中国科学技术大学
计算机学院
数据库课程设计

Student Management System

学生管理系统设计 Version1.0

黄业琦

2020-05-22

Contents

1. 项目简介

1.1 项目综述

在数据处理的及时性的高效性的今天，数据库系统的应用日益广泛。在各类商政业务的管理中，数据库系统已经成为了不可或缺的部分。在经历数据库课程的学习后，我们的组员对一般商用的数据库应用平台产生了浓厚的兴趣。结合我们已经掌握的理论知识，我们希望通过一个完整的项目，对我们的数据库理论知识和数据库开发技能进行全面的锻炼和提升。

1.2 项目一览

GITHUB 地址：https://github.com/KradNosnatef/996_SQL/

1.3 功能需求

1.3.1 前端需求

我们的管理系统需要有一个美观易用的前端页面作为承载。

这里前端我主要向 GITHUB 上已有的一个项目进行了吸纳和借鉴：

STUMANAGEMENT

1.3.2 后端需求

我们管理的主要实体为：

- 校区
- 专业
- 班级
- 学生

- 教师
- 学籍异动
- 课程

我们需要在对上述实体在不冲突的情况下，对他们进行数据库的常规操作，即增加、删除、修改。
此外，我们需要对各种不同的情况和需求进行查询。

1.4 技术背景

在对业界的数据库开发做了一些简单的调研之后，我们得到了如下的一些信息：

1. MySQL 是一般开发人员最喜欢用的数据库之一，也是我们课程推荐的数据库系统
2. 一般的 Web 应用开发人员更多的喜欢使用 Java 进行开发，具有优秀的可移植性
3. 现如今的 Web App 更推崇使用框架，注重敏捷开发

1.5 技术方案选择

在简单了解各个技术方案的优劣之后，我们最终并没有选择一些成熟的框架进行开发。而是使用了一些更加传统的技术方案。
我们的核心技术点为 Java Web，这使得我们可以使用安全的 JDBC 接口去保护我们的数据安全。此外，我们使用 JSP + Servlet + JavaBean 去规划处理我们的各种业务需求，使得开发可以有序推进。

1.6 开发工具

1. 数据库工具：MySQL Database 8.0+
2. Java 开发工具：IntelliJ IDEA 2019+
3. 前端工具：Brackets、VS Code、Vim
4. 后端承载平台：Tomcat 9.0
5. 文档工具：XeLatex
6. 测试工具：IntelliJ Junit
7. 管理工具：Git、GItg

这里我的选择注重了“经典和流行兼顾，便捷与规范具备”的原则。
Mysql 作为经典的数据库平台工具，是我们最初确定的技术核心。在确定了 Java Web 作为开发任务之后，选择了较为流行的开发工具：IntelliJ IDEA。我们配置的后端注重了规范性，向这行业的标准靠拢，

选择给予 Apache 的一个简单易用的版本 Tomcat 进行部署。而前端的编辑，我们不拘一格，各显身手，使用各自习惯的工具进行操作。

用到的编程语言主要为：Java CSS HTML JavaScript

1.7 组员分工

1.7.1 前端分工

页面设计布局：黄业琦

1.7.2 前后端对接

前后端对接、Tomcat 的配置：黄业琦

1.7.3 后端分工

数据库链接组件：黄业琦

校区、专业、成绩、个人基本信息、教师管理：黄业琦

复杂查询：张行健

学生管理、学籍异动管理：张行健

班级管理：张欣瑞

课程管理、开课管理：朱凡

1.7.4 其他

测试：张欣瑞、张行健、朱凡

文档与 demo 录制：黄业琦

2. 管理系统的设计

2.1 数据库设计

2.1.1 数据库设计需求

学校有多个校区, 各个校区均有其校区代码 (唯一)、校区名称和校区地址 (实体中包括但不限于上述属性, 下同)。学校开设多个专业, 各个专业均有其专业代码 (唯一)、专业名称、专业地址、专业负责人和所属校区 (一个专业仅属于一个校区)。学校建立多个班级, 各个班级均有其班级代码 (唯一)、班级名称、建班年月、班主任、所属年级 (年份) 和所属专业。

学校将所有教师和学生的基本个人信息统一存放, 包括身份证件号 (唯一)、身份证件类型 (身份证或护照)、中文名称、性别码 (女或男)、出生日期 (年月日) 和国籍 (中文名称)。如果教师和学生提供了家庭通讯方式, 包括家庭住址、家庭邮政编码和家庭电话, 学校也会记录。每个教师也有属于自己的工号 (唯一), 每个学生有属于自己的学号 (唯一)。学校记录学生的入学年月、电子邮箱和所属班级, 也记录教师的入职年月、电子邮箱、所属专业和职称 (教授或副教授)。学校允许学生转专业和降级 (二者不同时发生, 转专业和降级时均转班, 且只允许一次转专业和一次降级), 统称为学籍异动。学生发生学籍异动时需要记录异动编号 (唯一, 同一学生转专业和降级各有不同的异动编号)、异动日期 (年月)、原班级代码和现班级代码。转专业还需要记录是否已转出团员关系 (是、否或不是团员), 降级则还需要记录降级原因 (休学或支教)。

学校开设不同课程, 每门课程均有其课程号 (唯一, 与课程名称一一对应)、课程名称、开课专业和考核方式 (考试或当堂答辩, 满分均为 100)。当一门课程开课时, 需要记录其授课教师 (一门课仅有一个授课教师)、开课日期 (年)、开课学期 (春或秋)、开课时间 (每个课程一周只开一节课, 为周一至周五的第一节至第九节中的某一节, 自定义记录方式)。学校会记录每个学生的选课记录 (不允许重复选课), 包括选课日期 (同开课日期)、选课学期 (同开课学期) 和考试成绩。

2.1.2 数据库设计与改良

这里结合第二次实验的设计报告，去给出我们最初选择的数据库设计。

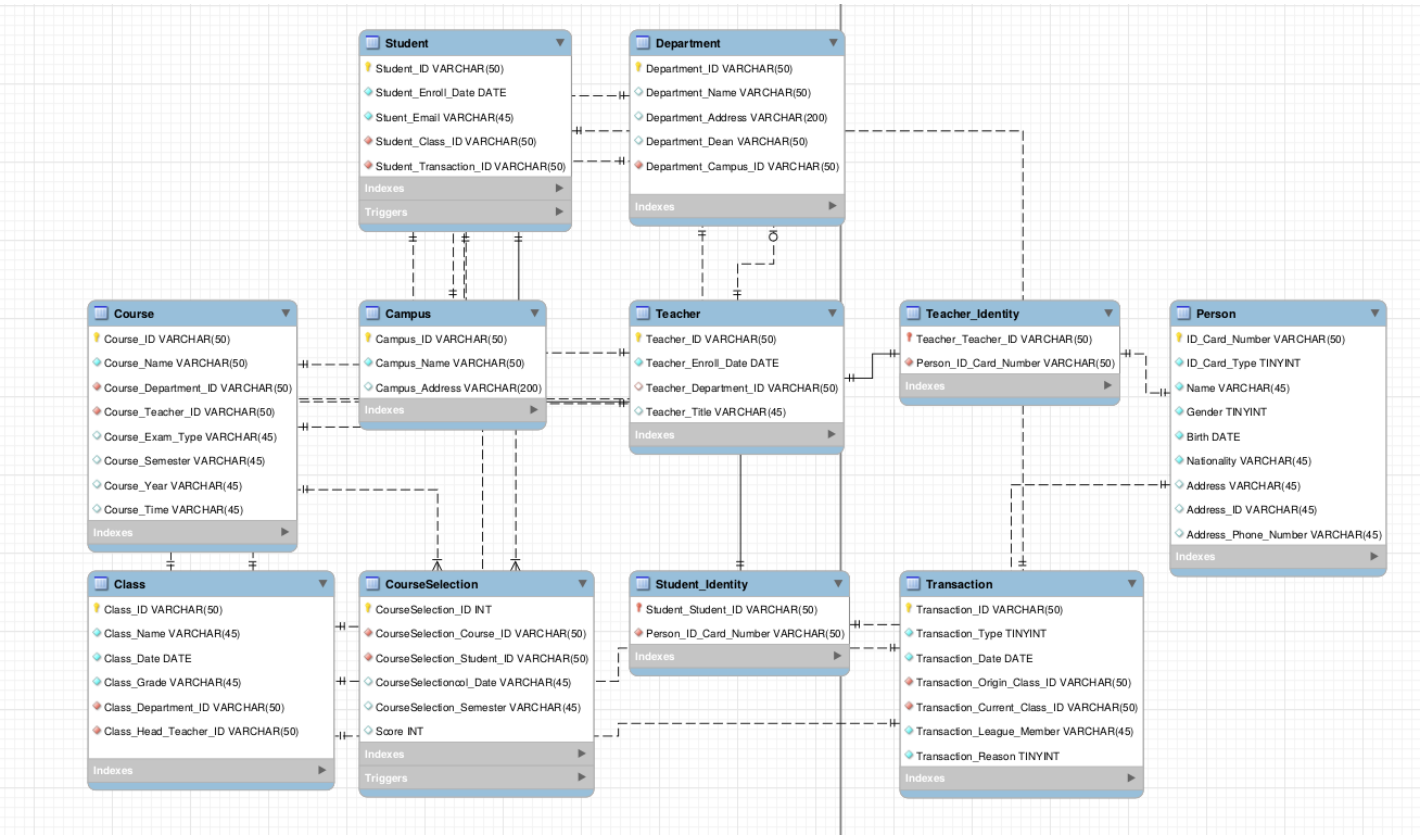

Figure 2.1: 数据库设计一览

在实际实现的时候，大家迅速发现了问题，单独列出 Student_Identity 表，对于我们的信息维护几位不方便。所以最终，在这一版数据库的基础上，我们给 Student 和 Teacher 分别增加了一个外健，用于和 Person 链接。

最终用于生成数据库框架的代码：database-init-github-link

2.2 管理系统总体框架的设计

我们的框架基本按照一般的前后端分离的办法操作。

Login Servlet 是我们应用的大门，用于处理登陆信息。

User-Type 是我们登陆后第一个需要审查的元素。我们通过他来判断我们的用户的类别。

我们的用户分为 3 个基本类别：

1. `Admin Account` 有所有功能的权限
2. `Student Account` 只有部分查询权限，可以对自己的个人信息修改
3. `Teacher Account` 只有煮粉查询权限，可以查询所有学生信息，可以对自己的个人信息修改

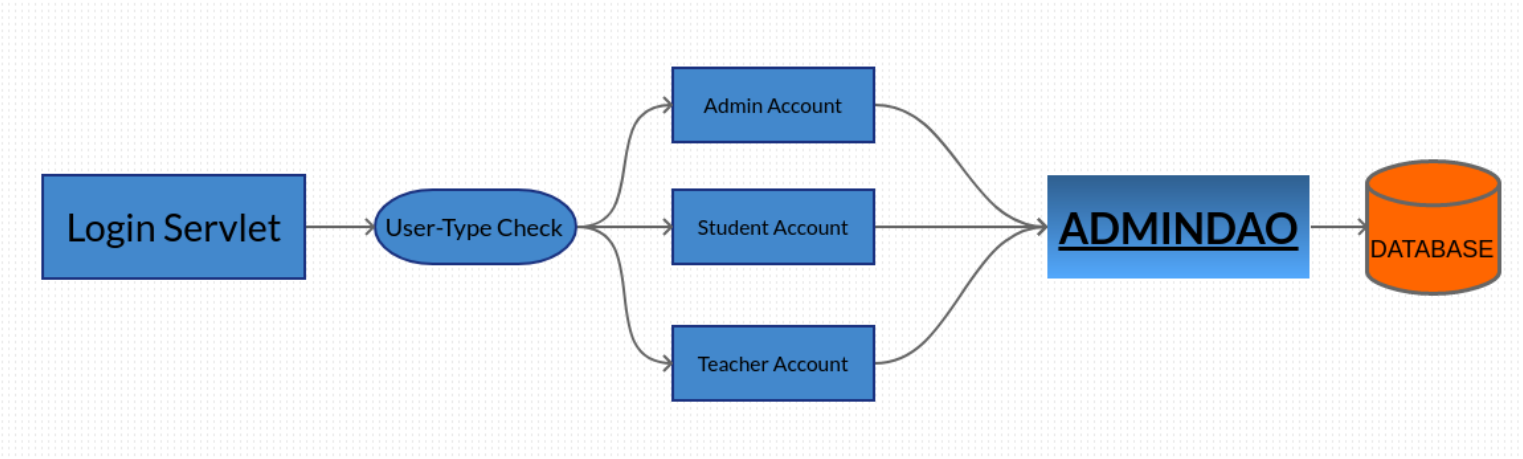

Figure 2.2: framework

我们管理权限主要是利用前端强制管理，这里是我们设计的不足之处。在对应的用户页面上之开放他能用的功能。但是，如果用户足够细心和好运，他可以通过观察我们发出的 HTTP Request 从而对我们的信息进行解析。继而通过 Postman 等工具，发送请求，就可以轻松获取自己权限以外的操作。

正确的做法是，处理所有 HTTP 请求是，处理带上 session 或者 token，记录下用户的类型。此外，对于 java servlet 发送的请求，我们应当适度做一些加密操作。

2.3 后端设计

2.3.1 一览

后端大体框架为 4 部分：

- DAO：Data Access Object 主要用来做各种实体的数据库操作
- model：存放各种实体的数据类型
- service：存放各种 Java Serrvlet 服务
- utils：常用 API 的单独存放

2.3.2 utils

这里我没有按照字典序介绍，而是理解由易到难的次序介绍。

首先是 utils 里的各个模块：

1. Datautils 处理我们 Data 等特殊格式的数据的格式标准化
2. DButils 处理我们的数据库连接和断开的操作

3. UnitTestSwitch 用于切换本地 JDBC 数据库和 Tomcat 的 JDBI 数据库连接方式，这一模块只存放一个常量，可以理解成 Java 下的条件编译，目的是为了方便的单元测试

2.3.3 model

对各个基本实体做存储，为各个实体做了一层封装。

1. Campus 存储校区信息
2. Class 存储班级信息
3. Course 存储课程信息
4. Department 存储专业信息
5. Person 存储个人基本信息
6. Student 存储学生特有信息
7. Teacher 存储教师特有信息
8. Transaction 存储异动信息
9. Users 存储用户信息，用于系统登陆

类型细节见程序注释。

2.3.4 DAO

这一部分为程序的核心，用于处理我们的数据库操作。我们的数据库和 Java 程序全是依靠他们进行连接。

我们这里不论我们的 DAO 母体，我们有一些公共的设计规则:

1. 插入方法的规则
   名称:
   insert[Model_Name]
   输入:
   对于所有的插入方法，参数为我们建立 Dao 对应母体的 Class Model 的所有需要的参数。
   输出:
   如果插入成功，返回非 0 数值，为 SQL DML(Data Manipulation Language) 的行号。
   如果插入失败,返回 0。如果插入成功,返回非 0 数值,为 SQL DML(Data Manipulation Language) 的行号。
   如果插入失败，返回 0。

2. 删除方法的规则

名称:

delete[Model_Name]

输出:

对于所有的删除方法，参数为 (String element_selector, int type)

element_selector 为我们的选择子，用于选定我们待删除的对象。

type 用于选择我们的选择子的含义。

我们只提供了“合理”的删除方法。这里的合理定义是符合一般常理的，例如根据院长去删除一个专业，这种事情听起来即不合理，且极容易发生操作失误。这类借口我们一律不予提供。

输出:

如果返回值为 0，待删除元素不存在。

如果返回值为 1，删除成功。

如果返回值为-1，数据库操作异常。

3. 查询方法的规则名称:

query[Model_Name]

输入:

对于所有的查询方法，参数为 (String element_selector, int type)

element_selector 为我们的选择子，用于选定我们待查询的对象。

type 用于选择我们的选择子的含义。

type = -1 表示查询所有元素的所有信息。

输出:

一个 ArrayList<Model>，Model 为 DAO 对应母体的类型。

4. 更新方法的规则名称:

update[Model_Name]

对于所有的更新方法，我们有两种类型:

- Update Detail Mode

  输入:

  参数为 (int old_type, int new_type, String old_value, String new_value)

  old_XXX 用于确定我们的更改目标，old_type 用于说明 old_value 的含义。old_value 则是我们的目标的选择子。

  new_type 指定说明我们的目标更改项,目标项的 new_type 代表的属性,需要更新为 new_value。

  输出:

  整数，1，代表成功；0，代表失败。

- Rebuild Update Mode

输入：
这里需要输入了一个基本的用于定位元素的键值，一般是主键，当然，特定情况也有其他的选择方案。其余的输入则是我们希望或者说允许更新的元素。
简而言之，我们这里所说的更新和插入基本是一样的。所以称为 Rebuild Update Mode。

下面我将用代码注释说明各个方法的作用。

2.3.4.1 CampusDao

用于对校区进行数据库操作。

1. 插入校区

   insertCampus(String id, String name, String address)

```
    // Insert new campus element
    // Input:
    // - id: campus id
    // - name: campus name
    // - address: campus address
    // Ouput:
    // - If insertion action succeed, return a number, which stands for the row count for SQL Data Manipulation
    // Language (DML) statements. Otherwise, it will return 0
```

2. 删除校区

   deleteCampus(String element_selector, int type)

```
    // Delete a campus
    // Input:
    // - element_selector: a string used to select element
    // - type: type is used to select the meaning of element_selector
    //   type = 0 -- delete by id
    //   type = 1 -- delete by name
    //   type = 2 -- delete by address (ILLEGAL)
    // Ouput:
    // - return 0 if the element is not existed
    // - return 1 if we delete the element successfully
```

```
    // - return -1 if our instruction is illegal
```

3. 查询校区

queryCampus(String element_selector, int type)

```
    // Query campus information
    // Input:
    // - element_selector: a string used to select element
    // - type: type is used to select the meaning of element_selector
    //   type = 0 -- select by id
    //   type = 1 -- select by name
    //   type = 2 -- select by address
    //   type = -1 -- selecat all elements
    // Ouput:
    // - return an array list for your query.
```

4. 更新校区

updateCampus(int old_type, int new_type, String old_value, String new_value)

```
    // Modify campus information
    // - old & new type: type is used to select the meaning of old_value &
  new_value
    //   type = 0 -- update id (if new_type == 0, ILLEGAL)
    //   type = 1 -- update name
    //   type = 2 -- update address
    // - old & new value:
    //   old value is used to select which element to update
    //   new value is used to update the database
    // Ouput:
    // - return 0 if element is not existed
    // - return 1 if update action succeed
    // - return -1 if type is illegal
```

2.3.4.2 ClassDao

用于对班级进行数据库操作。

1. 插入班级

```
// Note: You need to make sure the department_id and the head_teacher_id of the new class element exists
// Note: If you want to delete a class, you need to make sure the class does not have student or transaction.
// Return value: 1: succeed; -1: illegal; 0: fail(not exist);
// Type: 0: by id; 1: by name; 2: by date; 3: by grade; 4:by department; 5: by teacher;
// You can only delete by id or name;
```

2. 删除班级

3. 查询校区

4. 更新校区

2.3.4.3 CourseDao

用于对课程进行数据库操作。

1. 插入课程

```
// Insert new course element
// Input:
// - id: course id
// - name: course name
// - department_id: foreiggn key from Department.Department_ID
// - exam_type: course exam type
// Ouput:
// - If insertion action succeed, return 1
// - If insertion action is illegal, return -1
// - If insertion action failed, return 0
```

2. 删除课程

```
// Delete a course
// Input:
// - element_selector: a string used to select element
// - type: type is used to select the meaning of element_selector
//   type = 0 -- delete by id
//   type = 1 -- delete by name
//   type = 2 -- delete by department_id
//   type = 3 -- delete by exam_type (ILLEGAL)
// Ouput:
// - return 0 if the element is not existed
// - return 1 if we delete the element successfully
// - return -1 if our instruction is illegal
```

3. 查询校区

```
// Query course information
// Input:
// - element_selector: a string used to select element
// - type: type is used to select the meaning of element_selector
//   type = 0 -- select by id
//   type = 1 -- select by name
```

```
    //   type = 2 -- select by department_id
    //   type = 3 -- select by exam_type
    //   type = -1 -- selecat all elements
    // Ouput:
    // - return an array list for your query.
```

4. 更新校区

```
    // Modify course information
    // - old & new type: type is used to select the meaning of old_value &
    new_value
    //   type = 0 -- update id (if new_type == 0m ILLEGAL)
    //   type = 1 -- update name (if new_type == 1 ILLEGAL)
    //   type = 2 -- update department_id (check if existed before update)
    //   type = 3 -- update exam_type
    // Ouput:
    // - return 0 if element is not existed
    // - return 1 if update action succeed
    // - return -1 if type is illegal
```

2.3.4.4 DepartmentDao

用于对专业进行数据库操作。

1. 插入学院

```
    // Insert new department element
    // Input:
    // - id: department id
    // - name: department name
    // - address: department address
    // - dean: department dean
    // - campus_id: foreiggn key from Campus.Campus_ID
    // Ouput:
```

```
    // - If insertion action succeed, return 1
    // - If insertion action is illegal, return -1
    // - If insertion action failed, return 0
```

2. 删除学院

```
    // Delete a department
    // Input:
    // - element_selector: a string used to select element
    // - type: type is used to select the meaning of element_selector
    //   type = 0 -- delete by id
    //   type = 1 -- delete by name
    //   type = 2 -- delete by address (ILLEGAL)
    //   type = 3 -- delete by dean (ILLEGAL)
    //   type = 4 -- delete by campus_id
    // Ouput:
    // - return 0 if the element is not existed
    // - return 1 if we delete the element successfully
    // - return -1 if our instruction is illegal
```

3. 查询学院

```
    // Query department information
    // Input:
    // - element_selector: a string used to select element
    // - type: type is used to select the meaning of element_selector
    //   type = 0 -- select by id
    //   type = 1 -- select by name
    //   type = 2 -- select by address
    //   type = 3 -- select by dean
    //   type = 4 -- select by campus_id
    //   type = -1 -- selecat all elements
    // Ouput:
```

```
    // - return an array list for your query.
```

4. 更新学院

```
    // Modify department information
    // - old & new type: type is used to select the meaning of old_value &
    new_value
    //   type = 0 -- update id (if new_type == 0m ILLEGAL)
    //   type = 1 -- update name
    //   type = 2 -- update address
    //   type = 3 -- update dean
    //   type = 4 -- update campus_id (check if existed before update)
    // Ouput:
    // - return 0 if element is not existed
    // - return 1 if update action succeed
    // - return -1 if type is illegal
```

2.3.4.5 PersonDao

用于对个人进行数据库操作。内部操作，被 Student 和 Teacher 的 DAO 所控制。

1. 查询教师个人信息

```
    public Person queryStudentPerson(String id)
```

2. 查询学生个人信息

```
    public Person queryTeacherPerson(String id)
```

3. 更新个人信息

```
public int updatePerson(String id_card_number, String name, Boolean gender,
String birth, String nationality, String address, String address_postal_code,
String address_phone_number)
```

2.3.4.6 QueryToolKitsDao

用于对复杂查询进行数据库操作。

1. 根据学生 ID 查询课程

```
// Query course Selected by studentID
// Input:
// - id: student id
// output:
// - return a list of the course you want to query
public ArrayList<Course> queryCourseSelectedByStudentID(String studentID)
```

2. 根据课程 ID 查询学生

```
// Query student Selected by courseID
// Input:
// - id: course id
// output:
// - return a list of the student you want to query
public ArrayList<Student> queryStudentSelectedByCourseID(String courseID)
```

3. 根据教师 ID 查询课程

```
// Query course Selected by teacherID
// Input:
// - id: teacher id
// output:
```

```
    // - return a list of the course you want to query
    public ArrayList<Course> queryCourseSelectedByTeacherID(String teacherID)
```

2.3.4.7 SelectionDao

用于对选课进行数据库操作。

1. 插入选课信息

```
    // Insert a new course selecction information
    // student_id & course_id are foreign keys from other table
    // date is fetch from browser
    // Ouput:
    // - If insertion action succeed, return 1
    // - If insertion action is illegal, return -1
    // - If insertion action failed, return 0
```

2. 删除选课信息

```
    // Delete Selection
    // We have 3 deletion mode in total
    // type = 0 -- give student_id & course_id to delete a single selection
    // information
    // type = 1 -- give course_id only
    // type = 2 -- give student_id only
```

3. 查询选课信息

```
    // Query selection information
    // I am going to give out 2 methods here to query:
    // 1. Use student_id to query all the course he chose.
    // 2. Use course_id to quert all the student who chose this class.
```

```
    public ArrayList<Course> queryByStudent(String student_id)
    public ArrayList<Student> queryByCourse(String course_id)
```

2.3.4.8 StudentDao

用于对学生进行数据库操作。

1. 插入学生

```
    // Insert a new student
    // Input (Student Info) AND (Person Info)
    // TODO INITIAL WITH TRANSACTION NEED ADDITIONAL CHECK
    public int insertStudent(String id, String enrollment_date, String email,
String class_id,
                             String id_card_number, boolean card_type, String
name, boolean gender, String birthdate,
                             String nationality, String address, String
address_postal_code,
                             String address_phone_number)
```

2. 删除学生

```
    // Delete an existed student
    // We support 2 types to delete a student
    // type = 0 : delete by student_id
    // type = 1 : delete by id_card_number
```

3. 查询学生

```
    // Query student
    // type = -1 : list out all the students
    // type = 0 : use Student_ID
```

```
    // type = 1 : use ID_card_nubmer
    // type = 2 : use name
```

4. 更新学生

```
    // Update student
    // type = 1 : update enroll date
    // type = 2 : update class id
    // type = 3 : update email
```

2.3.4.9 TeacherDao

用于对教师进行数据库操作。

1. 插入教师

```
    // Insert a new teacher
    // Input (Teacher Info) AND (Person Info)
    public int insertTeacher(String id, String enrollment_date, String
department_id, String teacher_title,
            String id_card_number, boolean card_type, String name, boolean gender,
 String birthdate, String nationality,
            String address, String address_postal_code, String
address_phone_number)
```

2. 删除教师

```
    // Delete an existed teacher
    // We support 2 types to delete a teacher
    // type = 0 : delete by teacher_id
    // type = 1 : delete by id_card_number
```

3. 查询教师

```
// Query teacher
// type = -1 : List out all the teachers
// type = 0 : use Teacher_ID
// type = 1 : use ID_card_nubmer
// type = 2 : use name
```

4. 更新教师

```
// Update teacher
// type = 1 : update enroll date
// type = 2 : update department id
// type = 3 : update title
```

2.3.4.10 TransactionDao

用于对异动进行数据库操作。

1. 插入异动

```
// Insert new transaction element
// Input:
// - id: transaction id
// - type: transaction type
// - date: transaction date
// - and so on
// Ouput:
// - If insertion action succeed, return a number, which stands for the row
// count for SQL Data Manipulation
// Language (DML) statements. Otherwise, it will return 0
```

2. 删除异动

```
// Delete a transaction
// Input:
// - id: transaction id
// Ouput:
// - return 0 if the element is not existed
// - return 1 if we delete the element successfully
```

3. 查询异动

```
// Query traction information
// Input:
// - elementSelector: a string used to select element
// - type: type is used to select the meaning of element_selector
// type = 0 -- select by transactionID
// type = 1 -- select by studentID
// type = -1 -- selecat all elements
// Ouput:
// - return an array list for your query.
```

4. 更新异动

```
// Modify campus information
// - row & column selector: used to select the value you want to update , use
id
// to select row
// - columnSelector:
// 0 -- update type
// 1 -- update date
// 2 -- update originClassID
// 3 -- update CurrentClassID
// 4 -- update LeagueMember
```

```
    // 5 -- update Reason
    // - value:
    // used to update the database
    // Ouput:
    // - return 0 if element is not existed
    // - return 1 if update action succeed
    // - return -1 if action is illegal
```

2.3.4.11 UserDao

用于对用户进行数据库操作。

1. 插入用户

```
public int insertUser(String username, String password, int usertype, String
    foreign_id)
```

2. 删除用户

```
public int deleteUser(String username)
```

3. 查询用户
   查询所有用户

```
public ArrayList<User> queryUser()
```

4. 更新用户

```
public int updateUser (String username, String password, String usertype_string,
    String foreign_id)
```

3. 实现效果展示

完整的展示见 Demo。

这里提供一些截图。

Figure 3.1: 运行截图 1

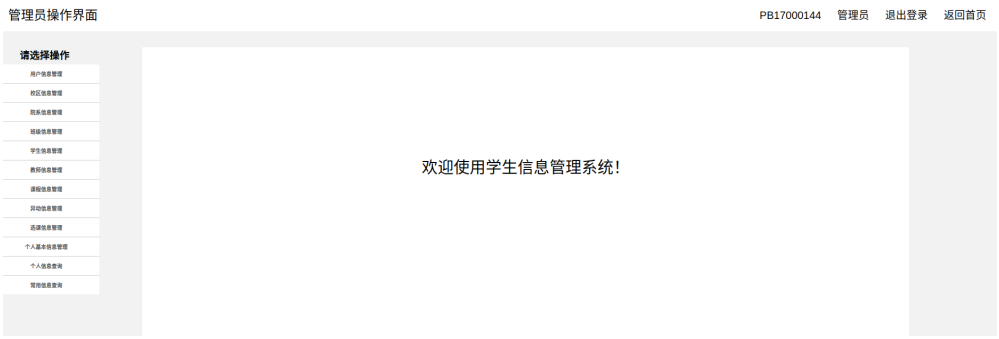

Figure 3.2: 运行截图 2

Figure 3.3: 运行截图 3

Figure 3.4: 运行截图 4

更多的展示信息见 Demo 视频。

4. 总结

通过这次实验，我个人体验熟悉了数据库应用开发，从前端到后端的全过程。

首先是学习了如何从前端到后端的构建，熟练了 IntelliJ IDEA 这一优秀开发工具。现在使用 IDEA 比起以前更加熟练了。

其次是学会了配置 Tomcat，其实这个并不是一个容易的工作，在配置的时候会遇到诸多问题。Tomcat 9.0 的文档也有一些不完善之处。多想多问，是我在这个实验过程中最大的收获。我也借助这个实验，在 StackOverflow 上结识了一位 Tomcat 大师，学到了很多不一样的东西，使得我对操作系统的认识也得到了加深。

此外，熟练了 SQL 的各种操作。各种跨表的增删查改得到了全方位的锻炼。这次实验中，我们也很难得的体验到了一次根据实际需求改良数据库的过程。这一过程让我真正感受到，设计数据库和设计好的数据库的差距。

未来，对于这个项目，其实我个人有一些想法，在代码的 TODO 部分注明。应用的健壮性、可移植性、安全性都有待提升。这个我将会在未来进行进一步的完善。