设计学籍管理系统

一，需求分析

1.1信息要求

学籍数据库的语义如下：

1. 学校有若干专业，每个专业每年招若干个班，每个班有若干学生
2. 每个专业有自己的教学计划，规定了该专业相关课程的性质（必修或选修）以及授课学期；例如，数据库课程对计算机专业为必修、在大三上学期，但对数学专业可能为选修、在大三下学期，而中文专业可能不学这门课
3. 一位教师可以给多个班带课，但不能给一个班带多门课
4. 一门课程最多允许学生一次补考；学生达到如下条件之一的被开除：不及格必修课累计达10学分、或不及格选修课累计达15学分

数据要求

- 系：系号，名称
- 专业：专业编号，名称，所在系，教学计划id
- 班级：班号，所在专业
- 学生：学号，姓名，所在班，课程分数
- 课程：课程号，课程名称，课程分数
- 教师：工号，姓名，所在系，教课班级，教课课程
- 教学计划：计划编号，所在系，所在专业，授课学期，课程，课程性质，课程编号。

查询功能：

- 按学号、姓名、专业三种方式查询学生基本信息
- 查询一位学生所修的课程、性质（必修或选修）、学期、学分及成绩；
- 查询他的必修课平均成绩、所有课程平均成绩（平均成绩应按学分加权）
- 查询一位学生被哪些教师教过课
- 查询快要被开除的学生（距被开除差3学分之内）

插入功能：

- 建库时应录入一定数量的（不能过少）学生、教师、课程、成绩等基本信息
- 录入一位学生，应包含学号、姓名、性别、出生年月、班级等信息
- 录入一位学生一门课的成绩

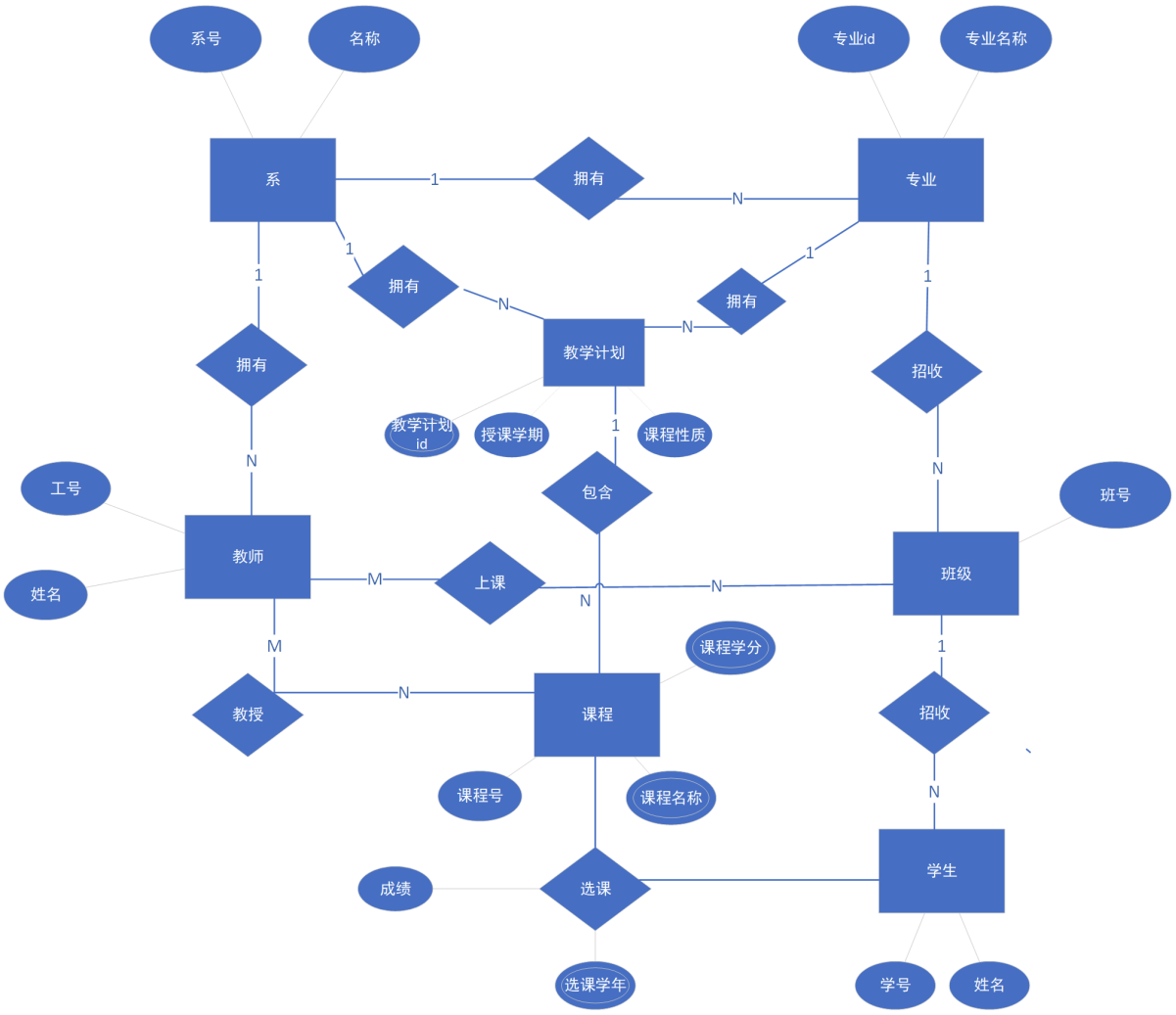
1.3数据库完整性要求

- **实体完整性**
  - 主键约束
- **参照完整性**
  - **外键约束**
  - **级联操作**，例如：

        - **删除级联**：当删除某个班级时，自动删除该班级的所有学生记录。
        - **更新级联**：当更新外键关联表的记录时，自动更新相关表的外键字段。
- **数据完整性**
    - **唯一性约束**：确保如学号、课程名称等字段在系统中唯一。
    - **非空约束**：对于关键字段（如 `student_id`、`course_id`）设置非空约束，防止插入空值。
    - **检查约束**：
        - **成绩范围**：确保成绩在 0-100 之间（`CHECK (score BETWEEN 0 AND 100)`）。
        - **课程性质**：如 `course_type` 只能为 '必修' 或 '选修'。
- **业务逻辑完整性**
    - **学籍开除规则**：
        - 必修课不及格累计达 10 学分，选修课不及格累计达 15 学分时，系统自动标记为开除状态。
        - 定期检测学生学分情况，自动生成学籍警告记录。
    - **成绩录入规则：**
        - 每门课程允许一次补考，补考成绩覆盖原成绩，但不允许超过补考次数。
        - 系里的教师可以给多个班带课，但是不能给一个班带多门课程
- **事务管理**
    - **事务：**对涉及多表的操作（如录入成绩、更新学籍状态）使用事务管理，确保操作的原子性。
    - **ACID属性**

二，概念结构设计

根据需求：设计学籍管理系统E-R图如下：



三，逻辑结构设计

转换原则：

（1）一个实体通常转换成一个数据表；
（2）一个多对多的关系，通常也转换成一个数据表；
（3）一个 1 对 1 ，或者 1 对多的关系，往往通过表的外键来表达，而不是设计一个新的数据表；
（4） 属性转换成表的字段。

将E-R图转换为关系模式如下：

四，详细实现及源代码

4.1 项目目录说明

```
卷 Data 的文件夹 PATH 列表
卷序列号为 7EBD-06B9                    文件说明
D:.
|   report.md                           ;md版本笔记
|   report.pdf                          ;pdf版本报告
|   tree.txt                            ;文件夹说明
|   建表.sql                             ;建表实现
|   查询功能.sql                          ;查询实现
|   建库(表+触发器+存储函数).sql            ;studdent_status项目库生成
|
├─assets                                ;md文件的图片目录
|       xxx.png
├─架构图
```

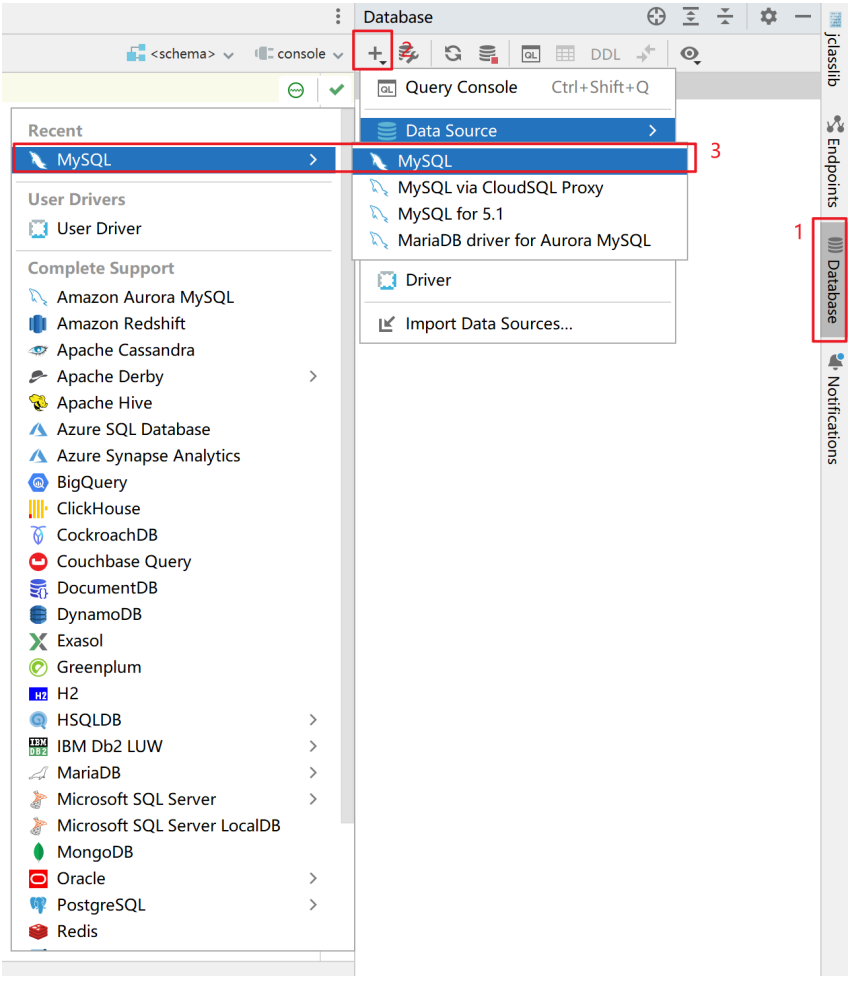
```
|       概念结构.png
|       概念结构.vsdx
|       逻辑结构设计.png
|       逻辑结构设计.vsdx
|
├─测试数据
|       class.sql                                ;班级mock数据插入
|       conduct.sql                              ;代课情况mock数据插入
|       course.sql                               ;course脚本数据插入
|       course.xlsx                              ;爬虫得到的学校课程信息，只有少部分
|       curricular_variable.sql                  ;学生选课mock数据插入
|       department.sql                           ;系mock数据插入
|       department.xlsx                          ;整理得到的西电系信息
|       major.sql                                ;专业mock数据插入
|       major.xlsx                               ;整理得到的西电专业信息
|       plan.sql                                 ;教学计划mock数据插入
|       student.sql                              ;学生mock数据插入
|       teacher.sql                              ;老师mock数据插入
|       测试数据是否插入正确.sql                    ; 测试文件，测试单个插入和查看表信息
|
└─触发器
        conduct插入检查.sql                      ;一门老师可以给多个班代课但不能给一个班代多
门课
        仅允许一次补考触发器.sql                  ;仅允许一次补考程序+测试代码
```

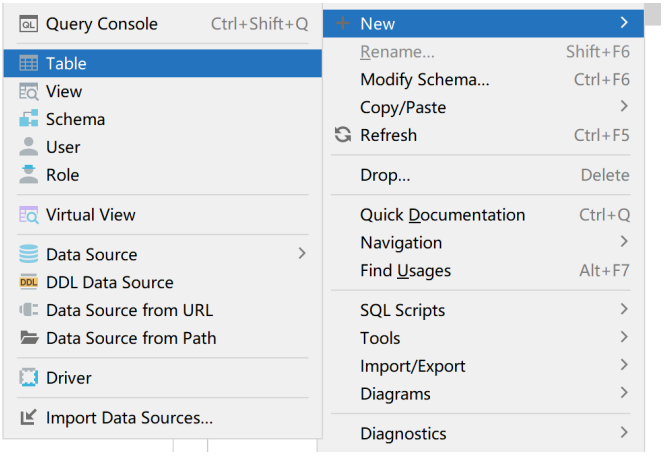
注：如何生成目录树

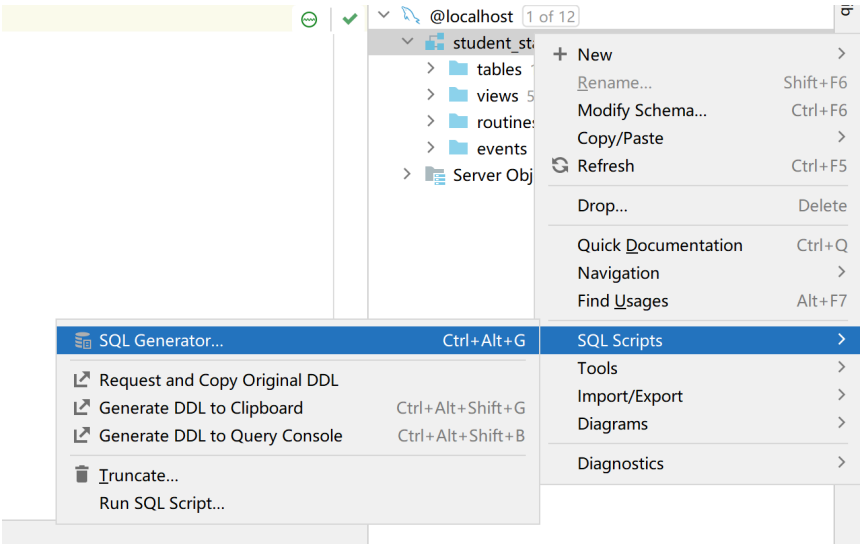
```
tree /F >tree.txt
首先打开当前目录
-- tree 生成目录树明令
/F 生成包括子文件下的文件树枝
> tree.txt 存储到当前目录的的tree.txt文件中
```

4.2根据建立的关系模式，编写sql语句

编写sql语句可以利用IDEA的图像画界面：连接数据库——建表——生成DDL


Database
<schema>
console
DDL
Query Console Ctrl+Shift+Q
Data Source
MySQL
MySQL via CloudSQL Proxy
MySQL for 5.1
MariaDB driver for Aurora MySQL
Driver
Import Data Sources...
Recent
MySQL
User Drivers
User Driver
Complete Support
Amazon Aurora MySQL
Amazon Redshift
Apache Cassandra
Apache Derby
Apache Hive
Azure SQL Database
Azure Synapse Analytics
BigQuery
ClickHouse
CockroachDB
Couchbase Query
DocumentDB
DynamoDB
Exasol
Greenplum
H2
HSQLDB
IBM Db2 LUW
MariaDB
Microsoft SQL Server
Microsoft SQL Server LocalDB
MongoDB
Oracle
PostgreSQL
Redis
1
3
jclasslib
Endpoints
Database
Notifications


Query Console Ctrl+Shift+Q
Table
View
Schema
User
Role
Virtual View
Data Source
DDL Data Source
Data Source from URL
Data Source from Path
Driver
Import Data Sources...
New
Rename... Shift+F6
Modify Schema... Ctrl+F6
Copy/Paste
Refresh Ctrl+F5
Drop... Delete
Quick Documentation Ctrl+Q
Navigation
Find Usages Alt+F7
SQL Scripts
Tools
Import/Export
Diagrams
Diagnostics


@localhost 1 of 12
tables
views
routines
events
Server Obj
New
Rename... Shift+F6
Modify Schema... Ctrl+F6
Copy/Paste
Refresh Ctrl+F5
Drop... Delete
Quick Documentation Ctrl+Q
Navigation
Find Usages Alt+F7
SQL Generator... Ctrl+Alt+G
Request and Copy Original DDL
Generate DDL to Clipboard Ctrl+Alt+Shift+G
Generate DDL to Query Console Ctrl+Alt+Shift+B
Truncate...
Run SQL Script...
SQL Scripts
Tools
Import/Export
Diagrams
Diagnostics

4.3插入测试数据

插入单个测试数据

我们首先插入单个测试数据，检查是否冲突，基础功能是否正常。

成批插入数据

我们这里选用三种方式，网站爬虫，借用mock生成工具，编写sql脚本，

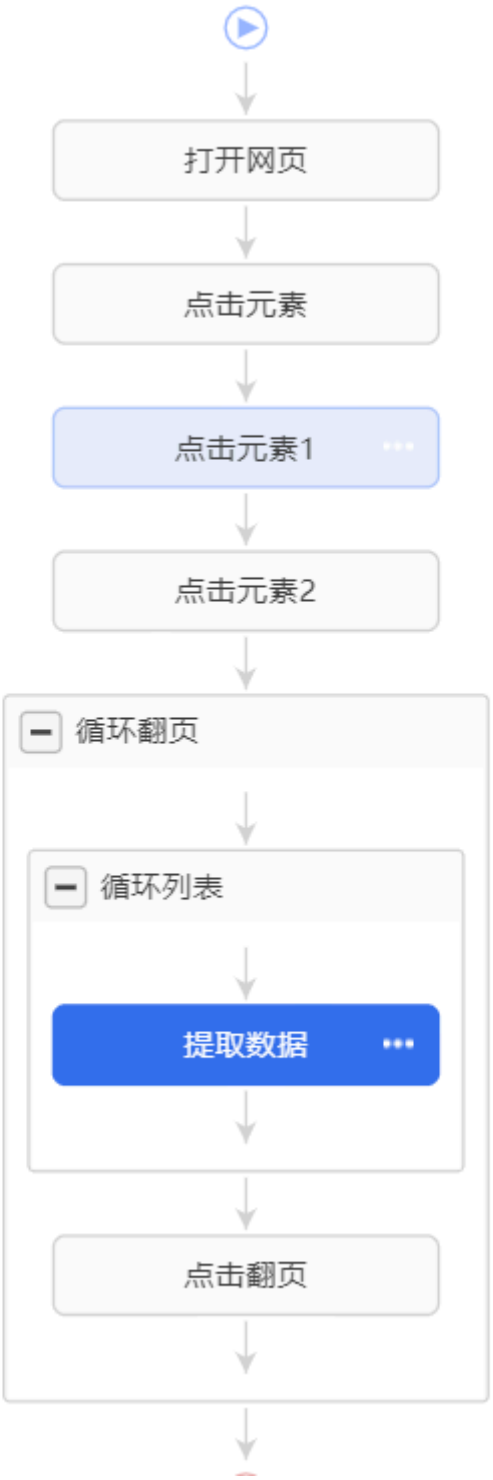
方式1——爬虫

以course为例

向八抓鱼软件复制全校课表系统链接：

```
https://ehall.xidian.edu.cn/jwapp/sys/kcbcx/*default/index.do?amp_sec_version_=1&gid_=aEZrVVJMcEdzbDRxZjhaNVhFK2dpNUxGQ2EzdmpVK1FTN25ZMXZLZUZiTDlKbUdkaER0OHh5NTFzbTVsSzA0cEJzM2RFVEJYQkpMY28wZEJidS9jN2c9PQ&EMAP_LANG=zh&THEME=cherry#/qxkcb
```

编辑抓取流程，选择去重。这里怎么使用需要根据网站指引摸索摸索。



导出csv文件后，在excel编辑好后导入：Convert Excel to Insert SQL Online - Table Convert Online

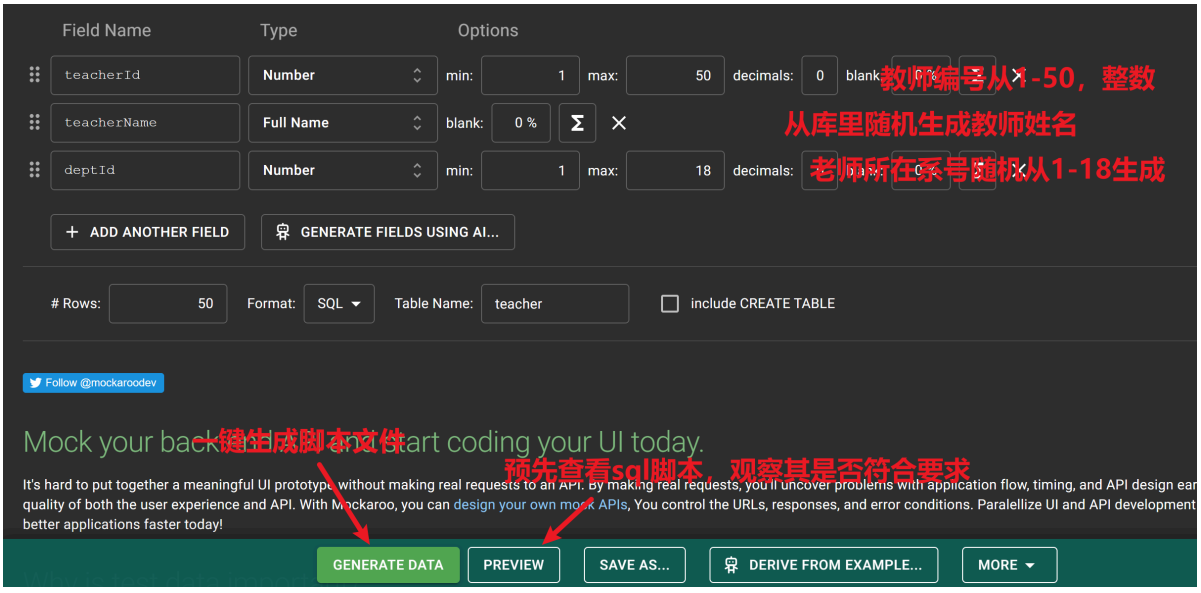
最后生成`sql`文件执行即可，插入成功：

courseId	courseName	credit
1	职业发展与就业指导（上）	1
2	新生研讨与学科导论(I)	5
3	班级指导	3
4	西电校史	4
5	Microwave Techniques/微波技术	2
6	High Frequency Circuit/高频电路	2
7	电路分析基础	4
8	数字电路与逻辑设计	4
9	电子线路实验（I）	3
10	程序思维：让人脑像电脑那样思考	1
11	生产实习	2
12	课程设计	1
13	电子系统设计实践	2
14	毕业设计	5
15	信号处理与系统分析	2
16	复变函数	2
17	微处理器系统与应用	3
18	高频电子线路	3
19	随机信号分析	3
20	通信原理	3

提取图片表格工具：[免费在线图片转 Excel 工具 - docsmall]

方式2——mock生成网站

[Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel]



2.方式2——sql脚本

- **班级生成脚本**

思路：遍历每个专业号i，生成5个班级号 i*100 +1*， *i*100 +2，

i*100 +3*， *i*100 +4， i*100 +5.

```
DELIMITER //

CREATE PROCEDURE generate_classes()
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE j INT; -- 将 j 的声明移到最前面
```

```
    WHILE i <= 65 DO
        SET j = 1; -- 每次外层循环重新初始化 j
        WHILE j <= 5 DO
            INSERT INTO class (classId, majorId)
            VALUES (i * 100 + j, i);
            SET j = j + 1;
        END WHILE;
        SET i = i + 1;
    END WHILE;
END;
//
DELIMITER ;

//调用
CALL generate_classes();

-- 查看生成的班级数据
SELECT * FROM class;
```

- **plan生成脚本：**

思路：一共生成65*5个planId，每5个planId对应一个majorId，majorId是从1-65，一个major生成5组（courseId，semester，property），并且majorid中的courseId不能重复。其中courseId是从1-27随机生成的，sesmester是从1-8随机生成的，peoperty是从1-3随机生成的。

```
-- 首先确保临时表不存在
DROP TEMPORARY TABLE IF EXISTS temp_course;

-- 创建临时表（只创建一次）
CREATE TEMPORARY TABLE temp_course (
    courseId INT PRIMARY KEY
);

DELIMITER //

CREATE PROCEDURE generate_plan_data()
BEGIN
    -- 将所有变量声明放在存储过程的开头
    DECLARE od INT DEFAULT 1;
    DECLARE j INT;
    declare i int default 1;
    DECLARE course_count INT;
    DECLARE random_courseId INT;
    DECLARE random_semester INT;
    DECLARE random_property INT;

    -- 外层循环，用于生成 65*5 个 planId 对应的 majorId
    WHILE od <= 65*5 DO
        -- 每个 planId 随机生成 5 门课程
        SET course_count = 5;
        -- 清空临时表
        TRUNCATE TABLE temp_course;

        -- 初始化 j
```

```
        SET j = 1;

        -- 内层循环，用于为每个 planId 生成 courseId, semester, property
        WHILE j <= course_count DO
            -- 随机生成 courseId（确保唯一）
            course_loop: LOOP
                SET random_courseId = FLOOR(1 + RAND() * 27); -- 生成 1-27 的随机数

                -- 检查是否已经存在于 temp_course 表中
                IF (SELECT COUNT(*) FROM temp_course WHERE courseId =
random_courseId) = 0 THEN
                    -- 如果不存在，则插入并退出循环
                    INSERT INTO temp_course (courseId) VALUES (random_courseId);
                    LEAVE course_loop;
                END IF;
            END LOOP course_loop;

            -- 随机生成 semester 和 property
            SET random_semester = FLOOR(1 + RAND() * 8); -- 生成 1-8 的随机数
            SET random_property = FLOOR(1 + RAND() * 3); -- 生成 1-3 的随机数

            -- 插入数据到 plan 表

            INSERT INTO plan (planId, majorId, courseId, semester, property)
            VALUES (od, i, random_courseId, random_semester, random_property);
            set od = od + 1;
            -- 增加 j 的值
            SET j = j + 1;
        END WHILE;
        SET i = i + 1;
    END WHILE;
END;
//
DELIMITER ;

-- 调用存储过程生成数据
CALL generate_plan_data();

-- 查看生成的数据
SELECT * FROM plan ORDER BY planId, courseId;
```

- 其余脚本见插入数据目录附件

4.4视图建立

- 创建`studetnt_baseinfo`，按学号、姓名、专业三种方式查询学生基本信息

```
-- 按学号、姓名、专业三种方式查询学生基本信息
CREATE VIEW student_baseinfo
AS
SELECT stuId '学号',stuName '姓名',class.classId '班级号',majorName '专业',deptName
'所在系'
FROM student,class,major,department
WHERE student.classId = class.classId AND
      class.majorId = major.majorId AND
      major.deptId = department.deptId;
```

- 查询一位学生所修的课程、性质（必修或选修）、学期、学分及成绩；

思路：以学号stuId为主键，首先拿着学生的stuId在curricularId_variable 查出对应的成绩和课程编号，拿着这些课程编号在course表中找出对应的学课程名，还需要拿着stuId在student表中找出classId班级号，再拿着classId在majorId中找到专业号，联合majorId和courseId在plan表中查找semester和property

```
-- 创建'student_courses_info'视图，查询一位学生所修的课程、性质（必修或选修）、学期、学分及
成绩;
CREATE VIEW student_courses_info AS
SELECT
    student.stuId,  -- 学生ID
    student.stuName,  -- 学生姓名
    course.courseId,  -- 课程ID
    course.courseName,  -- 课程名称
    course.credit,  -- 学分
    plan.property,  -- 课程性质（必修或选修）
    plan.semester,  -- 学期
    curricular_variable.grade  -- 成绩
FROM
    curricular_variable
INNER JOIN
    student ON curricular_variable.stuId = student.stuId  -- 关联学生
INNER JOIN
    course ON curricular_variable.courseId = course.courseId  -- 关联课程
INNER JOIN
    class ON student.classId = class.classId  -- 关联班级
INNER JOIN
    major ON class.majorId = major.majorId  -- 关联专业
INNER JOIN
    plan ON major.majorId = plan.majorId  -- 关联专业和课程
        AND curricular_variable.courseId = plan.courseId  -- 课程对应
ORDER BY
    student.stuId, course.courseId;  -- 按学生ID和课程ID排序
```

测试：

```
SELECT *
FROM student_courses_info
WHERE stuId = 1;-- 查询一号学生课程信息
```

- 查询他的必修课平均成绩、所有课程平均成绩（平均成绩应按学分加权）

```
```

```sql
-- 查询他的必修课平均成绩、所有课程平均成绩（平均成绩应按学分加权）
CREATE OR REPLACE VIEW student_average_scores AS
SELECT
    s.stuId,
    s.stuName,

    -- 计算必修课的平均成绩，保留四位小数
    ROUND(AVG(CASE WHEN sci.property = '必修' THEN sci.grade END), 4) AS 必修课平均成绩,

    -- 计算所有课程的加权平均成绩，保留四位小数
    ROUND(SUM(sci.grade * sci.credit) / SUM(sci.credit), 4) AS 所有课程加权平均成绩

FROM
    student_courses_info sci
INNER JOIN
    student s ON sci.stuId = s.stuId

GROUP BY
    s.stuId, s.stuName
ORDER BY
    s.stuId;
```

测试：

```sql
SELECT * FROM student_average_scores WHERE stuId = 1;
```

- 查询快要被开除的学生（距被开除差3学分之内）

每个学生的必修，限选和任选课程学分统计

```sql
-- 每个学生的必修、限选和任选课程通过学分统计
CREATE OR REPLACE VIEW student_credit_summary AS
SELECT
    s.stuId,
    s.stuName,

    -- 计算必修课通过学分
    SUM(CASE WHEN sci.property = '必修' AND sci.grade >= 60 THEN sci.credit ELSE 0 END) AS 必修通过学分,

    -- 计算限选课通过学分
    SUM(CASE WHEN sci.property = '限选' AND sci.grade >= 60 THEN sci.credit ELSE 0 END) AS 限选通过学分,

    -- 计算任选课通过学分
    SUM(CASE WHEN sci.property = '任选' AND sci.grade >= 60 THEN sci.credit ELSE 0 END) AS 任选通过学分

FROM
    student_courses_info sci
INNER JOIN
    student s ON sci.stuId = s.stuId
```

```
GROUP BY
    s.stuId, s.stuName
ORDER BY
    s.stuId;

SELECT * FROM student_credit_summary;
```

不及格学分统计

```
-- 不及格学分
CREATE OR REPLACE VIEW student_failed_credits AS
SELECT
    s.stuId,
    s.stuName,

    -- 统计不及格的必修课学分
    SUM(CASE WHEN sci.property = '必修' AND sci.grade < 60 THEN sci.credit ELSE 0
END) AS failed_required_credits,

    -- 统计不及格的选修课（包括限选和任选）学分
    SUM(CASE WHEN (sci.property = '限选' OR sci.property = '任选') AND sci.grade <
60 THEN sci.credit ELSE 0 END) AS failed_elective_credits

FROM
    student_courses_info sci
INNER JOIN
    student s ON sci.stuId = s.stuId

GROUP BY
    s.stuId, s.stuName;

select * from student_failed_credits;
```

筛选出待开除的学生（3学分以内）

```
SELECT
    sfc.stuId,
    sfc.stuName,
    sfc.failed_required_credits,
    sfc.failed_elective_credits
FROM
    student_failed_credits sfc
WHERE
    -- 筛选必修课不及格累计 7 到 9 学分的学生
    (sfc.failed_required_credits BETWEEN 7 AND 9)

    OR

    -- 筛选选修课不及格累计 12 到 14 学分的学生
    (sfc.failed_elective_credits BETWEEN 12 AND 14)
ORDER BY
    sfc.stuId;
```

- 查询一位学生被哪些教师教过课

思路：根据它的stuId，在student_baseInfo中找到班级号classId，再在student_courses_info中找到课程号courseId，联合classId和courseId在conduct中找到teacherId，根据这些teacherId在teacher表中找到老师信息（teacherId，teacherName，deptId）老师工号，老师姓名，老师所在系

```
SELECT
    t.teacherId AS '老师工号',
    t.teacherName AS '老师姓名',
    t.deptId AS '所在系'
FROM
    student_baseinfo sbi
INNER JOIN
    student_courses_info sci ON sbi.`学号` = sci.stuId
INNER JOIN
    conduct c ON c.classId = sbi.`班级号` AND c.courseId = sci.courseId
INNER JOIN
    teacher t ON t.teacherId = c.teacherId
WHERE
    sbi.`学号` = ? ;  -- 这里替换为要查询的学生ID
```

4.5触发器

老师代课约束

1. **完整性约束**
2. **设计逻辑**
    1. 插入`conduct`之前，检查是否有教师视图给同一班级教授多门课程。
    2. 如果违反约束，则抛出错误提示
    3. 否则，允许插入操作正常执行。

```
DELIMITER //

CREATE TRIGGER conduct_instead
BEFORE INSERT ON conduct
FOR EACH ROW
BEGIN
    -- 检查同一老师是否已经给该班级教授了其他课程
    IF EXISTS (
        SELECT 1
        FROM conduct c
        WHERE c.teacherId = NEW.teacherId
          AND c.classId = NEW.classId
          AND c.courseId != NEW.courseId
    ) THEN
        -- 如果违反约束，抛出错误
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = '同一教师不允许给同一个班级教授多门课程';
    END IF;
END//
```

```
DELIMITER ;
```

```
28  INSERT INTO conduct (teacherId, courseId, classId)
29  VALUES (1, 2, 601); -- 一个老师给多个班代课。
30
```

1 个配置文件 2 信息 3 表数据 4 信息

```
1 queries executed, 1 success, 0 errors, 0 warnings

查询：INSERT INTO conduct (teacherId, courseId, classId) VALUES (1, 2, 601)

共 1 行受到影响
```

```
30  INSERT INTO conduct (teacherId, courseId, classId)
31  VALUES (1, 3, 601); -- 一个老师给1个班代多门课。
```

1 信息 2 表数据 3 信息

```
1 queries executed, 0 success, 1 errors, 0 warnings

查询：INSERT INTO conduct (teacherId, courseId, classId) VALUES (1, 3, 601)

错误代码： 1644
同一教师不允许给同一个班级教授多门课程
```

仅允许一次补考

思路：创建一个 `BEFORE INSERT` 触发器来检查是否 `flag=2`，如果是则阻止插入。

创建一个存储过程来处理插入逻辑。如果记录存在：

- `flag=1` 时更新记录。
- `flag=2` 时则报错。

插入数据时调用存储过程而不是直接插入。

把更新逻辑放在存储函数，避免放在触发器中导致递归调用。

```
DELIMITER //

CREATE PROCEDURE insert_or_update_curricular(
    IN p_stuId INT,
    IN p_courseId INT,
    IN p_grade FLOAT
)
BEGIN
    DECLARE existing_flag INT DEFAULT 0;
    DECLARE existing_id INT DEFAULT 0;

    -- 查询是否有相同的 stuId 和 courseId 的记录
    SELECT flag, curricularId INTO existing_flag, existing_id
    FROM curricular_variable
    WHERE stuId = p_stuId AND courseId = p_courseId
    LIMIT 1;

    -- 情况1: 如果记录存在且 flag=2，抛出错误
    IF existing_flag = 2 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '考试次数已达上限，不允许再次补考';

    -- 情况2: 如果记录存在且 flag=1，更新成绩并将 flag 设为2
    ELSEIF existing_flag = 1 THEN
```

```
        UPDATE curricular_variable
        SET grade = p_grade, flag = 2
        WHERE curricularId = existing_id;

    -- 情况3: 如果没有记录，直接插入
    ELSE
        INSERT INTO curricular_variable (stuId, courseId, grade, flag)
        VALUES (p_stuId, p_courseId, p_grade, 1);
    END IF;
END;
//

DELIMITER ;



DELIMITER //

CREATE TRIGGER trg_check_flag_before_insert
BEFORE INSERT ON curricular_variable
FOR EACH ROW
BEGIN
    -- 如果已经有相同的 stuId 和 courseId 并且 flag=2，则抛出错误
    IF EXISTS (
        SELECT 1 FROM curricular_variable
        WHERE stuId = NEW.stuId AND courseId = NEW.courseId AND flag = 2
    ) THEN
        SIGNAL SQLSTATE '46000' SET MESSAGE_TEXT = '考试次数已达上限，不允许再次补考';
    END IF;
END;
//

DELIMITER ;
```

```
CALL insert_or_update_curricular(1, 9, 85);

-- 再次插入相同课程记录，更新 flag 为 2
CALL insert_or_update_curricular(1, 9, 95);

-- 再次插入相同课程，预期报错
CALL insert_or_update_curricular(1, 9, 98);
```

curricularId	stuId	courseId	grade	flag
2	1	11	90	2
3	1	14	53	1
4	1	15	98	2
5	1	21	29	2
1180	1	9	85	1

curricularId	stuId	courseId	grade	flag
2	1	11	90	2
3	1	14	53	1
4	1	15	98	2
5	1	21	29	2
1180	1	9	95	2

1 信息 2 表数据 3 信息

1 queries executed, 0 success, 1 errors, 0 warnings

查询：CALL insert_or_update_curricular(1, 9, 98)

错误代码： 1644
考试次数已达上限，不允许再次补考

五，问题解决

1. plan的主键设计

初始版本我选择将planId，majorId，deptId作为复合主键使用`primary key (planId, majorId, deptId)`，认为查询plan需要根据majorId，deptId联合去查，所以设置为复合主键，planId作为单独索引也将其作为主键。但是这完全错误，plan是唯一的，不应该在与其它字段组成主键，否则会造成复合主键使用不当。

加上majorId 和DeptId等于是增加了荣誉，因为planId本身已经能唯一标识一条记录，即使采用三者的复合主键，查询依然可以单独用planId来定位记录。

使用（majorId，deptId）作为联合查询条件可以通过UNIQUE来约束，而不是复合主键。

而且将三者作为复合主键，其他表在引用plan表时需要同时提供这三个字段，增加数据表关联复杂度。

2. curricular_variable主键设计顺序

最初我的主键顺序为`(courseId, stuId)`，测试插入后，发现同一门课程只能由同一学生选一次。但从业务上看，更符合实际情况的是，某个学生选修的所有课程应唯一，而不是课程为主键。因此纠正为(stuId, courseId)

```
primary key (stuId, courseId)
```

3. 枚举类型显示

为节省存储，将学期（大一上，大一下.....,大四下），以及(必修，限修，任选)设置为int型，认为扩展性比较好，枚举值在前后端做，习惯性在前端做直观展示，但我们仅用到数据库，发现用int型不够直观，故修改为ENUM型。索引从1开始

```
property enum('任选', '必修', '限选') not null comment '课程性质'
```

4. 语句作用域问题

在编写班级测试用例脚本时，初始版本如下：

```
DELIMITER //
CREATE PROCEDURE generate_classes()
BEGIN
    DECLARE i INT DEFAULT 1;
    WHILE i <= 65 DO
        DECLARE j INT DEFAULT 1;
        WHILE j <= 5 DO
            INSERT INTO class (classId, majorId)
            VALUES (i * 100 + j, i);
            SET j = j + 1;
        END WHILE;
        SET i = i + 1;
    END WHILE;
END;
//
```

```
DELIMITER ;
```

出现问题：

```
这段代码有错误，大概是在'DECLARE j INT DEFAULT 1;
        WHILE j <= 5 DO
            INSERT INTO class (' at line 5
```

该问题是 MySQL 存储过程中的 `DECLARE` **语句作用域** 引起的。在 MySQL 中，所有 `DECLARE` 语句都必须出现在 `BEGIN...END` 块的开头。也就是说，一旦开始编写逻辑（如 `WHILE` 循环），就不能再使用 `DECLARE` 语句。

解决方法：

将 `DECLARE j INT DEFAULT 1;` 移动到存储过程的最开始部分，确保它在所有逻辑之前定义。

5.删除带有外键限制的列

第一次删除后有提示外键名的信息

```
ALTER TABLE student DROP FOREIGN KEY student_classId_fk;
```

也可以使用这个命令查看外键约束

```
SHOW CREATE TABLE student;
```

再删除列：

```
ALTER TABLE student DROP COLUMN classId;
```

6.REPEAT UNTIL 循环格式

在编写plan脚本时：

```
REPEAT
    SET random_courseId = FLOOR(1 + RAND() * 27); -- 生成 1-27 的随机数
UNTIL (SELECT COUNT(*) FROM temp_course WHERE courseId = random_courseId) = 0;
```

每个planId教学计划内都有多门课程，所在在每个planId内部需要循环多门不重复的courseId，最初选用REPEAT UNTIL不断循环直到找到不重复的courseId，但是发现在这附近不断报错。排查其它无果后想着这个命令很少见，选择用LOOP替换成功解决问题。

```
-- 随机生成 courseId（确保唯一）
course_loop: LOOP
    SET random_courseId = FLOOR(1 + RAND() * 27); -- 生成 1-27 的随机数

    -- 检查是否已经存在于 temp_course 表中
    IF (SELECT COUNT(*) FROM temp_course WHERE courseId = random_courseId) = 0
THEN
        -- 如果不存在，则插入并退出循环
        INSERT INTO temp_course (courseId) VALUES (random_courseId);
        LEAVE course_loop;
    END IF;
END LOOP course_loop;
```

猜测可能是因为mysql8.0解析冲突吧。

7.游标返回

在生成选课表时，它的逻辑是遍历i = 230，每个循环对应个学生，stuId=i，courseId需要先根据它的classId在class表中找到majorId，再拿着这个majorId在major表中找到courseId，courseId不是单独一个，而可能有好几个，我们只能从查出来的courseId中挑选几个作为courseId。

```
            -- 获取一个随机的 courseId
            FETCH course_cursor INTO random_courseId;

            -- 如果已经遍历完所有课程，则退出循环
            IF done THEN
                CLOSE course_cursor;
                DROP TEMPORARY TABLE IF EXISTS temp_courses;
                LEAVE;
            END IF;
```

最后测试不能插入空数据，猜想FETCH可能会返回NULL，查询后发现：

> `FETCH` 语句在使用游标时是可能返回 `NULL` 或者没有数据的，尤其是在游标没有更多记录可读取时。当游标没有数据时，`FETCH` 操作会导致 `NOT FOUND` 错误，通常我们会通过一个异常处理机制来捕获这个问题。
>
> 在你的存储过程里，你使用了 `DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;`，这表明当游标遍历完成，没有更多记录时，`done` 变量会被设置为 `TRUE`，以便我们可以在代码中进行控制。
>
> 所以，`FETCH` 在没有更多数据时会成功执行，但是它会把变量 `random_courseId` 设置为 `NULL`，并触发 `NOT FOUND` 处理器来结束游标的读取过程。

解决办法：
可以在 `FETCH` 之后加一些判断逻辑，确保只有在游标返回有效数据时才进行后续操作。

```
-- 如果游标没有返回有效数据，random_courseId 可能是 NULL
-- 可做一些额外处理，确保数据有效性
IF random_courseId IS NOT NULL THEN
    -- 执行插入操作
    INSERT INTO curricula_variable (stuId, courseId, grade)
    VALUES (stuId, random_courseId, random_grade);
END IF;
```

8.仅允许一次补考

初始方案

```
-- 添加 flag 字段，默认为 1
ALTER TABLE curricular_variable
ADD COLUMN flag TINYINT DEFAULT 1;

-- 将已存储记录的 flag 随机设置为 1 或 2
UPDATE curricular_variable
SET flag = FLOOR(1 + (RAND() * 2));  -- 生成 1 或 2 的随机数

-- 触发器的创建
DELIMITER //

CREATE TRIGGER retest_records_trigger
BEFORE INSERT ON curricular_variable
FOR EACH ROW
BEGIN
    -- 检查是否存在该学生和课程的记录
    DECLARE existing_flag INT;

    -- 查询该学生和课程的现有记录的flag值
    SELECT flag INTO existing_flag
    FROM curricular_variable
    WHERE stuId = NEW.stuId AND courseId = NEW.courseId
    LIMIT 1;

    -- 如果记录存在且 flag 为 2，弹窗警告并阻止插入
    IF existing_flag = 2 THEN
        SIGNAL SQLSTATE '46000'
        SET MESSAGE_TEXT = '考试次数已达上限，不允许再次补考';
    ELSE
        -- 如果记录存在且 flag 不是 2，更新成绩并修改 flag 为 2
        IF existing_flag IS NOT NULL THEN
            UPDATE curricular_variable
            SET grade = NEW.grade, flag = 2
            WHERE stuId = NEW.stuId AND courseId = NEW.courseId;
        ELSE
            -- 如果记录不存在，插入新记录，默认 flag 为 1
            SET NEW.flag = 1;
        END IF;
    END IF;
END;
//
```

```
DELIMITER ;


-- 测试触发器
select * from curricular_variable;

-- flag为1 ，应成功
INSERT INTO curricular_variable (stuId, courseId, grade)
VALUES (1, 9, 85);
```

插入后出现如下错误：

```
Can't update table 'curricular_variable' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.
```

原因：

```
根本原因是mysql不允许插入新行后立即更新新行数据，只能分两次查询进行
```

解决方案：使用 `BEFORE INSERT` 触发器 + 存储过程

- **触发器** 仅用于检查 `flag=2` 的情况并阻止插入。
- **存储过程** 用于执行插入和更新逻辑，避免触发器的限制。避免触发器更新表自身导致的1442错误。

最后我们不用insert Into 语句插入数据，而是调用存储过程，

8. 删除UNIQUE

```
-- 删除外键约束
ALTER TABLE curricular_variable DROP FOREIGN KEY curricular_variable_ibfk_2;
-- 删除unique索引
ALTER TABLE curricular_variable DROP INDEX stuId;
-- 重新添加外键
ALTER TABLE curricular_variable
ADD CONSTRAINT curricular_variable_ibfk_2 FOREIGN KEY (stuId) REFERENCES
student(stuId);
```

六，总结

这次大作业我学习到了数据库设计的开发流程：

需求分析（数据/功能分析）——>项目调研（看别人怎么写的）——>概念设计——>逻辑结构设计——>建表——>添加mock数据——>增加功能——>添加规则（触发器）——>测试

关于建表：首先字段设计，以semester字段举例，我在设计时忽视了业务场景，只是在数据库层面做演示，所以就没有用enum字段，而选用int认为扩展性好，但是做到插入mock数据展示时发现可读性非常差，不得不修改字段，实际上是非常危险的事。

主键设计：对主键不理解·，想当然地将(planid，majorid,courseid)作为联合主键，仅仅是因为查询时需要一起查。但是完全可以依据planid查询。有好几张表也出现了类似的问题。最糟糕的是我是将ER图，逻辑结构画完之后，到正式建表时才发现的问题，不得不对着sql建表文件修改图，非常麻烦。

我认为出现这样的问题主要是由于数据库设计规范中的范式检查没有好好分析，直接略过，冗余字段的保留没有考虑好。因为在后期查询学生的major信息我不得不关联两个表才能确定它的专业号，试验结束后我认为这个冗余字段替换多表联查的时间和资源开销是非常值得的。

当然实验也有有趣的地方，我首先学习了几种自动生成mock数据的网站Mockaroo - Random Data Generator and API Mocking Tool | JSON / CSV / SQL / Excel，EXCEL自动转换为sql脚本的网站Convert Excel to Insert SQL Online - Table Convert Online，都是非常好用简洁的工具。但是针对具体业务还不是很灵活。关于数据源的选取我学习用八爪鱼从学校的"全校课程表"上爬虫，暂时知道这个软件的基础使用。然后就是清洗数据，也锻炼了我excel处理能力。当然最困难也最难理解的是针对设计的学籍管理系统编写存储函数，这是整个实验中最困难的部分，在实验中我想好思路然后利用chatGPT工具编写，再学习其中不懂的地方，比如游标使用，双层循环，生成随机数，我也学习到了很多。

关于是否允许再次补考，我设置次数字段1，2，再编写触发器插入时检查flag是否等于2，设置标志位让我联想到cache缓存机制中用于写策略的标志位，但是它在这个实验中并没有什么指导作用，因为我们不需要将补考数据做缓存，但是这个设计思想在非关系型数据库redis中的数据一致性是有所体现的，非常遗憾数据库课程是不讲这个的。