|  |  |
| --- | --- |
| **报告成绩** |  |
| **演示成绩** |  |
| **答辩成绩** |  |
| **总评** |  |

**中北大学**

**课 程 设 计 说 明 书**

数据库课程实验周

|  |  |  |  |
| --- | --- | --- | --- |
| **学生姓名:** | **赵琦** | **学 号：** | **1507084212** |
| **学 院:** | **计算机与控制学院** | | |
| **专 业:** | **网络工程** | | |
| **题 目:** | **宿舍管理系统** | | |
|  | **——后勤主管增删查模块** | | |
| **指导教师：** | **井超** | | |

2017年 6 月16日

目 录

[1系统设计 1](#_Toc485374149)

[1.1设计目标 1](#_Toc485374150)

[1.1.1设计过程 1](#_Toc485374151)

[1.2需求描述 1](#_Toc485374152)

[1.2.1信息要求 2](#_Toc485374153)

[1.2.2处理要求 2](#_Toc485374154)

[1.2.3安全性与完整性要求 2](#_Toc485374155)

[1.3数据字典 3](#_Toc485374156)

[1.4开发和运行环境选择 5](#_Toc485374157)

[2数据库设计 6](#_Toc485374158)

[2.1数据库概念设计 6](#_Toc485374159)

[2.2数据库逻辑设计 9](#_Toc485374160)

[2.3数据库实施 10](#_Toc485374161)

[2.3.1 公寓表的建立 10](#_Toc485374162)

[2.3.2 职工表的建立 11](#_Toc485374163)

[2.3.3 负责人表的建立 11](#_Toc485374164)

[2.3.4宿舍表的建立 11](#_Toc485374165)

[2.3.5学生表的建立 12](#_Toc485374166)

[2.3.6住宿表的建立 13](#_Toc485374167)

[2.3.7舍长表的建立 13](#_Toc485374168)

[2.3.8意见表的建立 13](#_Toc485374169)

[2.3.9后勤主管表的建立 14](#_Toc485374170)

[2.3.10公寓视图的建立 14](#_Toc485374171)

[3宿舍管理系统（后勤主管增删查模块及）详细设计 16](#_Toc485374172)

[3.1 StackFrame 16](#_Toc485374173)

[3.2 DBservice 18](#_Toc485374174)

[3.3 UserService 20](#_Toc485374175)

[3.4后勤主管的增删查模块 21](#_Toc485374176)

[3.4.1公寓标签页 22](#_Toc485374177)

[3.4.2宿舍标签页 30](#_Toc485374178)

[3.4.3职工标签页 35](#_Toc485374179)

[3.4.4学生标签页 37](#_Toc485374180)

[3.4工具函数 40](#_Toc485374181)

[4系统测试 42](#_Toc485374182)

[5个人体会 55](#_Toc485374183)

# 1 系统设计

## 设计目标

在学生学习了数据库原理和SQL Server 2008数据库管理系统后，采用Java编程语言开发工具，设计并实现信息管理系统。

培养学生数据库技术的综合应用能力，通过设计开发一个小型的数据库管理系统，将原理与应用相结合，锻炼学生实际问题的分析、设计与编程能力。

### 设计过程

采用小组设计总体框图、数据库结构和功能结构图。

## 1.2需求描述

本系统的最后用户为后勤主管，宿舍楼管理员，宿舍楼内的学生。

系统功能如图1.1所示：



图1.1 宿舍管理系统功能图

根据我们日常生活中的经验，和对实际情况的调查，得出用户的下列要求：

### 1.2.1信息要求

后勤主管能查询到所有公寓的所有相关信息，包括公寓号、公寓名、公寓地点、最大容纳学生数及已住人数及负责人信息；能查到所有宿舍的信息，包括宿舍的公寓号、宿舍号、宿舍内的床位数和已住学生人数及舍长的学号；能查到所有职工的信息，包括工号、姓名、性别、年龄、联系方式、管理公寓号、家庭地址、工龄等信息；能查到所有学生的信息，包括学号、姓名、性别、联系方式、学院、专业、所住公寓号、宿舍号、床位号及家庭地址。

宿舍管理员能查询到住在自己所管公寓的所有学生的基本信息，包括宿舍号、床号、学号、姓名、联系方式，能看到自己所管公寓的意见反馈栏的反馈信息，包括日期和内容，能查看自己的基本信息。

学生可以查看和管理自己的基本信息，能看到自己所住公寓的负责人信息，包括工号、姓名、联系方式。

### 1.2.2处理要求

后勤主管能管理所有的公寓信息，所有的宿舍信息，所有的职工信息和所有学生信息，包括增、删、查、改操作。

宿舍管理员能管理本宿舍内的住宿信息，对某宿舍某床位学生的添加和删除；能管理自己的个人信息，比如密码的修改和一些信息的修改。

学生能修改自己的个人信息和密码、能对宿舍的管理提出意见。

### 1.2.3安全性与完整性要求

系统对用户进行身份验证，保证相应的身份进入相应的子系统。

学生仅能查看自己所住公寓负责人的信息。

宿舍管理员仅对自己公寓的住宿情况进行管理，并不能删除或添加学生。

当公寓负责人被删除时，公寓的相关信息不会被删除；宿舍长被删除时，宿舍的相关信息不会被删除。

当公寓信息更改时，所有的牵扯到此公寓的信息都会随着更改。

当往宿舍表里添加信息时，会自动向住宿表中添加最大床位数个元组，并把所住学生信息初始为空，直到有学生住进来时。当某个学生添加进来时，会把学生信息自动添加到对应的公寓、宿舍、床位处。当学生被删除时，它之前所住的床位将被设置为空。

当某公寓负责人从职工表中删除时，会把公寓的负责人信息设置为空，直到设置一个新的负责人工号。

## 1.3数据字典

通过对收集的数据进行分析，得出了以下数据字典：

表1.1 公寓表（apartment）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 公寓号 | ApartNo | VARCHAR | 12 | Not | 主码 |
| 公寓名 | ApartName | VARCHAR | 10 | Not | 唯一 |
| 地址 | ApartLocation | VARCHAR | 30 | Not |  |
| 层数 | ApartFloor | SMALLINT | 2 | Not |  |
| 可容纳人数 | ApartmMax | SMALLINT | 2 | Yes |  |
| 当前人数 | ApartIn | SMALLINT | 2 | Yse |  |

表1.2 宿舍表（dorm）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 公寓号 | ApartNo | VARCHAR | 12 | Not | 主码、外码 |
| 宿舍号 | ApartNo | VARCHAR | 12 | Not | 主码 |
| 床位数 | DormBed | SMALLINT | 2 | Yes |  |
| 宿舍人数 | DormIn | SMALLINT | 2 | Yes |  |

表1.3 学生表（student）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | | 备注 |
| 学号 | SNo | VARCHAR | 12 | Not | 主码 | |
| 姓名 | SName | VARCHAR | 10 | Not | 唯一 | |
| 性别 | SSex | VARCHAR | 2 | Yes | m、f | |
| 联系方式 | STel | VARCHAR | 11 | Not |  | |
| 学院 | SAcademic | VARCHAR | 20 | Yes |  | |
| 专业 | SDept | VARCHAR | 20 | Yes |  | |
| 公寓号 | ApartNo | VARCHAR | 12 | Yes | 外码 | |
| 宿舍号 | DormNo | VARCHAR | 12 | Yes | 外码 | |
| 床号 | BedNo | SMALLINT | 2 | Yes | 外码 | |
| 密码 | password | VARCHAR | 16 | Not | 默认值为‘000000’ | |
| 家庭地址 | Saddress | VARCHAR | 50 | Yes |  | |

表1.4 宿舍管管理员表（adminstrator）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 工号 | AdminNo | VARCHAR | 12 | Not | 主码 |
| 姓名 | AdminName | VARCHAR | 10 | Not | 唯一 |
| 性别 | AdminSex | VARCHAR | 2 | Yes | 男、女 |
| 年龄 | AdminAge | SMALLINT | 2 | Not |  |
| 联系方式 | AdminTel | VARCHAR | 11 | Not |  |
| 公寓号 | ApartNo | VARCHAR | 12 | Not | 外码 |
| 家庭地址 | AdminAddr | VARCHAR | 50 | Yes |  |
| 工龄 | AdminWorkYear | SMALLINT | 2 | Yes |  |
| 密码 | password | VARCHAR | 16 | Not | 默认值‘123456’ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 公寓号 | Apartno | 字符 | 12 | Not | 主码 |
| 宿舍号 | DormNo | 字符 | 12 | Not | 主码 |
| 床号 | bedNo | 短整型 | 2 | Not | 主码 |
| 学号 | SNo | 字符 | 12 | Yes | 外码 |

表1.5 住宿表（live）

表1.6 寝室长表（dormmanager）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 公寓号 | ApartNo | VARCHAR | 12 | Not | 主码、外码 |
| 宿舍号 | DormNo | VARCHAR | 12 | Not | 主码、外码 |
| 寝室长床号 | BedNo | SMALLINT | 2 | Yes | 默认值为‘1’、外码 |

表1.7 公寓负责人表（apartmanager）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 公寓楼 | ApartNo | VARCHAR | 12 | Not | 主码、外码 |
| 负责人 | AdminNo | VARCHAR | 12 | Yes | 外码 |

表1.8 意见反馈表（advice）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 序号 | id | INT | 11 | Not | 主码 |
| 公寓号 | apartNo | VARCHAR | 12 | Yes | 外码 |
| 意见 | Advice | VARCHAR | 200 | Yes |  |
| 日期 | date | DATE |  | Yes |  |

表1.9 后勤主管账户表（supervisor）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 中文含义 | 字段名称 | 数据类型 | 长度 | 是否为空 | 备注 |
| 账号 | id | VARCHAR | 12 | Not | 主码、默认值为‘nucnuc’ |
| 密码 | password | VARCHAR | 12 | Yse | 默认值为‘nucnuc’ |

## 1.4开发和运行环境选择

开发工具：InteLLiJ IDEA 2017.1.4、MySql Workbench

前台语言：JAVA、Kotlin

后台数据库：mysql-5.7.14-winx64

运行环境：Windows2000及更高的操作系统

# 2数据库设计

本系统的业务描述如下：

⑴学生对自己信息的管理和对宿舍管理的建议

⑵宿舍管理员对公寓的管理功能

⑶后勤主管对各个公寓的管理

根据宿舍管理设计选题的业务情况，该系统分为学生登录，宿舍管理员登录和后勤主管三大系统模块。

㈠登录认证/退出功能

登录认证，根据用户身份，把权限作为变量传给相应的页面

㈡学生子系统

①对自己个人信息的修改

②宿舍管理员信息的查看

③对宿舍管理的建议

㈢宿舍管理员子系统

①查看该公寓学生的信息，比如联系方式之类等

②该公寓学生住宿信息的增删查

③查看自己所管公寓的意见反馈

④对自己个人信息的修改

㈣后勤主管子系统

①各公寓及其楼内宿舍信息的维护

②所有学生信息的维护

③所有宿管信息的维护

## 2.1数据库概念设计

下面用E-R图来表示宿舍管理系统的概念模型。

宿舍管理涉及以下几个实体：

⑴公寓：属性有公寓号、公寓名、地点、层数、可容纳人数、已住人数、负责人

⑵宿舍：属性有公寓号、宿舍号、床位数、已住人数、宿舍长

⑶宿舍管理员：属性有工号、姓名、性别、年龄、联系方式、管理公寓号、家庭地址、工龄、密码

⑷学生：属性有学号、姓名、性别、联系方式、学院、专业、所住公寓号、宿舍号、床位号、家庭地址、密码

⑸后勤主管：属性有账号和密码

⑹意见：序号、公寓号、内容、时间

这些实体之间的联系如下：

⑴一个公寓可以被多个宿管管理，一个宿管只能管理一个公寓，因此公寓和宿管是一对多的联系。

⑵一个公寓可以住多个学生，一个学生只能住在一个公寓，因此公寓和学生是一对多的联系。

⑶一个公寓可以有多个宿舍，一个宿舍只能属于一个公寓，因此公寓和宿舍是一对多的联系。

⑷一个宿管可以管理多个宿舍及住在公寓学生，一个学生被多个宿管管理，因此宿管和学生是多对多的联系。

⑸一个后勤主管管理多个公寓的信息，只有一个后勤主管，因此后勤主管和公寓是一对多的联系。

⑹一个宿舍可以住多个学生，一个学生只能住在一个宿舍，因此宿舍和学生是一对多的联系。

⑺一个学生可以给自己住的公寓反馈意见，一个公寓可以接受多个学生反馈的意见。系统E-R如图2.7所示。

公寓实体属性图如图2.1所示：



图2.1 公寓实体属性图

宿舍实体属性图如图2.2所示：



图2.2 宿舍实体属性图

宿舍管理员实体属性图如图2.3所示：



图2.3 宿舍管理员实体属性图

学生实体属性图如图2.4所示：



图2.4 学生实体属性图

后勤主管实体属性图如图2.5所示：



图2.5 后勤主管实体属性图

意见反馈属性图如图2.6所示：



图2.6 意见实体属性图



图2.7 宿舍管理系统E-R图

## 2.2数据库逻辑设计

把图2.7的E-R图转为关系模型为：

公寓（公寓号，公寓名，地点，楼层数，可容纳人数，学生数）

负责人（公寓号，负责人工号）

宿舍（公寓号，宿舍号，床位数，已住人数）

宿管（工号，姓名，性别，年龄，联系方式，管理公寓，家庭地址，工龄，密码）

学生（学号，姓名，性别，联系方式，学院，专业，公寓号，宿舍号，床位号，家庭地址，密码）

住宿（公寓号，宿舍号，床位号，学号）

宿舍长（公寓号，宿舍号，床位号）

后勤主管（账号，密码）

建议反馈（序号，公寓号，内容，日期）

## 2.3数据库实施

## 2.3.1 公寓表的建立

use apartments;

create table Apartment(

ApartNo varchar(12) not null primary key,

ApartName varchar(10) not null unique,

ApartLocation varchar(30),

ApartFloor smallint,

ApartMax smallint,

ApartIn smallint);

/\*当公寓插入一条元组时，会自动向apartmanager表插入一条元组，公寓号等于这条公寓号，职工号赋值为空\*/

CREATE DEFINER=`root`@`localhost`

TRIGGER `apartments`.`apartment\_AFTER\_INSERT`

AFTER INSERT ON `apartment`

FOR EACH ROW

BEGIN

insert into apartmanager(ApartNo)

values(new.ApartNo);

END

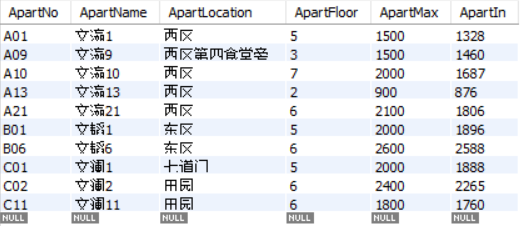


图2.8 基本表apartment

## 2.3.2 职工表的建立

create table Administartor(

AdminNo varchar(12) not null primary key,

AdminName varchar(10) not null unique,

AdminSex varchar(2) check(AdminSex in ('m','f')),

AdminAge smallint,

AdminTel varchar(11) not null,

ApartNo varchar(12),

AdminAddr varchar(50),

AdminWorkYead smallint,

password varchar(16) not null default '123456',

foreign key(ApartNo) references apartment(ApartNo));

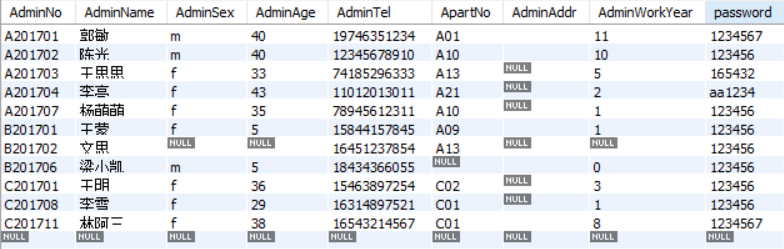


图2.9 基本表adminstrator

## 2.3.3 负责人表的建立

create table ApartManager(

ApartNo varchar(12) ,

AdminNo varchar(12) ,

foreign key(ApartNo) references Apartment(ApartNo));

alter table apartmanager

add constraint foreign\_Admin

foreign key(AdminNo) references administartor(AdminNo);

图2.10 基本表apartmanager

## 2.3.4宿舍表的建立

create table Dorm(

DormNo varchar(12),

ApartNo varchar(12),

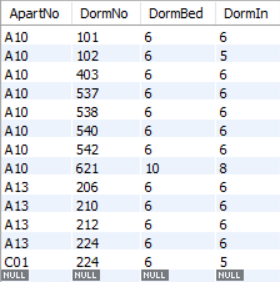
DormBed smallint,

DormIn smallint,

primary key(DormNo,ApartNo),

foreign key(ApartNo) references apartment(ApartNo));

/\*当插入一个元组到寝室表时，会自动向live添加他最大床位数个床位\*/

CREATE DEFINER=`root`@`localhost`

TRIGGER `apartments`.`dorm\_AFTER\_INSERT`

AFTER INSERT ON `dorm`

FOR EACH ROW

BEGIN

set @i=0;

lp1 : loop

set @i=@i+1;

insert into live(apartNo,dormNo,bedNo)

values(new.apartNo,new.dormNo,@i);

if @i>=new.DormBed

then 图2.11 基本表dorm

LEAVE lp1;

end if;

end loop;

insert into dormmanager(apartNo,DormNo) values(new.apartNo,new.dormNo);

END

## 2.3.5学生表的建立

create table Student(

SNo varchar(12) not null primary key,

SName varchar(10) not null unique,

SSex varchar(2) check ( SSex in ('m' and 'f')),

STel varchar(11) not null,

SAcademic varchar(20),

SDept varchar(20),

ApartNo varchar(12),

DormNo varchar(12),

BedNo smallint,

password varchar(16) not null default '000000',

Saddress varchar(50),

foreign key(ApartNo,DormNo) references Dorm(ApartNo,DormNo));

/\*当插入一个学生时，更新live表，把对应的床位号传入这个学生的学号\*/

CREATE DEFINER=`root`@`localhost`

TRIGGER `apartments`.`student\_AFTER\_INSERT`

AFTER INSERT ON `student`

FOR EACH ROW

BEGIN

update live set sno=new.sno

where apartNo=new.apartNo and dormno=new.dormNo and bedNo=new.bedNo;

END

/\*当一个学生被删除时，把live表原先住的床学号设置为空\*/

CREATE DEFINER=`root`@`localhost` TRIGGER `apartments`.`student\_AFTER\_DELETE`

AFTER DELETE ON `student`

FOR EACH ROW

BEGIN

update live set sno=''

where SNo=old.SNo;

END

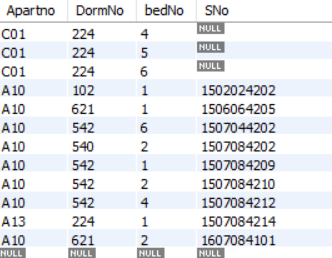


图2.12 基本表student

## 2.3.6住宿表的建立

create table live(

Apartno varchar(12),

DormNo varchar(12),

bedNo smallint check(bedNo >= 1 AND bedNo <= 10),

SNo varchar(12),

primary key(ApartNo,DormNo,bedNo),

foreign key(ApartNo,DormNo)

references Dorm(ApartNo,DormNo),

foreign key(SNo) references Student(Sno));

图2.13 基本表live

## 2.3.7舍长表的建立

create table DormManager(

ApartNo varchar(12),

DormNo varchar(12),

BedNo smallint,

primary key(ApartNo,DormNo),

foreign key(ApartNo, DormNo,BedNo)

references live(ApartNo,DormNo,BedNo));

图2.14 基本表dormmanager

### 2.3.8意见表的建立

create table Advice(

id int primary key,

apartNo varchar(12),

advice varchar(200),

date DATE,

foreign key(apartNo)

references apartment(apartNo));

图2.3.8 基本表advice

/\*每次插入前找到id最大值，把新元组id赋值为最大值+1\*/

CREATE DEFINER=`root`@`localhost`

TRIGGER `apartments`.`advice\_BEFORE\_INSERT`

BEFORE INSERT ON `advice`

FOR EACH ROW

BEGIN

set @i=(SELECT max(id) from advice);

set new.id=@i+1;

END

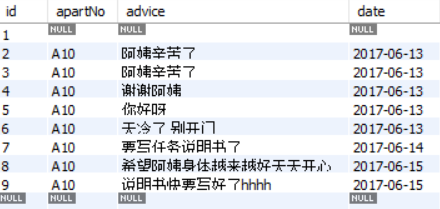


图2.15 基本表advice

### 2.3.9后勤主管表的建立

create table supervisor(

id varchar(12) primary key default 'nucnuc',

password varchar(16) not null default 'nucnuc');



图2.16 基本表supervisor

### 2.3.10公寓视图的建立

数据库管理员在每次公寓新建时会创建一个名为该公寓号的视图来查看该公寓内的学生信息：

CREATE

ALGORITHM = UNDEFINED

DEFINER = `root`@`localhost`

SQL SECURITY DEFINER

VIEW `a01` AS

SELECT

`dorm`.`DormNo` AS `dormNo`,

`live`.`bedNo` AS `bedNo`,

`live`.`SNo` AS `Sno`,

`student`.`SName` AS `Sname`,

`student`.`STel` AS `STel`

FROM

((`dorm`

JOIN `live`)

JOIN `student`)

WHERE

((`dorm`.`ApartNo` = `live`.`Apartno`)

AND (`dorm`.`DormNo` = `live`.`DormNo`)

AND (`live`.`SNo` = `student`.`SNo`)

AND (`dorm`.`ApartNo` = 'A01'))



图2.17 公寓A10住宿学生的信息

# 3宿舍管理系统（后勤主管增删查模块及）详细设计

## 3.1 StackFrame



图3.1 Frame生命周期

使用栈先进后出的存储结构来管理整个系统的Frame状态，使用Kotlin语言具体实现代码如下：**open abstract class** StackFrame : JFrame{ *//构造函数 The primary constructor cannot contain any code* **constructor**(title : String) : **super**(title)

*// companion对象列表 相当于static对象* **companion object**{  
 **private val sFrameStacks**: Stack<StackFrame> = Stack<StackFrame>()  
 }  
 */\*\*  
 \* 对象被实例化时执行 相当于Java构造函数*

*\* Initialization code can be placed in initializer blocks  
 \*/* **init** {  
 **sFrameStacks**.add(**this**)*//进栈* onStart()  
 **sFrameStacks**.*forEach* **{ it**.onOtherStart(**this**) **}** }  
 */\*\*  
 \* 重写dispose()方法。关闭本界面，并出栈  
 \*/* **override fun** dispose(){  
 **sFrameStacks**.pop()  
 onDispose()  
 **sFrameStacks**.*forEach* **{ it**.onOtherDispose(**this**) **}***//遍历通知关闭* **super**.dispose()  
 }  
 **open fun** onStart(){}  
 **open fun** onOtherStart(stackFrame: StackFrame){ }  
 */\*\*  
 \* 当自己关闭时调用  
 \*/* **open fun** onDispose(){}  
 */\*\*  
 \* 其他Frame关闭时调用  
 \*/* **open fun** onOtherDispose(stackFrame: StackFrame){}  
 **override fun** hide() {  
 **super**.hide()  
 **sFrameStacks**.*forEach* **{ it**.onOtherHide(**it**) **}** }  
 **open fun** onOtherHide(stackFrame: StackFrame){}  
 **override fun** show() {  
 **super**.show()  
 **sFrameStacks**.*forEach* **{ it**.onOtherShow(**it**) **}** }  
 **open fun** onOtherShow(stackFrame: StackFrame){}  
}

## 3.2 DBservice

使用DBService类把系统每次和数据库连接的方法封装起来。这个类在第一次被调用时实例化，有且只有一个实例，因为DBServiceHolder是私有的，所以系统只能通过getInstance()方法获得这个类的实例，返回的是一个静态实例所以也避免了同步带来的性能影响。具体实现代码如下：**public class** DBService {  
 **private static final** String ***MY\_SQL\_DRIVER*** = **"com.mysql.jdbc.Driver"**;  
 **private static final** String ***MYSQL*** = **"mysql"**;  
 **private static final** String ***JDBC*** = **"jdbc"**;  
 **private** Connection **mConnection** = **null**;  
 **private** Statement **mStatement** = **null**;  
 */\*\*  
 \* 获得完成的数据库URL  
 \** ***@return*** *\*/* **public** String getURL() {  
 **return *JDBC*** + **":"** + ***MYSQL*** + **"://"** + Config.***DBMS\_IP*** +**":"** +Config.***DBMS\_PORT*** + **"/"** + Config.***DB\_NAME***;  
 }  
 */\*\*  
 \*使用静态内部类来实现单例模式，避免同步带来的性能影响  
 \*/* **private final static class** DBServiceHolder{  
 **private static final** DBService ***sInstance*** = **new** DBService();  
 }  
 **public static** DBService getInstance(){  
 **return** DBServiceHolder.***sInstance***;  
 }

*/\*\*  
 \* 连接数据库  
 \** ***@throws*** *Exception  
 \*/* **public void** connect() **throws** Exception {  
 **try** {  
 Class.*forName*(***MY\_SQL\_DRIVER***);  
 **mConnection** = DriverManager.*getConnection*(getURL(),Config.***DBMS\_USERNAME***,Config.***DBMS\_PASSWORD***);  
 **mStatement** = **mConnection**.createStatement();  
 } **catch** (ClassNotFoundException e) {  
 **throw new** Exception(**"数据库驱动异常，可能是没有导入数据库驱动："** + e.getMessage());  
 } **catch** (SQLException e){  
 **throw new** Exception(**"数据库连接异常："** + e.getMessage());  
 } }  
 */\*\*\  
 \* 关闭数据库  
 \** ***@throws*** *Exception  
 \*/* **public void** close() **throws** Exception{  
 **try** {  
 **if** (**mConnection** != **null**){  
 **mConnection**.close();  
 **mConnection** = **null**;  
 }  
 } **catch** (SQLException e) {  
 **throw new** Exception(**"关闭数据库异常"** + e.getMessage());  
 }}  
 **public** Statement getStatement() {  
 **return mStatement**;  
 }}

## 3.3 UserService

每个用户使用本系统的时候只有一个身份类型，要么是学生，要么是宿舍管理员，要么是主管，每种身份对应不同的权限和操作，每个用户都有一个账户，所以这里依然使用单例模式创建一个UserService，当系统需要用到当前用户信息时，调用getInstance()方法获取当前用户实例。具体实现代码如下：

**public class** UserService {  
 **public static final int *TYPE\_NONE*** = 0x00;  
 **public static final int *TYPE\_STUDENT*** = 0x01;  
 **public static final int *TYPE\_ADMIN*** = 0x02;  
 **public static final int *TYPE\_SUPERVISOR*** = 0x03;  
 */\*\*  
 \* 使用单例模式创建类的实例。使得这个类的实例仅初始化一次并且有且仅有一个  
 \*/* **private static final class** UserServiceHolder{  
 **private static** UserService *sInstance* = **new** UserService();  
 }  
 **public static** UserService getInstance(){  
 **return** UserServiceHolder.*sInstance*;  
 }  
 **public** User **currentUser**; *//当前登陆的用户* **private int userType** = ***TYPE\_NONE***;  
 **public int** getUserType(){  
 **return userType**;  
 }  
 **public void** setUserType(**int** userType){  
 **this**.**userType** = userType;  
 }  
 */\*\*  
 \* 获取当前用户，返回一个账号  
 \** ***@return*** *\*/* **public** User getCurrentUser() {  
 **return currentUser**;  
 }  
 */\*\*  
 \* 登陆：设置当前用户  
 \** ***@param currentUser*** *\*/* **public void** signIn(User currentUser) {  
 **this**.**currentUser** = currentUser;  
 }}

## 3.4后勤主管的增删查模块

后勤主管界面分为四个标签页：公寓信息、宿舍信息、职工信息、学生信息，

每个界面有添加、修改、删除、查询功能。界面中有一个JTable显示数据库中所对应数据的简略信息，每当主管进入管理系统后，SupervisorBasicFrame初始化控件时，填充表中的数据为当前数据库的所有数据。当某行被选中时，JTextArea显示详细信息。可以通过输入信息查询数据，选中某行并点击修改或删除会对数据库进行相关处理。点添加按钮，会新建一个JFrame通过录入相关信息插入到数据库中。



图3.2 基本信息管理界面流程图

# 3.4.1公寓标签页

公寓实体类成员变量分别为公寓号、公寓名、地点、楼层数、可容纳人数、已住人数和负责人工号，实现代码如下：**class** Apartment (**var no** : String,  
 **var name** : String,  
 **var location** : String?,  
 **var floor** : Int?,  
 **var max** : Int?,  
 **var In** : Int?,  
 **var managerNo** : String?){  
 **override fun** toString(): String {  
 **return "公寓号："** + **no** +**"\n公寓名："**+ **name** + **"\n地点："** + **location** +  
 **"\n层数："**+ **floor**+**"\n可容纳人数："** + **max**+**"\n现住人数："** + **In**+**"\n负责人："** + **managerNo** }  
}



图3.3 公寓信息管理标签页

*/\*\*  
 \* 填充apartment表  
 \*/***fun** JTable.fillTable(res : Vector<Apartment>){  
 **var** tableModel : DefaultTableModel  
 tableModel = **this**.*model* **as** DefaultTableModel *//table中的元素转为Vector管理* tableModel.*rowCount* = 0*//清空表中的行数*  
 *//foreach遍历ApartmentVector向表中添加数据*  
 **for** ( r : Apartment **in** res){  
 **val** str = Array<String>(3,**{""}** )  
 str[0] = r.**no** str[1] = r.**name** str[2] = r.**managerNo**+**""** *//添加数据到apartTable* tableModel.addRow(str)  
 }  
 *//更新表格* **this**.invalidate()  
}  
*/\*\*  
 \*取得用户所选行的详细信息  
 \*/***fun** JTable.getApartInfo() : Apartment {  
 **var** tableModel = **this**.*model* **as** DefaultTableModel  
 **return** SelectApartments().search(tableModel.getValueAt(*selectedRow*,0) **as** String)  
}

*/\*\*  
 \* 选中多个时 获得所选行的第一个单元格  
 \*/***fun** JTable.getSelectRowsValue() : ArrayList<String> {  
 **var** selectRows = **this**.*selectedRows*.**size  
 var** tableModel = **this**.*model* **as** DefaultTableModel  
 **var** list = ArrayList<String>()  
 **if** (selectRows >= 1){  
 **val** array = **this**.*selectedRows//取得所选单元行* **for** (i **in** array){  
 list.add(tableModel.getValueAt(i,0).toString())  
 }  
 }  
 **return** list  
}

*//查询宿舍时get所选行的前两列的元素，用map存储***fun** JTable.getSelectMap() : Map<String,String> {  
 **var** selectRows = **this**.*selectedRows*.**size  
 var** tableModel = **this**.*model* **as** DefaultTableModel  
 **var** map = *hashMapOf*<String,String>()  
 **if** (selectRows >= 1){  
 **val** array = **this**.*selectedRows//取得所选单元行* **for** (i **in** array){  
 map.put(tableModel.getValueAt(i,0) **as** String,tableModel.getValueAt(i,1) **as** String)  
 }  
 }  
 **return** map  
}

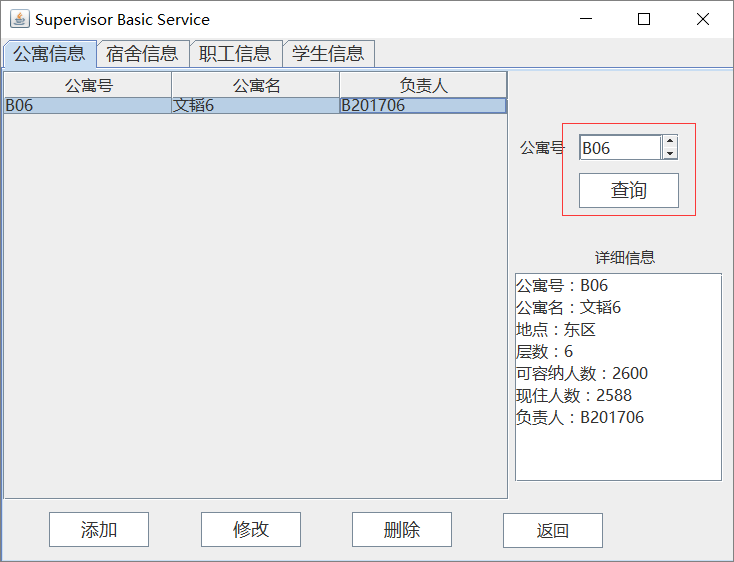


图3.4 公寓信息查询演示

*/\*\*  
 \* 公寓号查询 通过继承接口方式实现查询按钮的点击事件  
 \** ***@param evt*** *\*/***private void** apartSelectActionPerformed(java.awt.event.ActionEvent evt) {  
 String no = **apartTextFiled**.getText();*//得到输入的公寓号* **if** (no.isEmpty()){  
 ApartTableKt.*fillTable*(**apartTable**,**new** SelectApartments().searchAll());*//如果textTiled为空，向公寓表填充所有数据* }**else** {  
 Vector<Apartment> apartments = **new** Vector<>();  
 apartments.add(**new** SelectApartments().search(no));  
 ApartTableKt.*fillTable*(**apartTable**,apartments);  
 }  
}

*//公寓查询类*

**public class** SelectApartments {  
 **private** DBService **mDBMS** = **new** DBService().*getInstance*();  
 **private** Statement **mStatement** = **null**;  
 **private** Vector<Apartment> **apartments** = **new** Vector<>();  
 **private** Apartment **apartment** = **null**;  
 *//构造方法 连接数据库 实例化Statement*  
 **public** SelectApartments(){  
 **try** {  
 **mDBMS**.connect();  
 **mStatement** = **mDBMS**.getStatement();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 *//查询所有的公寓信息，连接公寓表和负责人表查询宿舍及其负责人信息*  
 **public** Vector<Apartment> searchAll() {  
 String sql = **"select apartment.ApartNo,ApartName,ApartLocation,ApartFloor,ApartMax,ApartIn,AdminNO "** +  
 **"from apartment,apartmanager where apartment.ApartNo=apartmanager.ApartNo"**;  
 **try** {  
 **apartments**=dealResultSet(**mStatement**.executeQuery(sql));  
 **mDBMS**.close();*//每个函数对应一个出口，否则connection过多将会出现connection too much异常*  
 }**catch** (Exception e){  
 e.printStackTrace();  
 }  
 **return apartments**;  
 }  
 **public** Vector<Apartment> dealResultSet(ResultSet resultSet){  
 **if** (resultSet != **null**){  
 **try** {  
 **while** (resultSet.next()){  
 **apartments**.add(**new** Apartment(resultSet.getString(1),  
 resultSet.getString(2),  
 resultSet.getString(3),  
 resultSet.getInt(4),  
 resultSet.getInt(5),  
 resultSet.getInt(6),  
 resultSet.getString(7)));  
 }  
 }**catch** (SQLException e){  
 e.printStackTrace();  
 }  
 }  
 **return apartments**;  
 }  
 *//通过公寓号查询某公寓的信息*   
 **public** Apartment search(String no){  
 String sql = **"select apartment.ApartNo,ApartName,ApartLocation,ApartFloor,ApartMax,ApartIn,adminNO "** +  
 **"from apartment,apartmanager where apartment.ApartNo=apartmanager.ApartNo and apartment.ApartNo LIKE'"** + no + **"%'"**;  
 **try** {  
 **apartment** = dealResult(**mStatement**.executeQuery(sql));  
 **mDBMS**.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return apartment**;  
 }  
  
 **private** Apartment dealResult(ResultSet resultSet) {  
 **if** (resultSet != **null**){  
 **try** {  
 **if** (resultSet.next()){  
 **apartment** = **new** Apartment(resultSet.getString(1),  
 resultSet.getString(2),  
 resultSet.getString(3),  
 resultSet.getInt(4),  
 resultSet.getInt(5),  
 resultSet.getInt(6),  
 resultSet.getString(7));  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }   
 }  
 **return apartment**;  
 }  
  
}

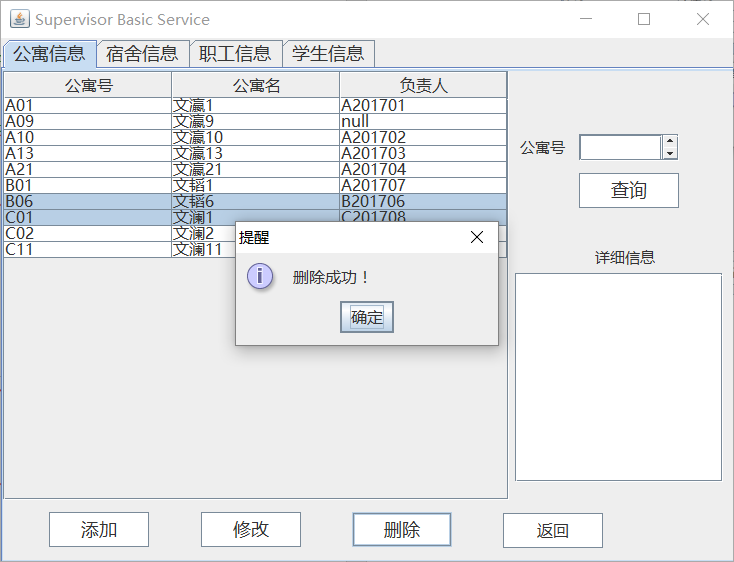


图3.5 公寓信息删除演示

**apartDelete**.addActionListener(**new** ActionListener() {

@Override  
 **public void** actionPerformed(ActionEvent e) {  
 ArrayList<String> arrayList = GetSelectRowsValueKt

.*getSelectRowsValue*(**apartTable**);

**for** (String str : arrayList){  
 System.***out***.println(str);  
 **int** i = **new** DeleteApartment().Delete(str);  
 **if** (i == -1){  
 JOptionPane.*showMessageDialog*(SupervisorBaseServiceFrame.**this**,  
 **"你不能删除这个楼！"**,**"警告"**,JOptionPane.***WARNING\_MESSAGE***);  
 }**else** {  
 JOptionPane.*showMessageDialog*(SupervisorBaseServiceFrame.**this**,  
 **"删除成功！"**,**"提醒"**,JOptionPane.***INFORMATION\_MESSAGE***);  
 ApartTableKt.*fillTable*(**apartTable**,**new** SelectApartments().searchAll());  
 }  
 }  
 }  
});

*//通过公寓号删除公寓信息***public int** Delete(String no){  
 **int** resultCode = 0;  
 String sql = **"delete from apartment where ApartNo='"** + no + **"';"**;  
 **try** {  
 **mStatement**.execute(sql);  
 **mDBMS**.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 resultCode = -1;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **return** resultCode;  
}

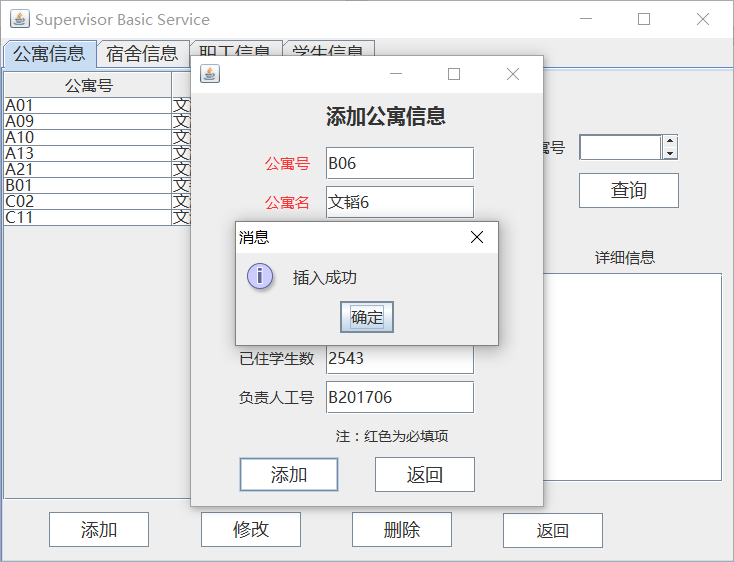


图3.6 公寓信息添加演示

*//添加公寓信息界面的添加按钮的点击事件*

**private void** bt\_InsertActionPerformed(java.awt.event.ActionEvent evt) {  
 Apartment apartment = **new** Apartment(**apartNoTextField**.getText(),  
 **apartNameTextField**.getText(),**apartLocationTextField**.getText(),  
 Integer.*parseInt*(**apartFloorTextField**.getText()),

Integer.*parseInt*(**apartMaxTextField**.getText()),  
 Integer.*parseInt*(**apartInTextField**.getText()),

**adminNoTextField**.getText());  
 **int** flag = **new** InsertApartment().insert(apartment);  
 **if** (flag < 0){  
 JOptionPane.*showMessageDialog*(InsertApartmentInfoFrame.**this**,**"插入失败"**);  
 }**else**{  
 JOptionPane.*showMessageDialog*(InsertApartmentInfoFrame.**this**,**"插入成功"**);  
 **apartNoTextField**.setText(**null**);  
 **apartNameTextField**.setText(**null**);  
 **apartLocationTextField**.setText(**null**);  
 **apartFloorTextField**.setText(**null**);  
 **apartMaxTextField**.setText(**null**);  
 **apartInTextField**.setText(**null**);  
 **adminNoTextField**.setText(**null**); }  
}

*//插入公寓信息。apartment有一个插入触发器，会自动向apartmanager表插入一个新的元组。当负责人不为空时，更新apartmanager表***public int** insert(Apartment apartment){  
 **int** result\_code = 0;  
 String sql1 = **"insert into apartment value('"** +apartment.getNo()+ **"','"** +  
 apartment.getName() + **"','"** +  
 apartment.getLocation() + **"',"** + apartment.getFloor() + **","** + apartment.getMax()  
 + **","** + apartment.getIn() + **");"**;  
 **try** {  
 **mStatement**.execute(sql1);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 result\_code = -1;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **if** (!apartment.getManagerNo().isEmpty()){  
 String sql2 = **"update apartmanager set AdminNo='"** + apartment.getManagerNo()  
 + **"' where ApartNo='"** + apartment.getNo() + **"'"**;  
 **try** {  
 **mStatement**.execute(sql2);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 result\_code = -1;  
 }  
 }  
 **try** {  
 **mDBMS**.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result\_code;  
}

# 3.4.2宿舍标签页

宿舍实体类成员变量为公寓号、宿舍号、床位数、已住人数和宿舍长学号，具体实现代码如下：**class** Dorm (**var apartNo**: String ,  
 **var dormNo** : String ,  
 **var bedNum** : Int?,  
 **var bedIn**: Int?,  
 **var managerNo** : String?){  
 **override fun** toString(): String {  
 **return "公寓号："** + **apartNo** +**"\n宿舍号："**+ **dormNo** + **"\n床位："** + **bedNum** +  
 **"\n已住人数："**+ **bedIn**+**"\n寝室长："** + **managerNo** }  
}



图3.7 宿舍管理标签页

填充宿舍表的方法、获取选中行信息的方法和公寓子标签页类似。

*/\*\*  
 \* 查找所有宿舍的信息  
 \** ***@return*** *\*/***public** Vector<Dorm> searchAllDorms(){  
 String sql = **"SELECT dorm.ApartNo,dorm.DormNo,dorm.DormBed,dorm.DormIn,dormmanager.bedNo\n"** +  
 **"from dorm,dormmanager\n"** +  
 **"where "** +  
 **"dorm.ApartNo=dormmanager.ApartNo\n"** +  
 **"and dorm.dormNo=dormmanager.DormNo"** ;  
 **try** {  
 **dorms** = dealAllResult(**mStatement**.executeQuery(sql));  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 **try** {  
 **mDBMS**.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return dorms**;  
}  
  
**private** Vector<Dorm> dealAllResult(ResultSet resultSet) {  
 **if** (resultSet != **null**){  
 **try** {  
 **while** (resultSet.next()){  
 **dorms**.add(**new** Dorm(resultSet.getString(1),  
 resultSet.getString(2),  
 resultSet.getInt(3),resultSet.getInt(4),  
 resultSet.getString(5)));  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **return dorms**;  
}

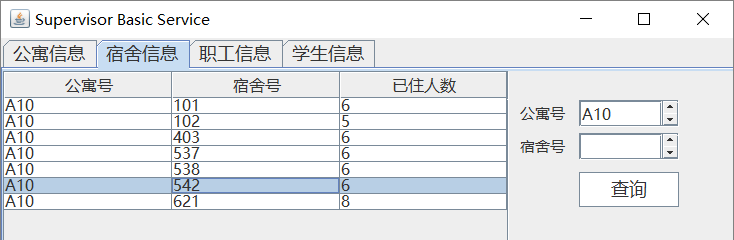
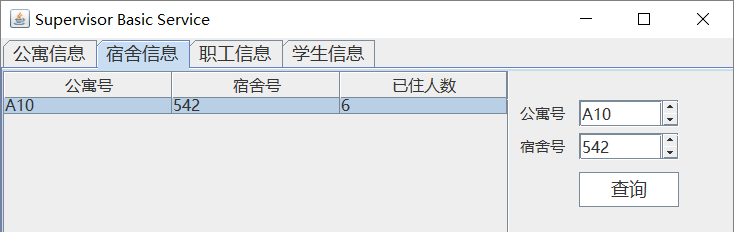
 

图3.8 宿舍查询演示

*//查询按钮点击事件*

**private void** dormSelectActionPerformed(java.awt.event.ActionEvent evt) {  
 String apartNo = **dormApartTextField**.getText();  
 String dormNo = **dormTextField**.getText();  
  
 **if** (apartNo.isEmpty() && dormNo.isEmpty()){  
 DormTableKt.*fillTable*(**dormTable**,**new** SelectDorms().searchAllDorms());  
 }**else if** (!apartNo.isEmpty() && dormNo.isEmpty()) {  
 Vector<Dorm> dorms = (**new** SelectDorms().search(apartNo,**"any(select dormNo from dorm)"**));  
 DormTableKt.*fillTable*(**dormTable**,dorms);  
 }**else if** (!apartNo.isEmpty() && !dormNo.isEmpty()){  
 Vector<Dorm> dorms = (**new** SelectDorms().search(apartNo,**"'"**+dormNo +**"'"**));  
 DormTableKt.*fillTable*(**dormTable**,dorms);  
 }  
}

*/\*\*  
 \* 通过公寓号和宿舍号查找具体的宿舍信息  
 \** ***@param apartNo*** *\** ***@param dormNo*** *\** ***@return*** *\*/***public** Vector<Dorm> search(String apartNo,String dormNo){  
 **try** {  
 String sql =**"SELECT dorm.ApartNo,dorm.DormNo,dorm.DormBed,dorm.DormIn,live.SNo\n"** +  
 **"from dorm,dormmanager,live\n"** +  
 **"where dormmanager.BedNo=live.bedNo \n"** +  
 **"and dormmanager.BedNo=live.bedNo\n"** +  
 **"AND dorm.ApartNo=dormmanager.ApartNo\n"** +  
 **"and dorm.dormNo=dormmanager.DormNo\n"** +  
 **"and dormmanager.ApartNo=live.Apartno\n"** +  
 **"and dormmanager.DormNo=live.DormNo\n"** +  
 **"and live.ApartNo='"**+ apartNo+**"'\n"** +  
 **"and live.dormNo="**+dormNo;  
 **dorms** = dealAllResult(**mStatement**.executeQuery(sql));  
 } **catch** (SQLException e) {}  
 **try** {  
 **mDBMS**.close();  
} **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return dorms**;  
}

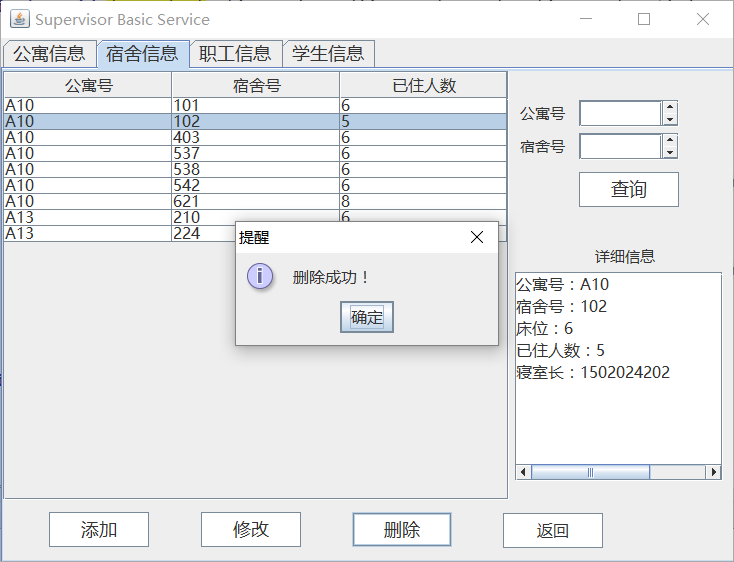


图3.9 宿舍删除演示

**public int** delete(String apartNo,String dormNo){  
 **int** result\_code = 0;  
 String sql = **"delete from dorm where apartNo='"**+ apartNo+**"' and DormNo='"**+dormNo+**"'"**;  
 **try** {  
 **mStatement**.execute(sql);  
 **mDBMS**.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 result\_code=-1;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result\_code;  
}



图3.10 宿舍添加演示

**try** {  
 String sql1 = **"insert into dorm(apartNo,DormNo,DormBed,DormIn) values('"**+dorm.getApartNo()+**"','"**+dorm.getDormNo()  
 +**"',"**+dorm.getBedNum()+**","**+dorm.getBedIn()+**")"**;  
 **mStatement**.execute(sql1);  
} **catch** (SQLException e) {  
 e.printStackTrace();  
 result\_code = -1;  
}  
**if** (!dorm.getManagerNo().isEmpty()){  
 String sql2 = **"update dormmanager set bedNo='"** + dorm.getManagerNo()+ **"' where ApartNo='"** + dorm.getApartNo() + **"' and DormNo='"** + dorm.getDormNo()+ **"'"**;  
 **try** {  
 **mStatement**.execute(sql2);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 result\_code = -1;}  
}

# 3.4.3职工标签页

职工实体类继承自User类，工号赋值为User的no，其他的成员变量分别为姓名、性别、年龄、联系方式、公寓号、家庭地址、工龄，实现代码如下：

**class** Admin(no : String, **var name**:String,  
 **var sex** : String?,**var age** :Int?,  
 **var tel** : String,**var apartNo**:String?,  
 **var address** : String?,  
 **var workAge** : Int?): User(no){  
  
 **override fun** toString(): String {  
 **return "工号："** + **no** +**"\n姓名："** + **name** + **"\n性别："**+ **sex** +**"\n年龄："**+ **age** +**"\n联系方式："**+ **tel** +  
 **"\n公寓号："** + **apartNo**+ **"\n家庭地址："**+ **address** + **"\n工龄："** + **workAge** }  
}

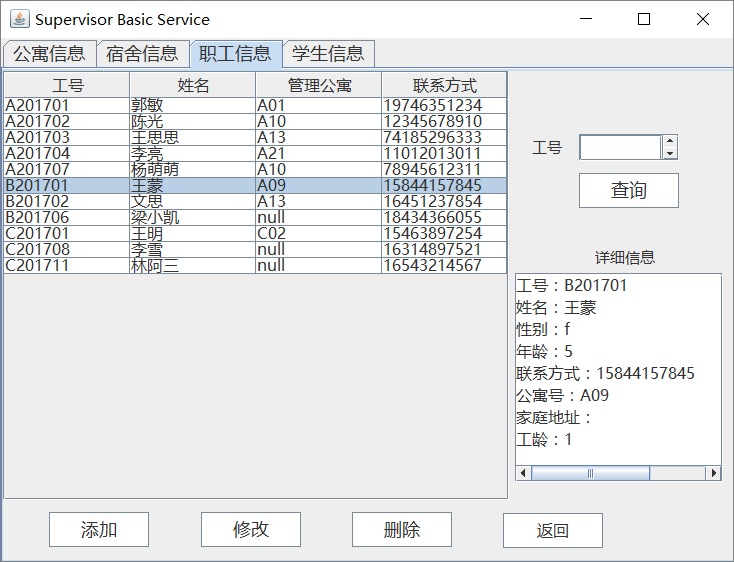


图3.11 职工信息标签页

**public** Vector<Admin> selectAll(){  
 **try** {  
 String sql = **"select \* from administrator"**;  
 **admins** = dealAllResult(**mStatement**.executeQuery(sql));  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return admins**;  
}  
**private** Vector<Admin> dealAllResult(ResultSet resultSet) {  
 **if** (resultSet != **null**){  
 **try** {  
 **while** (resultSet.next()){  
 **admins**.add(**new** Admin(resultSet.getString(1),  
 resultSet.getString(2),  
 resultSet.getString(3),  
 resultSet.getInt(4),  
 resultSet.getString(5),  
 resultSet.getString(6),  
 resultSet.getString(7),  
 resultSet.getInt(8)));  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 } }  
 **return admins**;}

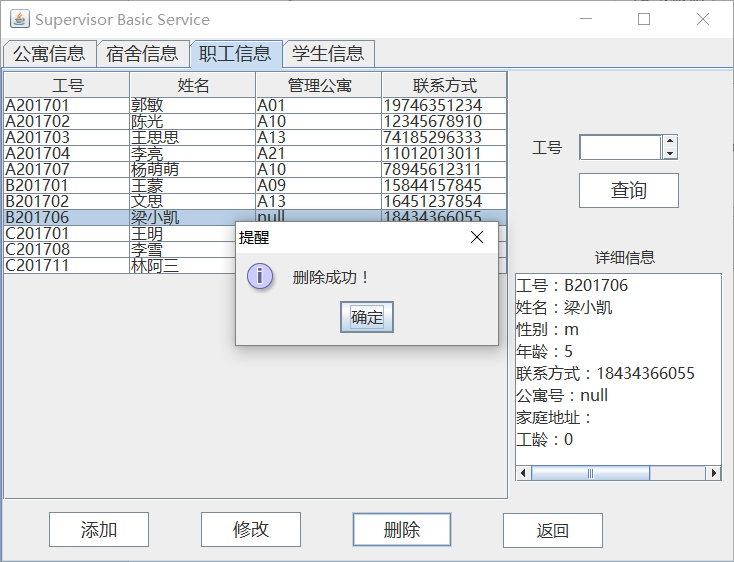


图3.12 职工信息删除演示

**public int** Delete(String no){  
 **int** resultCode = 0;  
 String sql = **"delete from administrator where AdminNo='"** + no + **"';"**;  
 **try** {  
 **mStatement**.execute(sql);  
 **mDBMS**.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 resultCode = -1;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** resultCode;  
}

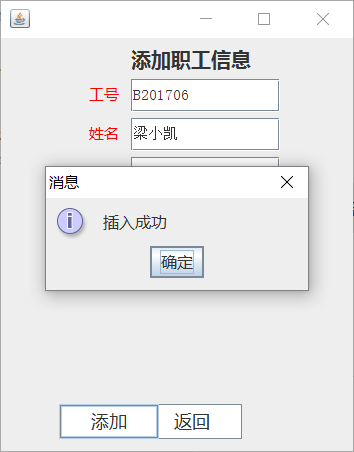


图3.13 职工添加演示

**public int** insert(Admin admin){

**int** result\_code = 0;  
 String sql = **"insert into administrator(AdminNo,AdminName,AdminTel,ApartNo) "** +  
 **"values('"** + admin.getNo() +**"','"** + admin.getName() + **"','"** + admin.getTel() +  
 **"','"**+admin.getApartNo()+**"')"**;  
 **try** {  
 **mStatement**.execute(sql);  
 **mDBMS**.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 result\_code = -1;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result\_code;  
}

# 3.4.4学生标签页

学生实体类继承自User类，学号赋值为User的no，其他的成员变量分别为姓名、性别、联系方式、学院、专业、公寓号、宿舍号、床位号、家庭地址，实现代码如下：

**class** Student(no : String ,**var name** : String,  
 **var sex** : String?,**var tel** : String,  
 **var academic** :String?, **var dept** : String?,  
 **var apartNO**: String?,**var dormNo** : String?,  
 **var bedNo** : Int? , **var address** : String?) : User(no){  
  
 **override fun** toString(): String{  
 **return "学号："** + **no** +**"\n姓名："** + **name** + **"\n性别："**+ **sex** +**"\n联系方式："**+ **tel** +  
 **"\n学院："**+ **academic** +**"\n专业："**+ **dept**+ **"\n公寓号："** + **apartNO**+  
 **"\n宿舍号："** + **dormNo** +**"\n床位："**+ **bedNo** +**"\n家庭地址："**+ **address** }  
}

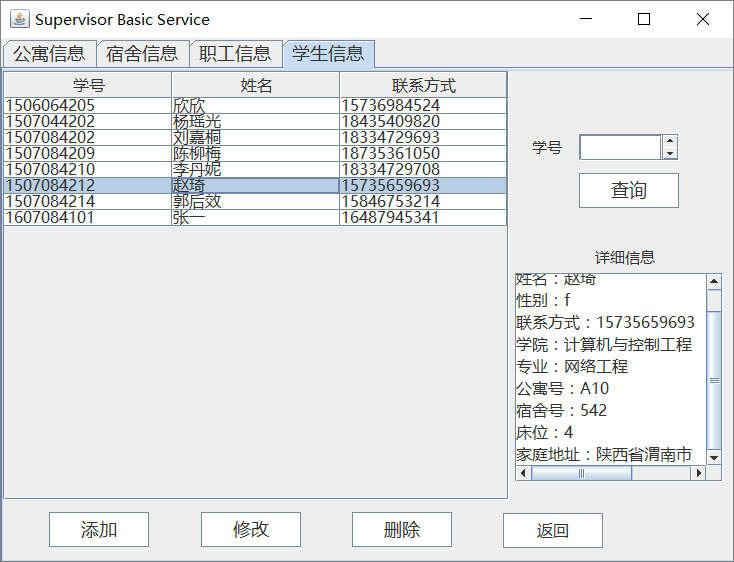


图3.14 学生信息标签页

*/\*\*  
 \* 查询所有学生信息  
 \** ***@return*** *\*/***public** Vector<Student> selectAll(){  
 **try** {  
 String sql = **"select student.SNo,student.SName,student.SSex,student.STel,student.SAcademic"** +**",student.SDept,live.Apartno,live.DormNo,live.bedNo,student.Saddress "** +  
**"from student,live "** +**"where student.sno=live.SNo"**;  
 **students** = dealAllResult(**mStatement**.executeQuery(sql));  
 **mDBMS**.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();}  
 **return students**;}

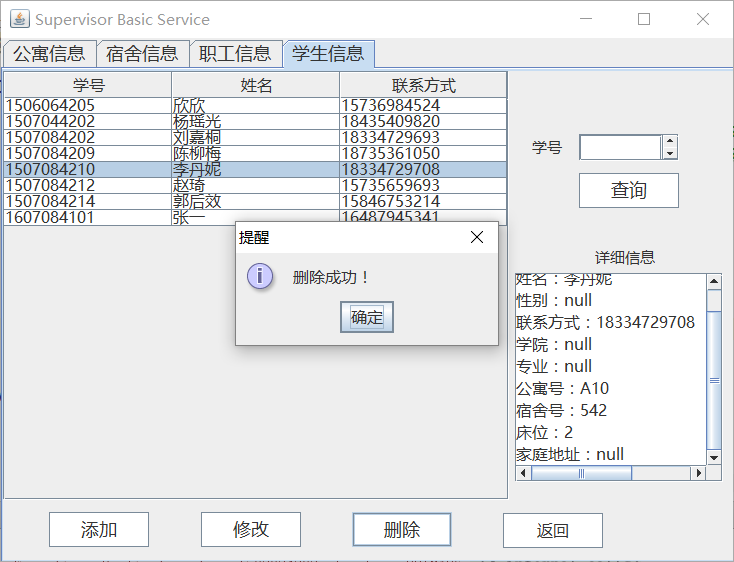


图3.15 学生信息删除演示  
 String sql = **"delete from student where SNo='"** + no + **"';"**;

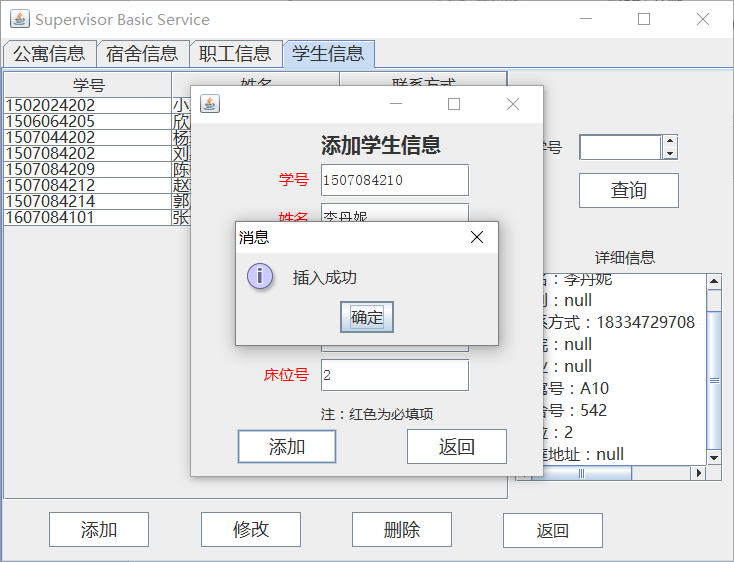


图3.16 学生信息添加演示

**public int** insertStuLive(Student student){  
 **int** result\_code = 0;  
 String sql = **"insert into student(SNo,SName,STel,**

**apartNo,dormNo,bedNo) values('"**+student.getNo()+

**"','"**+student.getName()+**"','"**+student.getTel()+**"','"**

+student.getApartNO()+**"','"**+student.getDormNo()+**"',"** +student.getBedNo()+**");"**;  
 **try** {  
 **mStatement**.execute(sql);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 result\_code = -1;  
 }

**try** {  
 **mDBMS**.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** result\_code;  
}

## 3.4工具函数

Kotlin支持函数式编程，新建一个kt文件，在文件中定义一下几个工具函数：

*/\*\*  
 \* 初始化Frame的基本属性  
 \*/***var** JFrame.*mWidth* : Int  
 **get**() = 750  
 **set**(value) {}  
**var** JFrame.*mHeight* : Int  
 **get**() = 530  
**set**(value) {}  
**fun** JFrame.init(width : Int, height : Int){  
 setSize(width, height)  
 *defaultCloseOperation* = WindowConstants.*EXIT\_ON\_CLOSE* **var** screenDimension = Toolkit.getDefaultToolkit().*screenSize //获得屏幕尺寸* setLocation((screenDimension.**width** - width)/2 ,(screenDimension.**height** - height)/2)  
 *isVisible* = **true**}  
**fun** JFrame.init(){  
 *init*(*mWidth* , *mHeight*)  
}  
**fun** JFrame.quit(){  
 **var** flag = 0  
 **var** message = **"您确定退出系统吗？"** flag = JOptionPane.showConfirmDialog(**this**,message,**"提示"**,JOptionPane.*YES\_NO\_OPTION*)  
 **if** (flag == JOptionPane.*YES\_OPTION*){  
 DBService.getInstance().close()  
 System.exit(0)  
 }  
}

*/\*\*  
 \* 设置btn背景为白色 varatg：可变参数  
 \*/***fun** setWhite(**vararg** btns : JButton){  
 **var** btn : JButton  
 **for** (btn **in** btns){  
 btn.*background* = Color.*WHITE* }  
}

*/\*\*  
 \* 设置所有字体为微软雅黑，字号为18  
 \*/***var** *heiti* : Font = Font(**"微软雅黑"** , Font.*PLAIN*,16)  
**val** *names* = *listOf*<String>(**"Label"**, **"CheckBox"**, **"PopupMenu"**,**"MenuItem"**, **"CheckBoxMenuItem"**,  
 **"JRadioButtonMenuItem"**,**"ComboBox"**, **"Button"**, **"Tree"**, **"ScrollPane"**,  
 **"TabbedPane"**, **"EditorPane"**, **"TitledBorder"**, **"Menu"**, **"TextArea"**,  
 **"OptionPane"**, **"MenuBar"**, **"ToolBar"**, **"ToggleButton"**, **"ToolTip"**,  
 **"ProgressBar"**, **"TableHeader"**, **"Panel"**, **"List"**, **"ColorChooser"**,  
 **"PasswordField"**,**"TextField"**, **"Table"**, **"Label"**, **"Viewport"**,  
 **"RadioButtonMenuItem"**,**"RadioButton"**, **"DesktopPane"**, **"InternalFrame"**)  
**fun** setFont(){  
 **var** item : String;  
 **for** (item **in** *names*){  
 UIManager.put(item + **".font"**, *heiti*)  
 }  
}

# 4系统测试



图4.1 选择界面

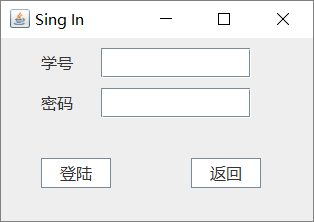
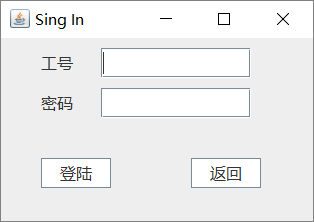
 

图4.2 学生登陆界面 图4.3 职工登陆界面

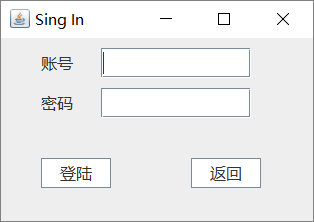


图4.4 主管登陆界面



图4.5 学生菜单界面



图4.6 学生个人信息界面

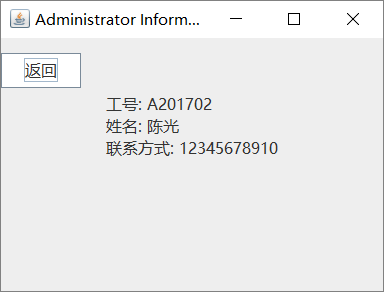


图4.7 公寓负责人信息界面



图4.8 学生个人信息修改测试

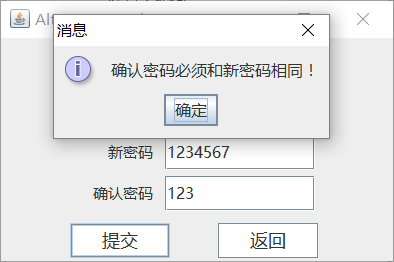
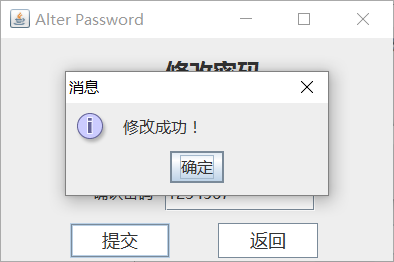
 

图4.9 修改密码测试a) 图4.10 修改密码测试b)

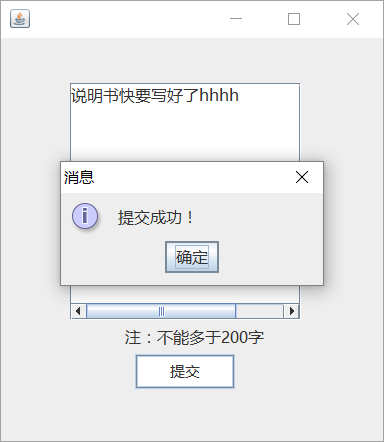


图4.11 意见反馈测试



图4.12 宿管菜单界面

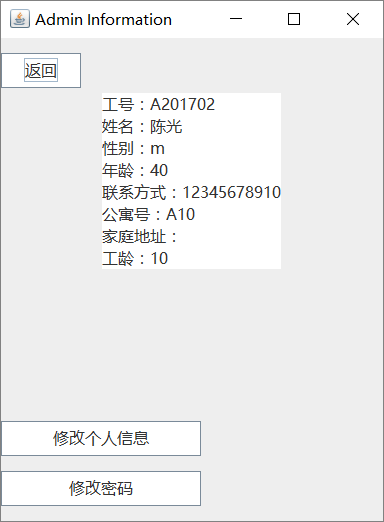
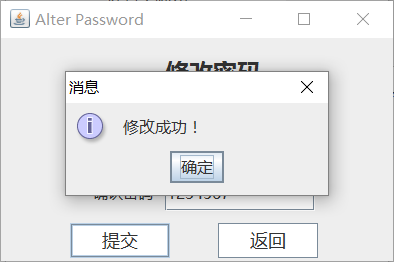
 

图4.13 宿管个人信息界面 图4.14 宿管密码修改测试

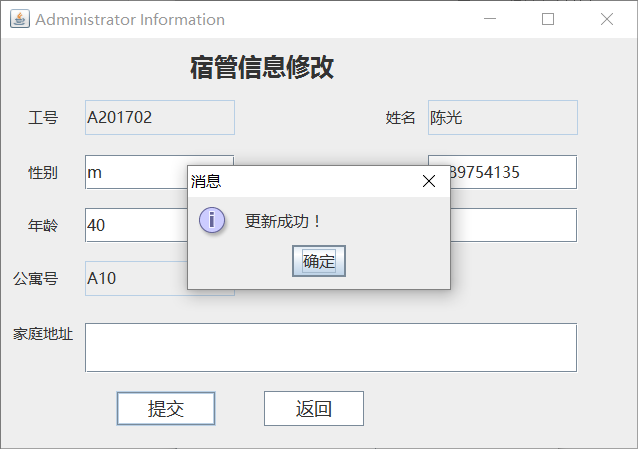


图4.15 宿管个人信息修改测试

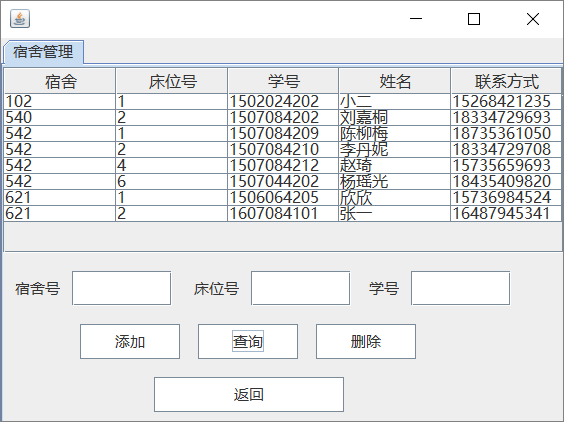


图4.16 公寓住宿管理界面

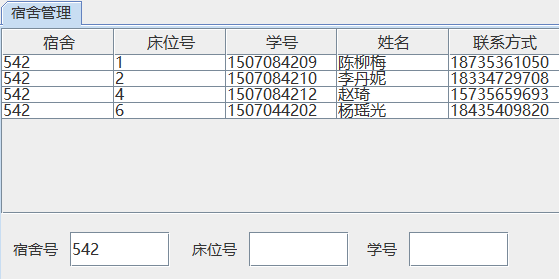


图4.17 住宿查询测试a)

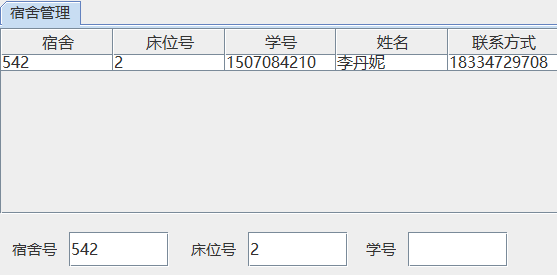


图4.18 住宿查询测试b)

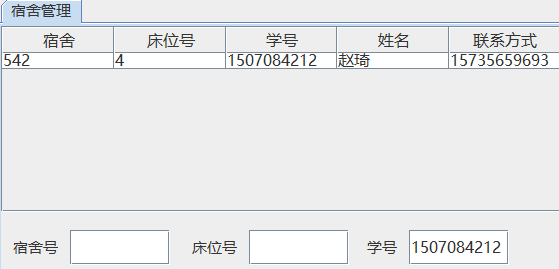


图4.19 住宿查询测试c)

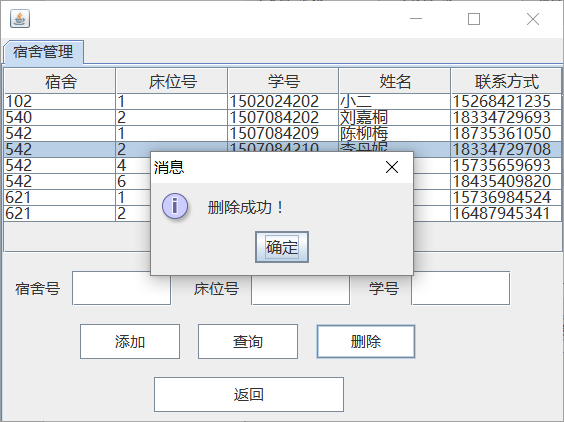


图4.20 住宿学生删除测试

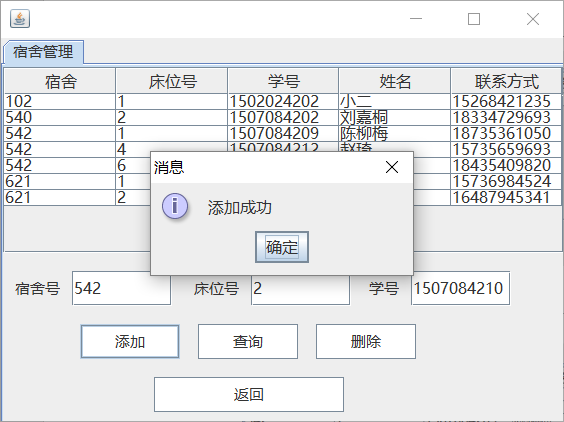


图4.21 住宿学生添加测试

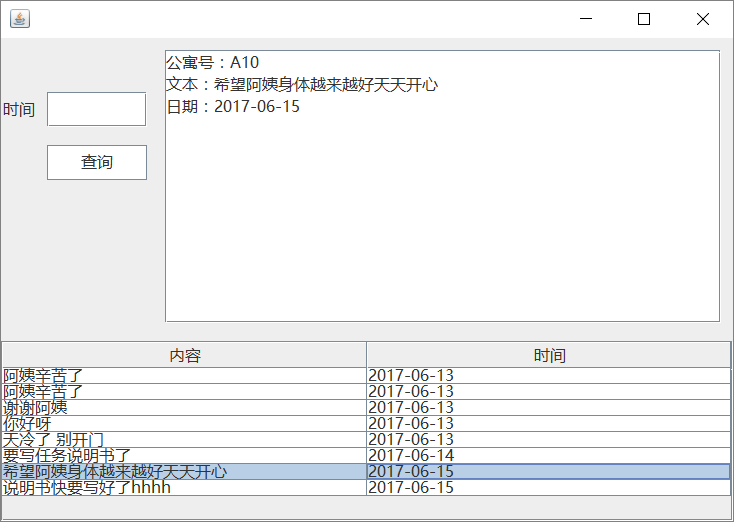


图4.22 意见反馈查看界面

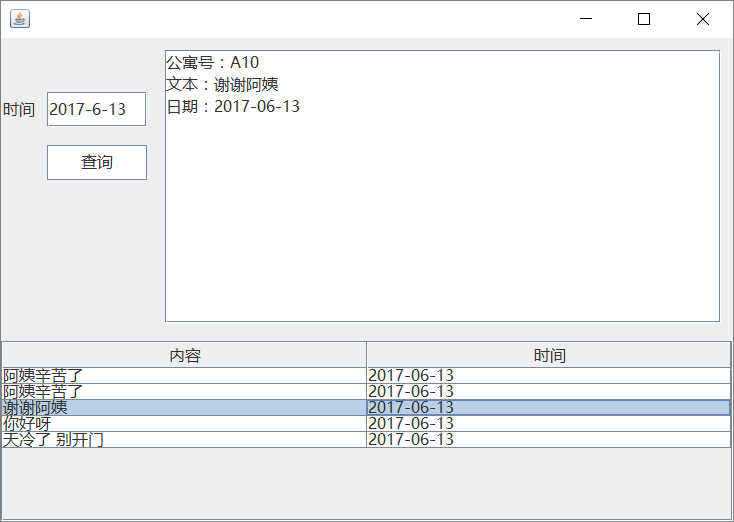


图4.23 意见反馈查询测试



图4.24 主管界面公寓信息标签栏

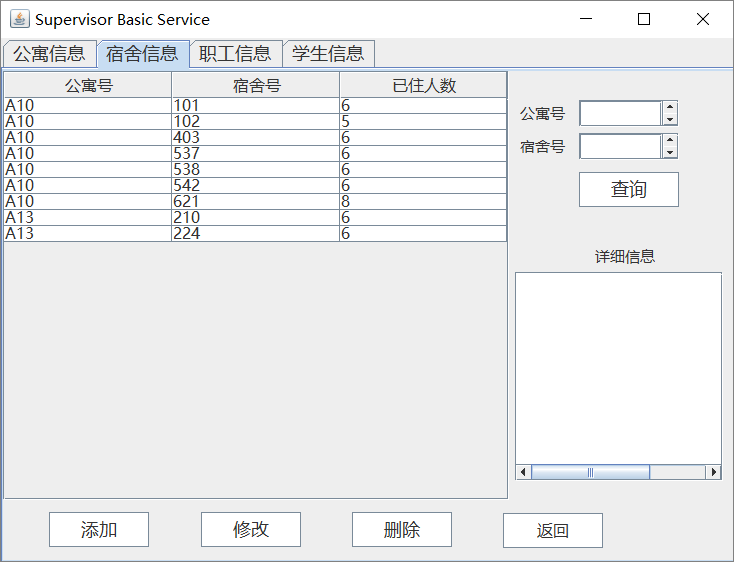


图4.25 主管界面宿舍信息标签栏

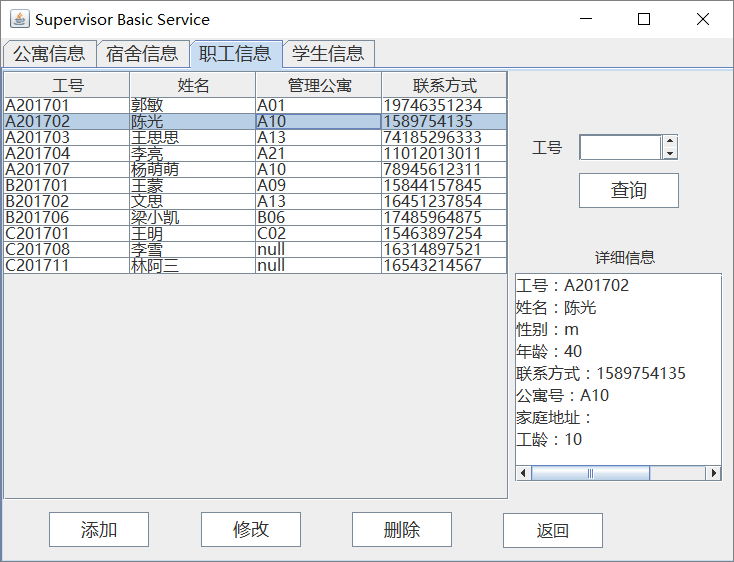


图4.26 主管界面职工信息标签栏

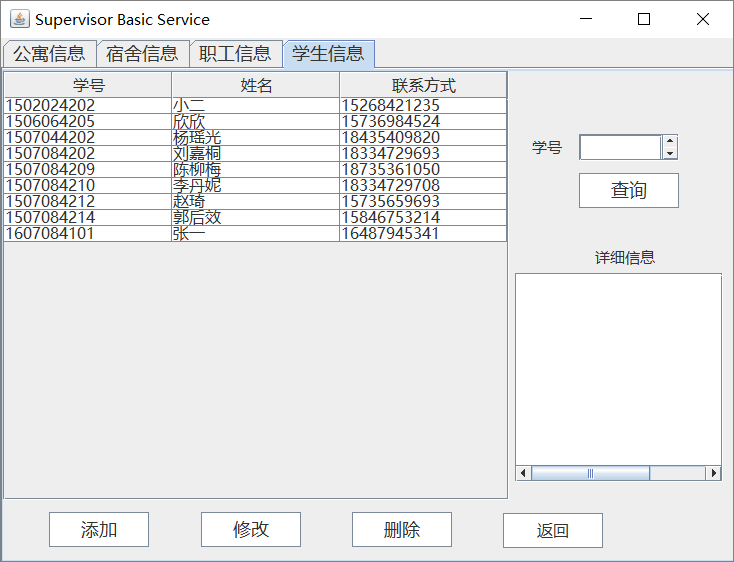


图4.27 主管界面学生信息标签栏

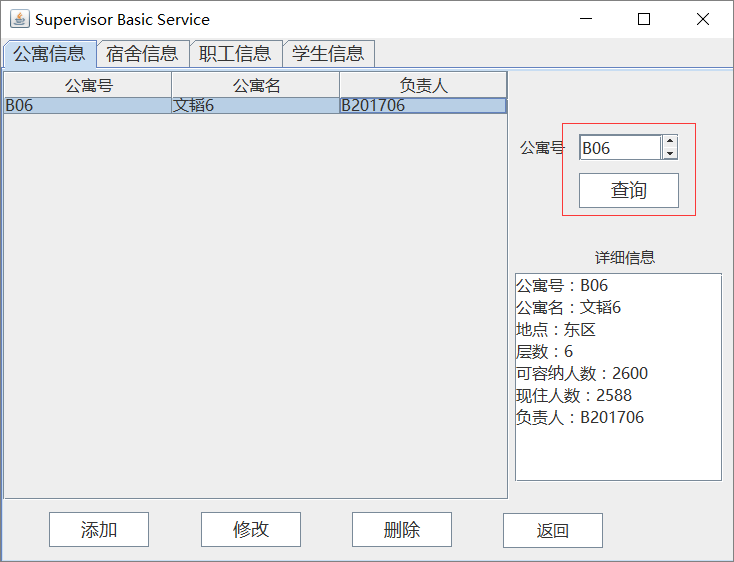


图4.28公寓信息查询测试

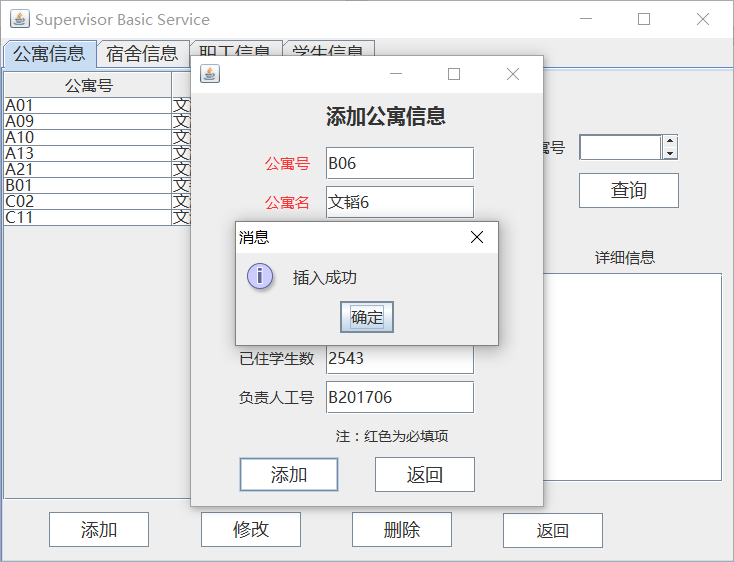


图4.29公寓信息添加测试



图4.30 公寓信息修改测试

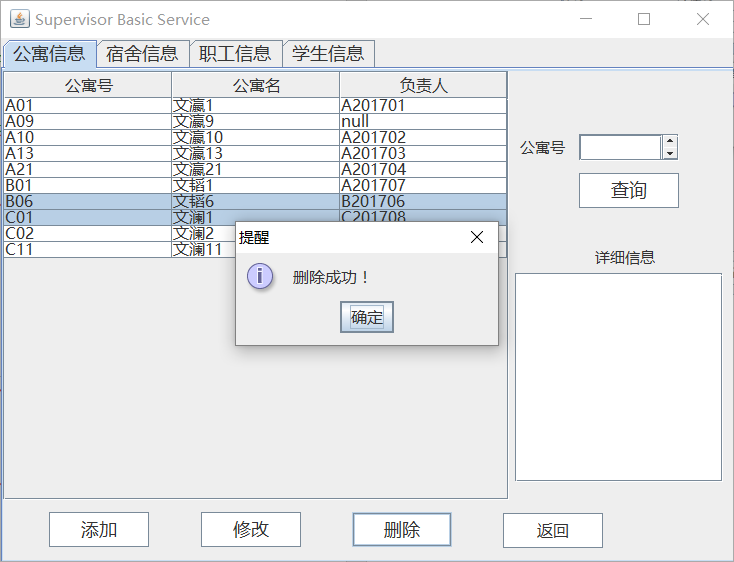


图4.31 公寓信息删除测试

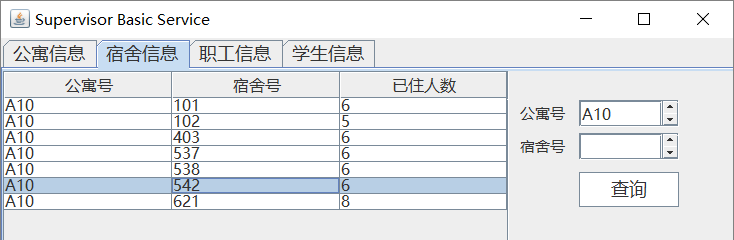
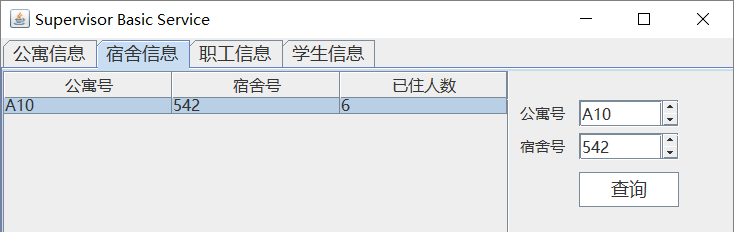
 

图4.32宿舍查询测试



图4.33 宿舍添加测试

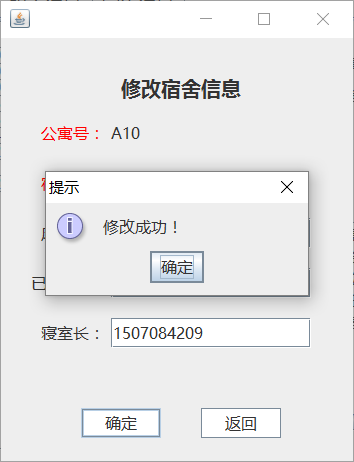


图4.34 宿舍修改测试

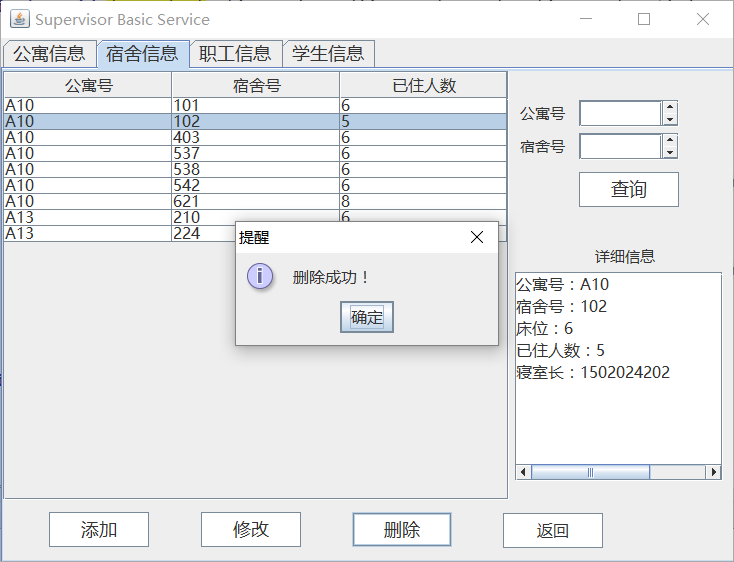


图4.35 宿舍删除测试

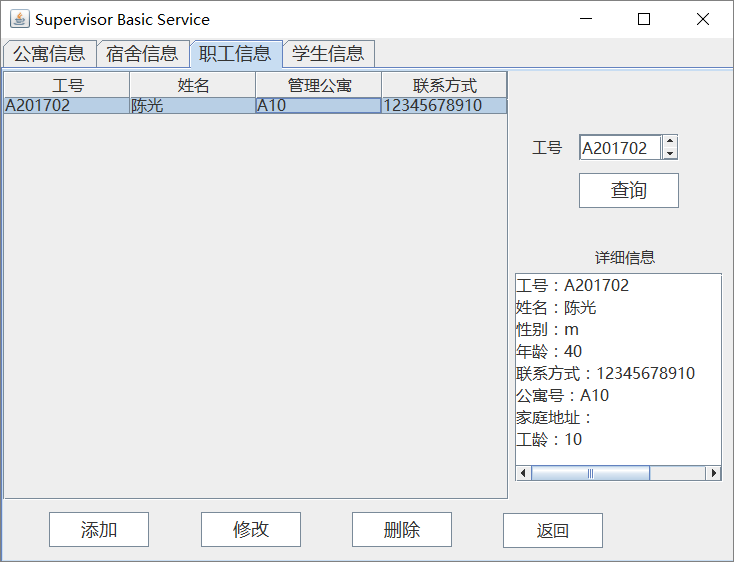


图4.36 职工查询测试

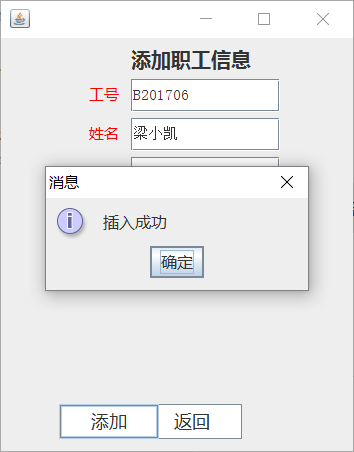


图4.37 职工添加测试

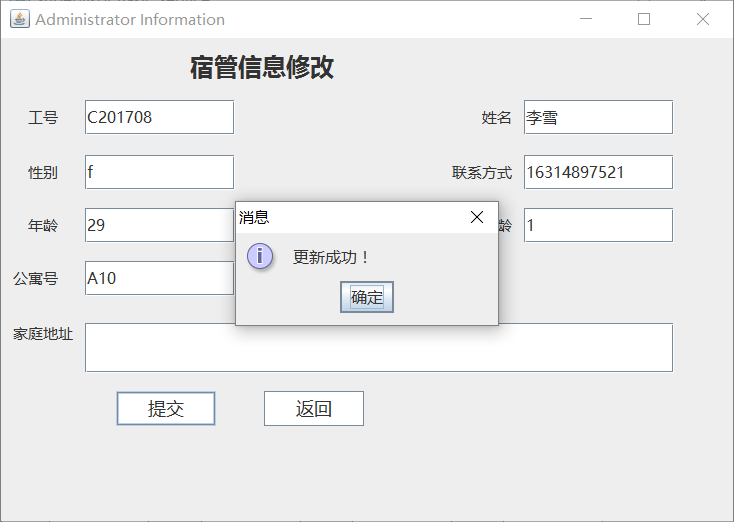


图4.38 职工信息修改测试

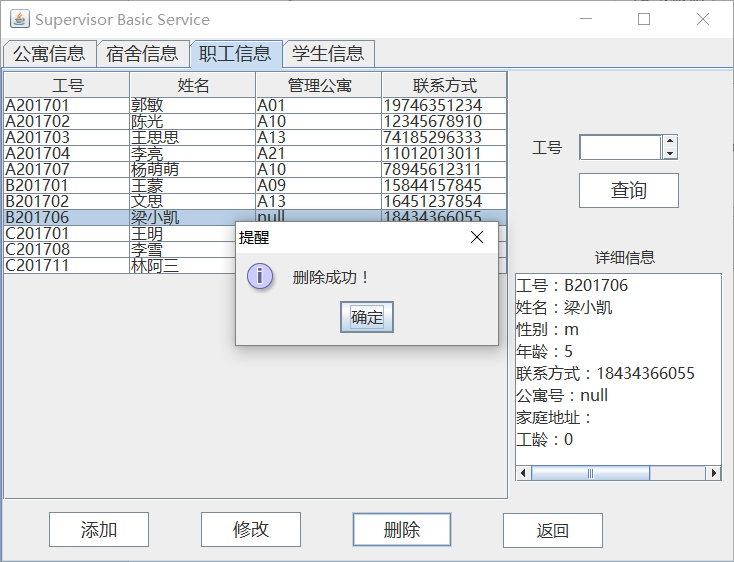


图4.39 职工删除测试

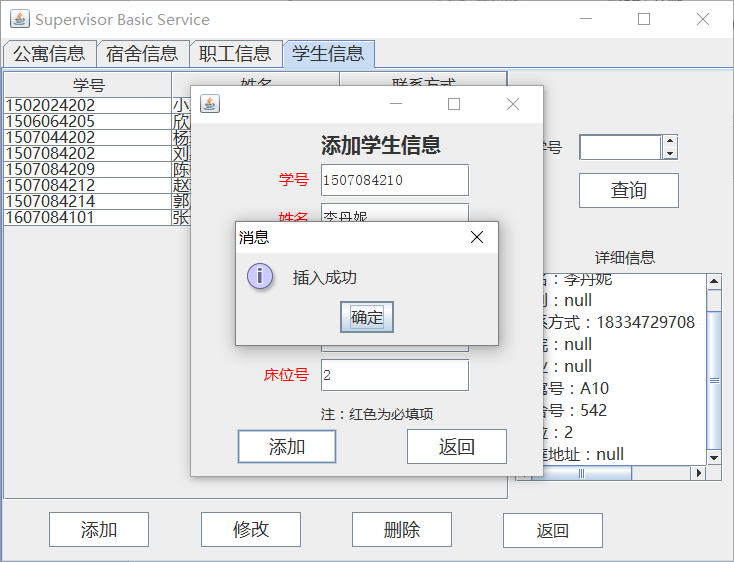


图4.40 学生添加测试

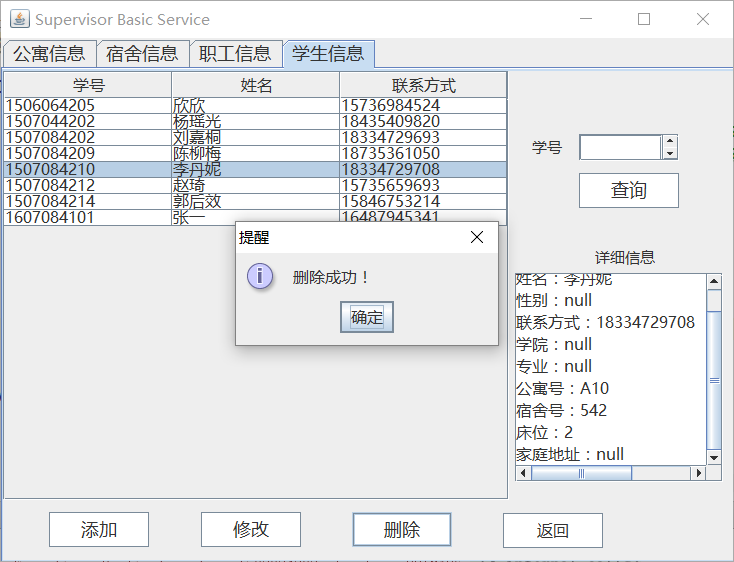


图4.41 学生删除测试



图4.42 学生信息修改测试

# 5个人体会

通过这次课设，我现在已经可以说是掌握了数据库编程的相关信息，对数据库完整性有了新的理解，视图、触发器、外键约束等机制我也可以熟练的使用了。从一开始配置MySQL数据库服务时各种爬坑到现在已经对MySQL这个数据库非常熟悉了，对MySql Workbench这个可视化工具也可以熟练操作了。

我在这次课设中应用了部分Kotlin语言，刚开始对这个语言只是了解一点基础的语法，用着用着就感受到了这个语言的魅力，语法简单而且支持函数式编程。宿舍管理系统使我把很多在课堂上学的运用到实际应用中了，比如设计模式中的单例模式，数据结构中的栈，这些对于很多项目来说都是非常方便的。

这两周的日子虽然苦了点，但是我却是异常的充实，学到了非常多的东西。在以后的学习和生活中，希望自己能继续努力，多学多看多动手，努力成为一个合格的社会主义接班人。