评语：

成绩：

**数据库原理**

**课程设计报告**

课题名称： 寝室管理系统

指导教师：

专业班级：

学 号：

姓 名：

**完成日期：2024年12月23日**

目 录

[1 问题的提出 2](#_Toc186046887)

[2 可行性分析 4](#_Toc186046888)

[2.1 技术可行性 4](#_Toc186046889)

[2.2 经济可行性 4](#_Toc186046890)

[2.3 操作可行性 4](#_Toc186046891)

[3 需求分析 5](#_Toc186046892)

[3.1 数据需求 5](#_Toc186046893)

[3.2 功能需求 5](#_Toc186046894)

[3.3 数据流图（DFD） 6](#_Toc186046895)

[3.4 数据字典 10](#_Toc186046896)

[4 数据库设计 15](#_Toc186046897)

[4.1 概念设计 15](#_Toc186046898)

[4.2 逻辑设计 15](#_Toc186046899)

[4.3 物理设计 18](#_Toc186046900)

[4.4 数据库建立 18](#_Toc186046901)

[4.5 数据初始化 29](#_Toc186046902)

[4.6 数据库用户权限管理 33](#_Toc186046903)

[5 数据库编程设计 35](#_Toc186046904)

[5.1 建立存储过程 35](#_Toc186046905)

[5.2 建立触发器 39](#_Toc186046906)

[5.3 视图设计 41](#_Toc186046907)

[6 系统实现 45](#_Toc186046908)

[6.1 系统功能模块图 45](#_Toc186046909)

[6.2 开发平台和开发工具 46](#_Toc186046910)

[6.3 寝室管理模块的实现 47](#_Toc186046911)

[7 系统测试 74](#_Toc186046912)

[7.1 存储过程的测试 74](#_Toc186046913)

[7.2 触发器的测试 77](#_Toc186046914)

[7.3 视图的测试 78](#_Toc186046915)

[总 结 82](#_Toc186046916)

[**参考文献** 84](#_Toc186046917)

# 1 问题的提出

学校后勤管理部门提出需要建立一套**寝室管理系统**进行计算机化的数据管理。寝室管理的现状如下：

1. 学校共有多个宿舍楼，每栋楼包含多个寝室，每间寝室提供若干床位。
2. 每个寝室有固定编号，学生信息包括学号、姓名、性别、联系方式和入住时间等。
3. 学生寝室的分配目前由管理员手动处理，寝室的空余床位和已入住学生情况难以实时查询。
4. 学生寝室的调整和更换过程中，需要人工更新记录，操作繁琐且容易出现错误。
5. 学生的寝室维修申请依赖手工记录，且处理进度无法追踪，造成维修效率较低。
6. 目前没有统一的系统来管理和统计寝室的使用情况、空余床位信息以及维修记录，管理人员需要花费大量时间和精力来维护数据。

**拟开发的系统要求**

1. **寝室管理功能：**

a.实现寝室基本信息的录入、修改和删除，包括寝室编号、楼栋、楼层、床位数等。

b.支持按楼栋、楼层等条件查询寝室的基本信息以及空余床位情况。

1. **学生管理功能：**

a.实现学生信息的录入、修改和删除，包括学号、姓名、性别、联系方式、寝室编号、入住时间等。

b.支持按学号、姓名等条件查询学生的寝室分配信息。

1. **寝室分配功能：**

a.支持学生入住寝室的分配操作，自动更新空余床位信息。

b.支持学生寝室调整或更换，方便管理员灵活处理特殊情况。

1. **维修申请管理功能：**

a.实现学生在线提交寝室维修申请，包括填写维修内容、寝室编号、申请时间等。

b.支持管理员查看和处理维修申请，并更新维修状态（待处理、处理中、已完成）。

1. **用户管理功能：**

a.支持管理员账户和学生账户的创建与删除。

b.不同用户权限明确，确保数据安全。

1. **系统设计与数据库功能：**

a.设计一个完整的数据库，包括学生、寝室、维修申请等核心数据表，确保数据的一致性和完整性。

b.掌握数据库设计的每个步骤，包括需求分析、概念设计、逻辑设计和物理设计。

c.熟练使用SQL语言，支持数据的增删改查操作，以及必要的约束条件和关系维护。

# 2 可行性分析

## 2.1 技术可行性

寝室管理系统的开发技术主要基于成熟的编程语言、数据库和开发工具，技术方面具备以下可行性：

1. 开发语言：Java等主流语言，均具有丰富的开发资源和强大的功能支持，可用于快速构建系统。
2. 数据库技术： SQL Server数据库系统具有广泛的使用基础，支持高效的数据存储、查询和管理，完全可以满足本系统的需求。
3. 网络和硬件：本系统可在学校现有的计算机硬件和网络基础上运行，无需额外购置设备，技术支持完全可行。

## 2.2 经济可行性

本系统的开发和运行经济上具备以下可行性：

1. **开发成本**：系统开发可利用现有的软件工具和开源技术（如Java、SQL Server等），避免额外的购买费用，仅需投入开发者的时间成本。
2. **运行成本**：系统可运行在现有的学校计算机设备上，无需新增硬件投资，维护成本较低。
3. **长远效益**：系统上线后可减少人工管理的时间和工作量，提高管理效率，从长期来看经济效益显著。

## 2.3 操作可行性

本系统在实际操作方面具备以下可行性：

1. **用户操作**：系统界面设计简单易用，学生和管理员通过直观的界面即可完成基本操作，减少了学习成本。
2. **管理员操作**：系统提供全面的管理功能，包括寝室分配、维修申请处理等，管理员可以轻松上手，提高工作效率。
3. **系统部署**：本系统支持在单机或局域网环境下部署，学校现有的计算机网络设施完全能够支持系统的运行和管理。

# 3 需求分析

## 3.1 数据需求

寝室管理系统的数据需求主要包括：  
① 学生基本信息的输入，包括学号、密码、姓名、性别、联系方式、学院、班级。

② 学生基本信息的查询、修改，包括学号、密码、姓名、性别、联系方式、学院、班级。

③ 寝室基本信息的输入，包括寝室编号、寝室人数、最大人数。

④ 寝室基本信息的查询、修改，包括寝室编号、寝室人数、最大人数。

⑤ 学生寝室分配信息的输入，包括学号、寝室编号。

⑥ 学生寝室分配信息的查询、修改，包括学号、寝室编号。

⑦ 来访人员信息的输入，包括来访人姓名、身份证号、联系方式。

⑧ 来访人员信息的查询、修改，包括来访人姓名、身份证号、联系方式。

⑨ 来访登记信息的输入，包括学号、来访人身份证号、来访编号、来访时间、离开时间、来访原因。

⑩ 来访登记信息的查询、修改，包括学号、来访人身份证号、来访编号、来访时间、离开时间、来访原因。

⑪ 维修申请信息的输入，包括报修编号、寝室编号、报修详细信息、报修时间、是否解决、解决备注。

⑫ 维修申请信息的查询、修改，包括报修编号、寝室编号、报修详细信息、报修时间、是否解决、解决备注。

⑬ 管理员信息管理，包括用户名、密码的创建、删除。

⑭ 管理员进行学生账户信息管理，包括学号、密码的创建、删除。

## 3.2 功能需求

寝室管理系统的功能需求主要包括：

① 在学生信息管理部分，要求：

a. 查询学生基本信息，包括学号、密码、姓名、性别、联系方式、学院、班级。

b. 对学生信息进行添加、修改、删除操作。

② 在寝室信息管理部分，要求：

a. 查询寝室基本信息，包括寝室编号、寝室人数、最大人数。

b. 对寝室信息进行添加、修改、删除操作。

③ 在寝室分配管理部分，要求：

a. 查询学生的寝室分配情况，包括学号与寝室编号的对应关系。

b. 实现寝室分配功能，包括为学生分配寝室编号。

c. 实现寝室调整功能，包括修改学生的寝室分配信息。

④ 在维修申请管理部分，要求：

a. 学生提交寝室维修申请，包括填写维修详细信息、寝室编号、申请时间。

b. 管理员查询维修申请信息，包括报修编号、学生信息、寝室编号、维修详细信息、报修时间、是否解决、解决备注。

c. 管理员修改维修申请状态（待处理、处理中、已完成），并添加解决备注。 ⑤ 在用户账户管理部分，要求：

a. 管理员创建和删除学生用户账户，包括学号、密码信息。

b. 管理员创建和删除管理员用户账户，包括用户名、密码信息。

## 3.3 数据流图（DFD）

通过对系统需求的调查分析，细化软件功能，把系统划分为几个主要模块，明确每个模块所要完的主要逻辑功能，采用以图形的方式描绘数据在系统中流动和处理。

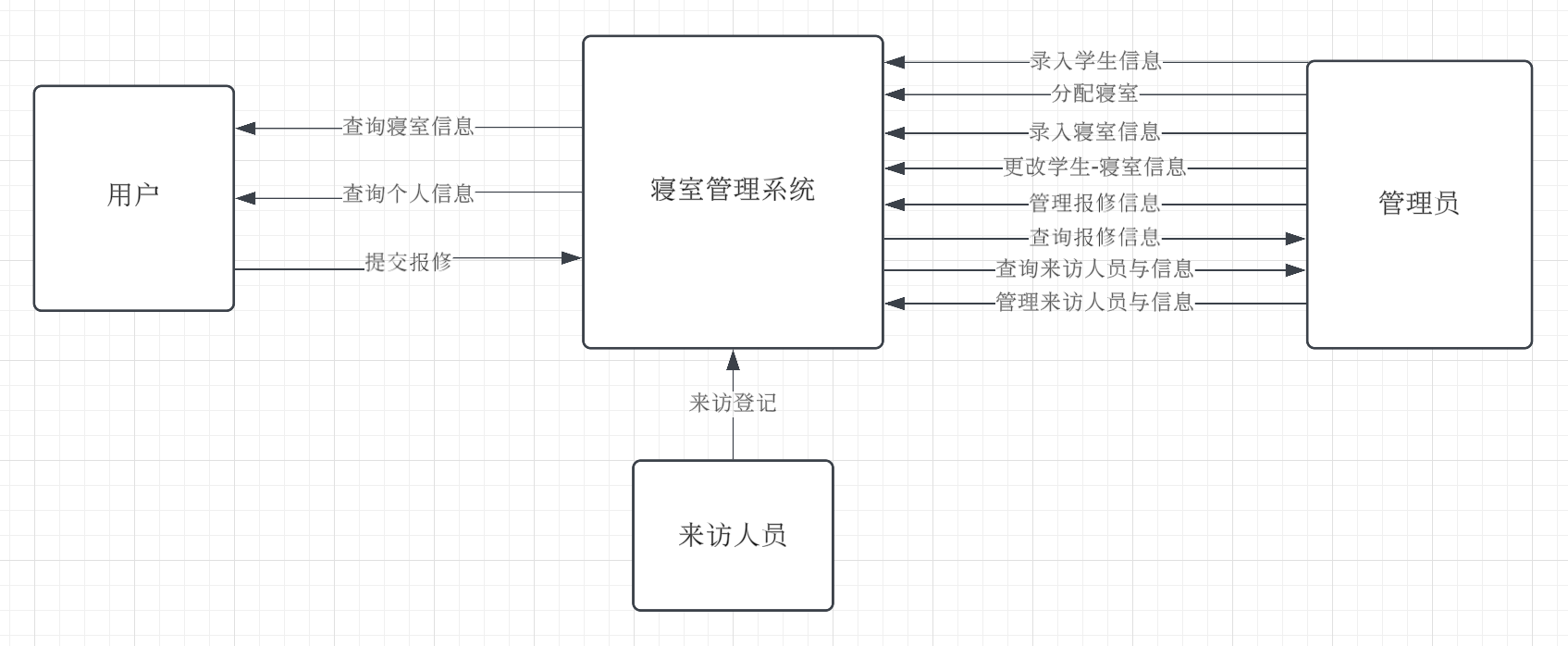


图1-1顶层数据流图

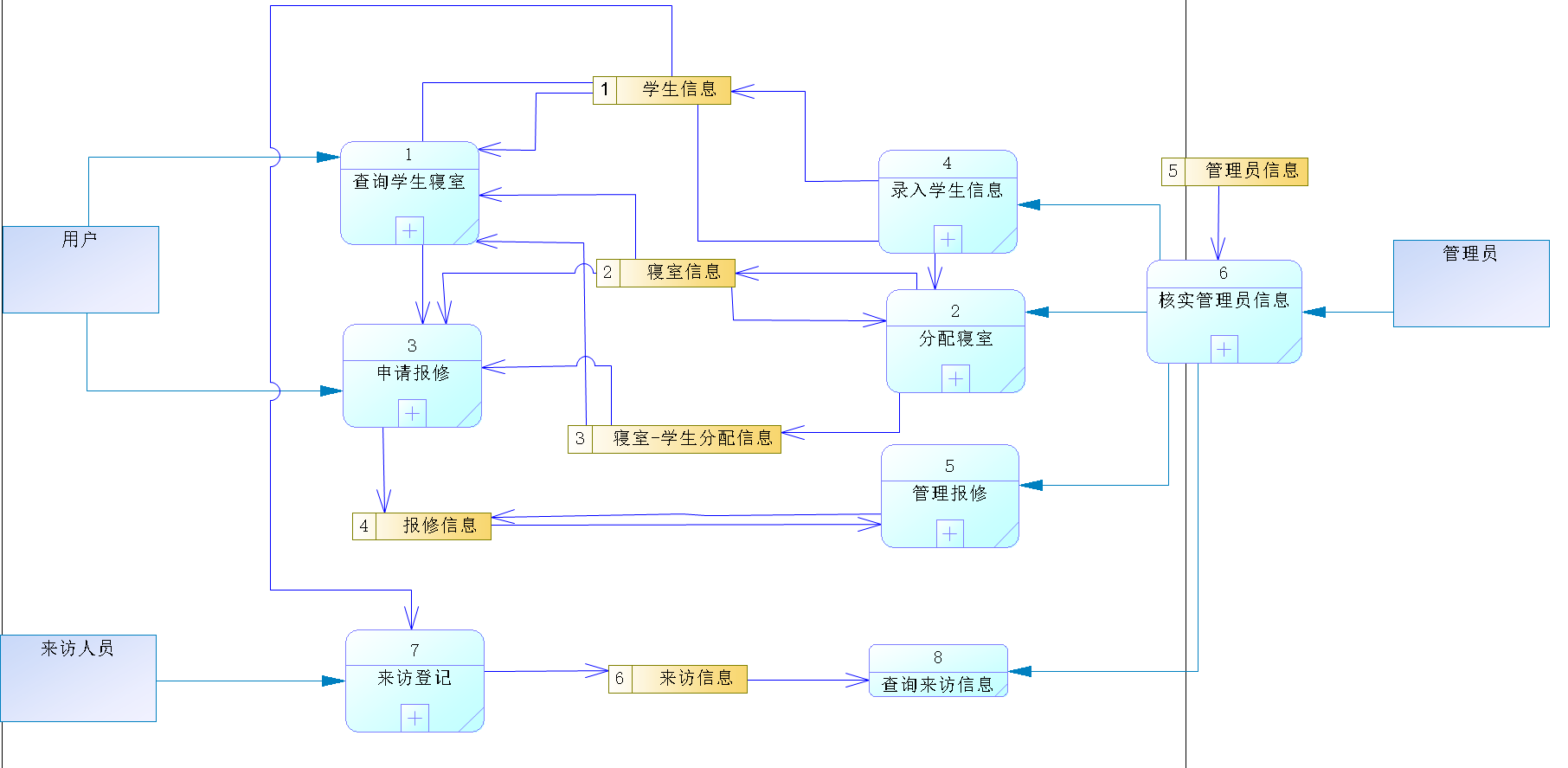


图1-2 第一层数据流图（整个系统）

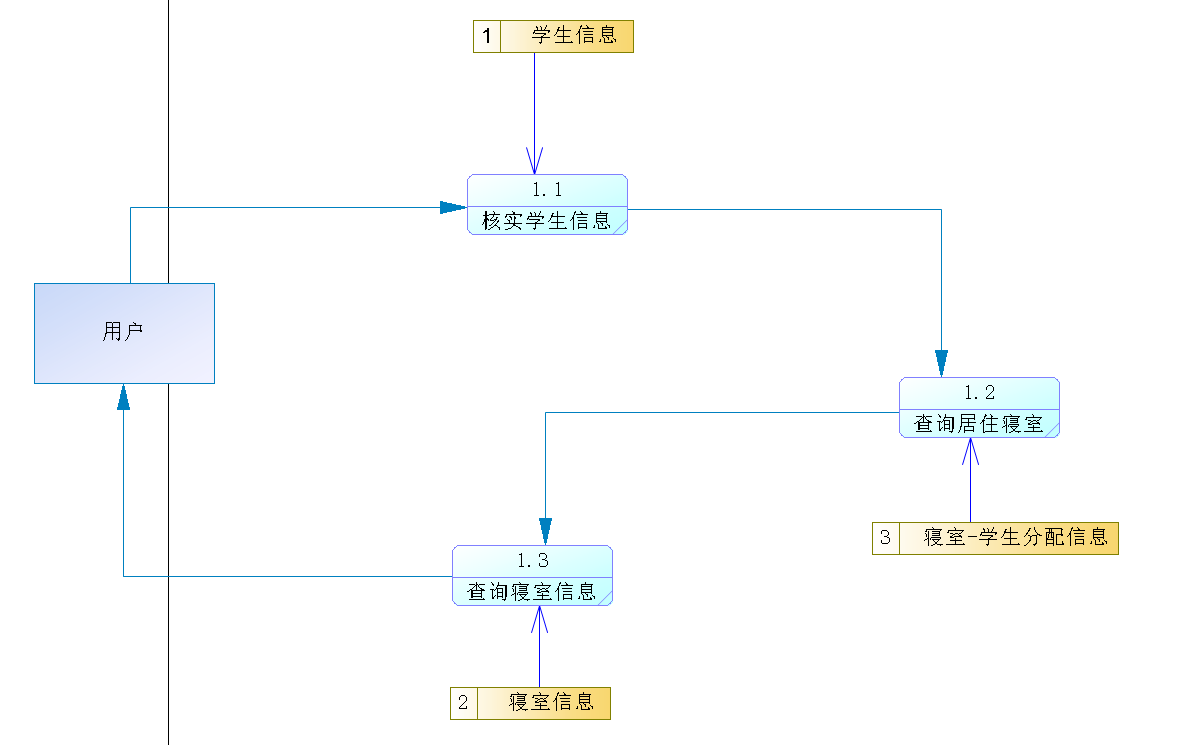


图1-3 第二层数据流图（细化“查询学生寝室”处理）

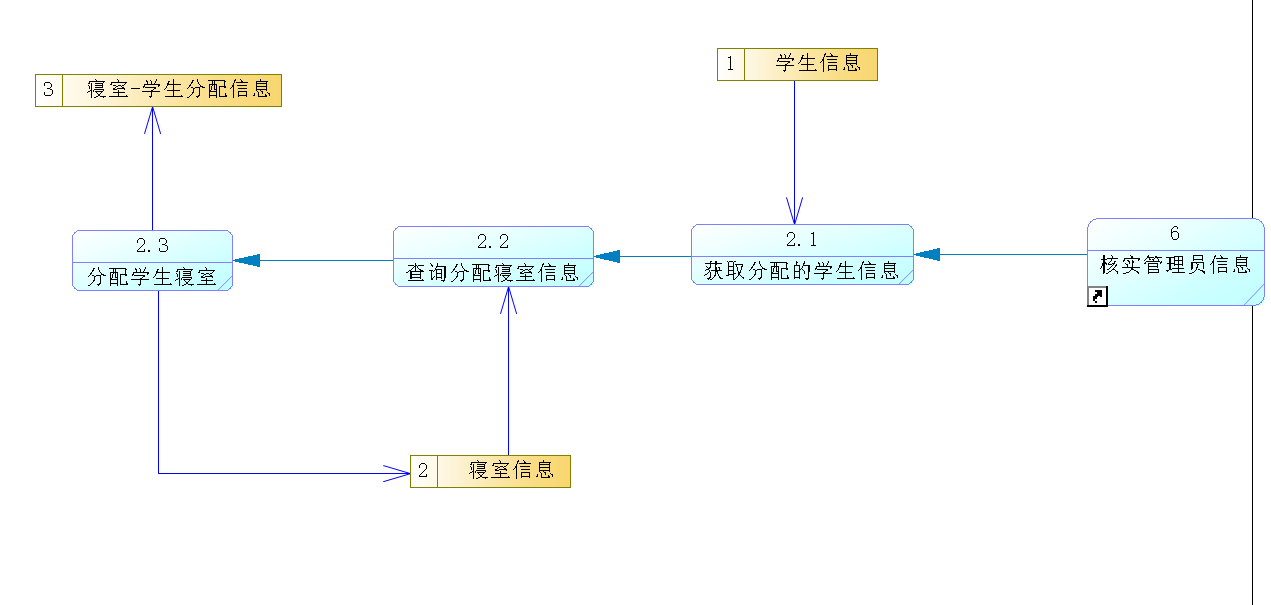


图1-4 第二层数据流图（细化“分配寝室”处理）

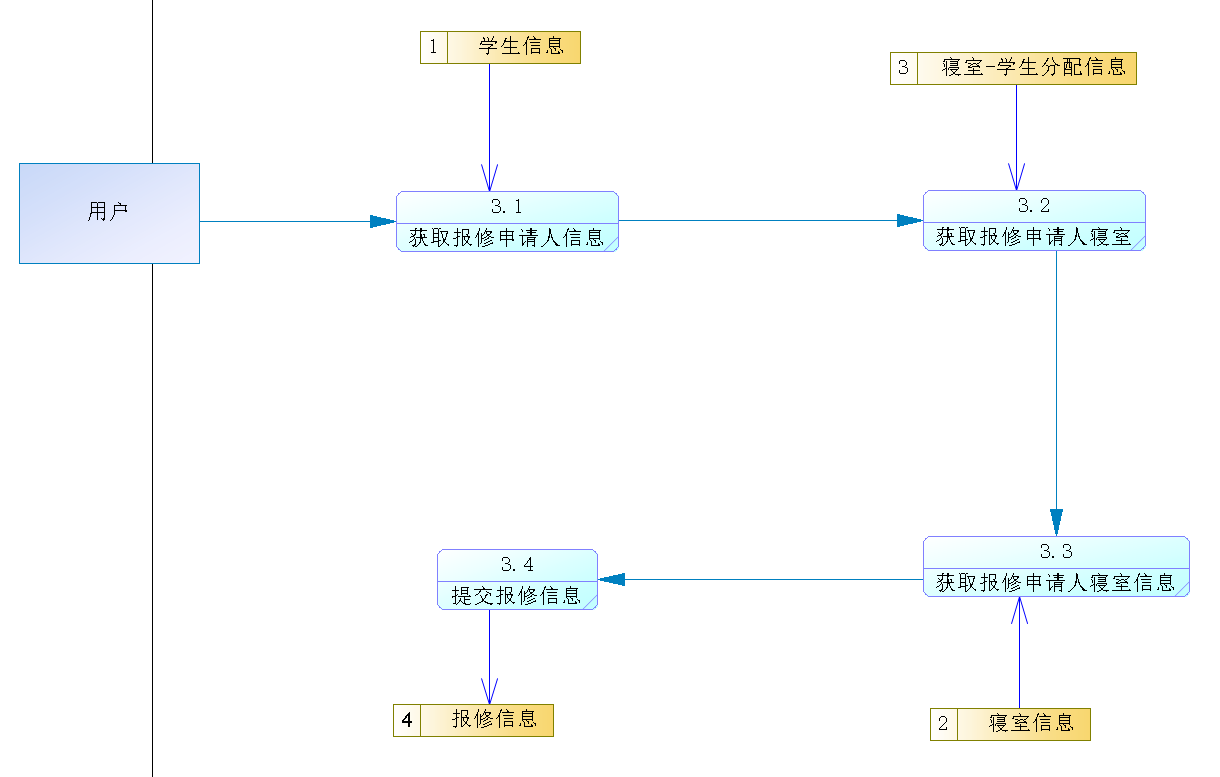


图1-5 第二层数据流图（细化“申请报修”处理）

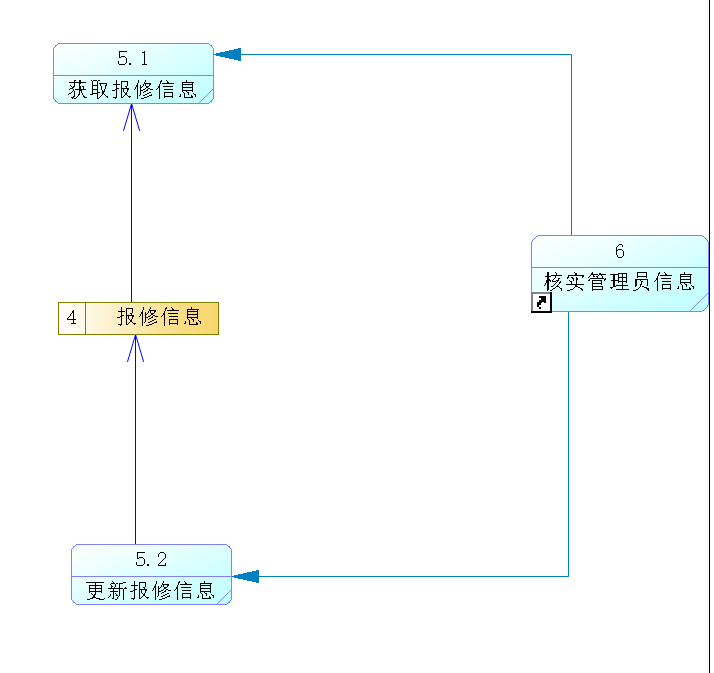


图1-6 第二层数据流图（细化“管理报修”处理）

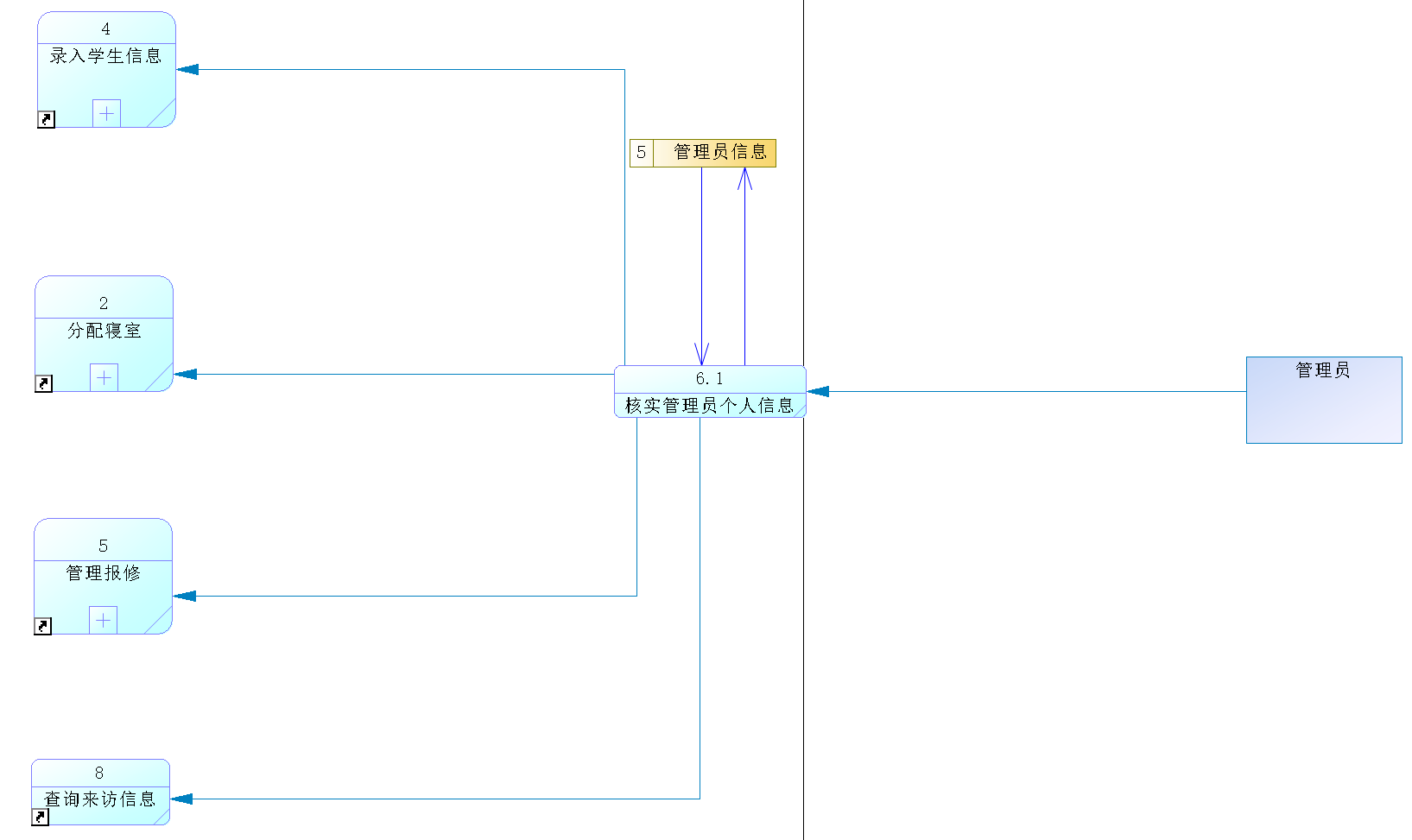


图1-7 第二层数据流图（细化“核实管理员信息”处理）

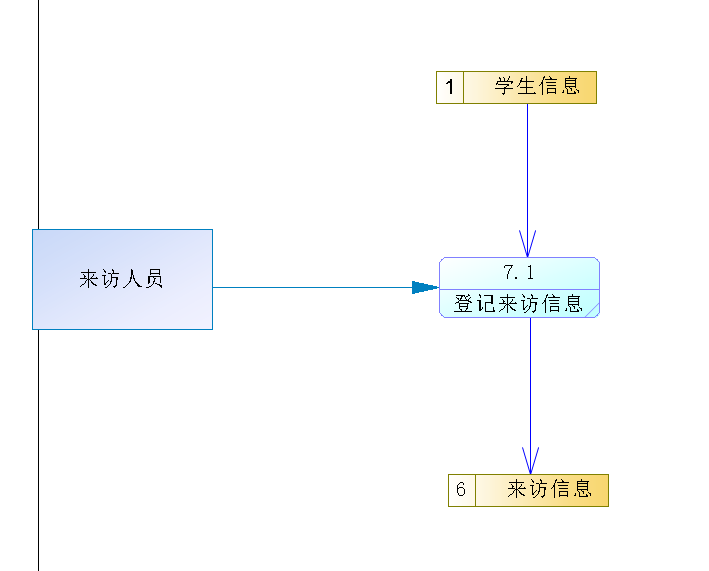


图1-7 第二层数据流图（细化“来访登记”处理）

## 3.4 数据字典

为了使各数据流，数据处理过程，存储过程不能反映其中的数据成，数据项目，数据特性，所以用数据字典来对数据流图中的各成份进行具体的定义，为系统的分析，设计及以后的实现提供有关元素一致性定义和详细的描述：

①数据流字典

数据流名称：查询学生寝室

别名：无

来源：用户

去向： 用户

数据组成：学生学号|学生姓名|寝室编号

数据流名称：申请报修

别名：无

来源：用户

去向： 报修信息

数据组成： 学号|寝室编号|报修信息

数据流名称：核实管理员信息

别名：无

来源：管理员

去向： 录入学生信息、分配寝室、管理报修

数据组成：管理员用户名|管理员密码

数据流名称：录入学生信息

别名：无

来源：管理员

去向： 学生信息

数据组成：学生学号|寝室编号|寝室人数

数据流名称：管理报修

别名：无

来源：管理员

去向： 管理员、报修信息

数据组成：报修编号|报修状态|学生学号|寝室编号|报修信息

数据流名称：分配寝室

别名：无

来源：管理员

去向： 寝室信息、寝室-学生分配信息

数据组成：学生学号|寝室编号|寝室人数

数据流名称：来访登记

别名：无

来源：来访人员

去向： 来访信息

数据组成：学生学号|来访者个人信息|来访时间

②数据存储字典：

数据存储名称：学生信息 标志符：D1

描述：学生的详细信息

涉及处理：P1、P2、P3、P7及下一层细化的处理

数据组成：学生学号|密码|姓名|性别|联系方式|学院|班级

数据存储名称：寝室信息 标志符：D2

描述：寝室的详细资料

涉及处理：P1、P2、P3及下一层细化的处理

数据组成：寝室编号|寝室人数|最大居住人数

数据存储名称：寝室-学生分配信息 标志符：D3

描述：学生及所分配的寝室的详细信息

涉及处理：P2、P3及下一层细化的处理

数据组成：学生学号|寝室编号

数据存储名称：报修信息 标志符：D4

描述：寝室报修的详细信息

涉及处理：P3、p5及下一层细化的处理

数据组成：学生学号|寝室编号|报修信息

数据存储名称：管理员信息 标志符：D5

描述：管理员的详细资料

涉及处理：P6及下一层细化的处理

数据组成：管理员用户名|管理员密码

数据存储名称：来访信息 标志符：D6

描述：来访人员及来访的详细信息

涉及处理：P7、P8及下一层细化的处理

数据组成：来访人员姓名|来访人员联系方式|来访人员身份证号|来访问的学生学号|来访时间|离开时间

③数据处理字典：

数据处理名称：查询居住寝室 标志符：P1.2

处理定义：查询学生居住寝室

激发条件：输入学生登录成功

输入：学生学号

输出：学生居住的寝室

数据处理名称：查询寝室信息 标志符：P1.3

处理定义：查新寝室信息

激发条件：学生登录成功并查找到寝室

输入：寝室号

输出：D2寝室信息

数据处理名称：获取分配学生的信息 标志符：P2.1

处理定义：获取学生信息

激发条件：管理员登录成功

输入：管理员要查询的学号

输出：D1学生信息

数据处理名称：获取报修申请人寝室信息 标志符：P3.3

处理定义：获取报修申请人寝室信息

激发条件：获取了申请人的个人信息

输入：申请人学号

输出：D2寝室信息

数据处理名称：获取报修信息 标志符：P5.1

处理定义：管理员获取学生报修信息

激发条件：管理员查询保修信息

输入：管理员查询

输出：D4报修信息

数据处理名称：登记来访信息 标志符：P7.1

处理定义：登记来访者及来访信息

激发条件：来访人员登记信息

输入：来访的详细信息

输出：无

# 4 数据库设计

## 4.1 概念设计

根据需求分析的结果，本系统的概念模型中有五个实体，分别是管理员、学生、寝室、来访人信息、报修信息，其中，学生和寝室之间有学生居住寝室关系，寝室与报修信息之间存在一对多关系，学生与来访人信息之间存在来访登记过程。

下面是概念模型图：

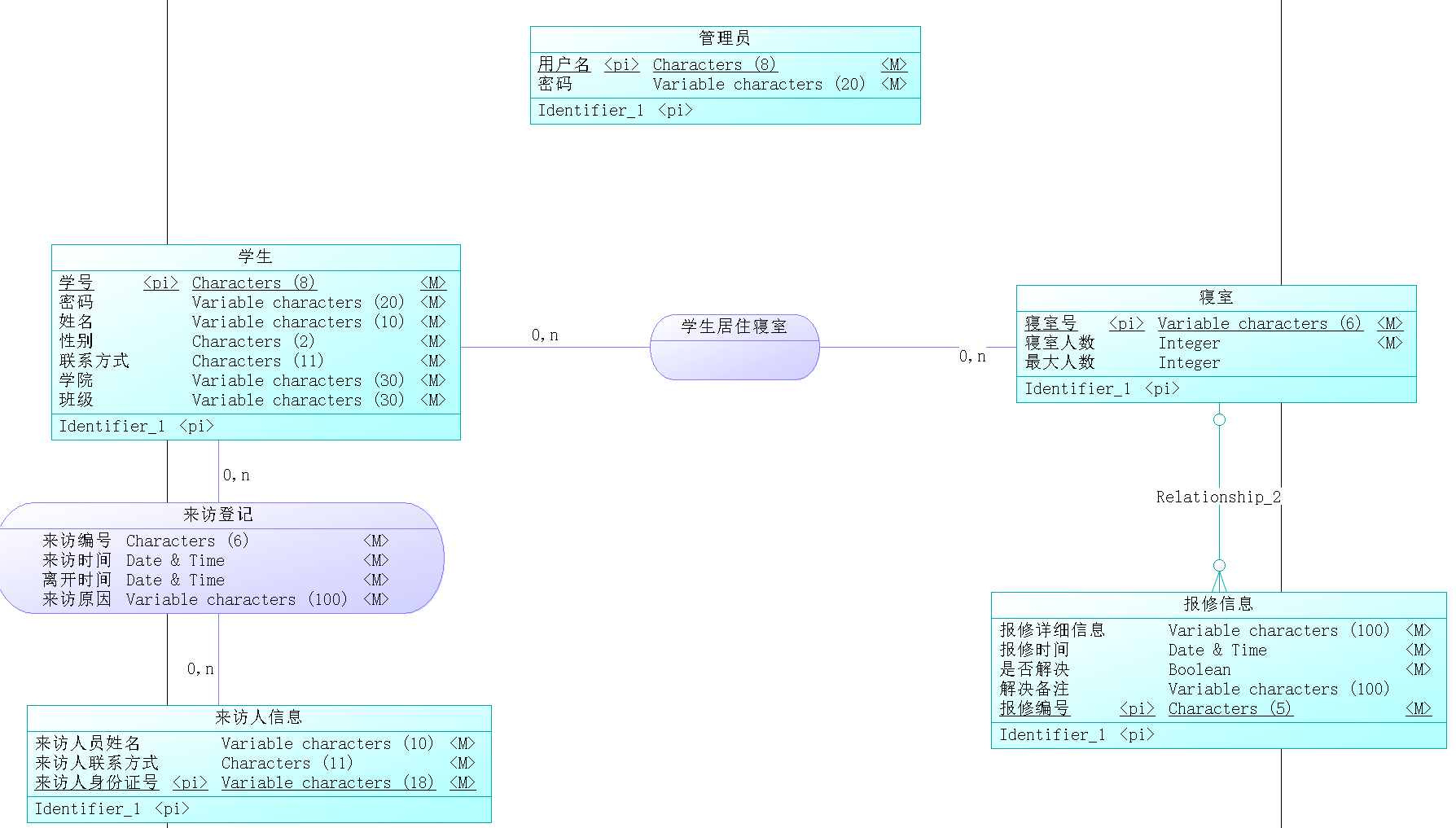
****

图4-1 图书管理系统的概念模型

## 4.2 逻辑设计

**4.2.1 表结构**

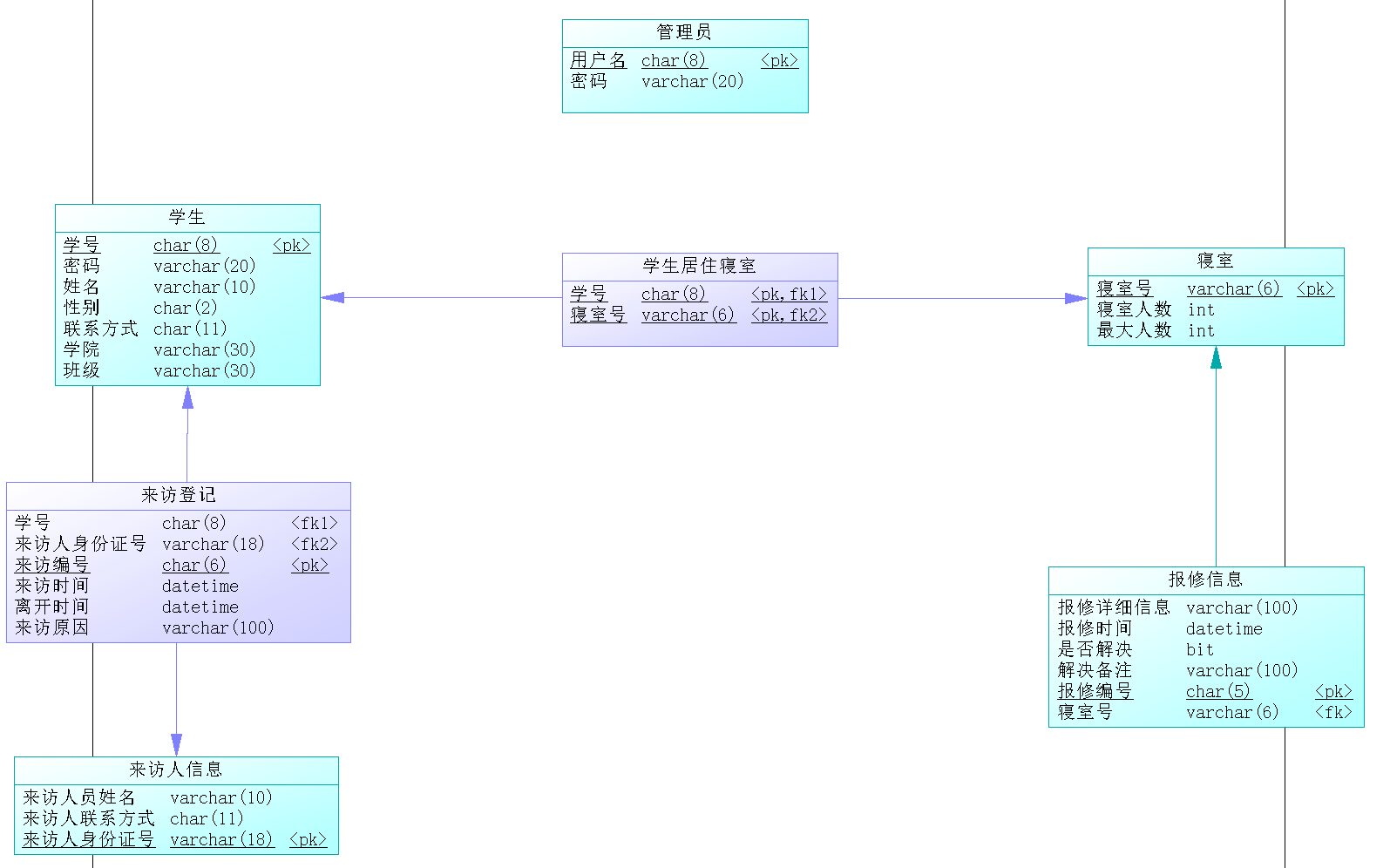


图4-2 寝室管理系统的逻辑模型

说明：这里的“来访登记”联系中，自动生成的是“学号”和“来访人身份证号”作为主键，但是一个来访者可以来访一个学生多次，因此，我将主键更改为“来访编号”，这样可以保证“来访登记表”的主键的唯一性。

具体每个表的结构如下列表格所示：

表4-1　 学生表S的结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表中列名** | **数据类型** | **可否为空** | **说明** | **约束** |
| **SNO** | **char(8)** | **not null(主键)** | **学号** |  |
| **SPassword** | **Varchar(20)** | **not null** | **学生密码** |  |
| **SName** | **Varchar(10)** | **not null** | **姓名** |  |
| **SSex** | **Char(2)** | **not null** | **性别(默认为男)** | **‘男’或‘女’** |
| **SPhone** | **Char(11)** | **not null** | **联系方式** |  |
| **SCollege** | **Varchar(30)** | **not null** | **学院** |  |
| **SClass** | **Varchar(30)** | **not null** | **班级** |  |

表4-2　 寝室表R的结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表中列名** | **数据类型** | **可否为空** | **说明** | **约束** |
| **RNO** | **Varchar(6)** | **not null(主键)** | **寝室号** |  |
| **RNumber** | **Int** | **not null** | **寝室人数(默认为0)** |  |
| **RMax** | **Int** | **not null** | **最大人数** | **2<=Rmax<=10** |

表4-3　 学生寝室表SR的结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表中列名** | **数据类型** | **可否为空** | **说明** | **约束** |
| **RNO** | **Varchar(6)** | **not null(主键、外键)** | **寝室号** |  |
| **SNO** | **Varchar(8)** | **not null(主键、外键)** | **寝室人数(默认为0)** |  |

表4-2　 来访人信息表VER的结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表中列名** | **数据类型** | **可否为空** | **说明** | **约束** |
| **VName** | **Varchar(10)** | **not null(主键)** | **来访人姓名** |  |
| **VPhone** | **Char(11)** | **not null** | **来访人联系方式** |  |
| **VSFZH** | **Char(18)** | **not null** | **来访人身份证号码** |  |

表4-2　来访登记表VS的结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表中列名** | **数据类型** | **可否为空** | **说明** | **约束** |
| **VNO** | **char(6)** | **not null(主键)** | **来访编号** |  |
| **SNO** | **Char(8)** | **not null(外键)** | **学号** |  |
| **VSFZH** | **Char(18)** | **not null(外键)** | **来访人身份证号** |  |
| **VTime** | **Datetime** | **not null** | **来访时间** |  |
| **Vleave** | **Datetime** | **not null** | **离开时间** |  |
| **VReason** | **Varchar(100)** | **not null** | **来访原因** |  |

表4-2　报修信息表F的结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **表中列名** | **数据类型** | **可否为空** | **说明** | **约束** |
| **FNO** | **Char(5)** | **not null(主键)** | **报修编号** |  |
| **FDetail** | **Varchar(100)** | **not null** | **报修详细** |  |
| **RNO** | **Varchar(6)** | **not null(外键)** | **寝室号** |  |
| **FTime** | **Datetime** | **not null** | **保修时间** |  |
| **FBool** | **Boolean** | **not null** | **是否解决(默认‘否’)** |  |
| **FRemark** | **Varchar(100)** | **not null** | **解决备注** |  |

**4.2.2 关系图**

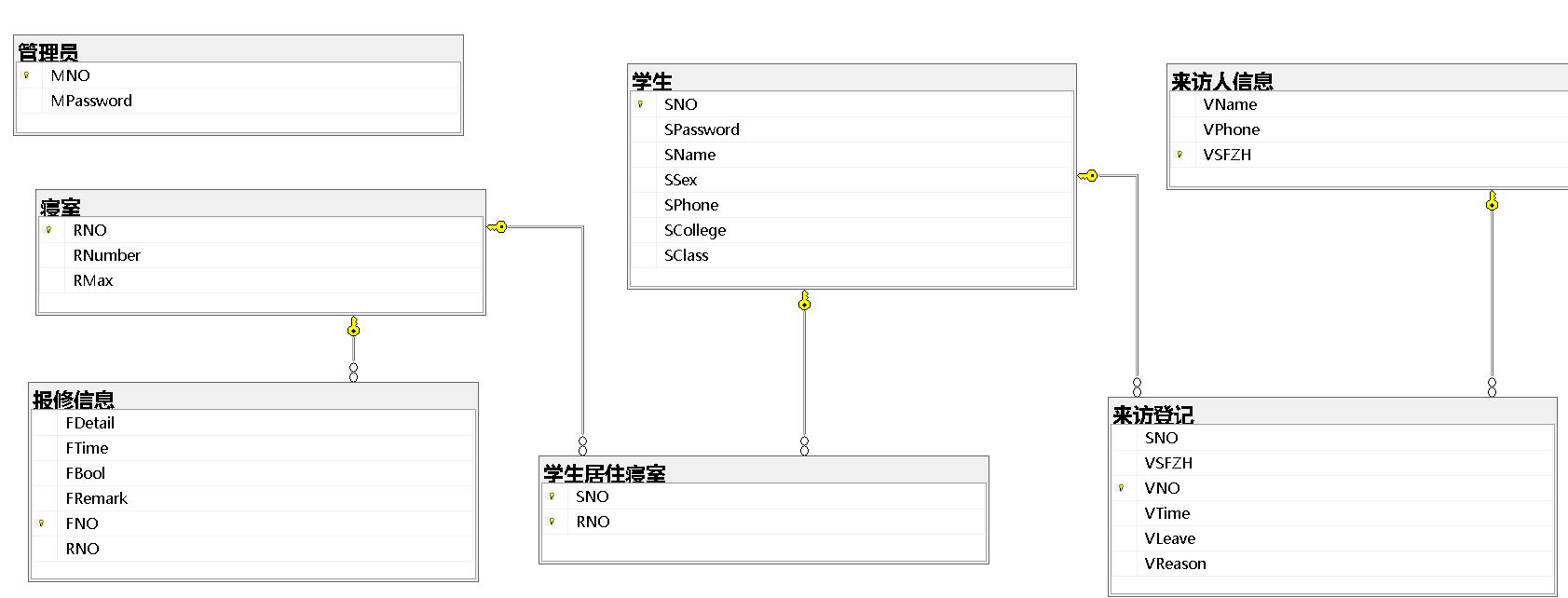


图4-3 SQL Server中的关系图

## 4.3 物理设计

数据库物理设计阶段的任务是根据具体计算机系统(DBMS和硬件等)的特点，为给定的数据库模型确定合理的存储结构和存取方法。所谓的“合理”主要有两个含义：一个是要使设计出的物理数据库占用较少的存储空间，另一个对数据库的操作具有尽可能高的速度。主要体现在后者。

(1)建立索引：

① 对学生表(Student)在学号(学号)属性列上建立聚集索引。

② 对寝室表(Dormitory)在寝室编号(寝室编号)属性列上建立聚集索引。

③ 对维修申请表(Repair)在报修编号(报修编号)属性列上建立聚集索引，在寝室编号(寝室编号)属性列上建立非聚集索引。

(2)存储结构

确定数据库的存储结构主要指确定数据的存放位置和存储结构，包括确定关系、索引、日志、备份等的存储安排及存储结构，以及确定系统存储参数的配置。

1.数据文件：主数据文件(MDF)：初始大小为10MB，增长方式为1MB，最大大小为无限制。

2.日志文件：初始大小为5MB，增长方式为512KB，最大大小为50MB。

## 4.4 数据库建立

4.4.1 创建数据库

USE master

GO

CREATE DATABASE dormitory\_management

ON

(

NAME = 'dormitory\_management',

FILENAME = 'C:\DatabaseFiles\dormitory\_management.mdf',

SIZE = 10MB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 64MB

)

LOG ON

(

NAME = 'dormitory\_management\_LOG',

FILENAME = 'C:\DatabaseFiles\dormitory\_management.ldf',

SIZE = 5MB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 64MB

)

GO

4.4.2 表的建立

/\*==============================================================\*/

/\* Table: 学生 \*/

/\*==============================================================\*/

create table 学生 (

SNO char(8) not null,

SPassword varchar(20) not null,

SName varchar(10) not null,

SSex char(2) not null default '男'

constraint CKC\_SSEX\_学生 check (SSex in ('男','女')),

SPhone char(11) not null,

SCollege varchar(30) not null,

SClass varchar(30) not null,

constraint PK\_学生 primary key nonclustered (SNO)

)

go

/\*==============================================================\*/

/\* Table: 学生居住寝室 \*/

/\*==============================================================\*/

create table 学生居住寝室 (

SNO char(8) not null,

RNO varchar(10) not null,

constraint PK\_学生居住寝室 primary key (SNO, RNO)

)

go

/\*==============================================================\*/

/\* Index: 学生居住寝室\_FK \*/

/\*==============================================================\*/

create index 学生居住寝室\_FK on 学生居住寝室 (

SNO ASC

)

go

/\*==============================================================\*/

/\* Index: 学生居住寝室2\_FK \*/

/\*==============================================================\*/

create index 学生居住寝室2\_FK on 学生居住寝室 (

RNO ASC

)

go

/\*==============================================================\*/

/\* Table: 寝室 \*/

/\*==============================================================\*/

create table 寝室 (

RNO varchar(10) not null,

RNumber int not null default 0,

RMax int null,

constraint PK\_寝室 primary key nonclustered (RNO)

)

go

/\*==============================================================\*/

/\* Table: 报修信息 \*/

/\*==============================================================\*/

create table 报修信息 (

FDetail varchar(100) not null,

FTime datetime not null,

FBool bit not null default 0,

FRemark varchar(100) null,

FNO char(5) not null,

RNO varchar(10) null,

constraint PK\_报修信息 primary key nonclustered (FNO)

)

go

/\*==============================================================\*/

/\* Index: Relationship\_3\_FK \*/

/\*==============================================================\*/

create index Relationship\_3\_FK on 报修信息 (

RNO ASC

)

go

/\*==============================================================\*/

/\* Table: 来访人信息 \*/

/\*==============================================================\*/

create table 来访人信息 (

VName varchar(10) not null,

VPhone char(11) not null,

VSFZH varchar(18) not null,

constraint PK\_来访人信息 primary key nonclustered (VSFZH)

)

go

/\*==============================================================\*/

/\* Table: 来访登记 \*/

/\*==============================================================\*/

create table 来访登记 (

SNO char(8) not null,

VSFZH varchar(18) not null,

VNO char(6) not null,

VTime datetime not null,

VLeave datetime not null,

VReason varchar(100) not null,

constraint PK\_来访登记 primary key (VNO)

)

go

/\*==============================================================\*/

/\* Index: 来访登记\_FK \*/

/\*==============================================================\*/

create index 来访登记\_FK on 来访登记 (

SNO ASC

)

go

/\*==============================================================\*/

/\* Index: 来访登记2\_FK \*/

/\*==============================================================\*/

create index 来访登记2\_FK on 来访登记 (

VSFZH ASC

)

go

/\*==============================================================\*/

/\* Table: 管理员 \*/

/\*==============================================================\*/

create table 管理员 (

MNO char(8) not null,

MPassword varchar(20) not null,

constraint PK\_管理员 primary key nonclustered (MNO)

)

go

alter table 学生居住寝室

add constraint FK\_学生居住寝室\_学生居住寝室\_学生 foreign key (SNO)

references 学生 (SNO)

go

alter table 学生居住寝室

add constraint FK\_学生居住寝室\_学生居住寝室2\_寝室 foreign key (RNO)

references 寝室 (RNO)

go

alter table 报修信息

add constraint FK\_报修信息\_RELATIONS\_寝室 foreign key (RNO)

references 寝室 (RNO)

go

alter table 来访登记

add constraint FK\_来访登记\_来访登记\_学生 foreign key (SNO)

references 学生 (SNO)

go

alter table 来访登记

add constraint FK\_来访登记\_来访登记2\_来访人信息 foreign key (VSFZH)

references 来访人信息 (VSFZH)

go

4.4.3 索引的创建

/\* 索引: 学生表\*/

CREATE INDEX IX\_学生\_SName\_SSex ON 学生 (

SName ASC,

SSex ASC

)

GO

/\* 索引: 学生居住寝室表\*/

CREATE INDEX IX\_学生居住寝室\_SNO\_RNO ON 学生居住寝室 (

SNO ASC,

RNO ASC

)

GO

/\* 索引: 寝室表\*/

CREATE INDEX IX\_寝室\_RMax ON 寝室 (

RMax ASC

)

GO

/\* 索引: 报修信息表\*/

CREATE INDEX IX\_报修信息\_FTime\_FBool ON 报修信息 (

FTime ASC,

FBool ASC

)

GO

/\* 索引: 来访人信息表\*/

CREATE INDEX IX\_来访人信息\_VName\_VPhone ON 来访人信息 (

VName ASC,

VPhone ASC

)

GO

/\* 索引: 来访登记表\*/

CREATE INDEX IX\_来访登记\_VTime\_VLeave ON 来访登记 (

VTime ASC,

VLeave ASC

)

GO

/\* 索引: 管理员表\*/

CREATE INDEX IX\_管理员\_MPassword ON 管理员 (

MPassword ASC

)

GO

## 4.5 数据初始化

-- 更新或插入寝室信息

-- 确保所有寝室的最大人数为6，并初始化当前人数为0（假设为统计动态更新）

INSERT INTO 寝室 (RNO, RNumber, RMax)

VALUES

('DY-20-2212', 0, 6),

('DY-21-3314', 0, 6),

('XY-03-4427', 0, 6),

('DY-22-6609', 0, 6),

('DY-5-2-101A', 0, 6),

('DY-5-2-101B', 0, 6),

('DY-22-6610', 0, 6),

('DY-22-6611', 0, 6),

('DY-23-216', 0, 6),

('DY-24-404', 0, 6)

-- 插入学生信息

INSERT INTO 学生 (SNO, SPassword, SName, SSex, SPhone, SCollege, SClass)

VALUES

('20233592', 'db20233592', '刘斌', '男', '18670397763', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233593', 'db20233593', '刘逸潇', '男', '18688747878', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233589', 'db20233589', '蒋东林', '男', '18670397763', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233590', 'db20233590', '李方斐', '男', '15314405998', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233588', 'db20233588', '何天顺', '男', '18273939767', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233585', 'db20233585', '邓家和', '男', '19870378041', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233597', 'db20233597', '粟祖杭', '男', '18747306083', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233599', 'db20233599', '王晟', '男', '18174270385', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233596', 'db20233596', '彭文杰', '男', '17887476894', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233594', 'db20233594', '罗振宇', '男', '17673922913', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233595', 'db20233595', '孟轶涵', '男', '18273703623', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233602', 'db20233602', '鄢郅恒', '男', '18684718681', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233606', 'db20233606', '朱法成', '男', '15839867583', '计算机与数学学院', '2023计算机科学与技术4班'),

('20233604', 'db20233604', '袁明琪', '男', '15637106726', '计算机与数学学院', '2023计算机科学与技术4班');

-- 插入学生居住寝室信息

INSERT INTO 学生居住寝室 (SNO, RNO)

VALUES

('20233592', 'DY-22-6609'),

('20233593', 'DY-22-6609'),

('20233589', 'DY-22-6609'),

('20233590', 'DY-22-6609'),

('20233588', 'DY-22-6609'),

('20233585', 'DY-22-6609'),

('20233597', 'DY-22-6610'),

('20233599', 'DY-22-6610'),

('20233596', 'DY-22-6610'),

('20233594', 'DY-22-6610'),

('20233595', 'DY-22-6610'),

('20233602', 'DY-22-6610'),

('20233606', 'DY-22-6611'),

('20233604', 'DY-22-6611');

-- 初始化报修信息表

INSERT INTO 报修信息 (FDetail, FTime, FBool, FRemark, FNO, RNO)

VALUES

('水管漏水', '2024-01-15 10:00:00', 0, '等待维修', '00001', 'DY-22-6609'),

('灯泡损坏', '2024-01-16 15:00:00', 1, '维修完成', '00002', 'DY-22-6610'),

('门锁松动', '2024-01-17 18:00:00', 0, '等待维修', '00003', 'DY-22-6611'),

('暖气故障', '2024-01-18 14:30:00', 1, '已解决', '00004', 'DY-23-216');

-- 初始化来访人信息表

INSERT INTO 来访人信息 (VName, VPhone, VSFZH)

VALUES

('赵六', '13812345678', '123456789012345678'),

('钱七', '13887654321', '234567890123456789'),

('孙八', '13911223344', '345678901234567890'),

('李九', '13999887766', '456789012345678901');

-- 初始化来访登记表

INSERT INTO 来访登记 (SNO, VSFZH, VNO, VTime, VLeave, VReason)

VALUES

('20233592', '123456789012345678', '0001', '2024-01-20 10:00:00', '2024-01-20 12:00:00', '探访同学'),

('20233593', '234567890123456789', '0002', '2024-01-21 14:00:00', '2024-01-21 15:30:00', '送文件'),

('20233594', '345678901234567890', '0003', '2024-01-22 09:00:00', '2024-01-22 11:00:00', '交还书籍'),

('20233595', '456789012345678901', '0004', '2024-01-23 16:00:00', '2024-01-23 17:00:00', '讨论项目');

-- 初始化管理员表

INSERT INTO 管理员 (MNO, MPassword)

VALUES

('admin', '001'),

## 4.6 数据库用户权限管理

该系统设置三种类型的用户

(1) 管理员(administrator)

管理员负责系统的全面管理与维护，权限范围覆盖了整个数据库的关键模块，具体包括：添加、修改和删除学生信息；添加、修改和删除寝室信息以及学生寝室分配关系；管理访客信息和访客登记，包括访客记录的增删改查；管理寝室报修信息，包括报修记录的增删改查。

权限描述： 管理员具备完整的读写权限，能够对所有数据表执行 SELECT、INSERT、UPDATE 和 DELETE 操作。

(2) 学生(student)

学生是系统的主要使用者之一，其权限被严格限定在与其相关的数据范围内，具体包括：查看自己的基本信息（如姓名、性别、学号、寝室号等）；查看自己所在寝室的报修信息及解决状态；提交寝室报修申请（如寝室设施损坏、需要维修等）。

权限描述： 学生只能对自己的数据和所在寝室相关数据进行操作：允许查询与自身相关的信息；允许向报修信息表提交报修记录，但不能修改或删除已有记录。

(3) 访客(visitor)

访客的权限范围最小，仅限于查看和登记来访记录，具体包括：添加新的访客登记信息（如来访时间、离开时间和来访原因）；查看与自身相关的访客历史记录（如曾访问的学生、访问时间和离开时间）。

权限描述： 访客可以：向访客登记表新增来访记录；查询与自己相关的历史来访记录。；但不能修改或删除任何记录。

# 5 数据库编程设计

## 5.1 建立存储过程

(1) 新增学生

此存储过程用于向 学生 表中新增学生记录，同时初始化学生与寝室的关系（插入到 学生居住寝室 表中）。需要提供学生的基本信息（学号、姓名、性别等）和寝室编号。具体程序如下：

CREATE PROCEDURE SP\_ADD\_STUDENT

@SNO CHAR(8), -- 学号

@SPassword VARCHAR(20), -- 密码

@SName VARCHAR(10), -- 姓名

@SSex CHAR(2), -- 性别

@SPhone CHAR(11), -- 电话

@SCollege VARCHAR(30), -- 学院

@SClass VARCHAR(30), -- 班级

@RNO VARCHAR(10) -- 寝室编号

AS

BEGIN

-- 向学生表插入记录

INSERT INTO 学生 (SNO, SPassword, SName, SSex, SPhone, SCollege, SClass)

VALUES (@SNO, @SPassword, @SName, @SSex, @SPhone, @SCollege, @SClass);

-- 向学生居住寝室表插入记录

INSERT INTO 学生居住寝室 (SNO, RNO)

VALUES (@SNO, @RNO);

END;

GO

(2) 删除学生

此存储过程用于删除学生记录。将从 学生 表和 学生居住寝室 表中移除指定学号的学生，并更新寝室表中的入住人数。具体程序如下：

CREATE PROCEDURE SP\_DELETE\_STUDENT

@SNO CHAR(8) -- 学号

AS

BEGIN

-- 获取学生所在的寝室编号

DECLARE @RNO VARCHAR(10);

SELECT @RNO = RNO FROM 学生居住寝室 WHERE SNO = @SNO;

-- 从学生居住寝室表删除记录

DELETE FROM 学生居住寝室

WHERE SNO = @SNO;

-- 更新寝室的入住人数

UPDATE 寝室

SET RNumber = RNumber - 1

WHERE RNO = @RNO;

-- 从学生表删除记录

DELETE FROM 学生

WHERE SNO = @SNO;

END;

GO

(3) 报修登记

此存储过程用于向 报修信息 表中添加一条新的报修记录。需要提供寝室编号、报修内容等信息。具体程序如下：

CREATE PROCEDURE SP\_ADD\_REPAIR

@RNO VARCHAR(10), -- 寝室编号

@FDetail VARCHAR(100), -- 报修详情

@FTime DATETIME, -- 报修时间

@FNO CHAR(5) -- 报修编号

AS

BEGIN

-- 向报修信息表插入记录

INSERT INTO 报修信息 (FDetail, FTime, FBool, FRemark, FNO, RNO)

VALUES (@FDetail, @FTime, 0, NULL, @FNO, @RNO);

END;

GO

(4) 查看寝室入住情况

此存储过程用于查询指定寝室的入住情况，返回寝室的入住人数、最大人数以及该寝室的学生列表。具体程序如下：

CREATE PROCEDURE SP\_GET\_ROOM\_STATUS

@RNO VARCHAR(6) -- 寝室编号

AS

BEGIN

-- 查询寝室基本信息

SELECT RNO, RNumber, RMax

FROM 寝室

WHERE RNO = @RNO;

-- 查询该寝室的学生列表

SELECT S.SNO, S.SName, S.SSex, S.SPhone, S.SCollege, S.SClass

FROM 学生居住寝室 JR

JOIN 学生 S ON JR.SNO = S.SNO

WHERE JR.RNO = @RNO;

END;

GO

(5) 登记来访记录

CREATE PROCEDURE SP\_ADD\_VISIT

@SNO CHAR(8), -- 学生学号

@VSFZH VARCHAR(18), -- 来访人身份证号

@VNO CHAR(6), -- 来访编号

@VTime DATETIME, -- 来访时间

@VLeave DATETIME, -- 离开时间

@VReason VARCHAR(100) -- 来访原因

AS

BEGIN

-- 向来访登记表插入记录

INSERT INTO 来访登记 (SNO, VSFZH, VNO, VTime, VLeave, VReason)

VALUES (@SNO, @VSFZH, @VNO, @VTime, @VLeave, @VReason);

END;

GO

## 5.2 建立触发器

(1)增加学生到寝室触发器

当 学生居住寝室 表中新增一条学生入住记录后，触发器会自动更新 寝室 表中的入住人数（RNumber+1）。具体程序如下：

CREATE TRIGGER TR\_ADD\_STUDENT\_TO\_ROOM

ON 学生居住寝室

FOR INSERT

AS

UPDATE 寝室

SET RNumber = RNumber + 1

WHERE RNO IN (SELECT RNO FROM inserted)

GO

(2)移除学生触发器

当 学生居住寝室 表中删除一条学生入住记录后，触发器会自动更新 寝室 表中的入住人数（RNumber-1）。具体程序如下：

CREATE TRIGGER TR\_REMOVE\_STUDENT\_FROM\_ROOM

ON 学生居住寝室

FOR DELETE

AS

UPDATE 寝室

SET RNumber = RNumber - 1

WHERE RNO IN (SELECT RNO FROM deleted)

GO

(3)设置学生密码触发器

当 学生表中新增一条学生入住记录后，触发器会自动更新 学生 表中的学生密码为db+学号。具体程序如下：

CREATE TRIGGER TR\_SetDefaultPassword

ON 学生

AFTER INSERT

AS

BEGIN

-- 明确指定更新操作中列名的来源

UPDATE 学生

SET SPassword = 'db' + i.SNO

FROM 学生 s

INNER JOIN inserted i ON s.SNO = i.SNO;

END;

## 5.3 视图设计

(1) 寝室入住情况视图

该视图用于展示每个寝室的基本信息，包括寝室编号、当前入住人数、最大容量以及已入住学生的学号和姓名。便于管理员快速了解寝室的入住情况。具体程序如下：

CREATE VIEW View\_Room\_Status AS

SELECT

寝室.RNO AS 寝室编号,

寝室.RNumber AS 当前入住人数,

寝室.RMax AS 最大容量,

学生.SNO AS 学号,

学生.SName AS 姓名

FROM

寝室

LEFT JOIN 学生居住寝室 ON 寝室.RNO = 学生居住寝室.RNO

LEFT JOIN 学生 ON 学生居住寝室.SNO = 学生.SNO;

GO

(2) 报修信息视图

该视图展示了所有的报修记录，包括寝室编号、报修内容、报修时间、是否已解决（布尔值）、备注信息等，方便管理员查看当前报修进度。具体程序如下：

CREATE VIEW View\_Repair\_Info AS

SELECT

报修信息.FNO AS 报修编号,

报修信息.RNO AS 寝室编号,

报修信息.FDetail AS 报修内容,

报修信息.FTime AS 报修时间,

CASE WHEN 报修信息.FBool = 1 THEN '已解决' ELSE '未解决' END AS 解决状态,

报修信息.FRemark AS 备注

FROM

报修信息;

GO

(3) 学生信息视图

该视图整合了学生的基本信息以及他们的寝室编号，便于快速查询学生与寝室的对应关系。具体程序如下：

CREATE VIEW View\_Student\_Info AS

SELECT

学生.SNO AS 学号,

学生.SName AS 姓名,

学生.SSex AS 性别,

学生.SPhone AS 电话,

学生.SCollege AS 学院,

学生.SClass AS 班级,

学生居住寝室.RNO AS 寝室编号

FROM

学生

LEFT JOIN 学生居住寝室 ON 学生.SNO = 学生居住寝室.SNO;

GO

(4) 来访记录视图

该视图展示所有的来访记录，包括来访人的姓名、身份证号、联系电话、来访的学生信息、来访时间、离开时间和来访原因。便于管理员跟踪所有来访记录。具体程序如下：

CREATE VIEW View\_Visit\_Record AS

SELECT

来访登记.VNO AS 来访编号,

来访人信息.VName AS 来访人姓名,

来访人信息.VSFZH AS 身份证号,

来访人信息.VPhone AS 来访人电话,

学生.SNO AS 学生学号,

学生.SName AS 学生姓名,

来访登记.VTime AS 来访时间,

来访登记.VLeave AS 离开时间,

来访登记.VReason AS 来访原因

FROM

来访登记

LEFT JOIN 来访人信息 ON 来访登记.VSFZH = 来访人信息.VSFZH

LEFT JOIN 学生 ON 来访登记.SNO = 学生.SNO;

GO

(5) 未满寝室视图

该视图展示当前未满的寝室，包括寝室编号、当前入住人数和最大容量，便于管理员安排新生入住。具体程序如下：

CREATE VIEW View\_Unfilled\_Room AS

SELECT

RNO AS 寝室编号,

RNumber AS 当前入住人数,

RMax AS 最大容量,

(RMax - RNumber) AS 剩余可入住人数

FROM

寝室

WHERE

RNumber < RMax;

GO

# 6 系统实现

## 6.1 系统功能模块图

本系统划分为5个子系统，系统功能模块图如图6-1所示。

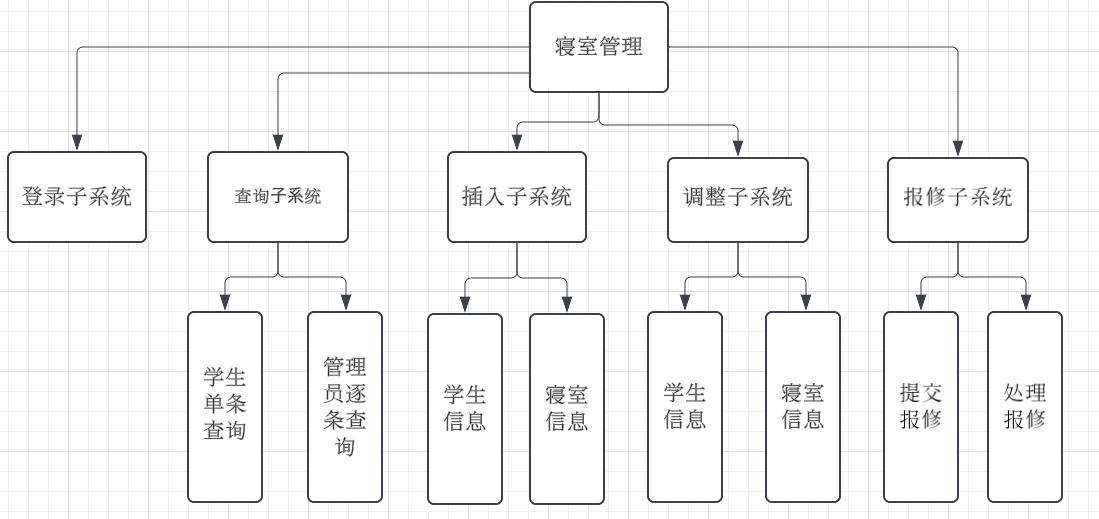


图6-1系统功能模块图

宿舍管理系统五个子系统的功能描述如下：

① 登录子系统

可以实现管理员和超级管理员登录系统，为宿舍系统的其他功能提供识别和权限验证。

功能包括：学生登录、管理员登录、提示错误的帐号或密码

② 查询子系统

为管理员和学生提供便捷的数据查询，包括单条查询和多条件综合查询。

功能包括：学生单条查询：可以根据学生的学号进行详细信息查询，包括学生姓名，学院，班级，请假信息等。宿舍单条查询：可查询指定宿舍的位置，现在入住人数，最大容量，连续状态。多条件查询：根据多种条件综合查询，如学生姓名和宿舍编号。

③ 插入子系统

用于新增学生和宿舍相关数据，实现入住和宿舍信息的快速插入。

功能包括：新增学生信息：学生基础信息（学号，姓名，性别，班级，身份证号，联系方式）插入。学生入住宿舍：列明宿舍编号，根据入住运行和管理员的指定全自动更新。

④ 调整子系统

调整子系统用于调整学生宿舍调谢，和更新宿舍列表。

功能包括：学生宿舍调整：根据学生需求，重新定位，宿舍进行更换。宿舍列表更新：学生调谢后，宿舍子系统自动更新，包括宿舍应容量，学生入住情况。

⑤ 报修子系统

报修子系统用于宿舍局部问题提交和处理，实现尽快更新。

功能包括：提交报修：提交问题的详细资料，如问题内容，报修时间，宿舍编号，报修编号。处理报修：报修管理员根据报修内容，尽快处理，并进行状态更新，如“已解决”或“応处理中”。

## 6.2 开发平台和开发工具

1. **前台开发平台的选择：idea（java）**
2. **java与数据库的连接**

JDBC是JAVA访问数据库的主要方式。

① JDBC 的数据访问方式

JDBC 提供了多种方式来访问不同类型的数据库：  
a) 通过ODBC桥接  
使用 sun.jdbc.odbc.JdbcOdbcDriver 驱动访问支持 ODBC 协议的数据库（已被废弃，较少使用）。

b) 直接通过JDBC驱动  
使用各数据库官方提供的驱动程序（如 SQL Server 的 com.microsoft.sqlserver.jdbc.SQLServerDriver）直接访问数据库。

c) 通过连接池技术  
使用连接池（如 HikariCP）提升数据库访问性能，适用于高并发场景。

② Java 访问数据库的具体过程

a) 创建数据库连接  
在 Java 中，使用 DriverManager 类与数据库建立连接。需要提供一个连接字符串，定义数据库的类型、位置以及其他信息。这些属性通常以分号分隔，包括以下关键属性：

Data Source（serverName 或 HostName）：指定 SQL Server 数据库所在计算机名称或 IP 地址；

databaseName：指定连接的 SQL Server 数据库名称；

User ID 和 Password：用于指定 SQL Server 的登录账号和密码；

具体实现代码如下：

import java.sql.\*;  
import java.util.ArrayList;  
public class DatabaseHelper {  
 public static Connection getConnection() {  
 try {  
 return DriverManager.*getConnection*("jdbc:sqlserver://localhost:1433;databaseName=dormitory\_management;user=sa;password=abc;encrypt=false");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }

}

## 6.3 寝室管理模块的实现

①宿舍管理系统登录界面如图10-2



图10-2宿舍管理系统登录界面图

②宿舍管理系统主界面如图10-3、4、5



图10-3宿舍管理系统管理员端主界面

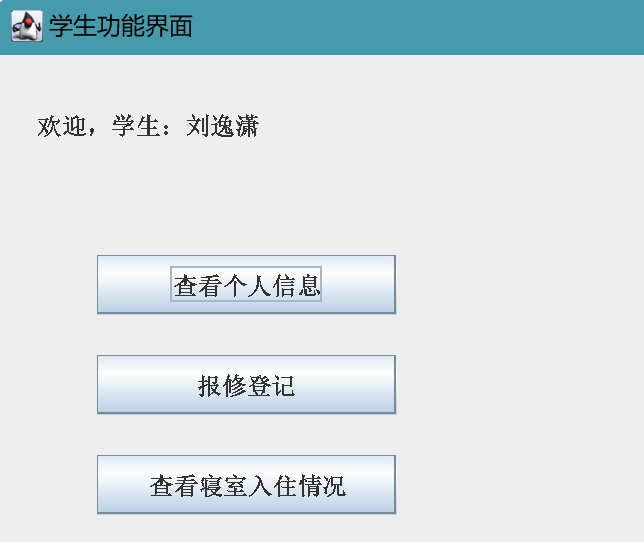


图10-4宿舍管理系统学生端主界面



图10-5宿舍管理系统访客端主界面

下面给出管理员的查询、插入、删除、更新模块的代码：

③查看界面如图10-6、7、8、9、10、11、12

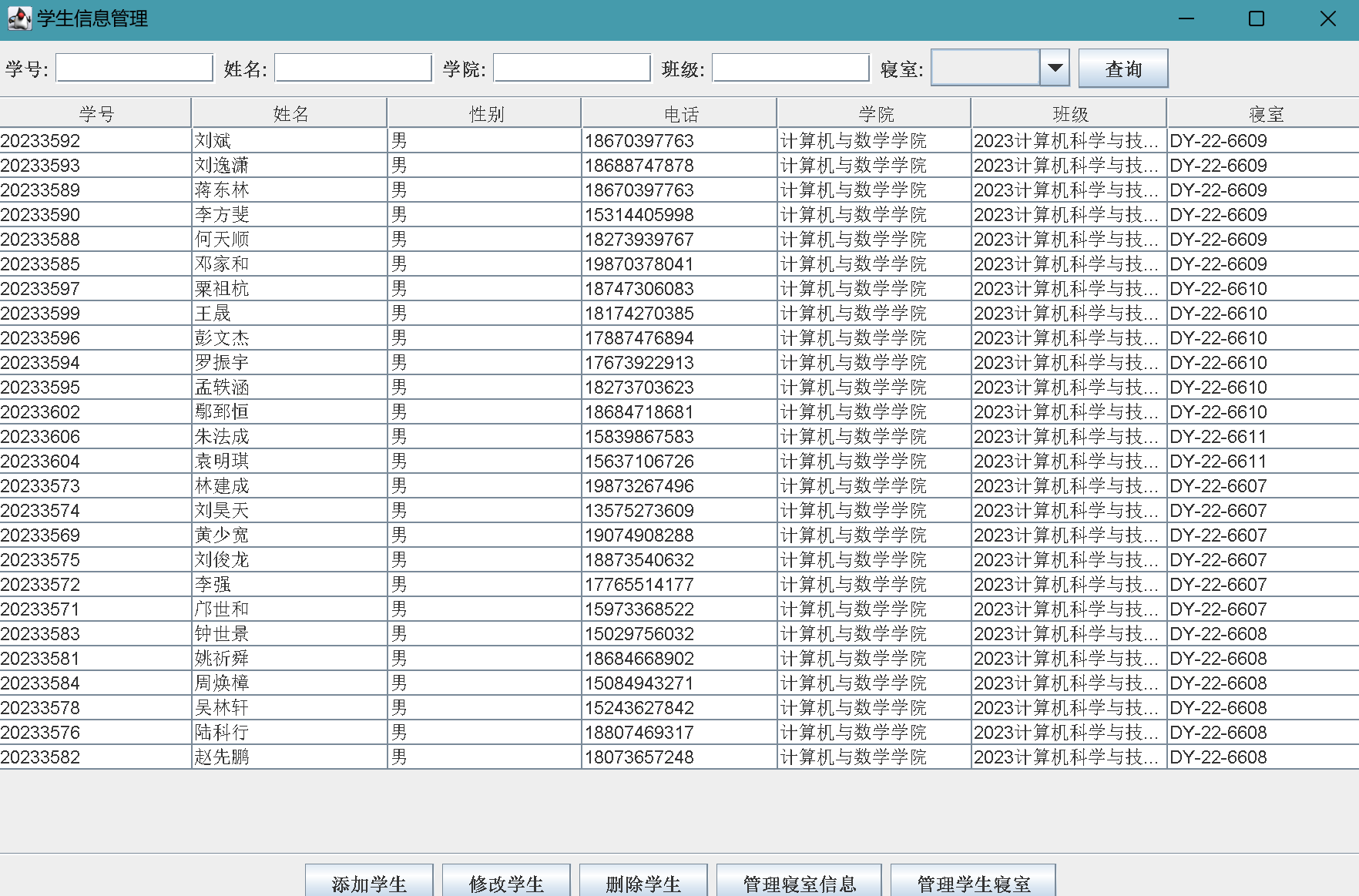


图10-6查看学生信息界面



图10-7查看报修信息界面

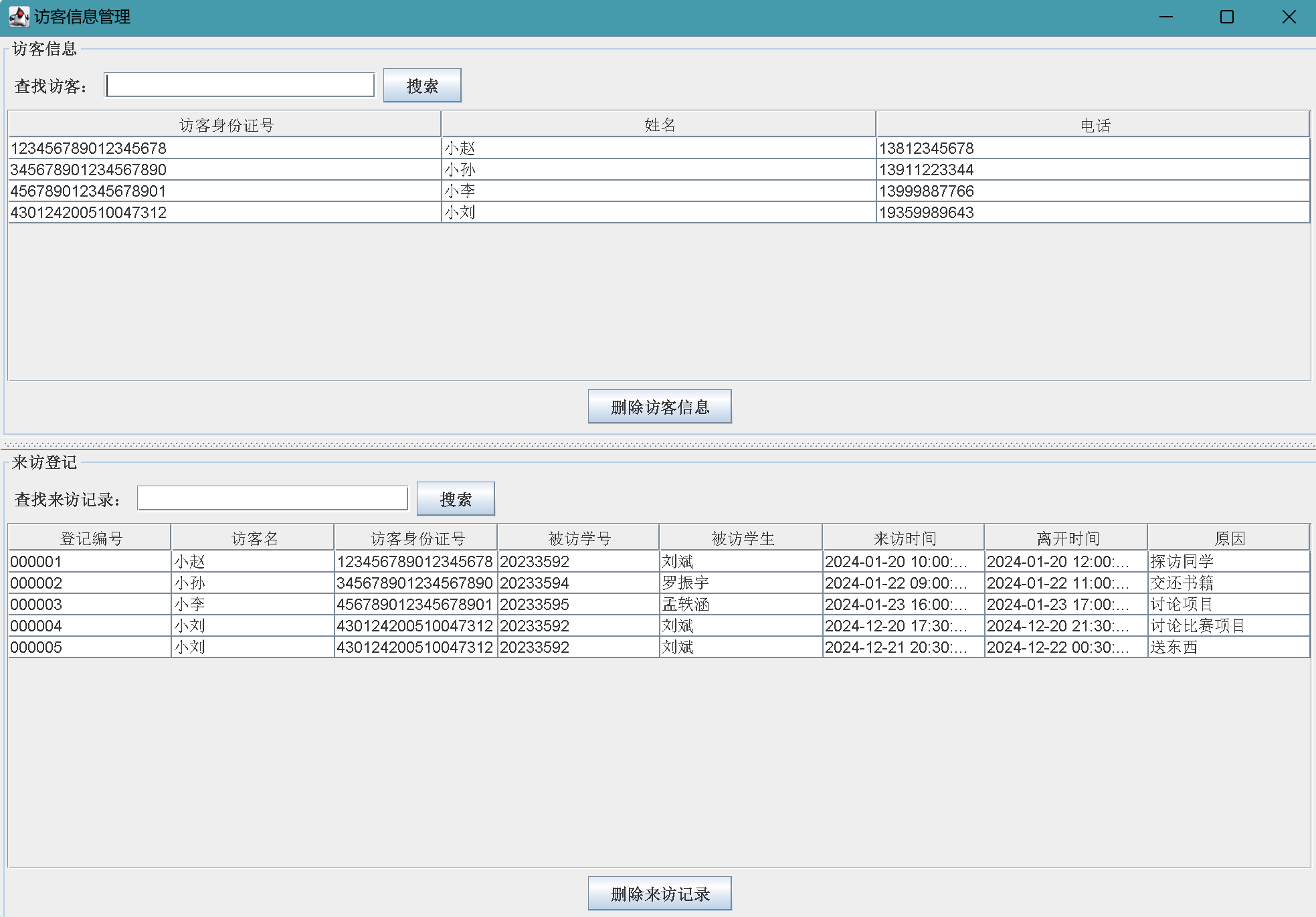


图10-8查看来访信息界面



图10-9学生查看个人信息界面

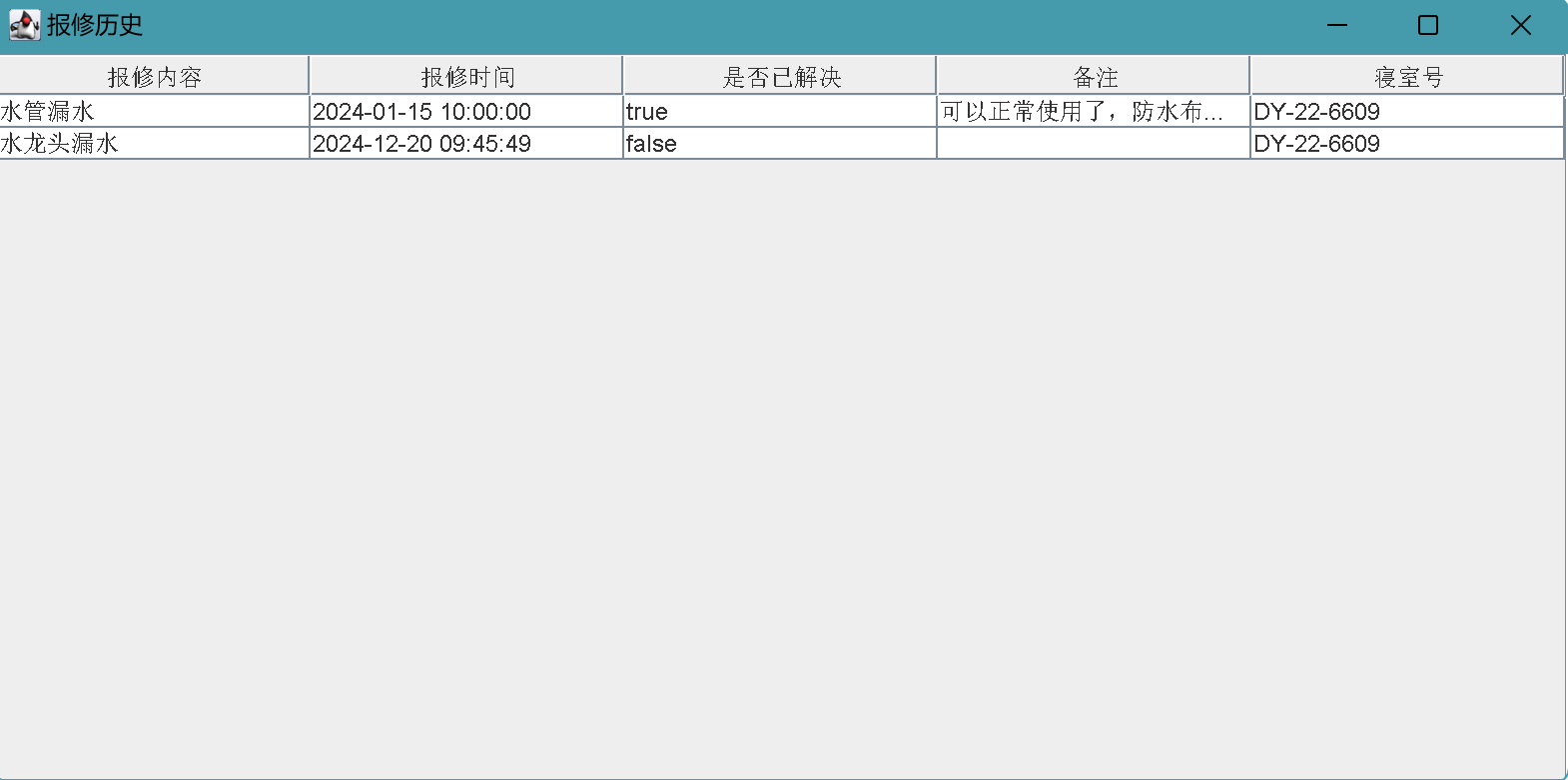


图10-10学生查看报修信息界面

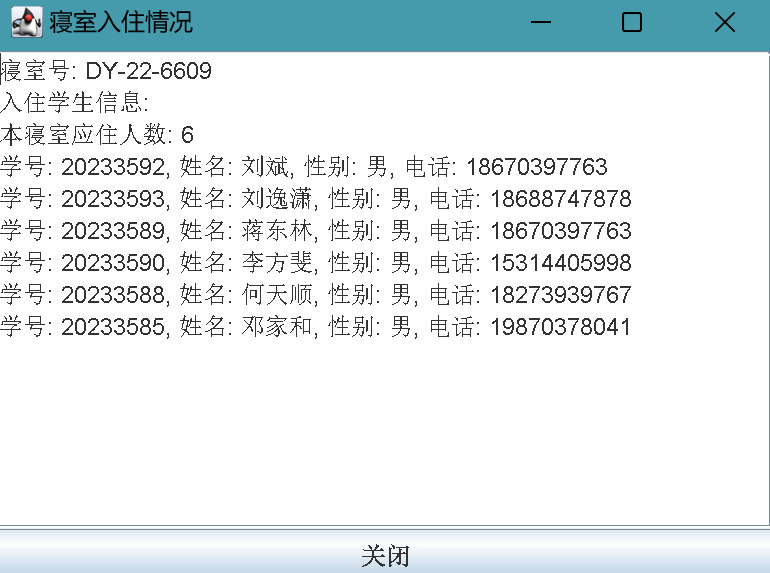


图10-11学生查看寝室入住信息界面



图10-12访客查看来访历史信息界面

④添加界面如图10-13、14、15、16、17所示：



图10-13管理员添加学生界面



图10-14管理员添加寝室运行界面

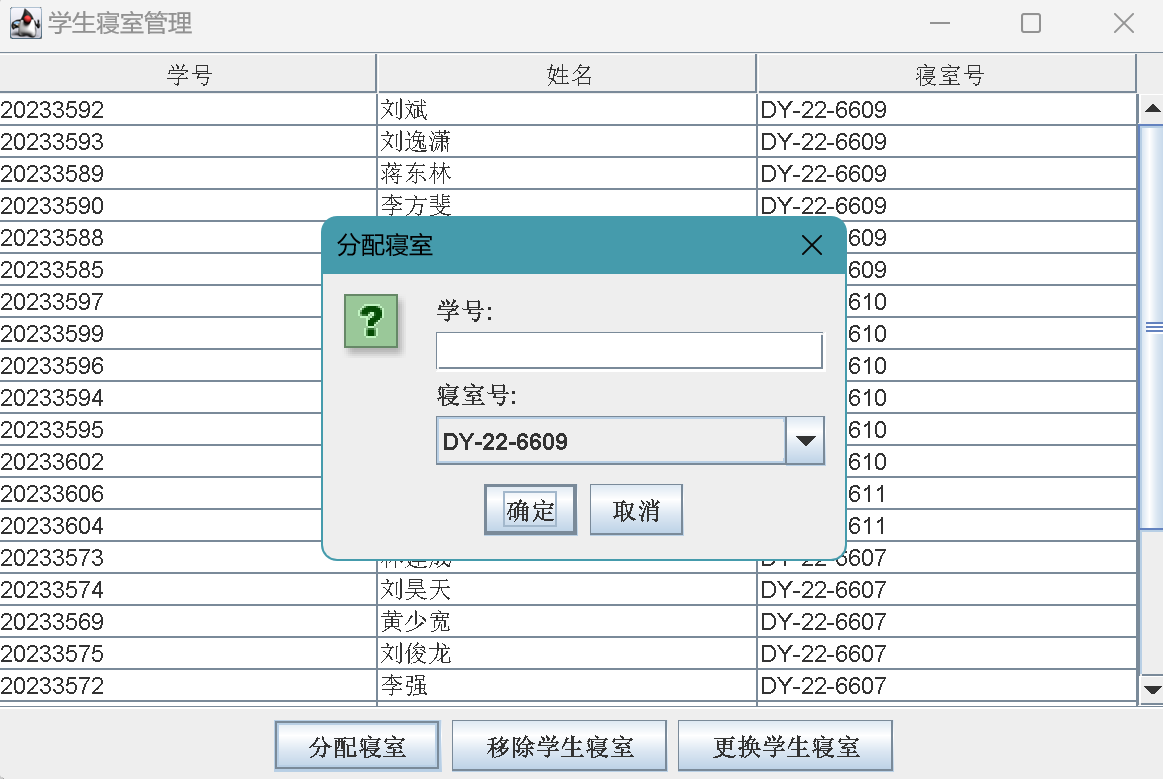


图10-15管理员添加学生-寝室界面



图10-16学生添加报修界面



图10-17访客添加来访记录界面

⑤删除运行界面如图10-18、19、20所示

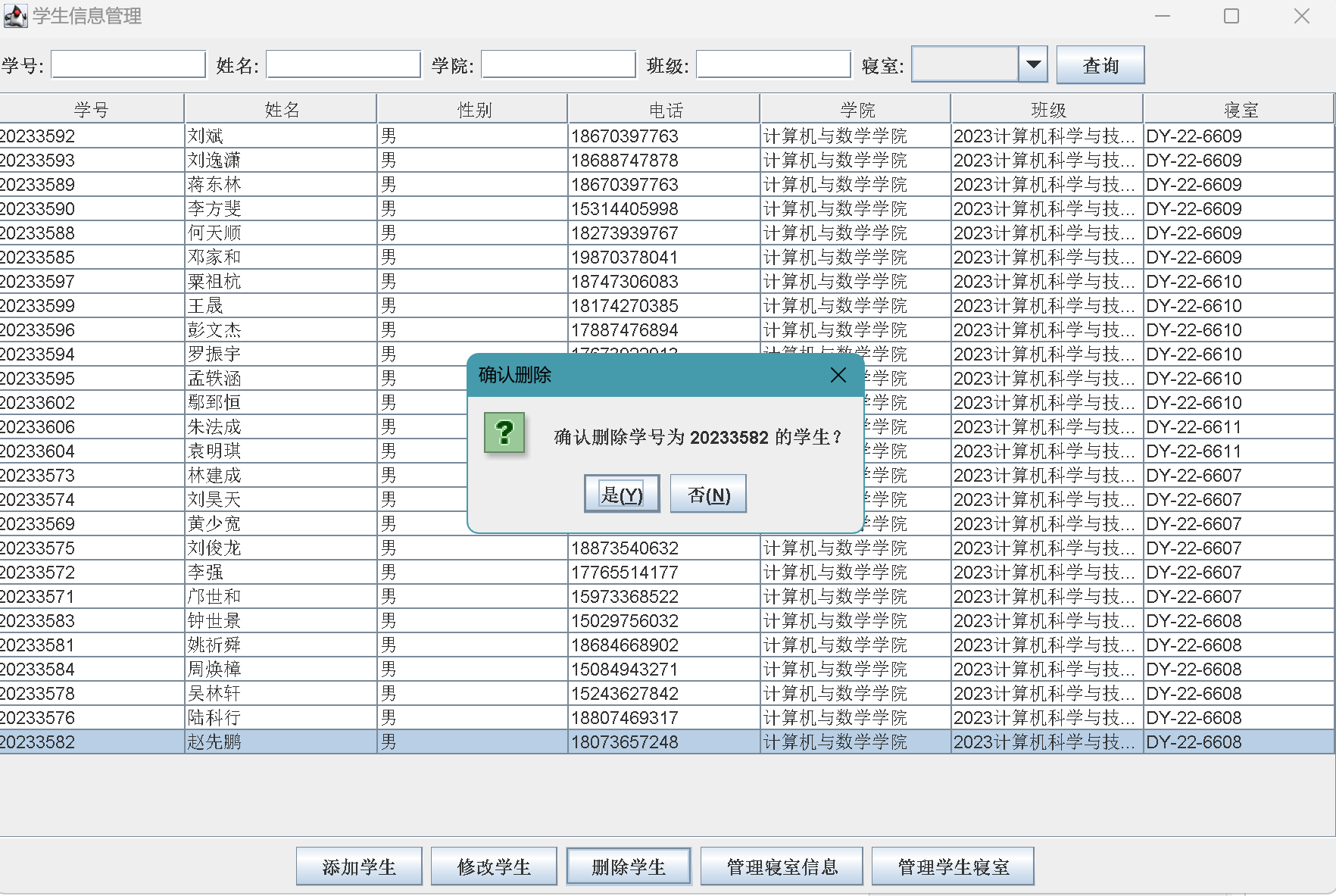


图10-18删除学生界面

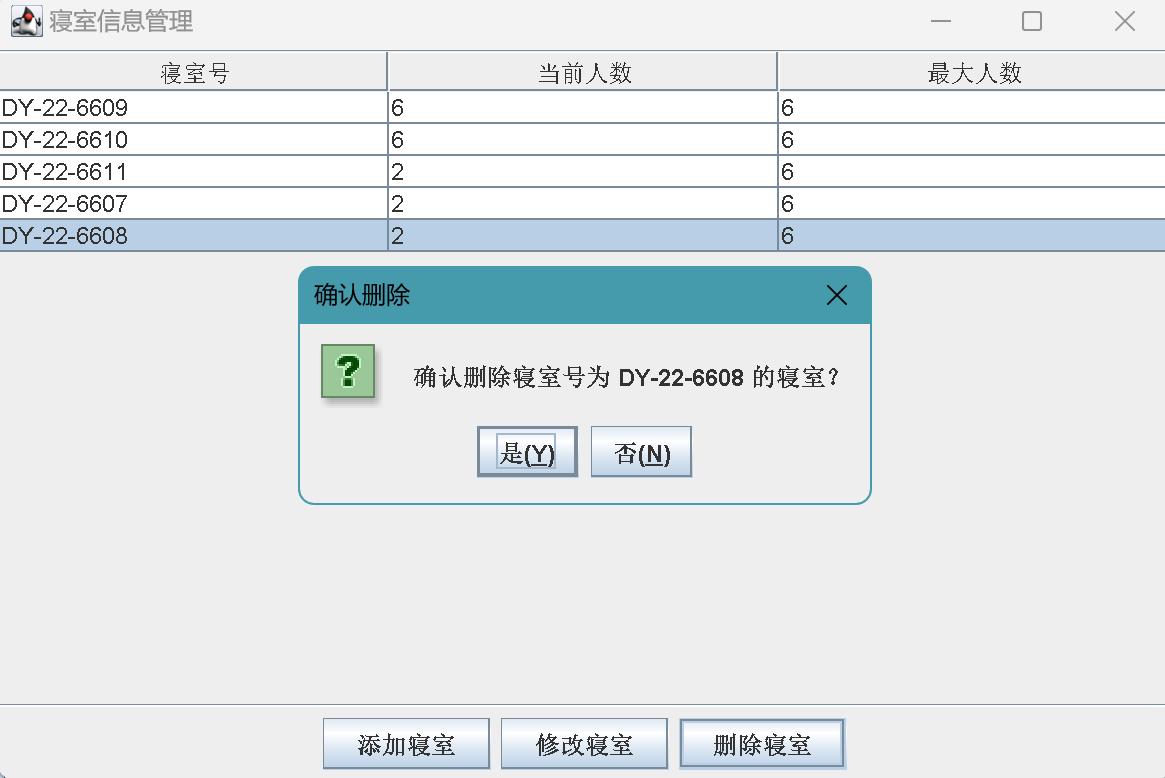


图10-19删除寝室界面



图10-20删除学生-寝室界面

⑥修改运行界面如图10-21、22、23、24所示

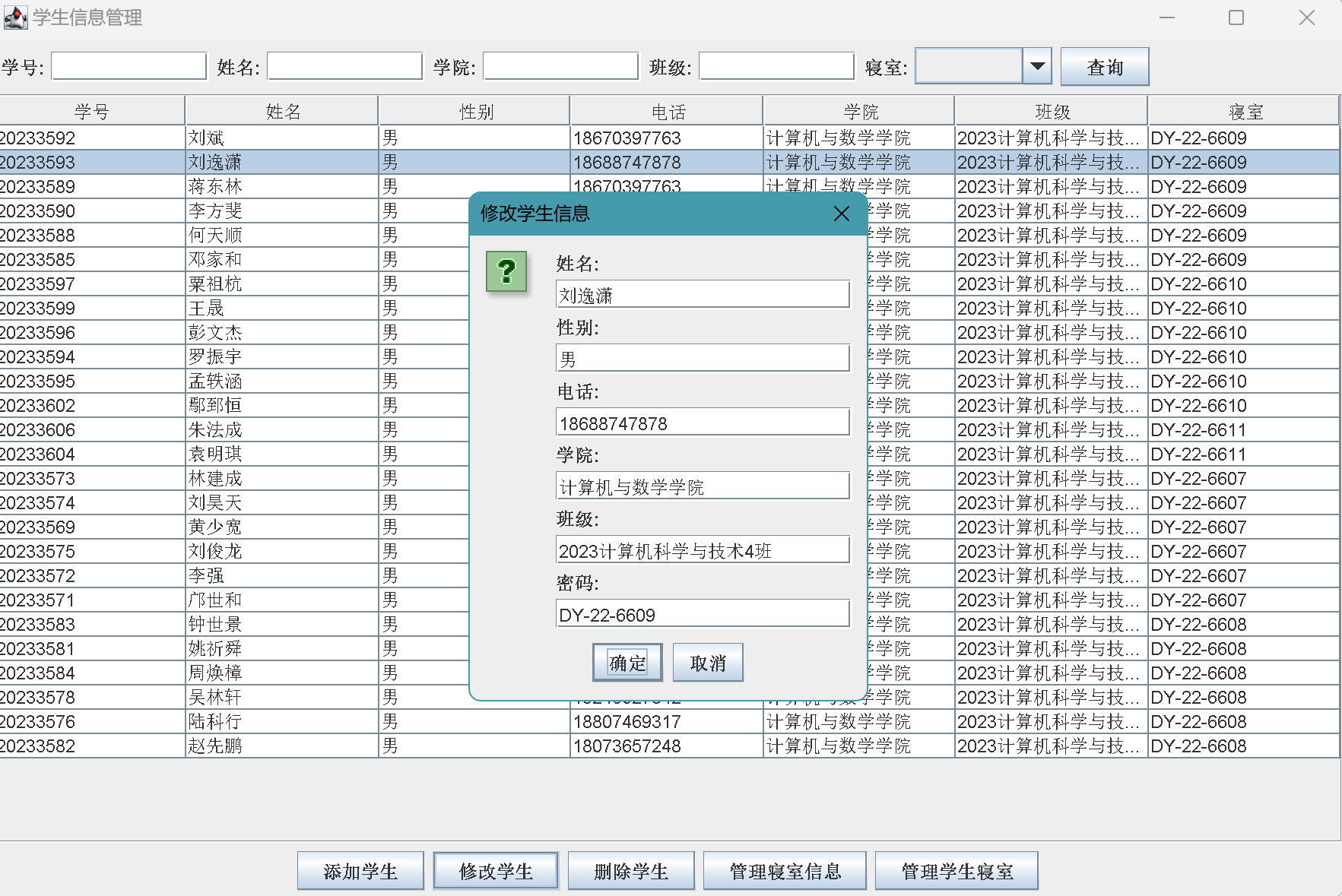


图10-21概念梨园修改学生信息界面



图10-22管理员修改寝室信息界面

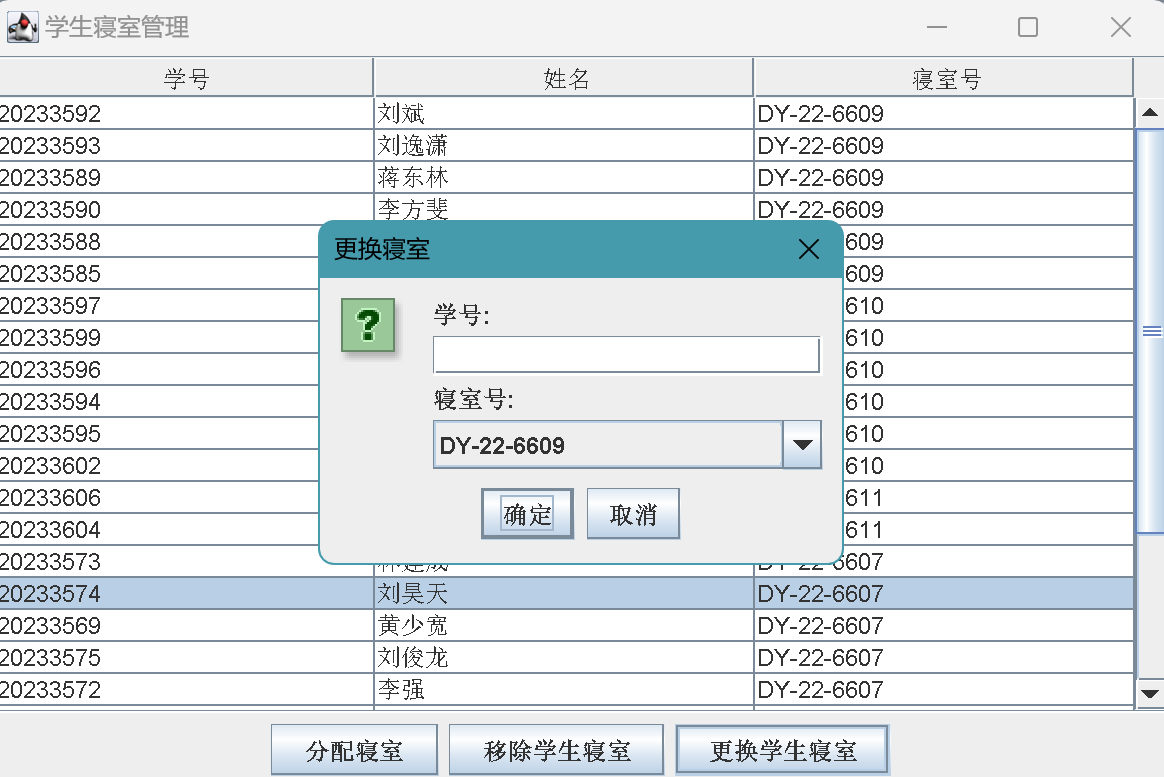


图10-7管理员修改学生-寝室分配界面

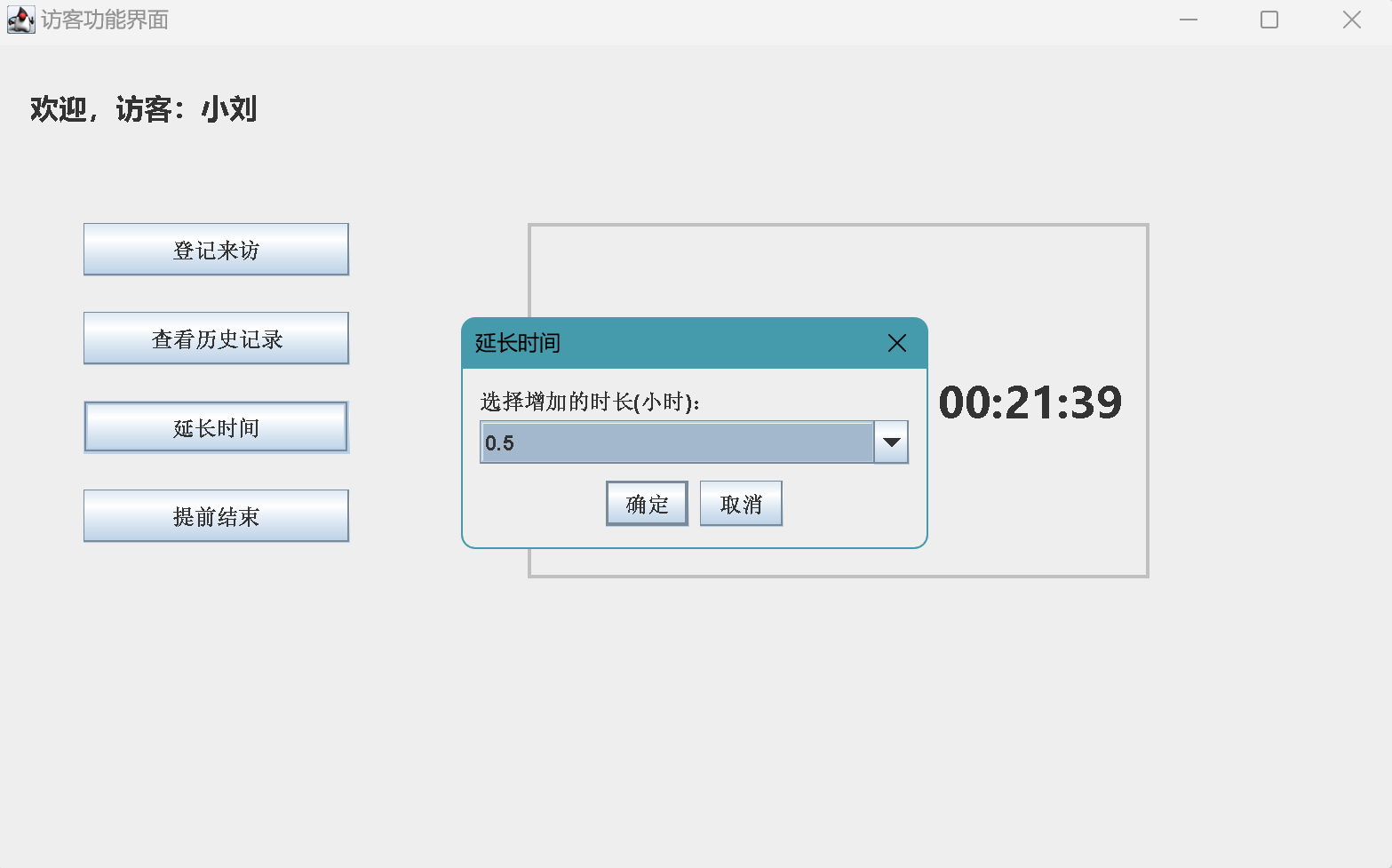


图10-7访客修改访问时长界面

⑥数据库总代码

import java.sql.\*;  
import java.util.ArrayList;  
import java.util.List;  
public class DatabaseHelper {  
 public static Connection getConnection() {  
 try {  
 return DriverManager.getConnection("jdbc:sqlserver://localhost:1433;databaseName=dormitory\_management;user=sa;password=abc;encrypt=false");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean login(String role, String username, String password) {  
 String sql = switch (role) {  
 case "管理员" -> "SELECT \* FROM 管理员 WHERE MNO = ? AND MPassword = ?";  
 case "学生" -> "SELECT \* FROM 学生 WHERE SNO = ? AND SPassword = ?";  
 default -> null;  
 };  
  
 if (sql == null) return false;  
  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, username);  
 stmt.setString(2, password);  
 ResultSet rs = stmt.executeQuery();//执行查询  
 return rs.next();//如果有下一行则返回true  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean registerVisitor(String username, String password) {  
 String sql = "INSERT INTO 来访人信息 (VName, VPassword) VALUES (?, ?)";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, username);  
 stmt.setString(2, password);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 /\*\*  
 \* 访客登记来访  
 \*/  
 public static boolean registerVisit(String id, String sno, String reason, java.sql.Timestamp startTime, java.sql.Timestamp endTime) {  
 String sql = "INSERT INTO 来访登记 (VSFZH, SNO, VReason, VTime, VLeave, VNO) VALUES (?, ?, ?, ?, ?, ?)";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, id);  
 stmt.setString(2, sno);  
 stmt.setString(3, reason);  
 stmt.setTimestamp(4, startTime); // 使用 setTimestamp 方法  
 stmt.setTimestamp(5, endTime); // 使用 setTimestamp 方法  
 // 生成来访编号,保证唯一性，取已有编号最大值加1  
 String getMaxVNOQuery = "SELECT MAX(VNO) AS MaxVNO FROM 来访登记";  
 ResultSet rs = conn.createStatement().executeQuery(getMaxVNOQuery);  
 String nextVNO = "000001"; // 默认从 00001 开始  
 if (rs.next() && rs.getString("MaxVNO") != null) {  
 String maxVNO = rs.getString("MaxVNO"); // 获取最大编号  
 int maxVNOInt = Integer.parseInt(maxVNO); // 转换为整数  
 nextVNO = String.format("%06d", maxVNOInt + 1); // 格式化为6位  
 }  
 stmt.setString(6, nextVNO);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static Timestamp getVisitorLeaveTime(String visitorId) {  
 String sql = "SELECT VLeave FROM 来访登记 WHERE VSFZH = ? AND VLeave > GETDATE()";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, visitorId);  
 ResultSet rs = stmt.executeQuery();  
 return rs.next() ? rs.getTimestamp("VLeave") : null;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean extendVisitorLeaveTime(String visitorId, double additionalHours) {  
 String sql = "UPDATE 来访登记 SET VLeave = DATEADD(HOUR, ?, VLeave) WHERE VSFZH = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setDouble(1, additionalHours);  
 stmt.setString(2, visitorId);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static List<Object[]> getVisitorHistory(String visitorId) {  
 String sql = "SELECT VNO, SNO, VReason, VTime, VLeave FROM 来访登记 WHERE VSFZH = ?";  
 List<Object[]> historyData = new ArrayList<>();  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, visitorId);  
 try (ResultSet rs = stmt.executeQuery()) {  
 while (rs.next()) {  
 historyData.add(new Object[]{  
 rs.getString("VNO"),  
 rs.getString("SNO"),  
 rs.getString("VReason"),  
 rs.getString("VTime"),  
 rs.getString("VLeave")  
 });  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return historyData;  
 }  
  
 public static ResultSet getStudentInfo(String studentSNO) {  
 String sql = "SELECT \* FROM 学生 WHERE SNO = ?";  
 try {  
 Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 stmt.setString(1, studentSNO);  
 return stmt.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean registerRepair(String roomNumber, String detail) {  
 String getMaxFNOQuery = "SELECT MAX(FNO) AS MaxFNO FROM 报修信息";//查询最大编号  
 String insertQuery = "INSERT INTO 报修信息 (FDetail, FTime, FBool, RNO, FNO) VALUES (?, GETDATE(), 0, ?, ?)";  
 try (Connection conn = getConnection();  
 PreparedStatement getMaxStmt = conn.prepareStatement(getMaxFNOQuery);  
 PreparedStatement insertStmt = conn.prepareStatement(insertQuery)) {  
 // 查询当前最大 FNO  
 ResultSet rs = getMaxStmt.executeQuery();//执行查询  
 String nextFNO = "00001"; // 默认从 00001 开始  
 if (rs.next() && rs.getString("MaxFNO") != null) {  
 String maxFNO = rs.getString("MaxFNO"); // 获取最大编号  
 int maxFNOInt = Integer.parseInt(maxFNO); // 转换为整数  
 nextFNO = String.format("%05d", maxFNOInt + 1); // 格式化为5位  
 }  
 // 插入新报修信息  
 insertStmt.setString(1, detail);  
 insertStmt.setString(2, roomNumber);  
 insertStmt.setString(3, nextFNO);  
 return insertStmt.executeUpdate() > 0;  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static ResultSet getRoomStatus(String roomNumber) {  
 String sql = "SELECT 学生.SNO, 学生.SName, 学生.SSex, 学生.SPhone " +  
 "FROM 学生居住寝室 " +  
 "JOIN 学生 ON 学生居住寝室.SNO = 学生.SNO " +  
 "WHERE 学生居住寝室.RNO = ?";  
 try {  
 Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 stmt.setString(1, roomNumber);  
 return stmt.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static ResultSet getRepairInfo() {  
 String sql = "SELECT 报修信息.FNO, 报修信息.FDetail, 报修信息.FTime, 报修信息.FBool, 寝室.RNO " +  
 "FROM 报修信息 " +  
 "JOIN 寝室 ON 报修信息.RNO = 寝室.RNO";  
 try {  
 Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 return stmt.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean updateRepairStatus(String repairID, boolean isResolved) {  
 String sql = "UPDATE 报修信息 SET FBool = ? WHERE FNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setBoolean(1, isResolved);  
 stmt.setString(2, repairID);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 / /管理员查看来访记录  
 public static ResultSet getVisitRecords() {  
 String sql = "SELECT 来访登记.VNO, 来访登记.VReason, 来访登记.VTime, 来访登记.VLeave, 学生.SName, 来访人信息.VName " +  
 "FROM 来访登记 " +  
 "JOIN 学生 ON 来访登记.SNO = 学生.SNO " +  
 "JOIN 来访人信息 ON 来访登记.VSFZH = 来访人信息.VSFZH";  
 try {  
 Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 return stmt.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static String getStudentRoomNumber(String studentSNO) {  
 String sql = "SELECT RNO FROM 学生居住寝室 WHERE SNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, studentSNO);  
 ResultSet rs = stmt.executeQuery();  
 return rs.next() ? rs.getString("RNO") : null;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static ResultSet getRoomInfo(String roomNumber) {  
 String sql = "SELECT \* FROM 寝室 WHERE RNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, roomNumber);  
 return stmt.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static ResultSet getRoomMembers(String roomNumber) {  
 String sql = "SELECT 学生.SNO, 学生.SName FROM 学生居住寝室 JOIN 学生 ON 学生居住寝室.SNO = 学生.SNO WHERE RNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, roomNumber);  
 return stmt.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static char[] getRoomCapacity(String roomNumber) {  
 String sql = "SELECT RMax FROM 寝室 WHERE RNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, roomNumber);  
 ResultSet rs = stmt.executeQuery();  
 return rs.next() ? rs.getString("RMax").toCharArray() : null;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static ResultSet getAllDormitories() {  
 String sql = "SELECT \* FROM 寝室";  
 try {  
 Connection conn = getConnection();  
 return conn.prepareStatement(sql).executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean addDormitory(String dormitoryNumber, int maxCapacity) {  
 String sql = "INSERT INTO 寝室 (RNO, RNumber, RMax) VALUES (?, 0, ?)";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, dormitoryNumber);  
 stmt.setInt(2, maxCapacity);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean updateDormitory(String dormitoryNumber, int maxCapacity) {  
 String sql = "UPDATE 寝室 SET RMax = ? WHERE RNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setInt(1, maxCapacity);  
 stmt.setString(2, dormitoryNumber);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean deleteDormitory(String dormitoryNumber) {  
 String sql = "DELETE FROM 寝室 WHERE RNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, dormitoryNumber);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static ResultSet getAllStudentDormitories() {  
 String sql = "SELECT 学生居住寝室.SNO, 学生.SName, 学生居住寝室.RNO " +  
 "FROM 学生居住寝室 " +  
 "JOIN 学生 ON 学生居住寝室.SNO = 学生.SNO";  
 try {  
 Connection conn = getConnection();  
 return conn.prepareStatement(sql).executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean assignStudentDormitory(String sno, String dormitoryNumber) {  
 String sql = "INSERT INTO 学生居住寝室 (SNO, RNO) VALUES (?, ?)";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, sno);  
 stmt.setString(2, dormitoryNumber);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean removeStudentDormitory(String sno) {  
 String sql = "DELETE FROM 学生居住寝室 WHERE SNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, sno);  
 return stmt.executeUpdate() > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static ResultSet getStudents(String sno, String name, String college, String sClass, String dormitory) {  
 StringBuilder sql = new StringBuilder(  
 "SELECT 学生.SNO, 学生.SName, 学生.SSex, 学生.SPhone, 学生.SCollege, 学生.SClass, 学生居住寝室.RNO " +  
 "FROM 学生 " +  
 "LEFT JOIN 学生居住寝室 ON 学生.SNO = 学生居住寝室.SNO " +  
 "WHERE 1=1"  
 );  
 if (sno != null && !sno.isEmpty()) sql.append(" AND 学生.SNO LIKE ?");  
 if (name != null && !name.isEmpty()) sql.append(" AND 学生.SName LIKE ?");  
 if (college != null && !college.isEmpty()) sql.append(" AND 学生.SCollege LIKE ?");  
 if (sClass != null && !sClass.isEmpty()) sql.append(" AND 学生.SClass LIKE ?");  
 if (dormitory != null && !dormitory.isEmpty()) sql.append(" AND 学生居住寝室.RNO = ?");  
 try {  
 Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql.toString());  
  
 int index = 1;  
 if (sno != null && !sno.isEmpty()) stmt.setString(index++, "%" + sno + "%");  
 if (name != null && !name.isEmpty()) stmt.setString(index++, "%" + name + "%");  
 if (college != null && !college.isEmpty()) stmt.setString(index++, "%" + college + "%");  
 if (sClass != null && !sClass.isEmpty()) stmt.setString(index++, "%" + sClass + "%");  
 if (dormitory != null && !dormitory.isEmpty()) stmt.setString(index++, dormitory);  
  
 return stmt.executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean addStudent(String sno, String name, String sex, String phone, String college, String sClass) {  
 String sql = "INSERT INTO 学生 (SNO, SName, SSex, SPhone, SCollege, SClass) VALUES (?, ?, ?, ?, ?, ?)";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, sno);  
 stmt.setString(2, name);  
 stmt.setString(3, sex);  
 stmt.setString(4, phone);  
 stmt.setString(5, college);  
 stmt.setString(6, sClass);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean updateStudent(String sno, String name, String sex, String phone, String college, String sClass,String password) {  
 String sql = "UPDATE 学生 SET SName = ?, SSex = ?, SPhone = ?, SCollege = ?, SClass = ? , SPassword=? WHERE SNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, name);  
 stmt.setString(2, sex);  
 stmt.setString(3, phone);  
 stmt.setString(4, college);  
 stmt.setString(5, sClass);  
 stmt.setString(6, password);  
 stmt.setString(7, sno);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean deleteStudent(String sno) {  
 String sql = "DELETE FROM 学生 WHERE SNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, sno);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean isStudentExist(String text) {  
 String sql = "SELECT \* FROM 学生 WHERE SNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, text);  
 ResultSet rs = stmt.executeQuery();  
 return rs.next();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean changeStudentDormitory(String sno, String newDormitory) {  
 String sql = "UPDATE 学生居住寝室 SET RNO = ? WHERE SNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, newDormitory);  
 stmt.setString(2, sno);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static ResultSet getAllRepairs() {  
 String sql = "SELECT FNO, RNO, FDetail, FTime, FBool, FRemark FROM 报修信息";  
 try {  
 Connection conn = getConnection();  
 return conn.prepareStatement(sql).executeQuery();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static boolean markRepairAsResolved(String repairId) {  
 String sql = "UPDATE 报修信息 SET FBool = 1 WHERE FNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, repairId);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 // 删除报修信息  
 public static boolean deleteRepair(String repairId) {  
 String sql = "DELETE FROM 报修信息 WHERE FNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, repairId);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean deleteVisitor(String visitorId) {  
 String sql = "DELETE FROM 来访人信息 WHERE VSFZH = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, visitorId);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean deleteVisitRecord(String visitId) {  
 String sql = "DELETE FROM 来访登记 WHERE VNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, visitId);  
 //更新来访登记表的编号，并保证编号是6位  
 String getMaxVNOQuery = "SELECT MAX(VNO) AS MaxVNO FROM 来访登记";  
 ResultSet rs = conn.createStatement().executeQuery(getMaxVNOQuery);  
 String nextVNO = "000001"; // 默认从 00001 开始  
 if (rs.next() && rs.getString("MaxVNO") != null) {  
 String maxVNO = rs.getString("MaxVNO"); // 获取最大编号  
 int maxVNOInt = Integer.parseInt(maxVNO); // 转换为整数  
 nextVNO = String.format("%06d", maxVNOInt + 1); // 格式化为6位  
 }  
 String updateVNOQuery = "UPDATE 来访登记 SET VNO = ? WHERE VNO = ?";  
 PreparedStatement updateStmt = conn.prepareStatement(updateVNOQuery);  
 updateStmt.setString(1, visitId);  
 updateStmt.setString(2, nextVNO);  
 updateStmt.executeUpdate();  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static boolean isVisitorExists(String id, String phone) {  
 String sql = "SELECT COUNT(\*) AS count FROM 来访人信息 WHERE VSFZH = ? AND VPhone = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, id);  
 stmt.setString(2, phone);  
 ResultSet rs = stmt.executeQuery();  
 if (rs.next()) {  
 return rs.getInt("count") > 0; // 如果存在记录，返回 true  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return false; // 默认返回 false  
 }  
  
 public static boolean registerVisitor(String id, String phone, String name) {  
 String sql = "INSERT INTO 来访人信息 (VSFZH, VPhone, VName) VALUES (?, ?, ?)";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, id);  
 stmt.setString(2, phone);  
 stmt.setString(3, name);  
 return stmt.executeUpdate() > 0; // 返回受影响的行数  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 public static String getVisitorName(String id) {  
 String sql = "SELECT VName FROM 来访人信息 WHERE VSFZH = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, id);  
 ResultSet rs = stmt.executeQuery();  
 return rs.next() ? rs.getString("VName") : null;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 //提前结束访问  
 public static void endVisitorVisit(String id) {  
 // 更新数据库中的离校时间  
 String sql = "UPDATE 来访登记 SET VLeave = GETDATE() WHERE VSFZH = ? AND VLeave > GETDATE()";//更新离校时间  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, id);  
 stmt.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static Object getStudentName(String sno) {  
 String sql = "SELECT SName FROM 学生 WHERE SNO = ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 stmt.setString(1, sno);  
 ResultSet rs = stmt.executeQuery();  
 return rs.next() ? rs.getString("SName") : null;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public static java.util.List<String[]> searchVisitors(String condition) {  
 java.util.List<String[]> visitorList = new ArrayList<>();  
 String sql = (condition == null || condition.isEmpty())  
 ? "SELECT \* FROM 来访人信息"  
 : "SELECT \* FROM 来访人信息 WHERE VName LIKE ? OR VSFZH LIKE ? OR VPhone LIKE ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 if (condition != null && !condition.isEmpty()) {  
 stmt.setString(1, "%" + condition + "%");  
 stmt.setString(2, "%" + condition + "%");  
 stmt.setString(3, "%" + condition + "%");  
 }  
 try (ResultSet rs = stmt.executeQuery()) {  
 while (rs.next()) {  
 visitorList.add(new String[]{  
 rs.getString("VSFZH"), // 身份证号  
 rs.getString("VName"), // 姓名  
 rs.getString("VPhone") // 电话  
 });  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return visitorList;  
 }

public static java.util.List<String[]> searchVisits(String condition) {  
 java.util.List<String[]> visitList = new ArrayList<>();  
 String sql = (condition == null || condition.isEmpty())  
 ? "SELECT \* FROM View\_Visit\_Record"  
 : "SELECT \* FROM View\_Visit\_Record WHERE 学生学号 LIKE ? OR 学生姓名 LIKE ? OR 来访人电话 LIKE ? OR 身份证号 LIKE ? OR 来访人姓名 LIKE ?";  
 try (Connection conn = getConnection();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
 if (condition != null && !condition.isEmpty()) {  
 stmt.setString(1, "%" + condition + "%");  
 stmt.setString(2, "%" + condition + "%");  
 stmt.setString(3, "%" + condition + "%");  
 stmt.setString(4, "%" + condition + "%");  
 stmt.setString(5, "%" + condition + "%");  
 }  
 try (ResultSet rs = stmt.executeQuery()) {  
 while (rs.next()) {  
 visitList.add(new String[]{  
 rs.getString("来访编号"), // 登记编号  
 rs.getString("来访人姓名"), // 访客姓名  
 rs.getString("身份证号"), // 访客身份证号  
 rs.getString("学生学号"), // 被访学号  
 rs.getString("学生姓名"), // 被访学生姓名  
 rs.getString("来访时间"), // 来访时间  
 rs.getString("离开时间"), // 离开时间  
 rs.getString("来访原因") // 来访原因  
 });  
 }  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return visitList;  
 }  
}

# 7 系统测试

## 7.1 存储过程的测试

1. 使用 SP\_ADD\_STUDENT 插入新学生，验证学生信息及寝室状态（寝室人数是否增加）是否更新。

该存储过程用于添加新学生并为其分配寝室，同时更新寝室的当前入住人数。存储过程接受多个参数，包括学生的学号、姓名、性别、联系方式、学院、班级、密码以及寝室编号。执行时，它会将学生信息插入 学生 表，同时在 学生居住寝室 表中添加该学生的寝室分配记录，并更新对应寝室的当前入住人数。

**主要功能**：

自动向 学生 表和 学生居住寝室 表添加学生及其寝室分配记录。

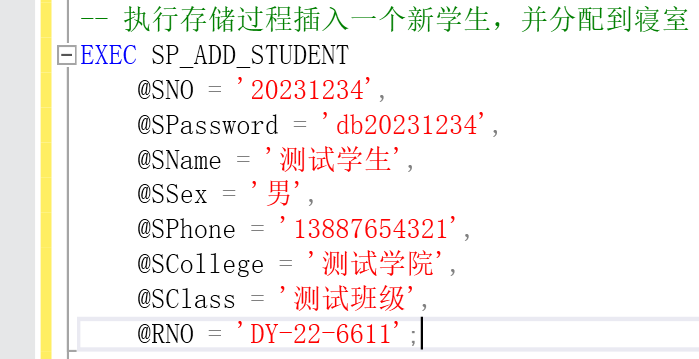
确保寝室的当前入住人数实时更新。

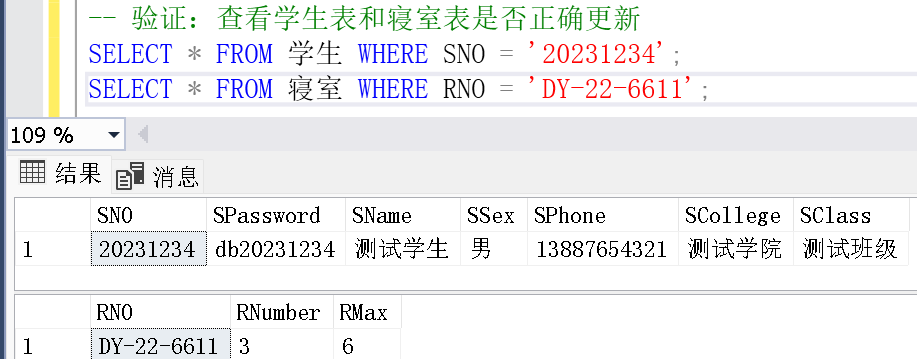
**适用场景**：

当新增学生时，同时完成寝室分配及入住人数调整。

**测试方法**：

1. 调用存储过程，传入学生信息及寝室编号。
2. 查询 学生 表和 学生居住寝室 表，验证新学生记录是否插入成功。
3. 查询寝室的当前入住人数，验证是否自动增加。（这里实际上是调用了触发器，不然就会重复增加人数）





2. 使用 SP\_DELETE\_STUDENT 删除学生，验证学生信息及寝室状态是否更新。

该存储过程用于删除学生记录，同时从其所在的寝室中移除该学生，并更新寝室的当前入住人数。执行时，它会先从 学生居住寝室 表中删除该学生的寝室分配记录，然后更新寝室的当前人数，最后从 学生 表中删除该学生的信息。

**主要功能**：

删除学生记录及其寝室分配记录。

确保寝室的当前入住人数实时减少。

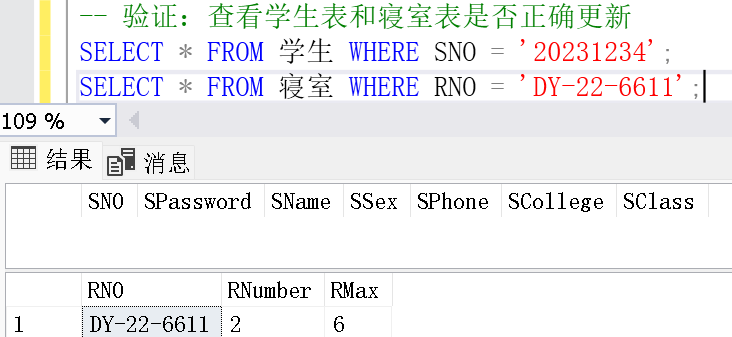
**适用场景**：

当学生退宿或离校时，使用该存储过程删除其信息并调整寝室人数。

**测试方法**：

1. 调用存储过程，传入学生学号。
2. 查询 学生 表和 学生居住寝室 表，验证记录是否删除成功。
3. 查询寝室的当前入住人数，验证是否自动减少。





3. 使用 SP\_ADD\_REPAIR 插入报修信息，验证报修信息是否完整记录。

该存储过程用于新增报修记录。它接受寝室编号、报修详情、报修时间和报修编号作为参数，执行时将数据插入 报修信息 表中。

**主要功能**：

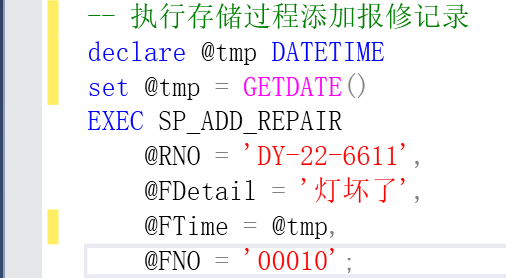
自动向 报修信息 表插入一条新的报修记录。

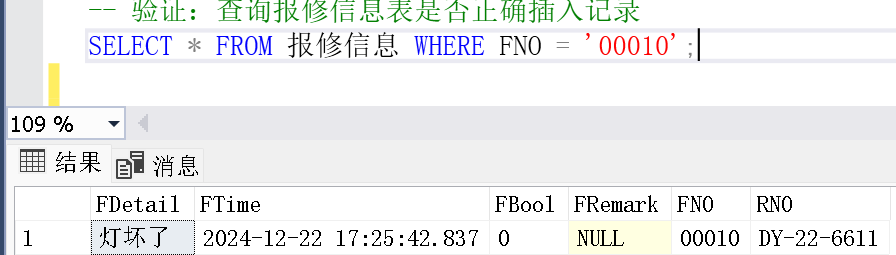
**适用场景**：

当寝室需要报修时，使用该存储过程记录报修信息。

**测试方法**：

1. 调用存储过程，传入报修信息（寝室编号、报修内容等）。
2. 查询 报修信息 表，验证报修记录是否正确插入。





4. 使用 SP\_ADD\_VISIT 插入来访记录，验证访客来访信息是否完整记录。

该存储过程用于记录来访登记信息。它接受学生学号、来访者身份证号、来访编号、来访时间、离校时间及来访原因作为参数，执行时将记录插入 来访登记 表中。

**主要功能**：

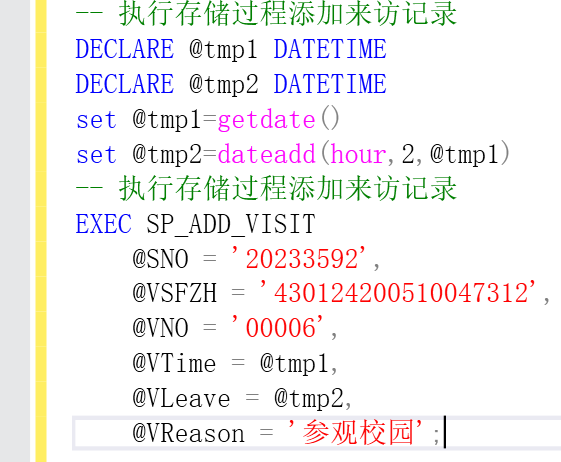
向 来访登记 表添加一条新的来访记录。

**适用场景**：

当学生有访客来访时，使用该存储过程记录访客信息。

**测试方法**：

1. 调用存储过程，传入来访登记信息。
2. 查询 来访登记 表，验证记录是否正确插入。





5. 使用 SP\_GET\_ROOM\_STATUS获取寝室信息。

该存储过程用于查询寝室的基本信息和入住学生列表。存储过程接受寝室编号作为参数，执行时会返回该寝室的编号、当前入住人数、最大容量，以及所有入住学生的详细信息。

**主要功能**：

查询寝室的状态信息（如当前入住人数、最大容量等）。

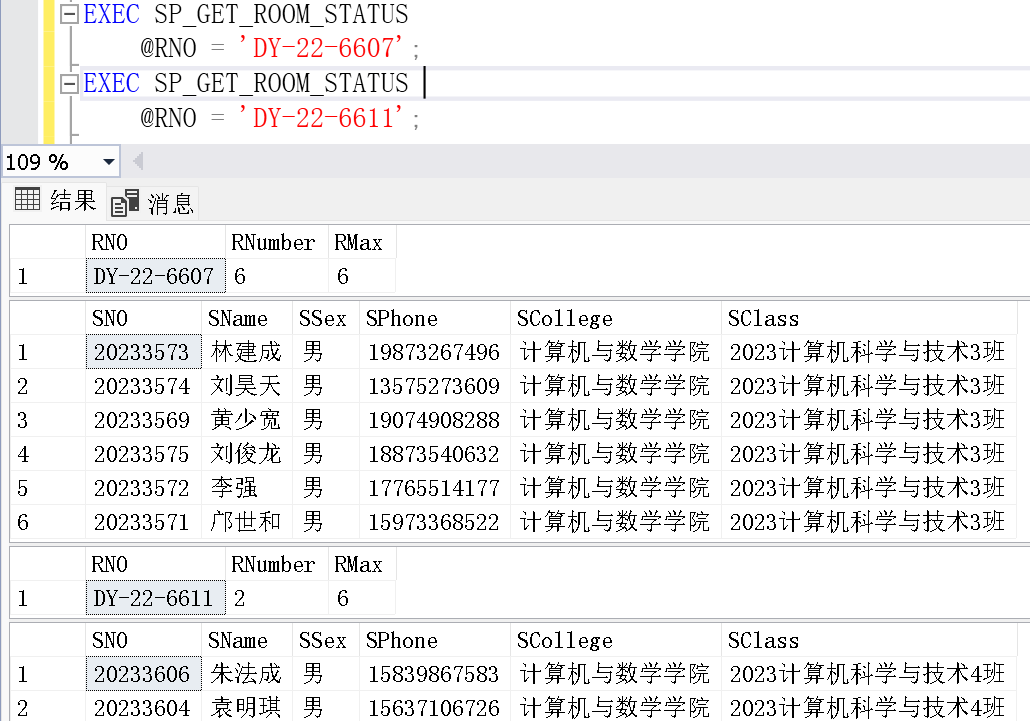
查询寝室内的学生详细信息（如学号、姓名、联系方式等）。

**适用场景**：

当需要查看寝室详细信息时，使用该存储过程进行查询。

**测试方法**：

1. 调用存储过程，传入寝室编号。
2. 验证返回结果是否包含寝室状态和入住学生的详细信息。



## 7.2 触发器的测试

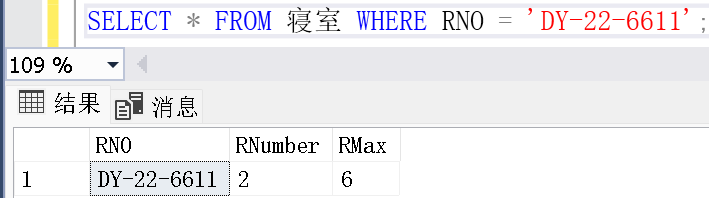
**1. TR\_ADD\_STUDENT\_TO\_ROOM 触发器说明**

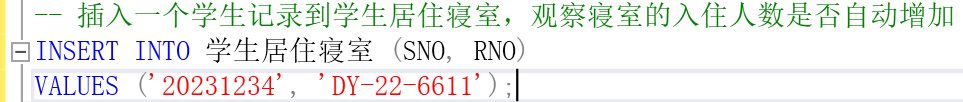
该触发器用于在向 学生居住寝室 表中插入新记录时，自动更新对应寝室的入住人数。触发器的主要功能是监听对 学生居住寝室 表的 INSERT 操作，并根据插入的新记录中的寝室编号，自动将相应寝室的当前人数加 1。  
**适用场景**：

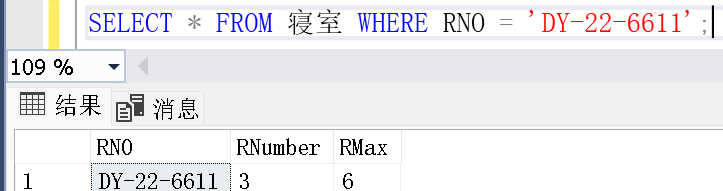
当有学生被分配到寝室时，该触发器自动调整寝室的入住人数，确保寝室信息的实时更新。

**测试方法**：

1. 向 学生居住寝室 表插入新记录。
2. 查询对应寝室的当前人数，验证人数是否自动增加。







**2. TR\_REMOVE\_STUDENT\_FROM\_ROOM 触发器说明**

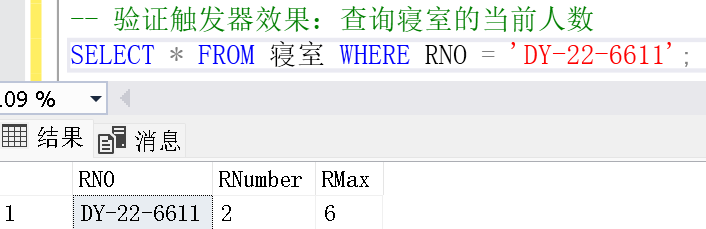
该触发器用于在从 学生居住寝室 表中删除记录时，自动更新对应寝室的入住人数。触发器的主要功能是监听对 学生居住寝室 表的 DELETE 操作，并根据删除记录中的寝室编号，自动将相应寝室的当前人数减 1。  
**适用场景**：

当学生退宿或被调离寝室时，该触发器自动调整寝室的入住人数，确保数据一致性。

**测试方法**：

1. 从 学生居住寝室 表删除指定记录。
2. 查询对应寝室的当前人数，验证人数是否自动减少。





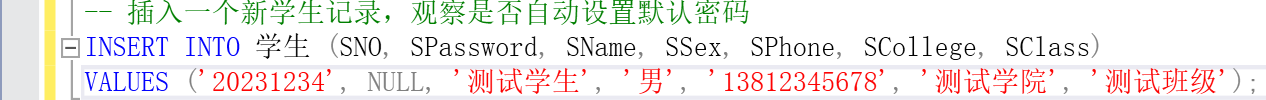
**3. TR\_SetDefaultPassword 触发器说明**

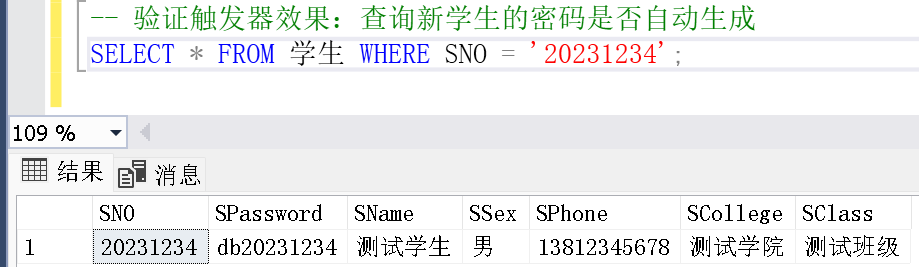
该触发器用于在向 学生 表插入新记录时，自动为新插入的学生生成默认密码。触发器的主要功能是监听对 学生 表的 INSERT 操作，并根据新插入记录的学号，自动设置默认密码为 db 加学号的形式（例如，学号为 20231234 的学生，默认密码为 db20231234）。  
**适用场景**：

当新增学生记录时，触发器确保每个学生都有一个默认初始密码，便于后续登录操作。

**测试方法**：

1. 向 学生 表插入一条新记录，密码字段设置为 NULL。
2. 查询新插入学生的密码字段，验证是否正确生成默认密码。





## 7.3 视图的测试

**1.视图 View\_Room\_Status**

作用：

该视图用于显示寝室的当前状态，包括寝室编号、当前入住人数、最大容量以及每个学生的详细信息。

数据来源：

寝室表（寝室）：提供寝室编号、当前入住人数（RNumber）和最大容量（RMax）。

学生居住寝室表（学生居住寝室）：提供学生和寝室之间的关联信息。

学生表（学生）：提供学生的详细信息，包括学号（SNO）、姓名（SName）等。

用途：

该视图适用于管理员快速查看寝室的整体入住情况以及每个寝室中学生的详细信息。方便管理寝室的使用状态以及学生分布情况。



**2. 视图View\_Visit\_Record**

**作用：**

该视图用于显示学生的来访记录，包括来访人信息、学生信息和访问时间。

**数据来源：**

来访登记表（来访登记）：提供来访记录的核心信息，包括来访编号（VNO）、访问时间（VTime）、离开时间（VLeave）等。

来访人信息表（来访人信息）：提供来访人的详细信息，包括姓名（VName）、身份证号（VSFZH）、电话（VPhone）。

学生表（学生）：提供被访问学生的学号（SNO）和姓名（SName）。

**用途：**

该视图适用于查询学生的历史访问记录，包括访问者的基本信息、访问时间以及访问原因。管理员可以快速追踪来访记录，确保寝室安全。



**3. 视图 View\_Unfilled\_Room**

**作用：**

该视图用于显示当前未满的寝室信息，包括寝室编号、当前入住人数、最大容量和剩余可入住人数。

**数据来源：**

寝室表（寝室）：提供寝室的编号（RNO）、当前入住人数（RNumber）和最大容量（RMax）。

**用途：**

该视图适用于管理员快速查看未满寝室的状态，以便合理分配和调整学生的住宿安排。通过剩余可入住人数，可以直观了解寝室容量利用率。



**4. 视图 View\_Repair\_Info**

**作用：**

该视图用于显示寝室的报修记录，包括报修内容、报修时间、解决状态以及备注。

**数据来源：**

报修信息表（报修信息）：提供报修的详细信息，包括报修编号（FNO）、寝室编号（RNO）、报修内容（FDetail）、报修时间（FTime）、解决状态（FBool）和备注（FRemark）。

**用途：**

该视图适用于管理员快速查看寝室的报修记录，包括是否已解决的问题状态。方便管理员跟踪和管理寝室的维修工作。



**5. 视图 View\_Student\_Info**

**作用：**

该视图用于显示学生的详细信息及其寝室分配情况，包括学号、姓名、性别、电话、学院、班级和寝室编号。

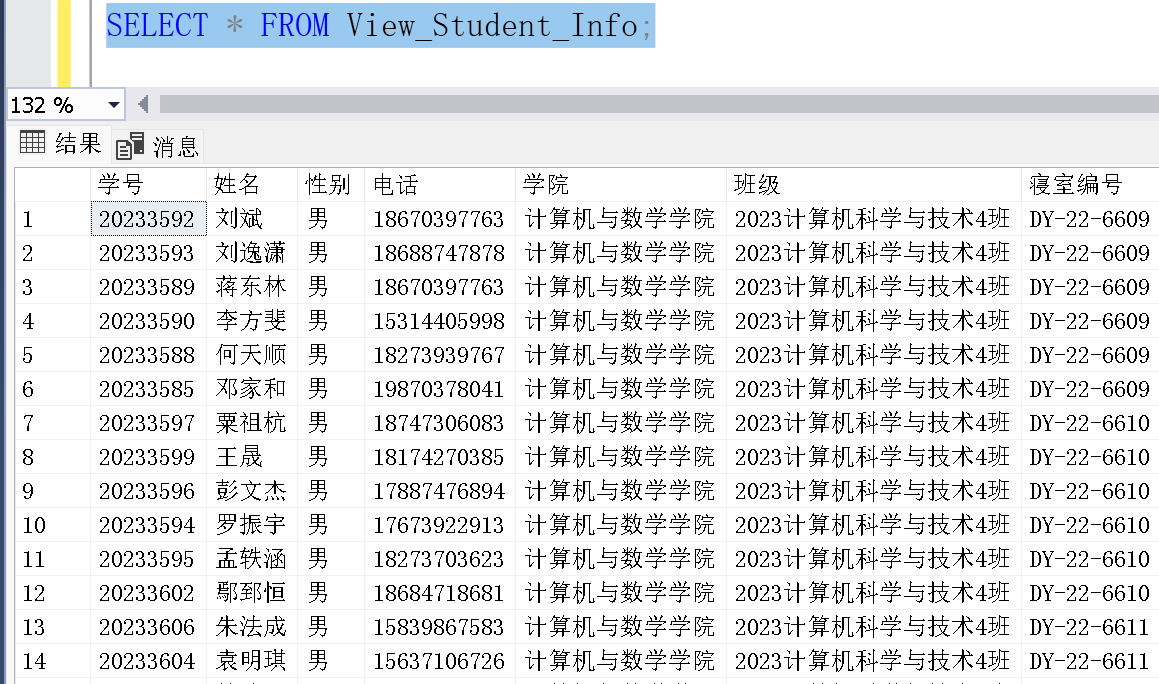
**数据来源：**

学生表（学生）：提供学生的详细信息，包括学号（SNO）、姓名（SName）、性别（SSex）、电话（SPhone）、学院（SCollege）和班级（SClass）。

学生居住寝室表（学生居住寝室）：提供学生与寝室的关联信息，包括寝室编号（RNO）。

**用途：**

该视图适用于快速查询学生的基本信息以及所在寝室的分配情况。便于管理员了解学生的住宿安排。



# 总 结

在本次课程设计过程中，我全面实践了一个寝室管理系统的设计与开发。我重新审视和综合运用了数据结构、程序设计、数据库系统以及软件工程的知识。。通过对实际问题的不断探索，我对诸多相关概念有了更加清晰且全面的理解。

项目中，我细致地分析了系统需求，明确了核心功能与数据需求，并基于寝室管理的现实需求设计了核心功能模块。需求分析阶段，我还绘制了系统的数据流图，编写了详尽的数据字典，这为接下来的数据库设计打下了坚实基础。

在数据库设计阶段，我分三步走——概念设计、逻辑设计与物理设计，逐层构建系统的数据库架构。在概念设计中，我以E-R图的形式抽象出寝室管理系统的关键实体（如学生、寝室、访客等）、属性以及它们之间的关系；随后，通过逻辑设计将E-R图转化为关系模型，设计了包括学生信息表、寝室信息表、访客登记表等在内的多个核心数据表，明确了主键、外键以及数据约束。在物理设计阶段，我选择了SQL Server作为实现平台，为数据库选择了适宜的存储结构，并设计了必要的索引以优化查询性能。

此外，为实现复杂的业务逻辑，我编写了多个存储过程，例如学生入住登记、寝室调整和访客管理等。与此同时，我还设计了触发器，用于自动维护寝室入住人数的更新等。为了简化复杂查询，我进一步构建了多种视图，例如寝室入住情况视图和访客记录视图，显著提高了数据查询的易用性。

系统测试是我重点投入的阶段之一。我精心设计了覆盖核心功能的测试用例，对存储过程、触发器与视图进行了全面测试与验证。在测试过程中，我发现了一些潜在的逻辑问题，并及时修复，确保了系统的稳定性和功能的正确性。测试的完成不仅让我对系统的整体架构更加自信，也让我认识到细致验证的重要性。

然而，项目的完成并不意味着结束。我也意识到，系统目前仍有不少需要优化和改进之处。比如，由于时间有限，应用程序的界面设计较为简陋，交互体验还有很大提升空间。此外，针对大规模数据处理与高并发场景的性能优化还需进一步研究与实践。

未来，我计划深入学习前端开发技术，设计更加直观和友好的用户界面，提升系统的用户体验。同时，我将专注于数据库性能优化技术的学习，包括查询优化、索引设计以及事务管理，以进一步提高系统的效率与可扩展性。

在功能扩展方面，我设想增加寝室物资管理、更多角色权限管理（如：寝室修理人员、学生辅导员等）等模块，以满足更多样化的管理需求。

通过本次课程设计，我不仅掌握了寝室管理系统的数据库设计与实现流程，更积累了宝贵的实践经验。这次项目让我深刻认识到数据库技术的实际应用价值，同时也为我今后的学习与开发指明了方向。

我相信，随着学习的深入和经验的积累，我能够设计出更加完善且功能丰富的寝室管理系统，为实际应用场景提供更高效的解决方案。

**参考文献**

[1] 王珊，萨师煊.数据库系统概论(第5版).高等教育出版社，2011.5

[2] Abraham Silberschatz，Henry F. Korth，S. Sudarshan. 数据库系统概念（原书第7版）[M]. 北京：机械工业出版社，2021.10

[3] 施瓦茨，Z. SQL学习指南（第2版）[M]. 北京：人民邮电出版社，2013

[4] 施瓦茨，Z. SQL必知必会（第4版）[M]. 北京：人民邮电出版社，2013

[5] Bruce Eckel. Java编程思想（第4版）[M]. 北京：机械工业出版社，2007

[6] Joshua Bloch. Effective Java（第2版）[M]. 北京：机械工业出版社，2009

[7] Roger S. Pressman. 软件工程：实践者的研究方法（第7版）[M]. 北京：机械工业出版社，2011.3