

中国科学技术大学计算机学院

《数据库系统实验报告》

实验题目：学籍管理系统

学生姓名：XXX

学生学号：XXX

完成时间：2024 年 5 月 23 日

需求分析

应用场景

学校的学籍管理系统是用于管理学生从入学到毕业期间所有学籍相关信息的系统。它服务于学校管理层、教师和学生，确保学生信息的准确性、及时性，并便于查询和统计。

数据需求

- **学生基本信息**：包括学生的姓名、性别、手机号、邮箱等。
- **学籍信息**：涵盖学生的专业、班级、学号等。
- **课程信息**：包括所有课程的基本信息，如课程名称、课程代码、学分、授课教师等。
- **成绩信息**：学生的百分制课程成绩。

具体功能

管理员可以实现下列信息的增删改查：

- **学生基本信息管理**：录入、修改、查询和删除学生的基本信息。
- **专业和班级管理**：设置和调整学生的专业和班级信息。
- **课程管理**：设置课程信息，包括课程的增加、修改、查询和删除。
- **成绩管理**：录入、修改、查询和删除学生的课程成绩。

学生可以实现以下功能：

- **查询个人信息**：学生可以查看自己的基本信息、学籍信息和课程成绩信息。
- **修改个人信息**：学生可以修改自己的部分信息，如密码、手机号和邮箱地址。
- **查询公开信息**：学生可以查询所有课程的详细信息，包括课程名称、课程代码等。
- **查询成绩**：学生可以查询自己每门课程的成绩。

其他需求

- 管理员账户的登录与鉴权
- 学生账户的登录与鉴权

总体设计

系统模块结构

系统采用 B/S 架构，分为前端、后端与数据库系统三大模块。前端使用流行的 HTML/CSS/JS 三大件，后端使用 Python + Flask 驱动，数据库系统采用所要求的 MySQL。

系统架构



系统工作流程

1. 浏览器请求网页，Flask 返回网页
2. 用户与网页交互
3. 网页上的脚本向后端发送请求
4. 后端通过 API 查询 MySQL 数据库，将结果返回前端
5. 前端将结果展示给用户

数据库设计

ER 图

MAJOR		
int	major_id	*Major ID
string	major_name	Major Name
int	major_stu_num	Major Student Count
string	dean	Dean

offers

CLASS		
int	class_id	*Class ID
string	class_name	Class Name
int	class_stu_num	Class Student Count
string	advisor	Advisor

enrolls

STUDENT		
string	stu_id	*Student ID
string	stu_name	Student Name
string	tel	Tel
string	email	Email
bool	sex	Sex (1 for male)

takes (int score)

COURSE		
string	course_id	*Course ID
string	course_name	Course Name
string	course_desc	Course Desc
string	semester	Semester
string	teacher	Teacher
float	credit	Credit
int	hours	Hours

模式分解

- ER 模型向关系模型转换
 - R_MAJOR(**major_id**, major_name, major_stu_num, dean)

- R_CLASS(**class_id**, class_name, class_stu_num, advisor, **major_id**)
- R_STUDENT(**stu_id**, stu_name, sex, tel, email, **class_id**)
- R_COURSE(**course_id**, course_name, course_desc, semester, teacher, credit, hours)
- R_SCORE(**stu_id**, **course_id**, score)
- 写出函数依赖
 - `major_id` → {`major_name`, `major_stu_num`, `dean`}
 - `class_id` → {`class_name`, `class_stu_num`, `advisor`, `major_id`}
 - `stu_id` → {`stu_password`, `stu_name`, `sex`, `tel`, `email`, `class_id`}
 - `course_id` → {`course_name`, `course_desc`, `semester`, `teacher`, `credit`, `hours`}
 - {`stu_id`, `course_id`} → `score`
- 因其没有局部函数依赖和传递依赖，显然满足 3NF

存储过程 & 事务

由于数据库内有外键约束，管理员难以“重命名”各表内的主键值 (例如 `major.major_id` 和 `class.major_id`)。因此，我设计了四个存储过程，方便这一流程：`rename_major_id`，`rename_class_id`，`rename_stu_id`，`rename_course_id`。为了保证数据的一致性，需要在这些存储过程中使用事务。它们的作用与实现方式基本一致，因此这里仅解释 `rename_major_id`。

`rename_major_id` 的用法为 `CALL rename_major_id(old_id, new_id, @state);`，其中 `old_id` 为要修改的 `major_id`，`new_id` 为想改成的 `major_id`，`@state` 为执行结果。`@state` 可能有如下几种值：

- -1: 未知错误
- 0: 执行成功
- 1: 未找到 `major_id = old_id` 的系
- 2: 已存在 `major_id = new_id` 的系

触发器

我们的数据库内有两个字段，`major_stu_num` 和 `class_stu_num`，显然需要在学生被添加/删除以及更改班级号时自动更新。那么，我们需要在 `student` 表内设置三个触发器，分别监控 `INSERT` 操作，`DELETE` 操作和 `UPDATE` 操作。

函数

为了便于查询，我设计了一个查询学生加权平均成绩的函数 `calculate_avg_score`，用法是 `SELECT calculate_avg_score('Student ID');`。

核心代码解析

仓库地址

目录

├── backend	-- 数据库用到的 SQL 文件
│ ├── create-db.sql	-- 初始化数据库的 SQL 文件
├── profile	-- 学生的个人资料照片
│ ├── default.jpg	-- 默认个人资料图片
├── scripts	-- 常用脚本
│ ├── console.sh	-- 打开 MySQL 控制台
│ ├── dev.sh	-- 以调试模式运行服务器
│ └── init-db.sh	-- 初始化数据库（执行
backend/create-db.sql)	
│ ├── prod.sh	-- 部署服务器
│ └── stop-mysql.sh	-- 终止 MySQL 数据库服务器
├── static	-- 网页用到的静态资源
│ ├── images	-- 图片
│ │ └── favicon.svg	-- 网站 Logo
│ ├── scripts	-- JavaScript 脚本
│ │ ├── admin.js	-- 管理员界面的脚本
│ │ ├── common.js	-- 通用脚本
│ │ ├── index.js	-- 主界面（登录界面）的脚本
│ │ ├── simple-datatables.js	-- 展示/编辑表格的依赖库
│ │ └── student.js	-- 学生界面的脚本
│ └── styles	-- CSS 样式表资源
│ ├── common.css	-- 通用样式表
│ ├── index.css	-- 主界面（登录界面）的样式表
│ ├── logged-in.css	-- 管理员/学生界面的样式表
│ ├── simple-datatables.css	-- simple-datatables 的样式
│ └── simple-datatables-editing.css	-- simple-datatables 的样式
├── admin.html	-- 管理员界面
├── app.py	-- 应用程序的主文件，包含了此 Web
应用的主要逻辑和路由	
├── config.json	-- 配置文件，存储应用的配置信息
├── index.html	-- 主界面（登录界面）
├── LICENSE	-- 此仓库的许可证
├── README.md	-- 此仓库的自述文件
├── requirements.txt	-- 此项目所需的 Python 依赖库
├── student.html	-- 学生界面
└── utils.py	-- app.py 所需的工具函数文件

数据库

存储过程

如 [数据库设计](#) 一节所述，这里仅解释 `rename_major_id`。此存储过程接受两个参数 `old_id` 和 `new_id`，并输出 `state` 表示执行结果。首先，它声明并初始化了两个变量 `s`

和 `tmp`，分别用于状态和临时存储。然后，它禁用外键检查以避免外键约束问题。

在主事务中，该过程首先检查 `major` 表中是否存在 `old_id`。如果不存在，设置状态 `s` 为 1；如果存在，则继续检查 `new_id` 是否已经存在于 `major` 表中。如果 `new_id` 已存在，设置状态 `s` 为 2。

如果状态 `s` 仍为 0（即没有错误），它将 `major` 表和 `class` 表中的 `major_id` 从 `old_id` 更新为 `new_id`，并提交事务。如果有任何错误或状态不为 0，则回滚事务。

最后，它重新启用外键检查并将状态 `s` 设置为输出参数 `state`。此存储过程的可能输出状态包括：0 表示成功，1 表示 `old_id` 未找到，2 表示 `new_id` 已存在，-1 表示意外错误。

```
-- backend/create-db.sql, line 95~

-- Stored procedure to rename `major_id`
-- Example: CALL rename_major_id(1, 2, @state); SELECT @state;
-- Parameters: `old_id` and `new_id`
-- Output: `state` to indicate the result
-- state: 0 for success, 1 for `old_id` not found, 2 for `new_id` already
exists, -1 for unexpected errors
DROP PROCEDURE IF EXISTS `rename_major_id`;
CREATE PROCEDURE `rename_major_id`(IN old_id INT, IN new_id INT, OUT state
INT) BEGIN
    -- Variables
    DECLARE s INT DEFAULT 0; -- State
    DECLARE tmp INT DEFAULT 0; -- Temporary variable to check if
`old_id`/`new_id` exists

    -- Error handling
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET s = -1; -- Other errors
    SET foreign_key_checks = FALSE;

    -- Main transaction: Update `major_id` in `major` and `class`
    START TRANSACTION;
    SELECT COUNT(*) INTO tmp FROM `major` WHERE `major_id` = old_id;
    IF tmp = 0 THEN
        SET s = 1; -- `old_id` not found
    ELSE
        SELECT COUNT(*) INTO tmp FROM `major` WHERE `major_id` = new_id;
        IF tmp > 0 THEN
            SET s = 2; -- `new_id` already exists
        END IF;
    END IF;
    IF s = 0 THEN
        UPDATE `major` SET `major_id` = new_id WHERE `major_id` = old_id;
        UPDATE `class` SET `major_id` = new_id WHERE `major_id` = old_id;
        COMMIT;
    ELSE
        ROLLBACK;
```

```
END IF;  
SET foreign_key_checks = TRUE;  
SET state = s;  
END; //
```

触发器

```
-- Trigger to update `major_stu_num` and `class_stu_num` when a student is  
added or removed  
CREATE TRIGGER `student_after_insert` AFTER INSERT ON `student`  
FOR EACH ROW BEGIN  
    UPDATE `class` SET `class_stu_num` = `class_stu_num` + 1 WHERE `class_id`  
= NEW.class_id;  
    UPDATE `major` SET `major_stu_num` = `major_stu_num` + 1 WHERE `major_id`  
= (SELECT `major_id` FROM `class` WHERE `class_id` = NEW.class_id);  
END;  
//  
CREATE TRIGGER `student_after_delete` AFTER DELETE ON `student`  
FOR EACH ROW BEGIN  
    UPDATE `class` SET `class_stu_num` = `class_stu_num` - 1 WHERE `class_id`  
= OLD.class_id;  
    UPDATE `major` SET `major_stu_num` = `major_stu_num` - 1 WHERE `major_id`  
= (SELECT `major_id` FROM `class` WHERE `class_id` = OLD.class_id);  
END; //  
-- Trigger to update `major_stu_num` and `class_stu_num` when a student's  
class_id is updated  
CREATE TRIGGER `student_after_update` AFTER UPDATE ON `student`  
FOR EACH ROW BEGIN  
    IF OLD.class_id <> NEW.class_id THEN  
        UPDATE `class` SET `class_stu_num` = `class_stu_num` - 1 WHERE  
`class_id` = OLD.class_id;  
        UPDATE `major` SET `major_stu_num` = `major_stu_num` - 1 WHERE  
`major_id` = (SELECT `major_id` FROM `class` WHERE `class_id` =  
OLD.class_id);  
        UPDATE `class` SET `class_stu_num` = `class_stu_num` + 1 WHERE  
`class_id` = NEW.class_id;  
        UPDATE `major` SET `major_stu_num` = `major_stu_num` + 1 WHERE  
`major_id` = (SELECT `major_id` FROM `class` WHERE `class_id` =  
NEW.class_id);  
    END IF;  
END; //
```

1. `student_after_insert` 触发器在 `student` 表中插入新记录后触发。
 - 将 `class` 表中对应 `class_id` 的 `class_stu_num` 增加 1。
 - 将 `major` 表中对应 `major_id` 的 `major_stu_num` 增加 1, 其中 `major_id` 通过 `class` 表中 `class_id` 查找得到。
2. `student_after_delete` 触发器在 `student` 表中删除记录后触发。

- 将 `class` 表中对应 `class_id` 的 `class_stu_num` 减少 1。
 - 将 `major` 表中对应 `major_id` 的 `major_stu_num` 减少 1, 其中 `major_id` 通过 `class` 表中 `class_id` 查找得到。
3. `student_after_update` 触发器在 `student` 表中更新记录后触发。
- 如果学生的 `class_id` 发生变化:
 - 将旧的 `class_id` 对应的 `class_stu_num` 减少 1。
 - 将旧的 `class_id` 对应的 `major_id` 的 `major_stu_num` 减少 1, 其中 `major_id` 通过 `class` 表查找得到。
 - 将新的 `class_id` 对应的 `class_stu_num` 增加 1。
 - 将新的 `class_id` 对应的 `major_id` 的 `major_stu_num` 增加 1, 其中 `major_id` 通过 `class` 表查找得到。
 - 否则不做任何操作。

函数

```
-- Function to calculate the weighted average score of a student
-- Example: SELECT calculate_avg_score('PB21114514');
DROP FUNCTION IF EXISTS `calculate_avg_score`;
CREATE FUNCTION `calculate_avg_score`(in_stu_id VARCHAR(10)) RETURNS FLOAT
DETERMINISTIC BEGIN
    -- sum(credit * score) / sum(credit)
    RETURN (SELECT SUM(`credit` * `score`) / SUM(`credit`) FROM `course`,
`score` WHERE `course`.`course_id` = `score`.`course_id` AND `stu_id` =
in_stu_id);
END; //
```

这个 MySQL 函数用于计算某个学生的加权平均成绩, 它接受一个参数 `stu_id`, 表示学生的 ID, 并返回一个 `FLOAT` 类型的平均成绩。

- 连接 `course` 表和 `score` 表, 根据 `course_id` 匹配两张表的数据。
- 使用 `stu_id` 作为过滤条件, 只计算特定学生的成绩。
- 计算所有课程的 `credit` (学分) 和 `score` (分数) 的乘积之和, 然后除以所有课程的 `credit` 之和, 得到加权平均成绩。

后端

此项目的后端服务器由 Flask 框架驱动, 通过 PyMySQL 与数据库建立连接。在 `utils.py` 文件中, 我定义了多个函数, 用于从数据库中读取/修改学籍信息, 以及记录用户的登录状态。例如更新学生信息的函数:

```
def updateStuInfo(conn: "Connection[Cursor]", stu_id: str, **kwargs) →
bool:
    """Update the student's information."""
    with conn.cursor() as cur: # Initialize a cursor
```

```

keys = []
values = []
for col in STUDENT_CAN_CHANGE: # Fields that a student can change
    if col in kwargs:
        keys.append(f"{col} = %s")
        values.append(kwargs[col])
r = cur.execute( # SQL query
    f"UPDATE student SET {'', ' '.join(keys)} WHERE stu_id = %s",
    (*values, stu_id)
)
conn.commit()
return bool(r)

```

记录用户的登入登出:

```

def loginUser(username: str, password: str, isAdmin: bool) → str:
    """Log in the user and return a token."""
    # Check if the user has already logged in
    for token, data in loggedInUsers.items():
        if (
            data["username"] == username
            and data["password"] == password
            and data["isAdmin"] == isAdmin
        ):
            return token
    # Generate a new token
    token = token_urlsafe(16)
    loggedInUsers[token] = {
        "username": username,
        "password": password,
        "isAdmin": isAdmin,
    }
    return token

def logoutUser(token: str) → bool:
    """Log out the user with the given token."""
    if token in loggedInUsers:
        del loggedInUsers[token]
        return True
    else:
        return False

```

在 `main.py` 文件中, 我实现了简易的登录鉴权功能; 同时使用 `utils.py` 中的方法, 对用户鉴权后再提供服务。例如用户的登入登出接口:

```

@app.route("/api/login", methods=["POST"]) # API entry
def login():
    data = request.json

```

```

    if not data:
        return abort(400)
    username = data.get("username")
    password = data.get("password")
    isAdmin = bool(data.get("is_admin", False))
    print(
        f'{"Admin" if isAdmin else "Student"} login attempt: "{username}" "
{password}"'
    )
    if isAdmin:
        data = queryAdmin(session, username, password) # Query if the user
is a valid admin
    else:
        data = queryStudent(session, username, password) # Query if the user
is a valid student
    success = bool(data)
    token = loginUser(username, password, isAdmin) if success else ""
    if success:
        log(token, "Login successful")
    return {"success": success, "isAdmin": isAdmin, "data": data, "token":
token}

@app.route("/api/logout", methods=["POST"]) # API entry
def logout():
    token = request.json.get("token")
    log(token, "Logout")
    return {"success": logoutUser(token)}

```

学生获取个人信息的接口：

```

def studentOnly(func):
    """Decorator to restrict access to student users only."""
    @wraps(func) # Preserve the original function's metadata
    def wrapper():
        token = request.json.get("token") # Get user's token from his
request
        user = loggedInQuery(token) # Is (s)he logged in?
        isAdmin = user.get("isAdmin") if user else False # Is (s)he an
admin?
        if isAdmin:
            return abort(403) # This API is intended for students ...
        stuId = user.get("username")
        return func(token, stuId) # Call original function
    return wrapper

# ...

@app.route("/api/student/info", methods=["POST"]) # API entry
@studentOnly # Restrict access to a valid logged-in student

```

```
def studentInfo(token, stuId):
    log(token, "StudentInfo", stuId)
    result = getStuInfo(session, stuId)
    return {"success": bool(result), "data": result}
```

前端

此项目的前端使用了原生的 HTML/CSS/JS 技术栈，通过 JavaScript 动态请求数据来在浏览器渲染界面。为了便于代码复用，我将学生和管理员用户都使用的相同代码放进一个文件。由于学生和管理员用户的实现逻辑相似，这里以学生用户为例讲解一部分关键代码。

```
<!-- student.html -->
<!DOCTYPE html>
<html>
<!-- Omitted document head -->
<body>
    <h1>Student Management System - Student</h1>
    <div id="nav"></div> <!-- Navbar generated by JavaScript -->
    <div id="panels">
        <div id="info"> <!-- Info panel -->
            <div id="info-display"> <!-- Info display -->
                <ul>
                    <!-- Omitted, displays student's information in a list -
→
                </ul>
                <div id="profile-display">
                    <img id="profile" title="Profile Picture"> <!-- Displays
student's profile image -->
                    <input id="profile-upload" type="file" accept=".jpg"
title="Drag and drop or click to upload a new profile picture"> <!-- Upload
new profile image -->
                </div>
            </div>
            <h2>Update Info</h2>
            <form action="/api/student/update" method="POST"
class="update">
                <!-- Omitted, form used to update student's information -->
            </form>
        </div>
        <div id="courses">
            <table>
                <!-- Omitted, displays all courses -->
            </table>
        </div>
        <div id="scores">
            <table>
                <!-- Omitted, displays studen's scores in each course -->
            </table>
```

```

        <h2>Statistics</h2>
        <div id="statistics"> ←!— Displays weighted average score →
            <b>Weighted Average Score</b>: <span id="avg-
score">null</span>
        </div>
    </div>
</div>
<div id="float-buttons">
    <a href="javascript:void(0);" title="Logout" id="op-logout"> 🚪 </a>
    <a href="javascript:void(0);" title="Refresh" id="op-
refresh"> 🔄 </a>
    <a href="javascript:void(0);" title="Go to top" id="op-top"> ⬆️ </a>
</div>
</body>

</html>

```

```

// static/scripts/student.js

// ...

async function reloadTable(panel) { // Reloads a given table, such as
"course" or "score"
    initTable(panel); // Initialize the table if not already initialized
    const dataTable = dataTables[panel.id]; // Get the dataTable object
    dataTable.setMessage("Loading ... "); // Displays a message to show that
data is being fetched
    dataTable.wrapperDOM.toggleAttribute("data-busy", true); // Make the
panel "look busy"
    const data = await
window.common.postWithToken(`/api/student/${panel.id}`); // Fetches data
from our Flask server
    dataTable.data.data = []; // Clear old data
    dataTable.insert({ data: data.data }); // Insert new data
    dataTable.wrapperDOM.toggleAttribute('data-busy', false);
    log(`Reloaded table "${panel.id}"`);
}

// ...

```

实验与测试

依赖

- 所需的库
 - flask
 - PyMySQL

- 运行环境
 - Python: 3.11
 - MySQL: 8.0.36-0ubuntu0.20.04.1

部署

1. 安装 Python 3.8+
2. 安装 MySQL 8.0+
3. 安装依赖: `pip install -r requirements.txt`
4. 允许脚本执行: `chmod +x ./scripts/*`
5. 更改 `config.json` , 提供 MySQL 的用户名和密码
6. 执行 `./scripts/init-db.sh` 来初始化数据库

运行

1. 执行 `./scripts/prod.sh`
2. 浏览器打开 `http://127.0.0.1:5000`

实验结果

查

登录后即可在对应面板进行数据查询:

Student Management System - Admin

Major

Class

Student

Course

Score

10 entries per page

Search...

Stu ID	Password	Stu Name	Sex	Tel	Email	Class ID
PB21114514	alicep@ssw0rd	Alice	0	1234567870	alice@example.com	1
PB21114515	bob's p@ssw0rd	Bob	1	1234567891	bob@example.com	1
PB21114516	charliep@ssw0rd	Charlie	1	1234567892	charlie@example.com	2

Showing 1 to 3 of 3 entries

全文搜索功能:

Student Management System - Admin

Major	Class	Student	Course	Score		
10 entries per page				ch		
Stu ID	Password	Stu Name	Sex	Tel	Email	Class ID
PB21114516	charliep@ssw0rd	Charlie	1	1234567892	charlie@example.com	2
Showing 1 to 1 of 1 entries						

增

在下方 Insert 区域输入想要插入的数据:

10 entries per page

Search...

Stu ID	Password	Stu Name	Sex	Tel	Email	Class ID
PB21114514	alicep@ssw0rd	Alice	0	1234567870	alice@example.com	1
PB21114515	bob's p@ssw0rd	Bob	1	1234567891	bob@example.com	1
PB21114516	charliep@ssw0rd	Charlie	1	1234567892	charlie@example.com	2

Showing 1 to 3 of 3 entries

Insert

PB21114517

johnp@ssw0rd

John

0

1234567893

john@example.com

2

Insert

提交后等待刷新完成即可：

10 entries per page

Search...

Stu ID	Password	Stu Name	Sex	Tel	Email	Class ID
PB21114514	alicep@ssw0rd	Alice	0	1234567870	alice@example.com	1
PB21114515	bob's p@ssw0rd	Bob	1	1234567891	bob@example.com	1
PB21114516	charliep@ssw0rd	Charlie	1	1234567892	charlie@example.com	2
PB21114517	johnp@ssw0rd	John	0	1234567893	john@example.com	2

Showing 1 to 4 of 4 entries

Insert

PB21114517

johnp@ssw0rd

John

0

1234567893

john@example.com

2

Insert

若想要复制某一行的数据，作为新数据的模板，可以在对应行右键，选择 "Duplicate"：

PB21114514	alicep@ssw0rd	Alice	0	1234567870
PB21114515	bob's p@ssw0rd	Bob	1	1234567891
PB21114516	charliep@ssw0rd		1	1234567892
PB21114517	johnp@ssw0rd		0	1234567893

Showing 1 to 4 of 4 entries

Duplicate

Rename

Remove

之后这一行的数据就会被复制到 Insert 这里：

PB21114517	johnp@ssw0rd	John	0	1234567893	john@example.com	2
------------	--------------	------	---	------------	------------------	---

Showing 1 to 4 of 4 entries

Insert

PB21114515

bob's p@ssw0rd

Bob

1

1234567891

bob@example.com

1

Insert

删

右键想要删除的行，选择 Remove：

PB21114517	johnp@ssw0rd		0	1234567893	john@example.com	2
------------	--------------	--	---	------------	------------------	---

Showing 1 to 4 of 4 entries

Duplicate

Rename

Remove

Insert

PB21114515

bob's p@ssw0rd

Bob

1

1234567891

bob@example.com

1

Insert

Rename

Old Stu ID

->

New Stu ID

Rename

在弹出的提示框中选择“确定”：

localhost:5000 显示

Are you sure?

确定

取消

刷新后即可看到效果：

10 entries per page

Search...

Stu ID	Password	Stu Name	Sex	Tel	Email	Class ID
PB21114514	alicep@ssw0rd	Alice	0	1234567870	alice@example.com	1
PB21114515	bob's p@ssw0rd	Bob	1	1234567891	bob@example.com	1
PB21114516	charliep@ssw0rd	Charlie	1	1234567892	charlie@example.com	2

Showing 1 to 3 of 3 entries

Insert

PB21114515

bob's p@ssw0rd

Bob

1

1234567891

bob@example.com

1

Insert

Rename

Old Stu ID

->

New Stu ID

Rename

改

若想要修改非主键的字段，双击此字段即可进行编辑：

Stu ID	Password	Stu Name	Sex	Tel
PB21114514	alicep@ssw0rd	Alice	0	1234567870
PB21114515	bob's p@ssw0rd	Bob	1	1234567891
PB21114516	charliep@ssw0rd	Charlie	1	1234567892

编辑完成后按下回车，数据即完成更新（按下 ESC 可以取消操作）：

Stu ID	Password	Stu Name	Sex
PB21114514	alicep@ssw0rd	Alice	0
PB21114515	bobp@ssw0rd	Bob	1
PB21114516	charliep@ssw0rd	Charlie	1

若想修改主键，则需右键此列，选择 "Rename"：

PB21114516charliep@ssw0rd11234567892charlie@example.com2

Showing 1 to 3 of 3 entries

Duplicate

Rename

Remove

Insert

随后在 "Rename" 区域输入新的主键值：

PB21114516charliep@ssw0rdCharlie11234567892charlie@example.com2

Showing 1 to 3 of 3 entries

Insert

PB21114515

bob's p@ssw0rd

Bob

1

1234567891

bob@example.com

1

Insert

Rename

PB21114516

->

PB21114517

Rename

提交后等待刷新，即可看到数据更新：

MajorClassStudentCourseScore

10 entries per page

Search...

Stu ID	Password	Stu Name	Sex	Tel	Email	Class ID
PB21114514	alicep@ssw0rd	Alice	0	1234567870	alice@example.com	1
PB21114515	bobp@ssw0rd	Bob	1	1234567891	bob@example.com	1
PB21114517	charliep@ssw0rd	Charlie	1	1234567892	charlie@example.com	2

Showing 1 to 3 of 3 entries

Insert

PB21114515

bob's p@ssw0rd

Bob

1

1234567891

bob@example.com

1

Insert

Rename

PB21114516

->

PB21114517

Rename

存储过程

在上述 [改](#) 这一节中，我们更新了 PB21114516 学号为 PB21114517，这就是通过存储过程实现的。因为在 score 表中外键引用了此学号，根据我们的存储过程，score 表应跟随着更新。那么我们查看 score 表：

10 entries per page

Search...

Stu ID	Course ID	Score
PB21114514	1	90
PB21114514	2	85
PB21114515	1	95
PB21114515	2	80
PB21114516	3	88
PB21114516	4	92

Showing 1 to 6 of 6 entries

此时学号未被更改，这是因为此时数据在服务器上已更新，但是客户端这边 (浏览器缓存) 并未更新。我们需要手动点击一下刷新按钮 ，待其刷新完成后即可看到学号已更新：

Stu ID	Course ID	Score
PB21114514	1	90
PB21114514	2	85
PB21114515	1	95
PB21114515	2	80
PB21114517	3	88
PB21114517	4	92

触发器

根据我们触发器的描述，我们可以通过插入一条学生数据来验证其正确性。可以看到插入之前有两个系，每个系各一个班：

17 / 20

Major		Class	Student	Course	Score
<div><div>10</div> entries per page</div>					<div>Search...</div>
Major ID	Major Name	Major Student Count	Dean		
1	Computer Science	2	Dr. Smith		
2	Software Engineering	1	Dr. Johnson		
Showing 1 to 2 of 2 entries					

Major		Class	Student	Course	Score
10 entries per page					Search...
Class ID	Class Name	Class Student Count	Advisor	Major ID	
1	CS101	2	Dr. Smith	1	
2	SE101	1	Dr. Johnson	2	
Showing 1 to 2 of 2 entries					

那么当我们新建一条 Class ID 为 2 的学生数据时，所有系/班级的学生数都应当为 2。那么我们构造这么一个学生：

Insert

PB21114519

qwquwu

qwq

0

1234567899

qwq@example.com

2

Insert

刷新 major 和 class 表后可以看到数据已经按照预期被更新：

Major		Class	Student	Course	Score
10 entries per page					Search...
Major ID	Major Name	Major Student Count	Dean		
1	Computer Science	2	Dr. Smith		
2	Software Engineering	2	Dr. Johnson		
Showing 1 to 2 of 2 entries					

Major		Class	Student	Course	Score
10 entries per page					Search...
Class ID	Class Name	Class Student Count	Advisor	Major ID	
1	CS101	2	Dr. Smith	1	
2	SE101	2	Dr. Johnson	2	
Showing 1 to 2 of 2 entries					

同理可验证学生的删除/更改班级，此处不再赘述。

函数

查询学生加权平均成绩的函数 `calculate_avg_score` 在学生的 `scores` 面板被使用，我们可以手工计算加权平均成绩并与之对比来验证其正确性。首先我们登录为学生 PB21114514，查看其成绩：

Info

Courses

Scores

10 entries per page

Search...

Course ID	Course Name	Score
1	Math	90
2	Physics	85

Showing 1 to 2 of 2 entries

Statistics

Weighted Average Score: 88.00

查看这两门课的学分：

Info

Courses

Scores

10 entries per page

Search...

Course ID	Course Name	Course Desc	Semester	Teacher	Credit	Hours
1	Math	Mathematics	2021 Spring	Dr. White	3	48
2	Physics	College Physics	2021 Spring	Dr. Black	2	40
3	CSAPP	Computer Systems: A Programmer's Perspective	2021 Fall	Dr. Smith	3	48
4	SEAPP	Software Engineering: A Programmer's Perspective	2021 Fall	Dr. Johnson	3	48

Showing 1 to 4 of 4 entries

手动计算其平均成绩： $avg_score = \frac{3*90+2*85}{3+2} = 88$ ，这与函数给出的平均成绩相吻合，说明了其正确性。

文件管理

在学生的 Info 面板，学生可以查看当前头像并更改。例如这是未上传头像时的界面：


Student Management System - Student

Info

Courses

Scores

- Student ID: PB21114514
- Password: [alicep@ssw0rd](#)
- Name: Alice
- Sex: 0
- Tel: 1234567870
- Email: [alice@example.com](#)
- Class ID: 1
 - Class Name: CS101
 - Class Student Count: 3
 - Advisor: Dr. Smith
- Major ID: 1
 - Major Name: Computer Science
 - Major Student Count: 3
 - Dean: Dr. Smith



未选择文件

Update Info

alicep@ssw0rd

1234567870

alice@example.com

Update

点击并选择图片，或将图片拖入指定区域即可上传新头像：

Info

Courses

Scores

File Explorer

此电脑 > Data (D:) > Download > stuAvatarSample

在 stuAvatarSample 中搜索

排序

查看

设置为背景

向左旋转

向右旋转

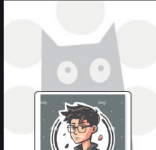
...

详细信息

default.jpg

sample1.jpg

sample2.jpg



未选择文件

sample2.jpg

+ 复制

等待刷新完成后即可看到新头像：

- **Student ID:** PB21114514
- **Password:** [alicep@ssw0rd](#)
- **Name:** Alice
- **Sex:** 0
- **Tel:** [1234567870](#)
- **Email:** [alice@example.com](#)
- **Class ID:** 1
 - **Class Name:** CS101
 - **Class Student Count:** 3
 - **Advisor:** Dr. Smith
- **Major ID:** 1
 - **Major Name:** Computer Science
 - **Major Student Count:** 3
 - **Dean:** Dr. Smith



未选择文件

Update Info

参考

- 此项目的数据库为 [MySQL](#)
- 此项目的后端是使用 [Python](#) 语言编写的，使用 [Flask](#) 作为后端框架，使用 [PyMySQL](#) 连接 MySQL 数据库
- 此项目的前端使用了 [simple-datatables](#) 来在表格中展示/修改数据
- 此项目的图标来自于 [SVG Repo](#)
- 学生的默认头像来自于 [ICourse](#)