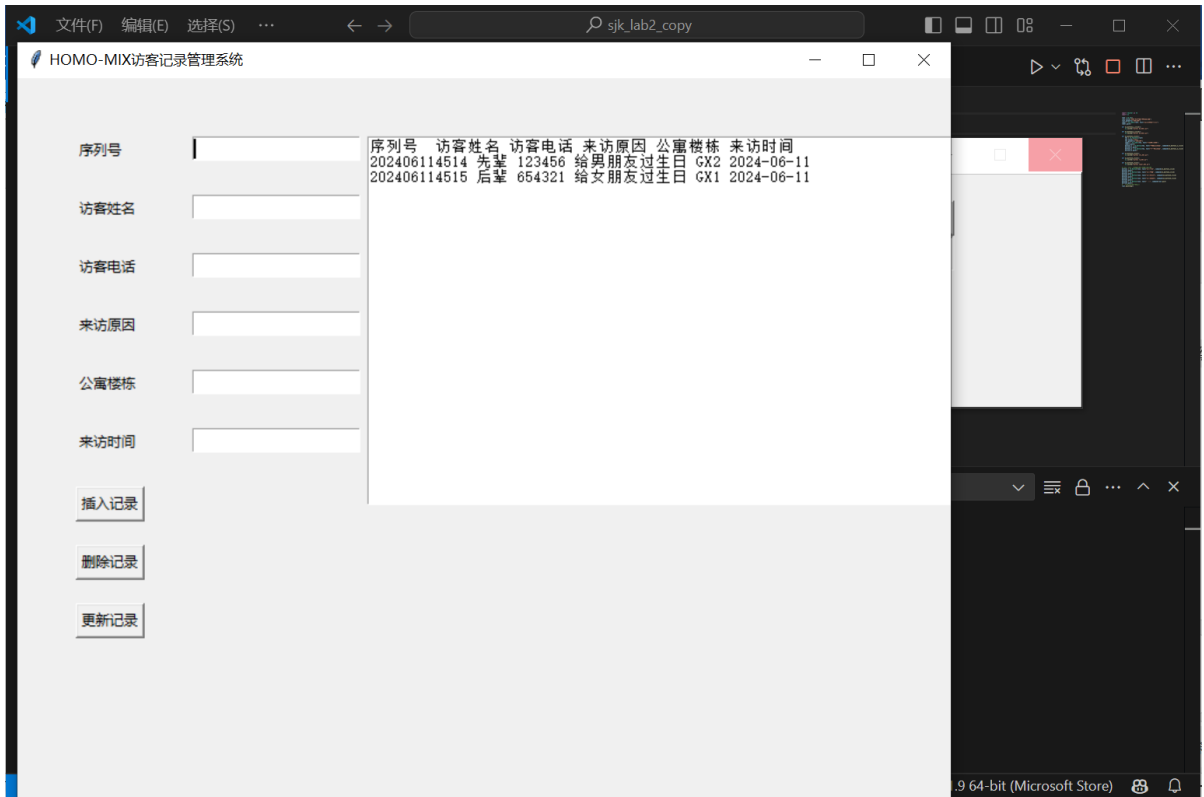
- 主界面：



- 界面1：



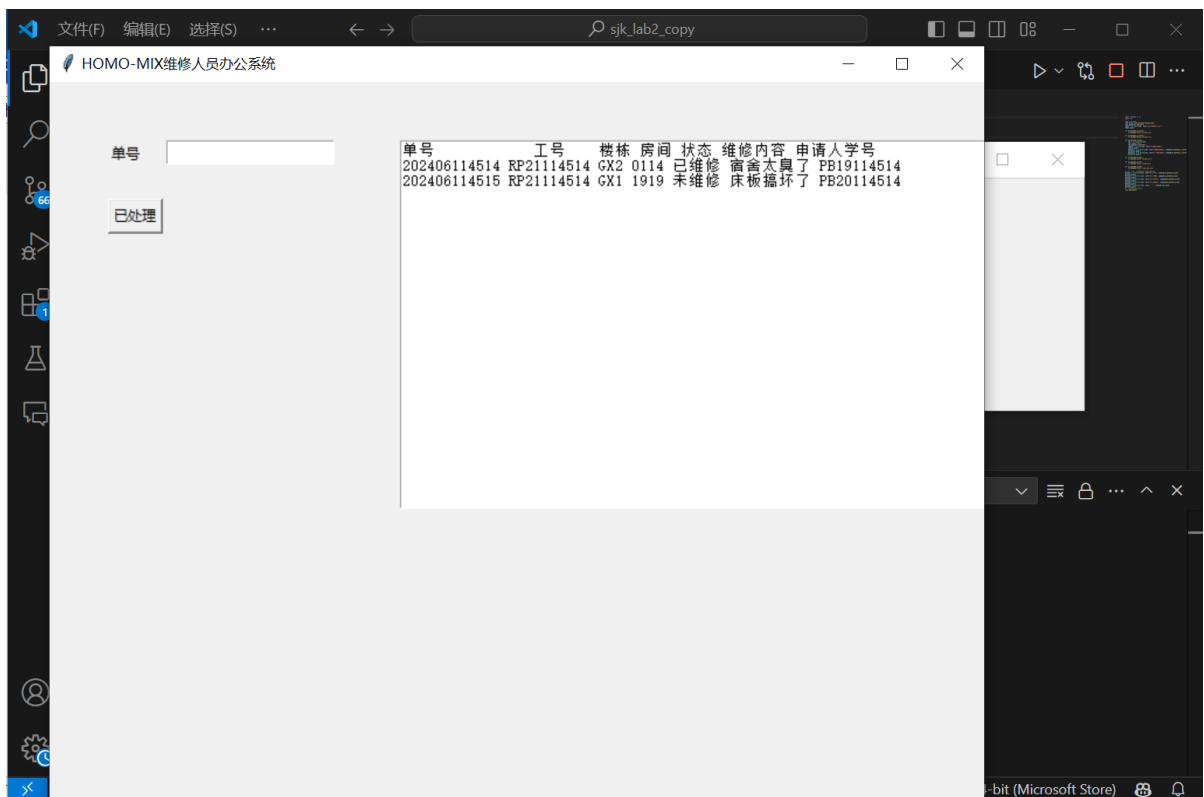
- 界面2:



- 界面3:



- 界面4:

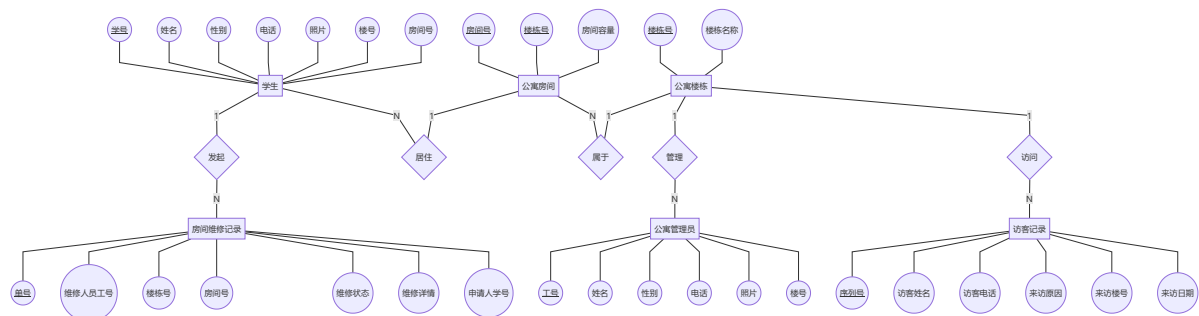


- 界面5:



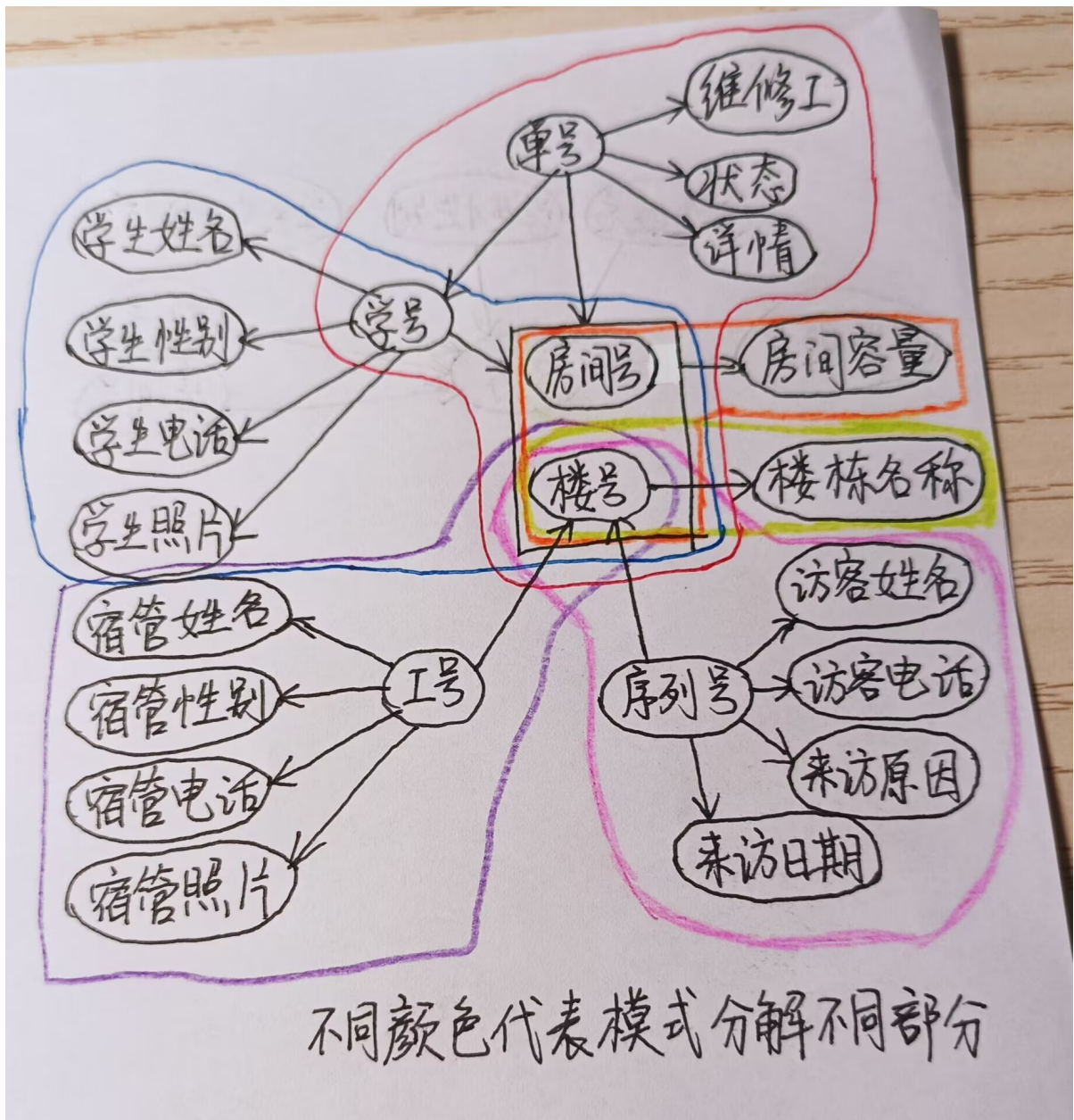
数据库设计

ER图



模式分解

该数据库的模式满足2NF。之所以没有满足3NF，是因为学生可能会搬宿舍，如果不在维修表中存放房间号和楼号信息，可能导致维修记录指向错误的宿舍。这样一来，就产生了“维修单号->学号->房间号和楼号”的传递依赖，不再是3NF。



存储过程、触发器、函数等设计思路

- 存储过程：输入要删除的维修记录单号。如果对应记录状态为未维修，删除相应的记录；否则不做任何事。

```

delimiter //
create procedure delete_repair(
    in r_id char(24)
)
begin
    delete from Repair where Repair.r_id = r_id and r_state = 'F';
end //
delimiter ;

```

- 触发器：删除学生信息后，也删除该学生发起的维修记录。

```

delimiter //
create trigger delete_student after delete on Student for each row
begin
    delete from Repair where Repair.r_stu_id = old.s_id;
end //
delimiter ;

```

- 函数：输入维修记录的状态代码，输出状态的全称。

```

DELIMITER //
create FUNCTION Repair_state(state char(2)) RETURNS varchar(10)
BEGIN
    declare result varchar(10);
    if state = 'T' then
        set result = '已维修';
    else
        set result = '未维修';
    end if;
    return result;
END //
DELIMITER ;

```

核心代码解析

仓库地址

https://github.com/LingLingMao/USTC_Database_lab2_2024.git

目录

```

| database_script.py --数据库连接与事务提交
| dk_GUI1.py --学生信息管理界面
| dk_GUI2.py --访客信息管理界面
| GUI.py --主界面
| README.md
| rp_GUI.py --维修工办公界面
| stu_GUI.py --学生宿舍维修申请界面
| super_GUI.py --物业管理界面
|
|--photos --测试用图片
|   1.jpg
|   10.jpg
|   11.jpg
|   12.jpg
|   2.jpg
|   3.jpg
|   4.jpg
|   5.jpg
|   6.jpg
|   7.jpg
|   8.jpg
|   9.jpg
|
|--sqls

```



```
| create.sql --表的建立
| delete_rp.sql --过程: 删除维修记录
| repair_state.sql --函数: 显示维修状态
| triggers.sql --触发器: 删除学生后, 删除相关维修记录
|
└─__pycache__
    botton1.cpython-311.pyc
    botton2.cpython-311.pyc
    botton3.cpython-311.pyc
    botton4.cpython-311.pyc
    database_script.cpython-311.pyc
```

主界面功能实现

在主界面放置按钮, 点击后直接编译其他的py文件, 例如:

```
# GUI.py
def on_button2_click():
    os.system("python stu_GUI.py")

botton2 = tk.Button(root, text="我是学生", command=on_button2_click)
botton2.pack()
```

其他界面功能实现 (以学生信息管理为例)

前端通过ds.pub_in和ds.pub_out两个全局变量与后端交互。输入框中的字符串直接通过ds.pub_out传给后端, 然后再接收ds.pub_in的值打印到输出框:

```
# dk_GUI1.py
import database_script as ds

def update_text(text): # 输出框
    text.delete(1.0, "end")
    text.insert("insert", "学号\t姓名\t性别\t电话\t宿舍号\t宿舍楼\n")
    ds.query_student()
    for row in ds.pub_out:
        text.insert("insert", str(row[0]) + "\t" + str(row[1]) + "\t" +
str(row[2]) + "\t" + str(row[3]) + "\t" + str(row[4]) + "\t" + str(row[5]) +
"\t\n")
    while len(ds.pub_out) > 0:
        ds.pub_out.pop()

def on_button1_click(): # 插入学生
    ds.pub_in.append(entry1.get())
    ds.pub_in.append(entry2.get())
    ds.pub_in.append(entry3.get())
    ds.pub_in.append(entry4.get())
    ds.pub_in.append(entry5.get())
    ds.pub_in.append(entry6.get())
    ds.pub_in.append(entry7.get())
    ds.insert_student()
    while len(ds.pub_in) > 0:
        ds.pub_in.pop()
    update_text(text)
```

特别地，当需要显示图片时，需要先把二进制数据转换成图片格式：

```
# dk_GUI1.py
def on_button4_click():
    ds.pub_in.append(entry1.get())
    ds.query_student_image()

    # 显示blob字段的二进制图片
    blob_data = ds.pub_out[0][6]
    image = Image.open(io.BytesIO(blob_data))
    photo = ImageTk.PhotoImage(image)
    label = tk.Label(root, image=photo)
    label.image = photo
    label.place(x=500, y=400)
    while len(ds.pub_in) > 0:
        ds.pub_in.pop()
    while len(ds.pub_out) > 0:
        ds.pub_out.pop()
```

实现与数据库的连接

后端接收全局变量pub_in的值后，通过mysql.connector库直接与数据库连接，生成并提交事务，再把返回的值赋给pub_out变量：

```
# database_script.py
def delete_student(): # 删除学生
    # 创建数据库连接
    cnx = mysql.connector.connect(user='root', password='031006',
    host='localhost', database='sys')
    cursor = cnx.cursor()

    # 执行一个删除
    delete_student = ("DELETE FROM Student WHERE Student.s_id = '" +
    str(pub_in[0]) + "'")
    cursor.execute(delete_student)
    cnx.commit() # 提交事务

    # 关闭连接
    cursor.close()
    cnx.close()

def query_student(): # 查询所有学生记录
    # 创建数据库连接
    cnx = mysql.connector.connect(user='root', password='031006',
    host='localhost', database='sys')
    cursor = cnx.cursor()

    # 执行一个查询
    query = ("SELECT * FROM Student")
    cursor.execute(query)

    # 获取查询结果
    results = cursor.fetchall()
    for row in results:
        pub_out.append(row)
```

```
# 关闭连接
cursor.close()
cnx.close()
```

特别地，在插入图片时，需要先把图片转换成二进制格式，再用blob字段保存：

```
# database_script.py
def convertToBinaryData(filename): # 把文件转换成二进制
    # Convert digital data to binary format
    with open(filename, 'rb') as file:
        binaryData = file.read()
    return binaryData

def insert_student():# 插入学生信息
    # 创建数据库连接
    cnx = mysql.connector.connect(user='root', password='031006',
    host='localhost', database='sys')
    cursor = cnx.cursor()

    # 为方便测试，图片均从固定路径选取，只需输入图片名
    if pub_in[6] == '': # 图片为空
        blob = 'NULL'
    else: # 把图片转换成二进制
        blob = convertToBinaryData("./photos/"+pub_in[6])

    # 执行一个插入
    add_student = ("INSERT INTO Student values (%s, %s, %s, %s, %s, %s, %s)")
    cursor.execute(add_student, (pub_in[0], pub_in[1], pub_in[2], pub_in[3],
    pub_in[4], pub_in[5], blob))
    cnx.commit() # 提交事务

    # 关闭连接
    cursor.close()
    cnx.close()
```

数据库的表结构

表的结构如模式分解部分所示，所以不再赘述。由于本人忙于期末考试，为了简化测试流程，仅对表添加少量约束，在实际应用中我不会这么做。

create table代码如下：

```
-- create.sql
use sys;

drop table if exists Student;
drop table if exists DormKeeper;
drop table if exists Dormitory;
drop table if exists Building;
drop table if exists Repair;
drop table if exists Visitor;

set sql_safe_updates=0;
create table Student(
```

```

        s_id char(20) primary key,
        s_name varchar(40),
        s_sex varchar(2),
        s_tele varchar(30),
        s_dorm_id char(8),
        s_build_id varchar(10),
        s_photo blob,

        constraint CK_Student1 check(s_sex in ('M', 'F')),
        constraint CK_Student2 check(s_name is not null)
    );

create table DormKeeper(
    dk_id char(20) primary key,
    dk_name varchar(40),
    dk_sex varchar(2),
    dk_tele varchar(30),
    dk_build_id varchar(10),
    dk_photo blob,

    constraint CK_DormKeeper1 check(dk_sex in ('M', 'F')),
    constraint CK_DormKeeper2 check(dk_name is not null)
);

create table Dormitory(
    d_id char(8),
    d_build varchar(10),
    d_max int,

    constraint PK_Dormitory primary key(d_id, d_build)
);

create table Building(
    b_id varchar(10) primary key,
    b_name varchar(40),

    constraint CK_Building1 check(b_name is not null)
);

create table Repair(
    r_id char(24) primary key,
    r_man_id char(20),
    r_build varchar(10),
    r_dorm char(8),
    r_state char(2),
    r_content varchar(100),
    r_stu_id char(20),

    constraint CK_Repair1 check(r_state in ('T', 'F'))
);

create table Visitor(
    v_id char(24) primary key,
    v_name varchar(40),
    v_tele varchar(30),
    v_content varchar(100),

```

```
v_build_id varchar(10),
v_date date,

constraint CK_Visitor1 check(v_name is not null)
);
```

实验与测试

依赖

所需的py库：tkinter, mysql, PIL, io, os

运行环境：Windows10, Python 3.11

部署

```
python GUI.py
```

实验结果

代码能实现基本的增删改查功能，存储过程、函数、触发器能达到预期效果，也能对图片（学生照片和宿管照片）进行增删改查的操作。全部功能已经在检查实验时演示完毕，由于截图过多，仅在此展示部分结果。

- 增删改查学生信息（查询的照片显示在右下角）：

HOMO-MIX学生信息管理系统

学号

PB24114514

姓名

李阔秒

性别

M

电话

555555

宿舍号

0114

宿舍楼

GX2

照片名

10.jpg


插入学生信息

删除学生信息

更新学生信息

查询学生照片

学号	姓名	性别	电话	宿舍号	宿舍楼
PB18114514	李田	M	191919	0114	GX2
PB19114514	李填	M	114514	0114	GX2
PB20114514	李甜	F	114114	1919	GX1
PB21114514	李天	M	514114	0114	GX2
PB22114514	李恬	M	514514	0514	GX2
PB23114514	李添	F	810810	0810	GX1
PB24114514	李阔	M	555555	0114	GX2



• 宿舍维修申报和撤销：

HOMO-MIX学生宿舍维修申报系统

单号

202406114516

工号

RP21114514

楼栋

GX2

房间

0114

状态

F

维修内容

宿舍door坏了

学号

PB24114514

申报

撤销申报

单号

工号

楼栋

房间

状态

维修内容

申请人学号

202406114514

RP21114514

GX2

0114

已维修

宿舍太臭了

PB19114514

202406114515

RP21114514

GX1

1919

未维修

床板搞坏了

PB20114514

202406114516

RP21114514

GX2

0114

未维修

宿舍door坏了

PB24114514

• 宿舍维修状态修改：

HOMO-MIX维修人员办公系统

单号

202406114516

已处理

单号

工号

楼栋

房间

状态

维修内容

申请人学号

202406114514

RP21114514

GX2

0114

已维修

宿舍太臭了

PB19114514

202406114515

RP21114514

GX1

1919

未维修

床板搞坏了

PB20114514

202406114516

RP21114514

GX2


0114

已维修

宿舍door坏了

PB24114514

- 删除学生后自动删除维修记录:

 HOMO-MIX维修人员办公系统

单号

已处理

单号	工号	楼栋	房间	状态	维修内容	申请人学号
202406114514	RP21114514	GX2	0114	已维修	宿舍太臭了	PB19114514
202406114515	RP21114514	GX1	1919	未维修	床板搞坏了	PB20114514

参考

本项目前后端均未使用现成模板，但部分参考自大模型文心一言4.0和copilot给出的回答，在此感谢这两个大模型的开发团队。

测试图片中的暹罗猫来自小豆泥系列表情包，在此感谢小豆泥的作者；兔子图片来源未知，也感谢兔子表情包的作者。

所需py库的官网：

- tkinter：<https://customtkinter.tomschimansky.com/>
- mysql：<https://pypi.org/project/MySQL-python/>
- PIL：<https://pillow.readthedocs.io/en/latest/>
- io：<https://docs.python.org/zh-cn/3/library/io.html#module-io>
- os：<https://docs.python.org/zh-cn/3/library/os.html?highlight=os#module-os>