学籍管理系统设计

姓名：王越洋 学号：22009200894

题目

设计并实现一个学生成绩管理系统的数据库，满足以下功能需求：

- 录入和管理学生基本信息。
- 按学号、姓名、专业等多种方式查询学生信息。
- 录入和管理学生的课程成绩。
- 查询学生所修课程的详细信息及计算加权平均成绩。
- 查询学生被哪些教师教授过课程。
- 查询接近被开除标准的学生。

通过该系统，学校管理人员可以高效地管理学生信息、课程安排、教师授课情况及学生成绩，及时发现并对学业不佳的学生进行学业预警。

需求分析

为了实现上述功能，数据库需要包含以下主要实体：

1. **学生（Student）**：存储学生的基本信息。
2. **班级（Class）**：管理班级信息及其所属专业。
3. **专业（Major）**：记录不同专业的基本信息。
4. **教师（Teacher）**：存储教师的基本信息。
5. **课程（Course）**：管理课程信息及其属性（必修或选修）。
6. **成绩（Grade）**：记录学生在各课程中的成绩。

7. **课程-教师关系（Course_Teacher）**：表示教师可教授哪些课程。
8. **班级-课程-教师关系（Class_Course_Teacher）**：确保一位教师只能为一个班级教授一门课程。
9. **教学计划（Plan）**：管理各专业的必修和选修学分要求。

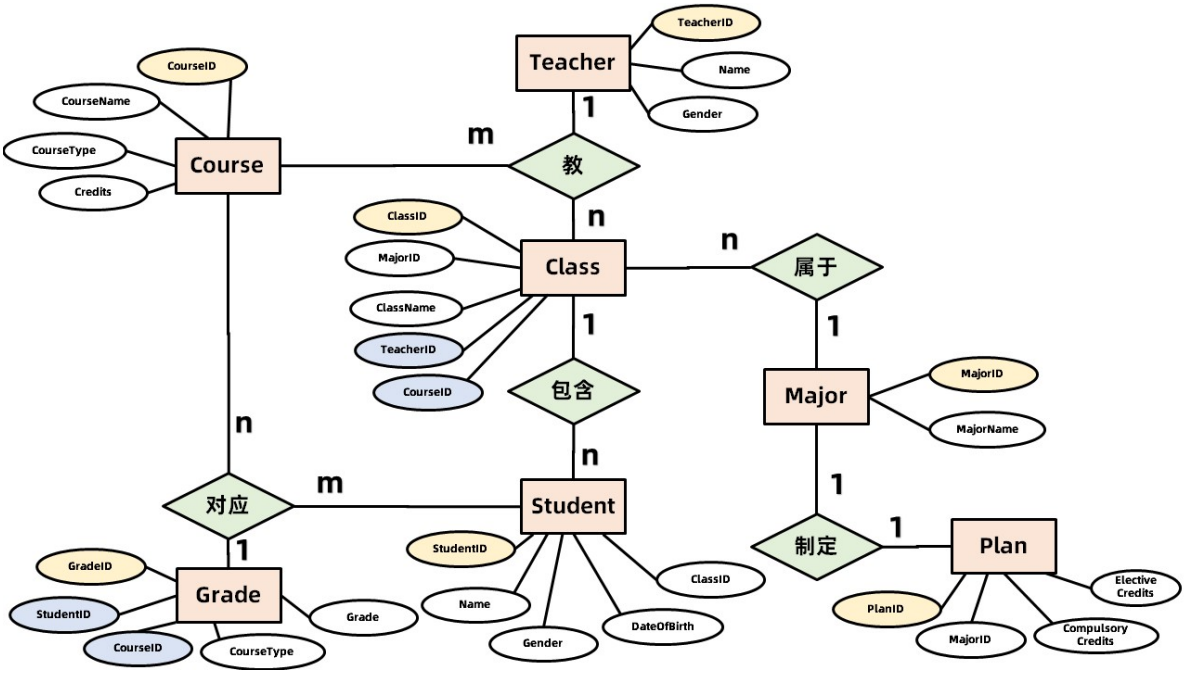
此外，系统需满足以下约束：

- **教师授课限制**：一位教师可以给多个班级带课，但不能给同一个班级带多门课。
- **开除标准**：不及格必修课累计达到10学分，或不及格选修课累计达到15学分。

数据库设计

使用MySQL 作为数据库管理系统。

实体-关系模型（E-R 图）



- **实体**：
  - **Student**：学生，属性包括StudentID、Name、Gender、DateOfBirth、ClassID。

- **Class**：班级，属性包括ClassID、MajorID、ClassName。
- **Major**：专业，属性包括MajorID、MajorName。
- **Teacher**：教师，属性包括TeacherID、Name、Gender。
- **Course**：课程，属性包括CourseID、CourseName、CourseType、Credits。
- **Grade**：成绩，属性包括GradeID、StudentID、CourseID、Grade、ResitGrade。
- **Course_Teacher**：课程-教师关系，属性包括CourseID、TeacherID。
- **Class_Course_Teacher**：班级-课程-教师关系，属性包括ClassID、CourseID、TeacherID。
- **Plan**：教学计划，属性包括PlanID、Year、Grade、MajorID、CompulsoryCredits、ElectiveCredits。

- **关系**：
  - **Student** 属于 **Class**。
  - **Class** 属于 **Major**。
  - **Student** 与 **Course** 通过 **Grade** 表建立多对多关系。
  - **Course** 与 **Teacher** 通过 **Course_Teacher** 表建立多对多关系。
  - **Class**, **Course**, **Teacher** 通过 **Class_Course_Teacher** 表建立复合关系，确保教师授课限制。
  - **Plan** 关联 **Major**，定义每个专业的教学计划。

数据库表结构

以下是各表的详细结构及约束：

1. **专业表（Major）**

属性名	数据类型	描述
MajorID	INT	主键，自增
MajorName	VARCHAR(50)	专业名称

2. **班级表（Class）**

属性名	数据类型	描述
ClassID	INT	主键，自增
MajorID	INT	外键，关联Major
ClassName	VARCHAR(50)	班级名称

3. **学生表（Student）**

属性名	数据类型	描述
StudentID	INT	主键，自增
Name	VARCHAR(50)	学生姓名
Gender	CHAR(1)	性别
DateOfBirth	DATE	出生日期
ClassID	INT	外键，关联Class

4. **教师表（Teacher）**

属性名	数据类型	描述
TeacherID	INT	主键，自增
Name	VARCHAR(50)	教师姓名
Gender	CHAR(1)	性别

5. **课程表（Course）**

属性名	数据类型	描述
CourseID	INT	主键，自增
CourseName	VARCHAR(50)	课程名称
CourseType	CHAR(1)	课程类型（'C' 必修，'E' 选修）
Credits	INT	学分

6. **课程-教师关系表（Course_Teacher）**

属性名	数据类型	描述
CourseID	INT	外键，关联Course
TeacherID	INT	外键，关联Teacher

   - **主键**：(CourseID, TeacherID)
7. **班级-课程-教师关系表（Class_Course_Teacher）**

属性名	数据类型	描述
ClassID	INT	外键，关联Class
CourseID	INT	外键，关联Course
TeacherID	INT	外键，关联Teacher

   - **主键**：(ClassID, CourseID)
   - **唯一约束**：(ClassID, TeacherID) 确保一个班级中同一教师不能教授多门课程。
8. **成绩表（Grade）**

属性名	数据类型	描述
GradeID	INT	主键，自增
StudentID	INT	外键，关联Student
CourseID	INT	外键，关联Course
Grade	INT	成绩
ResitGrade	INT	补考成绩（可选）

9. **教学计划表（Plan）**

属性名	数据类型	描述
PlanID	INT	主键，自增
Year	INT	年份
Grade	INT	年级（如大一、大二等）
MajorID	INT	外键，关联Major
CompulsoryCredits	INT	必修学分
ElectiveCredits	INT	选修学分

数据库实现

创建数据库和表

以下是完整的SQL脚本，用于创建数据库及其所有相关表，包括外键约束和唯一约束。

```
create
database if not exists xue;

use
```

```
xue;

-- 学生表
CREATE TABLE Student (
     StudentID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     Name VARCHAR(50) NOT NULL,
     Gender CHAR(1) NOT NULL,
     DateOfBirth DATE NOT NULL,
     ClassID INT
);


-- 专业表
CREATE TABLE Major (
   MajorID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   MajorName VARCHAR(50) NOT NULL
);

-- 教师表
CREATE TABLE Teacher (
     TeacherID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     Name VARCHAR(50) NOT NULL,
     Gender CHAR(1) NOT NULL
);

-- 班级-课程-教师关系表
CREATE TABLE Class (
    ClassID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ClassName VARCHAR(50) NOT NULL,
    MajorID INT NOT NULL,
    CourseID INT NOT NULL,
    TeacherID INT NOT NULL,
    UNIQUE (ClassID, CourseID, TeacherID)
);

```

```
40
41  -- 课程表
42  CREATE TABLE Course (
43      CourseID INT NOT NULL AUTO_INCREMENT PRIMARY
    KEY,
44      CourseName VARCHAR(50) NOT NULL,
45      CourseType CHAR(1) NOT NULL,   -- 'C' for
    compulsory, 'E' for elective
46      Credits INT NOT NULL
47  );
48
49  -- 成绩表
50  CREATE TABLE Grade (
51     GradeID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
52     StudentID INT,
53     CourseID INT,
54     Grade INT,
55     CourseType CHAR(1) ,-- 'C' for compulsory, 'E'
    for elective
56     UNIQUE (StudentID,CourseID)
57  );
58
59  -- 课程-教师
60  CREATE TABLE Course_Teacher (
61      CourseID INT,
62      TeacherID INT,
63      PRIMARY KEY (CourseID, TeacherID)
64  );
65
66  -- 教学计划表
67  CREATE TABLE Plan (
68        PlanID INT NOT NULL AUTO_INCREMENT PRIMARY
    KEY,
69        MajorID INT,
70        CompulsoryCredits INT NOT NULL,
71        ElectiveCredits INT NOT NULL
72  );
```

插入示例数据

为了测试系统功能，需要插入一些示例数据。

```
use xue;
INSERT INTO Major (MajorName) VALUES
('计算机科学与技术'),
('软件工程'),
('电子信息工程'),
('通信工程'),
('人工智能');


INSERT INTO Teacher (Name, Gender) VALUES
('张老师', 'M'),
('李老师', 'F'),
('王老师', 'M'),
('赵老师', 'F'),
('孙老师', 'M');

INSERT INTO Class (CourseID,MajorID, ClassName, TeacherID) VALUES
(1,1, '计算机科学与技术-1班', 1),  -- MajorID 1对应计算机科学与技术专业
(2,2,'软件工程-1班', 2),           -- MajorID 2对应软件工程专业
(3,3,'电子信息工程-1班', 3),       -- MajorID 3对应电子信息工程专业
(4,1,'通信工程-1班', 4),           -- MajorID 4对应通信工程专业
(5,1,'人工智能-1班', 5);           -- MajorID 5对应人工智能专业

INSERT INTO Student (Name, Gender, DateOfBirth, ClassID) VALUES
('张三', 'M', '2001-03-15', 1),  -- StudentID自动增长
('李四', 'F', '2001-06-21', 1),
```

```
('王五', 'M', '2001-07-11', 2),
('赵六', 'F', '2001-08-10', 2),
('孙七', 'M', '2001-09-01', 3),
('周八', 'F', '2001-12-12', 3),
('吴九', 'M', '2002-01-30', 4),
('郑十', 'F', '2002-03-10', 4),
('钱十一', 'M', '2002-05-05', 5),
('陈十二', 'F', '2002-07-20', 5);

INSERT INTO Course (CourseName, CourseType, Credits)
VALUES
('数据结构', 'C', 4),  -- Compulsory
('操作系统', 'C', 4),
('计算机网络', 'C', 3),
('数据库原理', 'C', 3),
('算法设计与分析', 'E', 2),  -- Elective
('人工智能基础', 'E', 3),
('软件工程导论', 'E', 3),
('数字电路', 'E', 2);

INSERT INTO Course_Teacher (CourseID, TeacherID)
VALUES
(1, 1),  -- 数据结构由张老师讲授
(2, 1),  -- 操作系统由张老师讲授
(3, 2),  -- 计算机网络由李老师讲授
(4, 3),  -- 数据库原理由王老师讲授
(5, 4),  -- 算法设计与分析由赵老师讲授
(6, 5),  -- 人工智能基础由孙老师讲授
(7, 2),  -- 软件工程导论由李老师讲授
(8, 3);  -- 数字电路由王老师讲授


INSERT INTO Grade (StudentID, CourseID, Grade,
CourseType) VALUES
(1, 1, 85, 'C'),  -- 张三-数据结构
(1, 2, 98, 'C'),  -- 张三-操作系统

```

```
(2, 1, 12, 'C'),  -- 李四-数据结构
(2, 3, 10, 'C'),  -- 李四-计算机网络
(3, 4, 18, 'C'),  -- 王五-数据库原理
(4, 5, 15, 'E'),  -- 赵六-算法设计与分析
(5, 6, 10, 'E'),  -- 孙七-人工智能基础
(6, 7, 12, 'E'),  -- 周八-软件工程导论
(7, 8, 13, 'E'),  -- 吴九-数字电路
(8, 1, 18, 'C');  -- 郑十-数据结构


INSERT INTO Plan (MajorID, CompulsoryCredits, ElectiveCredits) VALUES
(1, 10, 10),  -- 计算机科学与技术专业的教学计划
(2, 10, 10),   -- 软件工程专业的教学计划
(3, 10, 10),   -- 电子信息工程专业的教学计划
(4, 10, 10),   -- 通信工程专业的教学计划
(5, 10, 10);   -- 人工智能专业的教学计划
```

查询功能实现

1. 录入一位学生

操作：插入一位新学生，包括学号、姓名、性别、出生年月、班级等信息。

```
-- 插入一位新学生
INSERT INTO Student (Name, Gender, DateOfBirth, ClassID)
VALUES ('孙七', 'M', '2003-04-04', 1);  -- 学生5，计算机101班
```

2. 按学号、姓名、专业三种方式查询学生基本信息

2.1 按学号查询学生

```sql
SELECT
    S.StudentID,
    S.Name,
    S.Gender,
    S.DateOfBirth,
    C.ClassName,
    M.MajorName
FROM
    Student S
JOIN
    Class C ON S.ClassID = C.ClassID
JOIN
    Major M ON C.MajorID = M.MajorID
WHERE
    S.StudentID = 1;
```

- 通过`Student`表、`Class`表和`Major`表的联接，获取学生的详细信息。
- 替换`S.StudentID = 1`为需要查询的具体学号。

2.2 按姓名查询学生

```sql
SELECT
    S.StudentID,
    S.Name,
    S.Gender,
    S.DateOfBirth,
    C.ClassName,
    M.MajorName
FROM
    Student S
```

```
JOIN
    Class C ON S.ClassID = C.ClassID
JOIN
    Major M ON C.MajorID = M.MajorID
WHERE
    S.Name = '张三';
```

- 通过`Name`字段过滤，获取特定姓名的学生信息。

2.3 按专业查询学生

```
SELECT
    S.StudentID,
    S.Name,
    S.Gender,
    S.DateOfBirth,
    C.ClassName,
    M.MajorName
FROM
    Student S
JOIN
    Class C ON S.ClassID = C.ClassID
JOIN
    Major M ON C.MajorID = M.MajorID
WHERE
    M.MajorName = '计算机科学与技术';
```

- 通过`MajorName`字段过滤，获取特定专业的所有学生信息。

3. 录入一位学生一门课的成绩

操作：为学生录入一门课程的成绩。

```
-- 为学生学号为1的学生录入一门课程成绩
INSERT INTO Grade (StudentID, CourseID, Grade,
CourseType)
VALUES (1, 3, 80, 'E');  -- 学生1，线性代数，选修，80分
```

- `StudentID = 1`：指明成绩属于哪位学生。
- `CourseID = 3`：指明是哪门课程（线性代数）。
- `Grade = 80`：成绩为80分。
- `CourseType = 'E'`：课程类型为选修。

4. 查询一位学生所修的课程

4.1 查询学生所修课程的详细信息

```sql
SELECT
    C.CourseName,
    C.CourseType,
    C.Credits,
    G.Grade
FROM
    Grade G
JOIN
    Course C ON G.CourseID = C.CourseID
WHERE
    G.StudentID = 1;
```

4.2 查询学生的必修课加权平均成绩

```sql
SELECT
    SUM(C.Credits * G.Grade) / SUM(C.Credits) AS AvgCompulsoryGrade
FROM Grade G
         JOIN Course C ON G.CourseID = C.CourseID
WHERE G.StudentID = 1 AND G.CourseType = 'C';

```

4.3 查询学生的所有课程加权平均成绩

```sql
SELECT
    SUM(C.Credits * G.Grade) / SUM(C.Credits) AS AvgAllCoursesGrade
FROM Grade G
         JOIN Course C ON G.CourseID = C.CourseID
WHERE G.StudentID = 1;
```

- 计算学生在所有课程中的加权平均成绩。

5. 查询一位学生被哪些教师教过课

```sql
SELECT DISTINCT T.Name AS TeacherName
FROM Grade G
         JOIN Course_Teacher CT ON G.CourseID = CT.CourseID
         JOIN Teacher T ON CT.TeacherID = T.TeacherID
WHERE G.StudentID = 1;
```

7.查询一个专业的学生

```sql
SELECT
    S.StudentID,
    S.Name,
    S.Gender,
    S.DateOfBirth,
    C.ClassName,
    M.MajorName
FROM
    Student S
        JOIN
    Class C ON S.ClassID = C.ClassID
        JOIN
    Major M ON C.MajorID = M.MajorID
WHERE
    M.MajorName = '计算机科学与技术';
```

6. 查询快要被开除的学生

```sql
SELECT
    s.StudentID,  -- 学生ID
    s.Name AS StudentName,  -- 学生姓名
    compulsory.CompulsoryCredits AS EarnedCompulsoryCredits,  -- 已修的必修学分
    elective.ElectiveCredits AS EarnedElectiveCredits,  -- 已修的选修学分
    p.CompulsoryCredits,  -- 专业要求的必修学分
    p.ElectiveCredits,  -- 专业要求的选修学分
    (p.CompulsoryCredits - IFNULL(compulsory.CompulsoryCredits, 0)) AS CompulsoryCreditDifference,  -- 必修学分差距，NULL按0计算
    (p.ElectiveCredits - IFNULL(elective.ElectiveCredits, 0)) AS ElectiveCreditDifference  -- 选修学分差距，NULL按0计算
FROM
    Student s
        JOIN Class c ON s.ClassID = c.ClassID  -- 连接班级表
        JOIN Major m ON c.MajorID = m.MajorID  -- 连接专业表
        JOIN Plan p ON m.MajorID = p.MajorID  -- 连接教学计划表
        LEFT JOIN
    (  -- 计算每个学生已修的必修学分，成绩>=60的才计入学分
        SELECT g.StudentID, SUM(co.Credits) AS CompulsoryCredits
        FROM Grade g
                 JOIN Course co ON g.CourseID = co.CourseID  -- 连接课程表
        WHERE co.CourseType = 'C'  -- 仅计算必修课程
          AND g.Grade >= 60  -- 仅计算成绩大于等于60的课程
```

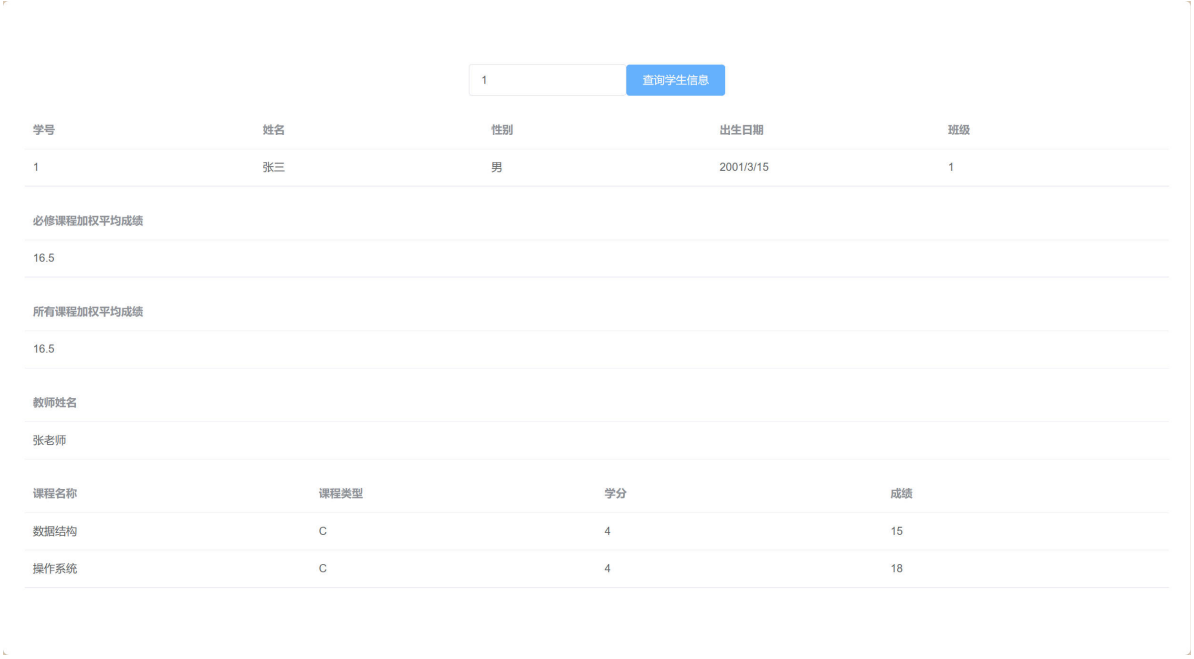
```
        GROUP BY g.StudentID  -- 按学生ID分组
    ) compulsory ON s.StudentID = compulsory.StudentID  -- 将必修学分与学生信息连接
        LEFT JOIN
    (  -- 计算每个学生已修的选修学分，成绩>=60的才计入学分
        SELECT g.StudentID, SUM(co.Credits) AS ElectiveCredits
        FROM Grade g
                 JOIN Course co ON g.CourseID = co.CourseID  -- 连接课程表
        WHERE co.CourseType = 'E'  -- 仅计算选修课程
          AND g.Grade >= 60  -- 仅计算成绩大于等于60的课程
        GROUP BY g.StudentID  -- 按学生ID分组
    ) elective ON s.StudentID = elective.StudentID  -- 将选修学分与学生信息连接
HAVING
    (p.CompulsoryCredits - IFNULL(compulsory.CompulsoryCredits, 0)) BETWEEN 0 AND 3  -- 必修学分差距不超过 3 分
    OR
    (p.ElectiveCredits - IFNULL(elective.ElectiveCredits, 0)) BETWEEN 0 AND 3;  -- 选修学分差距不超过 3 分
```

前端界面实现

使用Vue的ElementUI和Axios路由实现，利用JS操纵数据库。

具体界面如下：

学生信息界面



专业查询界面

不允许插入重复课程的成绩

查询即将被开除的学生

主要问题 & 解决办法

1. 数据唯一性问题。

如一个老师不能教一个班的多门课，同一学生同一课程成绩不能存在多个相同的。使用`UNIQUE`限制。

2. 前端js错误处理和json数据解析和组件化页面调用。

配置路由解决多页面展示问题，从`response.data`里获取需要结果。

3. **即将被开除学生查询较为繁琐。**

需要注意成绩不足60分视为没有获得学分，还要注意课程性质要分开算。

主要逻辑：

关联学生与其班级、专业和教学计划。

通过子查询计算每个学生已修的必修和选修学分（成绩≥60分）。

计算专业要求与学生已修学分之间的差距。

筛选出学分差距在0到3学分之间的学生，即那些即将满足或已经满足学分要求的学生。

细节：

选择字段

- `s.StudentID`：学生的唯一标识符。
- `s.Name AS StudentName`：学生的姓名。
- `compulsory.CompulsoryCredits AS EarnedCompulsoryCredits`：学生已修的必修学分。
- `elective.ElectiveCredits AS EarnedElectiveCredits`：学生已修的选修学分。

- `p.CompulsoryCredits`：专业要求的必修学分。
- `p.ElectiveCredits`：专业要求的选修学分。
- `CompulsoryCreditDifference`：专业要求与已修必修学分之间的差距。
- `ElectiveCreditDifference`：专业要求与已修选修学分之间的差距。

主要表及连接

- **主表**：`Student`（学生表）作为主表。
- **连接班级表**：`JOIN Class c ON s.ClassID = c.ClassID`，将学生与其所属班级关联。
- **连接专业表**：`JOIN Major m ON c.MajorID = m.MajorID`，将班级与其所属专业关联。
- **连接教学计划表**：`JOIN Plan p ON m.MajorID = p.MajorID`，获取该专业对应的教学计划（包括必修和选修学分要求）。

计算已修学分

- **已修必修学分**：

```
LEFT JOIN (
    SELECT g.StudentID, SUM(co.Credits) AS CompulsoryCredits
    FROM Grade g
    JOIN Course co ON g.CourseID = co.CourseID
    WHERE co.CourseType = 'C' AND g.Grade >= 60
    GROUP BY g.StudentID
) compulsory ON s.StudentID = compulsory.StudentID
```

子查询：计算每个学生在必修课程（`CourseType = 'C'`）中成绩达到或超过60分的总学分。

LEFT JOIN：将计算出的必修学分与学生信息关联，如果某学生没有达到条件，则`CompulsoryCredits`为`NULL`。

- **已修选修学分**：

```
LEFT JOIN (
    SELECT g.StudentID, SUM(co.Credits) AS
ElectiveCredits
    FROM Grade g
    JOIN Course co ON g.CourseID = co.CourseID
    WHERE co.CourseType = 'E' AND g.Grade >= 60
    GROUP BY g.StudentID
) elective ON s.StudentID = elective.StudentID
```

子查询：计算每个学生在选修课程（`CourseType = 'E'`）中成绩达到或超过60分的总学分。

LEFT JOIN：将计算出的选修学分与学生信息关联，如果某学生没有达到条件，则`ElectiveCredits`为`NULL`。

计算学分差距

- **必修学分差距**：

```
(p.CompulsoryCredits -
IFNULL(compulsory.CompulsoryCredits, 0)) AS
CompulsoryCreditDifference
```

- 计算专业要求的必修学分与学生已修必修学分之间的差距。使用`IFNULL`函数将`NULL`值转换为`0`，确保差距计算正确。

选修学分差距：

```
(p.ElectiveCredits - IFNULL(elective.ElectiveCredits,
0)) AS ElectiveCreditDifference
```

- ◦ 计算专业要求的选修学分与学生已修选修学分之间的差距。使用`IFNULL`函数将`NULL`值转换为`0`，确保差距计算正确。

筛选条件

```
HAVING
    (p.CompulsoryCredits -
IFNULL(compulsory.CompulsoryCredits, 0)) BETWEEN 0
AND 3
    OR
    (p.ElectiveCredits -
IFNULL(elective.ElectiveCredits, 0)) BETWEEN 0 AND 3;
```

目的：筛选出那些在必修或选修学分上距离专业要求不超过3学分的学生。

逻辑：

- **必修学分差距在0到3之间**：意味着学生在必修学分上接近满足或已经满足要求。
- **选修学分差距在0到3之间**：意味着学生在选修学分上接近满足或已经满足要求。

使用`OR`：只要满足其中一个条件，即可被包含在查询结果中。

总结

本次数据库设计构建了一个学生成绩管理系统，涵盖了学生、班级、专业、教师、课程、成绩等多个实体及其关系。通过系统的需求分析、概念结构设计（E-R图）、逻辑结构设计（关系模型转换）以及功能实现，成功完成了数据库的设计与搭建。

主要收获：

1. **数据库设计的全面理解**：深入理解了数据库设计的基本步骤，包括需求分析、E-R图设计、关系模型转换等，掌握了如何将实际需求转化为数据库结构。
2. **SQL编程技能的提升**：通过编写和优化SQL语句，增强了在MySQL环境下进行数据库操作的能力，包括表的创建、数据插入、复杂查询等。

3. **解决实际问题的能力**：在设计过程中遇到了主键冲突、外键约束调整、数据冗余等问题，通过分析和调整设计方案，提升了问题解决能力。
4. **数据一致性与完整性的维护**：学会了通过外键约束、唯一性约束等手段，确保数据的一致性和完整性，理解了数据库设计中数据规范化的重要性。

未来改进与优化方向：

1. **增强数据验证机制**：尽管移除了一些外键约束，但可以通过触发器、存储过程或应用层验证机制，进一步确保数据的有效性和一致性。
2. **性能优化**：为常用查询字段添加索引，如 `Student.Name`、`Teacher.Name`、`Course.CourseName` 等，提高查询效率。同时，定期进行数据库维护，优化表的存储和索引结构。
3. **权限管理**：实现基于角色的权限控制，确保不同用户（如管理员、教师、学生）只能访问和操作其权限范围内的数据，提升系统的安全性。
4. **扩展功能**：根据实际需求，添加更多功能模块，如补考成绩管理、课程评价系统、学费管理等，进一步完善学生成绩管理系统的功能。
5. **数据备份与恢复策略**：制定并实施定期的数据备份策略，确保数据的安全性和可恢复性，防止数据丢失或损坏。