数据库第四次课程大作业

姓名：* 学号：***

一、 分析与说明

当今社会信息化程度越来越高，大量个人信息和机构商业数据都被存储在数据库中，这些信息的泄露将给用户和机构带来极大的损失。因此，在数据库系统应用为公众提供信息服务过程中，对于运营者和维护管理人员实施严格的法规与职业道德约束显得尤为重要——

1.1 要求运营者和维护管理人员实施严格法规与职业道德约束原因

要求运营者和维护管理人员实施严格法规与职业道德约束原因主要是为了保障数据的安全和隐私。在当今信息化时代，数据已经成为一种极其重要的资源，它包含了个人或组织的各种敏感信息，一旦数据被不法分子窃取滥用，将会造成巨大损失，甚至是不可弥补的灾难。

因此运营者和维护管理人员需要实施严格法规与职业道德约束，具体可能有如下几点的考虑：

1 避免滥用和泄露用户个人隐私信息

作为提供信息服务的运营者和维护管理人员，其手上掌握着大量的用户信息，如不加以约束和监管，可能会导致用户的隐私泄露和信息安全问题，进而引发一系列的法律纠纷，给用户和社会造成不良影响。因此，对于这些敏感信息的获取、存储、使用和发布，必须要遵守相关的法律法规和道德规范，严格加以限制和约束。

2 保护机构商业机密

在数据库中存储了许多机构的商业机密信息，如客户信息、销售数据等，如果没有严格的保护措施，这些信息很有可能会被运营者和维护管理人员泄露出去，这会给公司带来重大损失，无论是信誉上的还是信息价值上的。

3 维护公平竞争环境

运营者和维护管理人员还需要维护公平竞争，避免滥用数据或打压竞争对手。相关从业人员需要保持公正、透明的态度，避免任何不当行为，让市场竞争的环境更加健康。

4 提高社会信任和企业形象

对于运营者和维护管理人员来说，接受严格的法规和职业道德约束，有助于保护用户隐私和信息安全，提高社会对其的信任度和企业形象。

只有在公众对其产生信任和认可之后，才能够更好地为用户提供质量可靠的

信息服务，并获得更好的商业回报。因此，对于运营者和维护管理人员来说，遵守相关法规和职业道德规范，是提升自身价值和社会声誉的必经之路。

1.2 从技术层面保护用户个人隐私和机构商业机密的方法

在数据库系统应用为公众提供信息服务的过程中，为了保护用户个人隐私和机构商业机密，可以从技术层面上采取以下几点措施：

1 数据加密

数据加密可以保证存储在数据库中的信息只有经过授权才能访问，即使数据被盗取也无法查看其中的敏感内容。可以使用一些流行的加密算法，如 AES、RSA 等。

一般使用的加密方法有对称加密和非对称加密。对称加密是指加密和解密使用同一个密钥，而非对称加密是指加密和解密使用不同的密钥。通常情况下，对称加密速度较快，适合加密大量数据；而非对称加密虽然速度慢，但更安全可靠。

2 采用访问控制策略

访问控制是数据库系统保护信息安全的重要手段之一，通过对用户和管理员的身份验证和权限的授予，从根本上控制数据库的访问范围。在访问敏感数据之前，系统需要先对请求者进行身份验证，并检查其权限是否足够。

访问控制的策略有很多种，如强制访问控制、自主访问控制、可选访问控制等。此外，数据库管理员还可以采用角色和权限分离的方式，将不同的用户分配到不同的角色中，然后为不同的角色设置不同的访问权限。

3 数据脱敏

数据脱敏是指通过对敏感数据进行处理，去除其中的个人身份信息和商业机密信息，将原始数据中的个人身份信息替换为随机的编码，从而保护数据的隐私性和安全性。在不得不共享敏感数据的情况下，可以减少直接暴露其中的个人身份信息的风险。

4 采用安全协议传输

安全协议是指在数据传输过程中使用一些特定的技术手段来加密传输数据、身份认证等，从而保护数据传输的安全性。例如 SSL/TLS 等协议可以在网络上实现安全传输。

5 实现安全审计与日志记录

安全审计是指对访问数据库的用户和管理员进行日志记录和监视，以便在需要时对其行为进行审计。通过对记录的日志数据进行分析和比对，可以及时发现不当行为或者异常行为，从而更好的保护用户个人隐私和机构商业机密。

1.3 总结

总之，数据库系统应用为公众提供信息服务时，必须要从技术层面上加强对用户个人隐私和机构商业机密的保护，采取科学有效的技术措施，确保数据得到充分保护并符合相关的法规和规范。同时，也需要对运营者和维护管理人员进行定期培训，加强其法规与职业道德意识，提高其意识并遵从相关规定和标准。

二、 实践操作题

2.1 建表与数据插入

创建数据库以及创建表的语句为——

```
--创建数据库
CREATE DATABASE GradeDB;
-- 创建 STUDENT 表
CREATE TABLE STUDENT(
    SID     varchar(10)    PRIMARY KEY,
    SName   varchar(10)    NOT NULL,
    Age     int            NOT NULL,
    Sex     char(2)        NOT NULL CHECK (Sex in ('男', '女'))
);
-- 创建 COURSE 表
CREATE TABLE COURSE(
    CID     varchar(10)    PRIMARY KEY,
    CName   varchar(50)    NOT NULL,
    Teacher varchar(20)    NOT NULL
);
-- 创建 GRADE 表
CREATE TABLE GRADE(
    SID     varchar(10)    NOT NULL,
    CID     varchar(10)    NOT NULL,
    Score   int            NOT NULL,
    Note    varchar(50)    NOT NULL,
    CONSTRAINT pk_grade PRIMARY KEY (SID, CID),
    CONSTRAINT SID_FK FOREIGN KEY (SID) REFERENCES STUDENT(SID)
        ON DELETE CASCADE,
    CONSTRAINT CID_FK FOREIGN KEY (CID) REFERENCES COURSE(CID)
        ON DELETE CASCADE
);
```

相关截图如下图，可以看到左侧已经成功创建了三个表，返回成功创建。

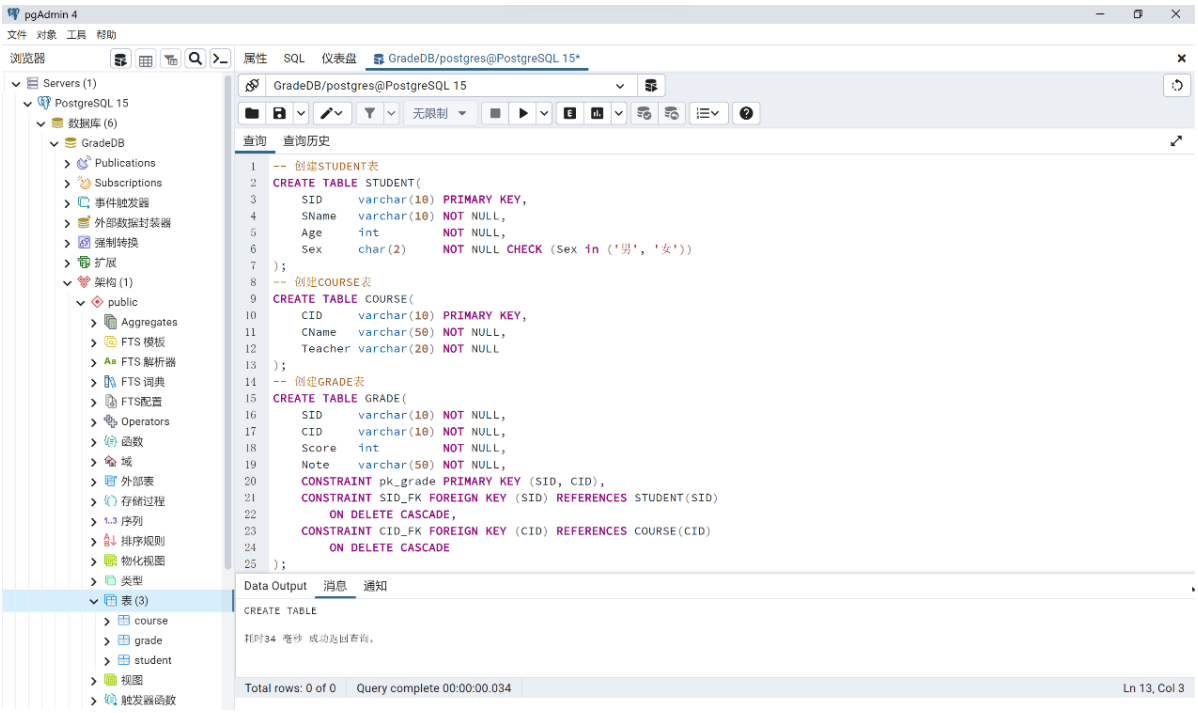

图 1：创建数据库以及表

接下来，插入 20 个学生的信息，插入语句为：

```
-- 插入 20 个学生数据
INSERT INTO STUDENT VALUES
    ('1001', '张三', 18, '男'),
    ('1002', '李四', 19, '女'),
    ('1003', '王五', 20, '男'),
    ('1004', '赵六', 18, '女'),
    ('1005', '钱七', 19, '男'),
    ('1006', '孙八', 20, '女'),
    ('1007', '周九', 18, '男'),
    ('1008', '吴十', 19, '女'),
    ('1009', '郑一', 20, '男'),
    ('1010', '王二', 18, '女'),
    ('1011', '林三', 19, '男'),
    ('1012', '陈四', 20, '女'),
    ('1013', '刘五', 18, '男'),
    ('1014', '黄六', 19, '女'),
    ('1015', '张七', 20, '男'),
    ('1016', '王八', 18, '女'),
    ('1017', '李九', 19, '男'),
    ('1018', '赵十', 20, '女'),
    ('1019', '钱一', 18, '男'),
    ('1020', '孙二', 19, '女');
-- 插入 2 门课程数据
INSERT INTO COURSE VALUES
    ('C001','数据库原理及应用','陆老师'),
    ('C002','数据结构与算法','白老师');
-- 插入 20 个学生的 2 门课程成绩数据
INSERT INTO GRADE VALUES
    ('1001','C001',85,'优秀'),
    ('1001','C002',78,'良好'),
```

```
('1002','C001',90,'优秀'),
('1002','C002',76,'良好'),
('1003','C001',80,'良好'),
('1003','C002',80,'良好'),
('1004','C001',85,'优秀'),
('1004','C002',90,'优秀'),
('1005','C001',58,'不及格'),
('1005','C002',85,'优秀'),
('1006','C001',59,'不及格'),
('1006','C002',76,'良好'),
('1007','C001',90,'优秀'),
('1007','C002',55,'不及格'),
('1008','C001',85,'优秀'),
('1008','C002',78,'良好'),
('1009','C001',90,'优秀'),
('1009','C002',85,'优秀'),
('1010','C001',78,'良好'),
('1010','C002',80,'良好'),
('1011','C001',82,'良好'),
('1011','C002',46,'不及格'),
('1012','C001',56,'不及格'),
('1012','C002',85,'优秀'),
('1013','C001',68,'及格'),
('1013','C002',72,'及格'),
('1014','C001',57,'不及格'),
('1014','C002',56,'不及格'),
('1015','C001',90,'优秀'),
('1015','C002',85,'优秀'),
('1016','C001',86,'优秀'),
('1016','C002',92,'优秀'),
('1017','C001',80,'良好'),
('1017','C002',52,'不及格'),
('1018','C001',78,'良好'),
('1018','C002',50,'不及格'),
('1019','C001',60,'及格'),
('1019','C002',55,'不及格'),
('1020','C001',85,'优秀'),
('1020','C002',80,'良好');
```

相关截图如下图，可以看到左侧已经成功插入了数据，返回成功插入。

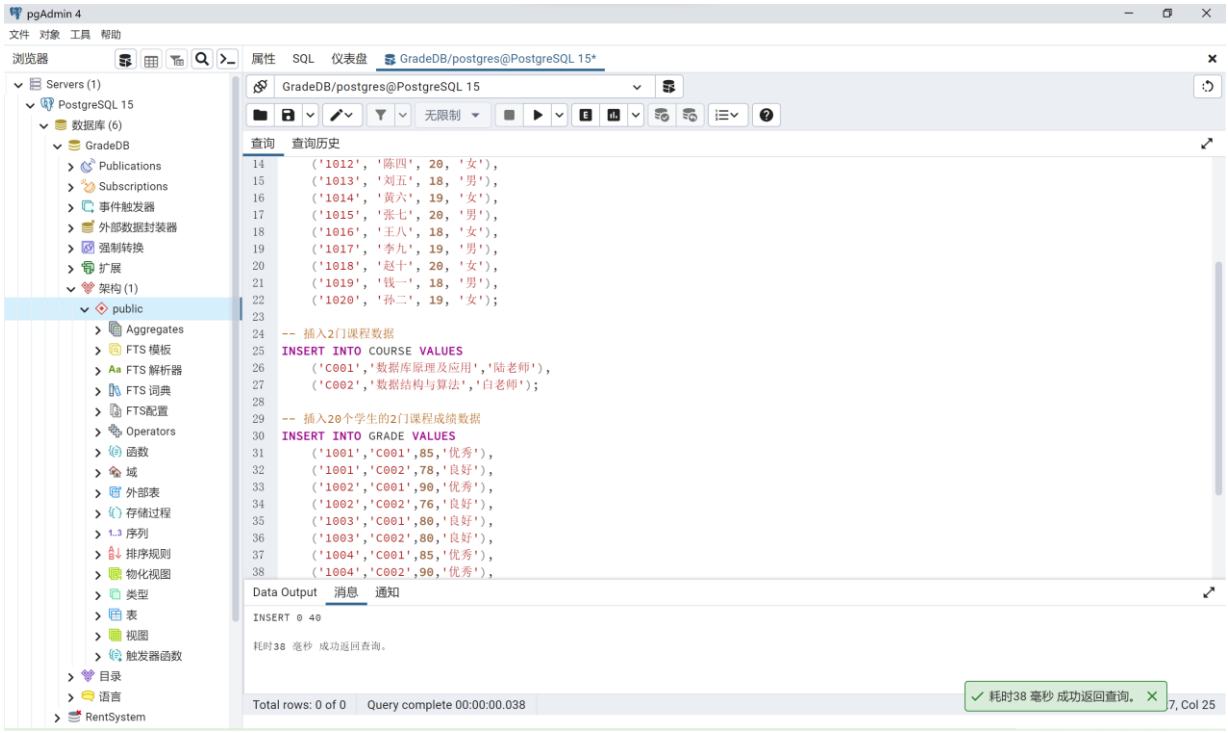

图 2：插入数据

2.2 触发器程序实现日志记录

首先我们建立 GradeLog 表，建表语句为：

```
CREATE TABLE GradeLOG (
    logID      serial     PRIMARY KEY,
    userID     varchar(20) NOT NULL,
    SID        varchar(20) NOT NULL,
    CID        varchar(20) NOT NULL,
    updateTime timestamp  NOT NULL,
    oldScore   int        DEFAULT NULL,
    newScore   int        DEFAULT NULL
);
```

相关截图见下一页：

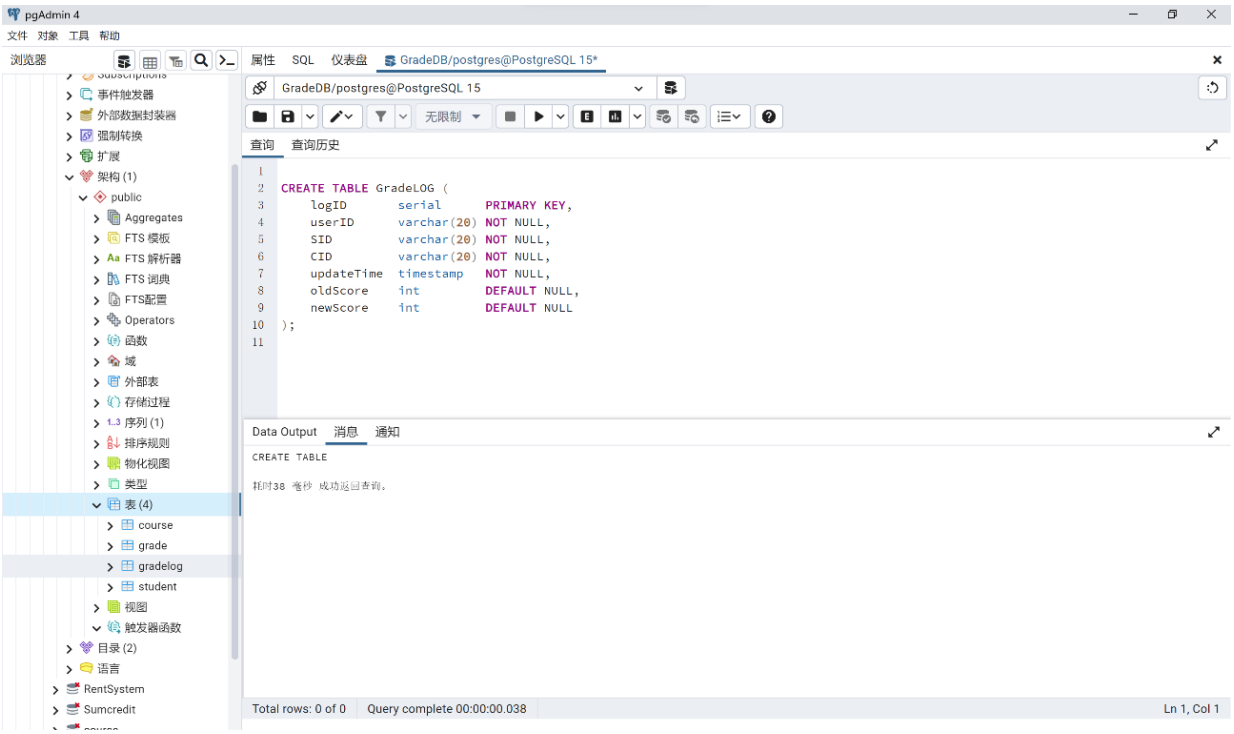

图 3：建立 GradeLog 表

随后，进程触发器函数编程，编写函数 log_grade_changes()

```
    CREATE OR REPLACE FUNCTION log_grade_changes()
    RETURNS trigger AS $log_grade_changes$
    BEGIN
      -- 判断是 insert 还是 delete 还是 update 操作
      IF (TG_OP = 'INSERT') THEN
        -- 记录插入操作前的成绩
        INSERT INTO GradeLOG (userID, SID, CID, updateTime, oldScore,
newScore)
         values (USER, NEW.SID, NEW.CID, CURRENT_TIMESTAMP, NULL,
NEW.Score);

      ELSIF (TG_OP = 'DELETE') THEN
        -- 记录删除操作前的成绩
        INSERT INTO GradeLOG (userID, SID, CID, updateTime, oldScore,
newScore)
         values (USER, OLD.SID, OLD.CID, CURRENT_TIMESTAMP, OLD.Score,
NULL);

      ELSE
        -- 记录更新操作前和更新操作后的成绩
        INSERT INTO GradeLOG (userID, SID, CID, updateTime, oldScore,
newScore)
         values (USER, OLD.SID, OLD.CID, CURRENT_TIMESTAMP, OLD.Score,
```

```
NEW.Score);
      END IF;

      RETURN NEW;
    END;
    $log_grade_changes$ LANGUAGE plpgsql;
```

在触发器函数中，一旦触发触发器，首先判断触发的事件（insert/update/delete）即会调用对应的 INSERT 语句，插入相关信息至 GradeLOG 中。

其中 USER 是 PostgreSQL 的内置变量，用以存储操作的用户信息，CURRENT_TIMESTAMP 会自动获取当前状态的时间戳，不需要用户自行输入时间戳值，从而避免了因为时区、格式等问题导致的错误。

创建的截图如下，可以看到，创建触发器函数成功。

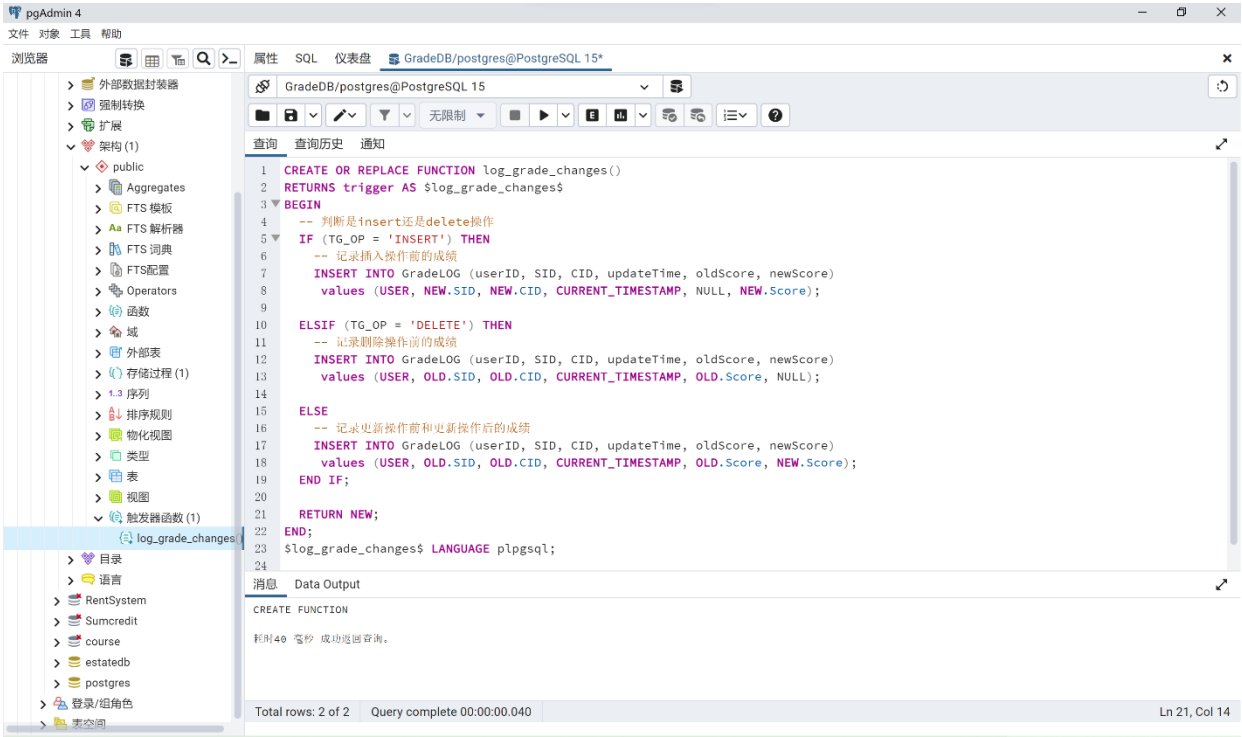

图 4：创建触发器函数

随后创建触发器

```
    -- 创建触发器：在更新 GRADE 表中的记录时，自动在 GradeLOG 表中插入日志
    CREATE TRIGGER grade_change_trigger
    AFTER INSERT OR UPDATE OR DELETE ON GRADE
    FOR EACH ROW
  EXECUTE PROCEDURE log_grade_changes();
```

相关截图如下：

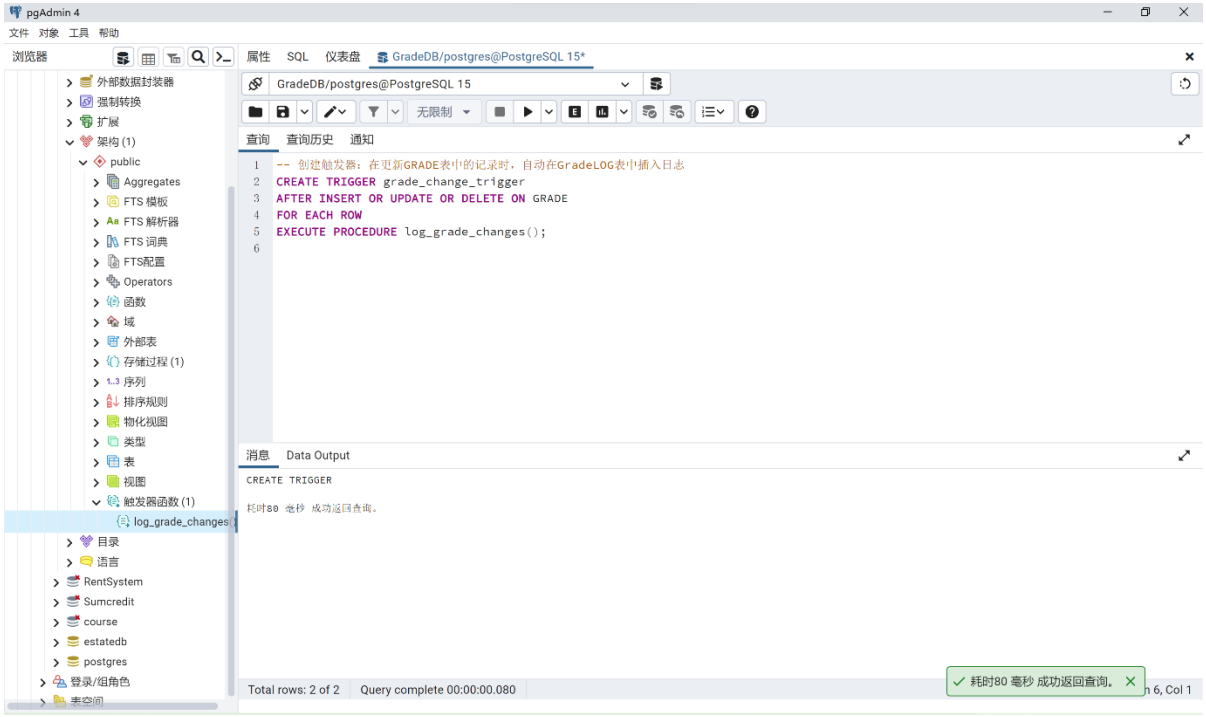

图 5：触发器创建

触发器实现了在更新后自动调用触发器函数 log_grade_changes，以插入数据。该触发器是一个行级触发器。

要验证这个触发器函数是否有效，我们可以使用以下语句在 GRADE 表中进行如下修改操作：

```
UPDATE GRADE SET Score = 100 WHERE SID='1001' AND CID='C001';
DELETE FROM GRADE WHERE SID = '1002' AND CID = 'C001';
INSERT INTO GRADE (SID, CID, Score, Note) VALUES ('1002', 'C001', 90, '优秀');
```

以上分别对成绩进行修改、删除和新增。

此时，该触发器函数将被激活，并向 GradeLOG 表中插入 3 条新的记录，记录了修改前和修改后的成绩等信息。

为了验证日志表是否成功记录了这个改变，您可以使用如下 SQL 查询语句来检查：

```
SELECT * FROM GradeLOG;
```

若触发器函数编写正确，则应该能够得到包含上述修改日志信息的查询结果。实验验证的结果截图如下：

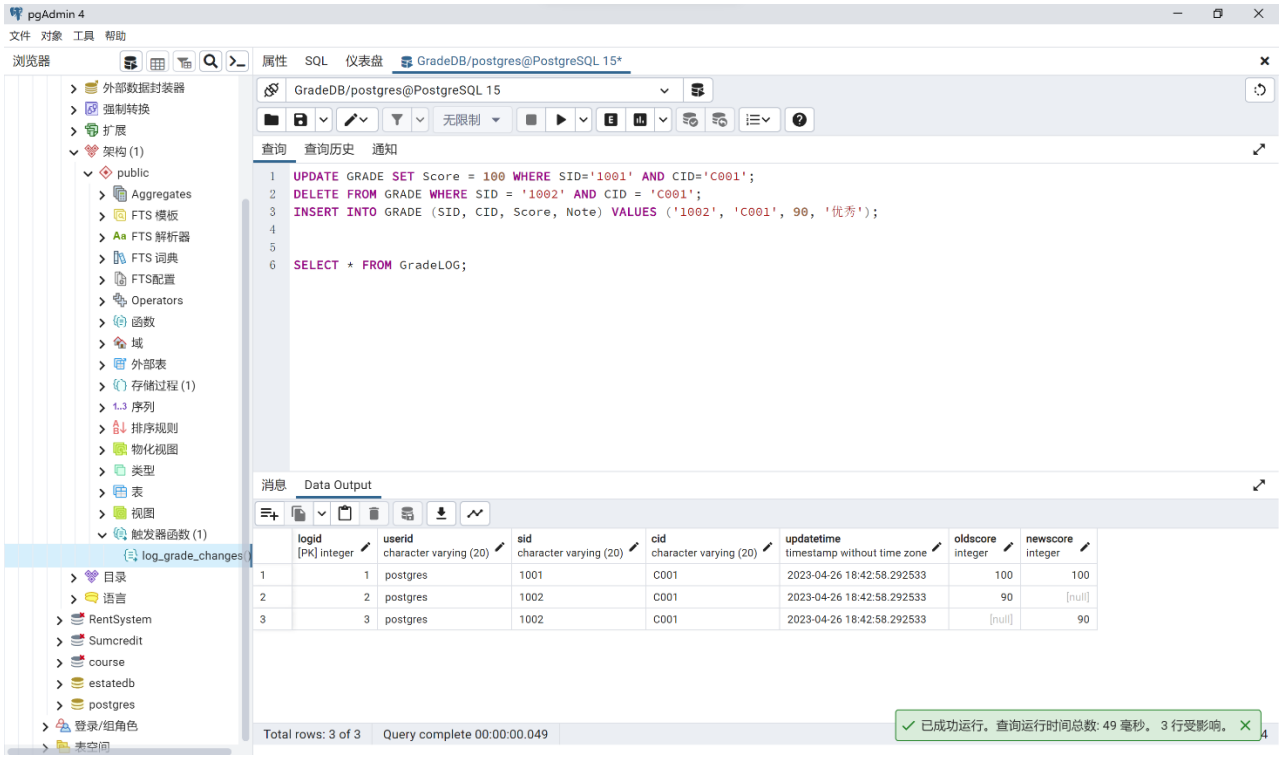

图 6：结果验证

可以看到，在 UPDATE 之后，GradeLOG 自动记录了一条修改日志信息，成绩从 85 分修改为 100 分。

在 DELETE 之后，GradeLOG 自动记录了第二条修改日志信息，原始成绩记录为 90，新成绩为 NULL，符合预期。

在 INSERT 之后，GradeLOG 自动记录了第三条修改日志信息，原始成绩记录为 NULL，新插入成绩为 90，符合预期。

其余列均记录正确，表明触发器编程正确，结果无误。

2.3 存储过程程序实现统计各课程不及格学生人数

编写存储过程 count_failed_students()

其中使用游标通过一个循环遍历所有的课程信息，当查询到某课程对应的成绩信息小于 60 分时，studentcount 进行自增，以实现统计各个课程不及格学生人数的功能和作用。

源代码见如下：

```
CREATE OR REPLACE PROCEDURE count_failed_students()
AS $$
DECLARE
  courses CURSOR FOR SELECT CID, CName FROM COURSE;
  courseid CHAR(4);
```

```
    coursename VARCHAR(10);
    studentcount INT;
  BEGIN
    OPEN courses;
    LOOP
      FETCH courses INTO courseid, coursename;
      EXIT WHEN NOT FOUND;
      studentcount := 0;
      SELECT COUNT(*) INTO studentcount FROM GRADE WHERE CID = courseid
AND Score < 60;
      RAISE NOTICE '课程序号：% ，课程名称：% ，挂科人数：% ', courseid,
coursename, studentcount;
    END LOOP;
    CLOSE courses;
  END;
  $$ LANGUAGE plpgsql;
```

从下图可知，存储过程无错误，创建成功

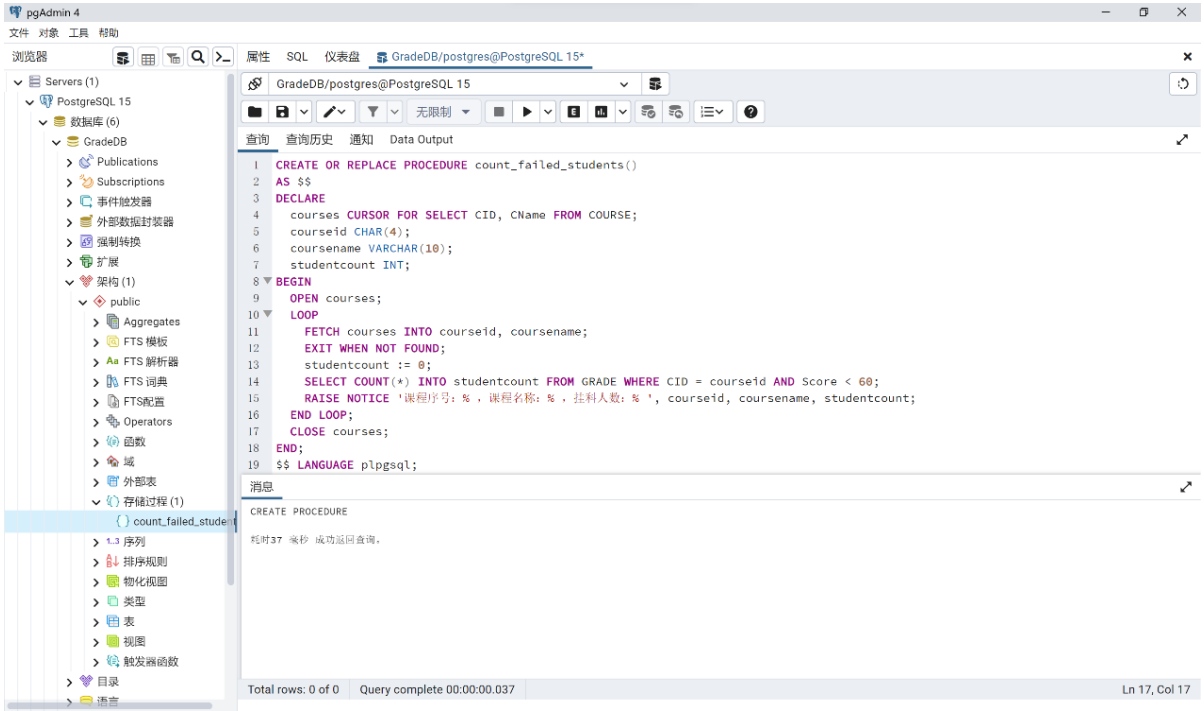

图 7：创建存储过程

我们使用如下的 SQL 语句对存储过程进行调用：

```
  call count_failed_students();
```

调用结果可参考下一页截图。

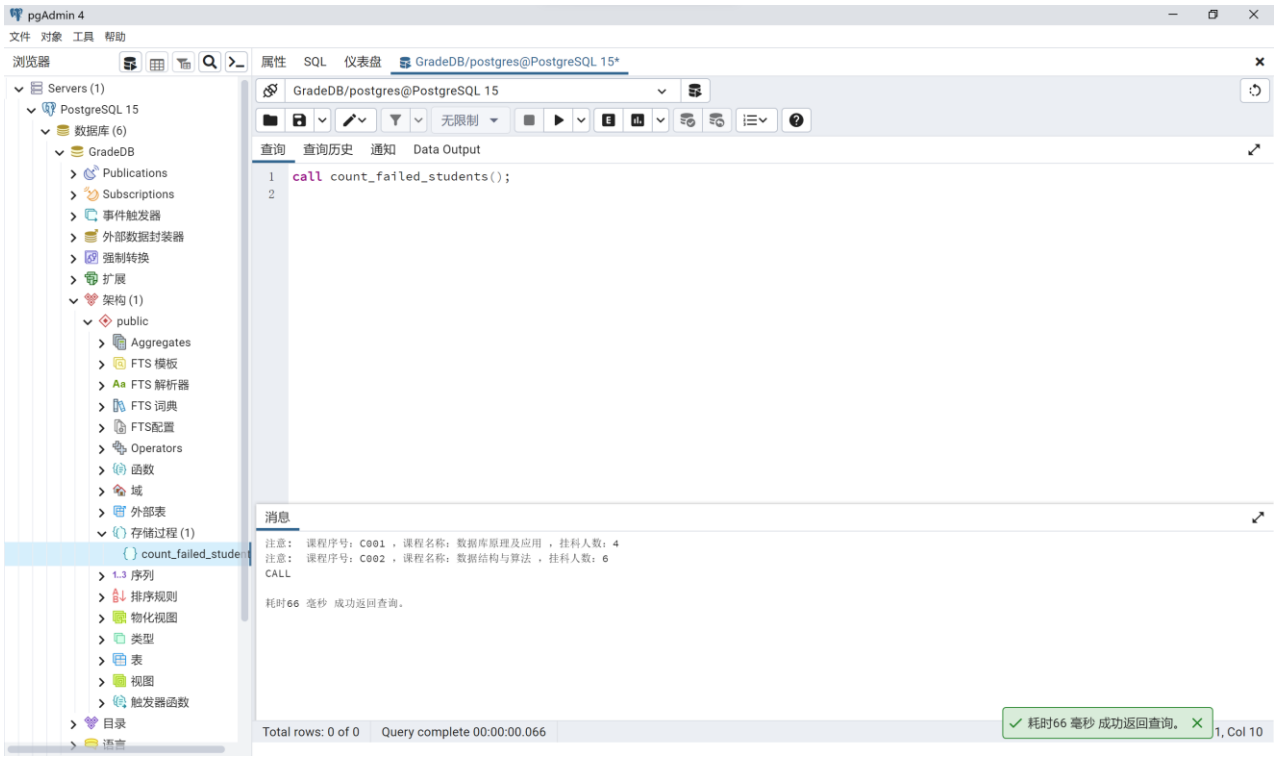

图 8：调用存储过程

我们看到程序输出——课程序号为 C001 的数据库课程挂科 4 人次，课程序号为 C002 的数据结构课程挂科 6 人次，与我们插入的 insert 语句相符合，预期结果一致，表明存储过程编写完全正确。

三、 挑战性问题研讨

3.1 技术方案概要

在挑战性问题解决时，我们主要用到了如下几个技术

①后端层面：

- SpringWeb(SpringMVC):一个基于 mvc 的 web 框架,方便前后端数据的传输。
- Mybatis：一款优秀的持久层框架，支持定制化 SQL、存储过程以及高级映射，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。
- Lombok：是一个在 Java 开发过程中用注解的方式，简化了 JavaBean 的编写，避免了冗余和样板式代码而出现的插件，让编写的类更加简洁。

②前端层面：

- Vue：构建用户界面的 JavaScript 框架。它基于标准 HTML、CSS 和

JavaScript 构建，并提供了一套声明式的、组件化的编程模型，帮助开发者高效地开发用户界面。

- ElementUI：一套为开发者准备的基于 Vue 2.0 的 web 端组件库
- Axios：一个基于 promise 的网络请求库，常用于前端获取网络请求

3.2 后端系统实现——基于 Spring+Mybatis 框架

后端系统主要用于对前端发送的网络请求进行业务处理。使用 IDEA 进行开发。首先在 IDEA 中创建新项目，命名为“GradeSearch”，使用 Spring Initializer 进行初始化。

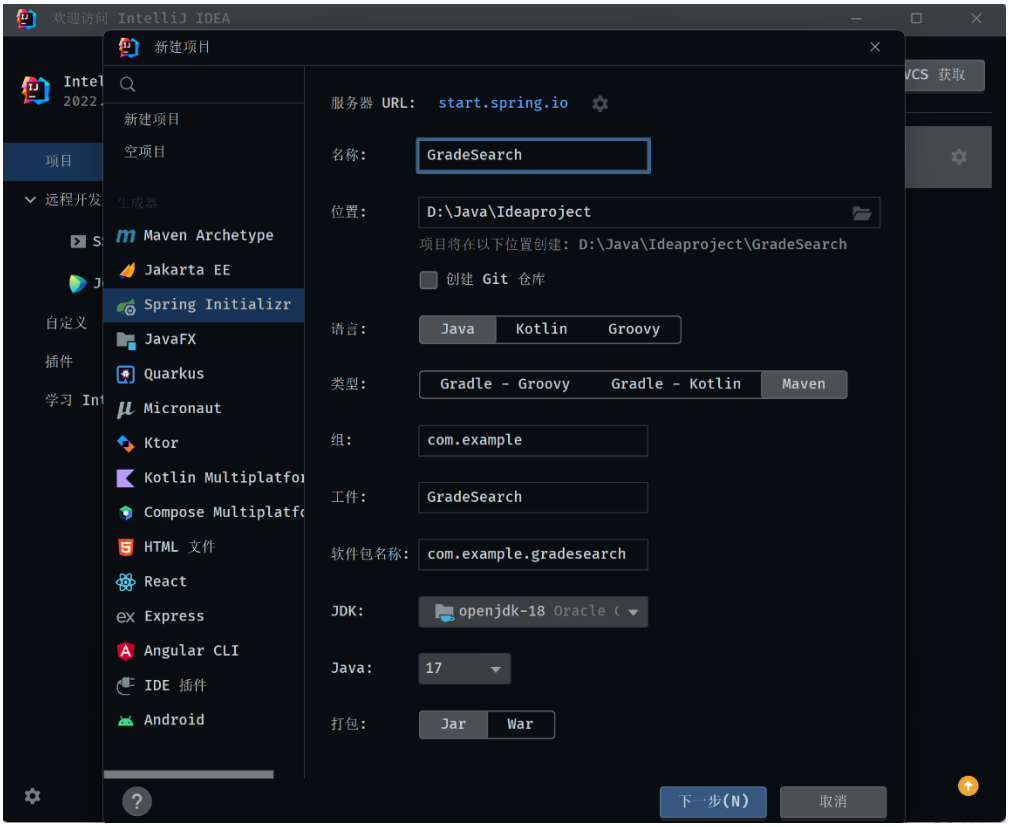

图 9：创建新项目

随后根据项目技术方案选入本项目可能需要的依赖，其中主要的为以下几项：

- Spring Web
- Lombok
- PostgreSQL Driver
- Mybatis Framework
- JDBC API

相关截图见下一页。

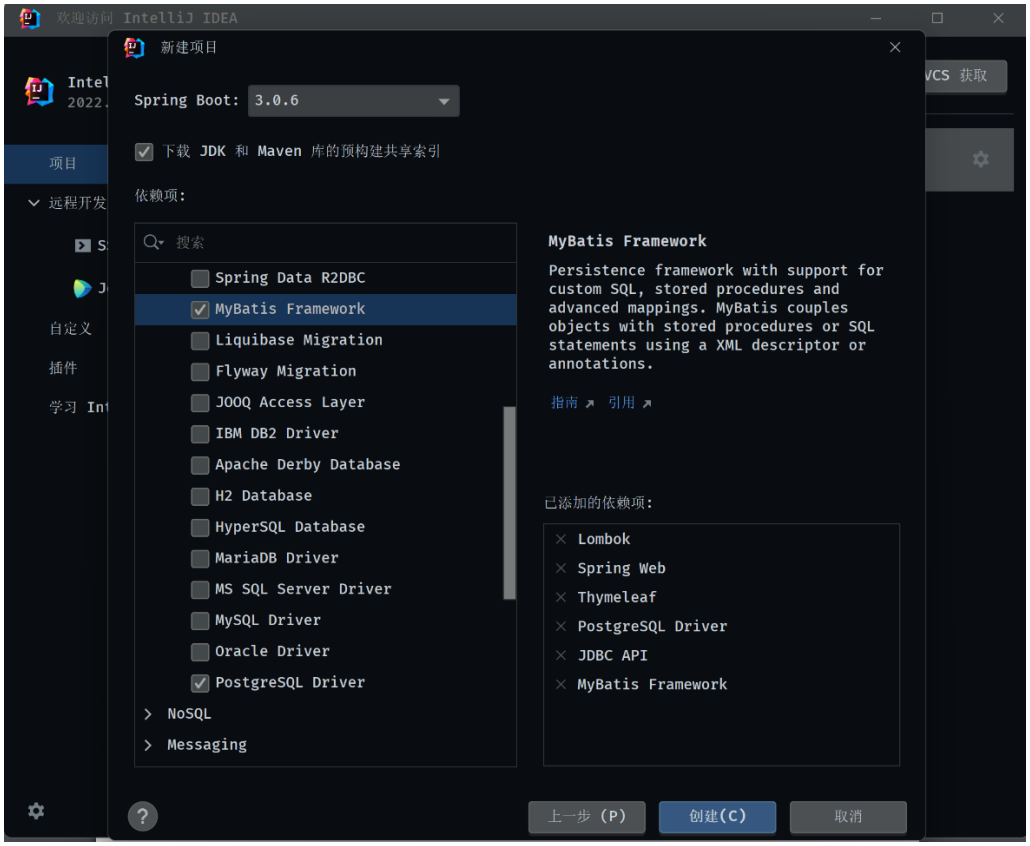

图 10：导入依赖

接下来，找到 application.yml 文件，进行数据库连接以及服务器创建的基础配置，包括服务器端口、数据库账户和密码等。

相关配置项见下图。

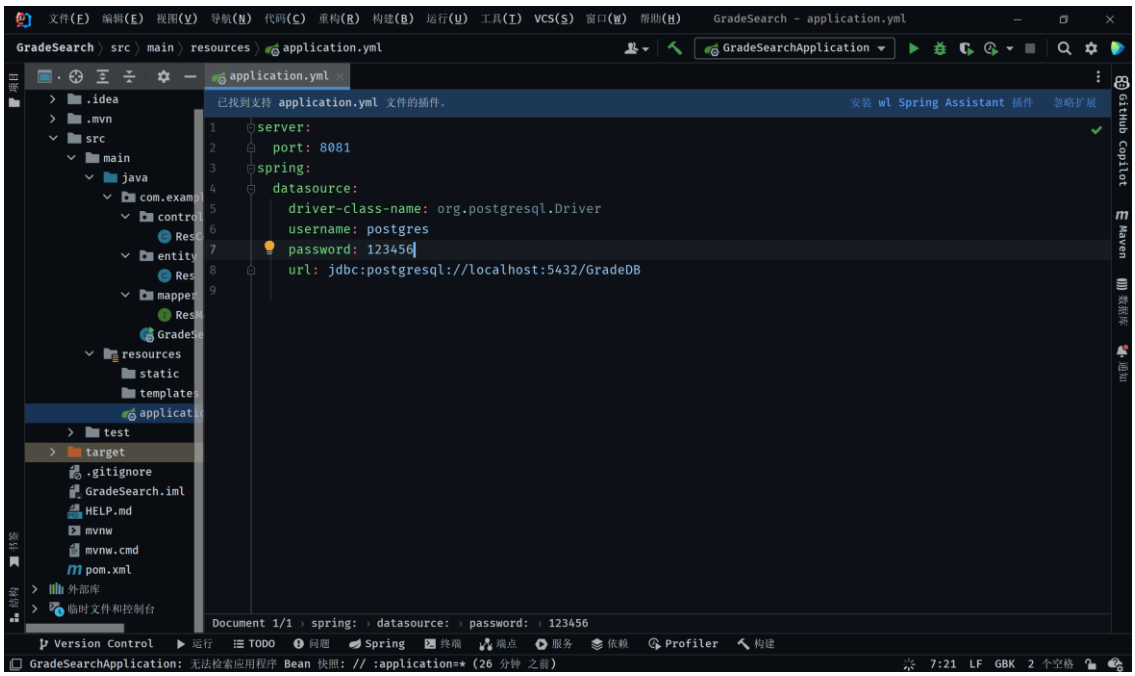

图 11：配置基础信息

配置完毕后，即可进行业务逻辑的编写。

首先，定义实体（entity）类 Res，用以保存查询的结果集合。每个结果元素均为一个 Res 对象，包含学号、姓名、成绩、课程的信息。其类型需要和数据库定义的信息一致。

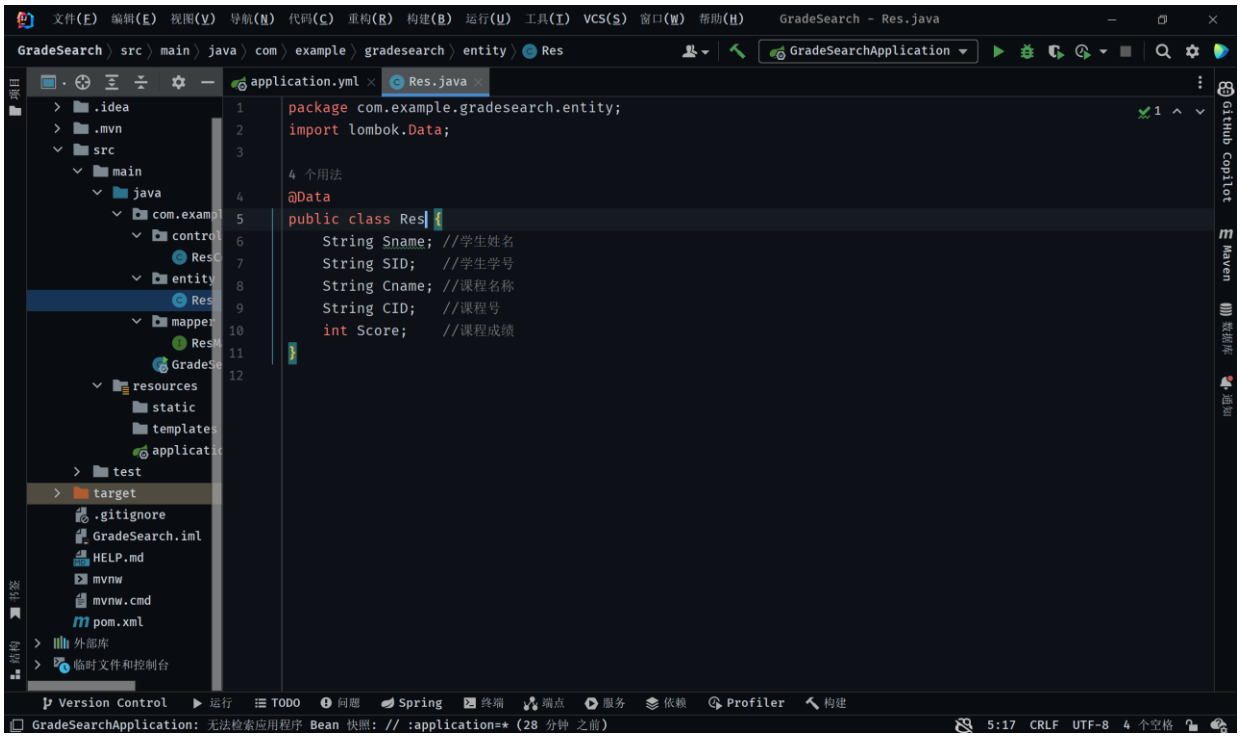

图 12：定义实体类 Res

随后，定义 Mapper 类 ResMapper

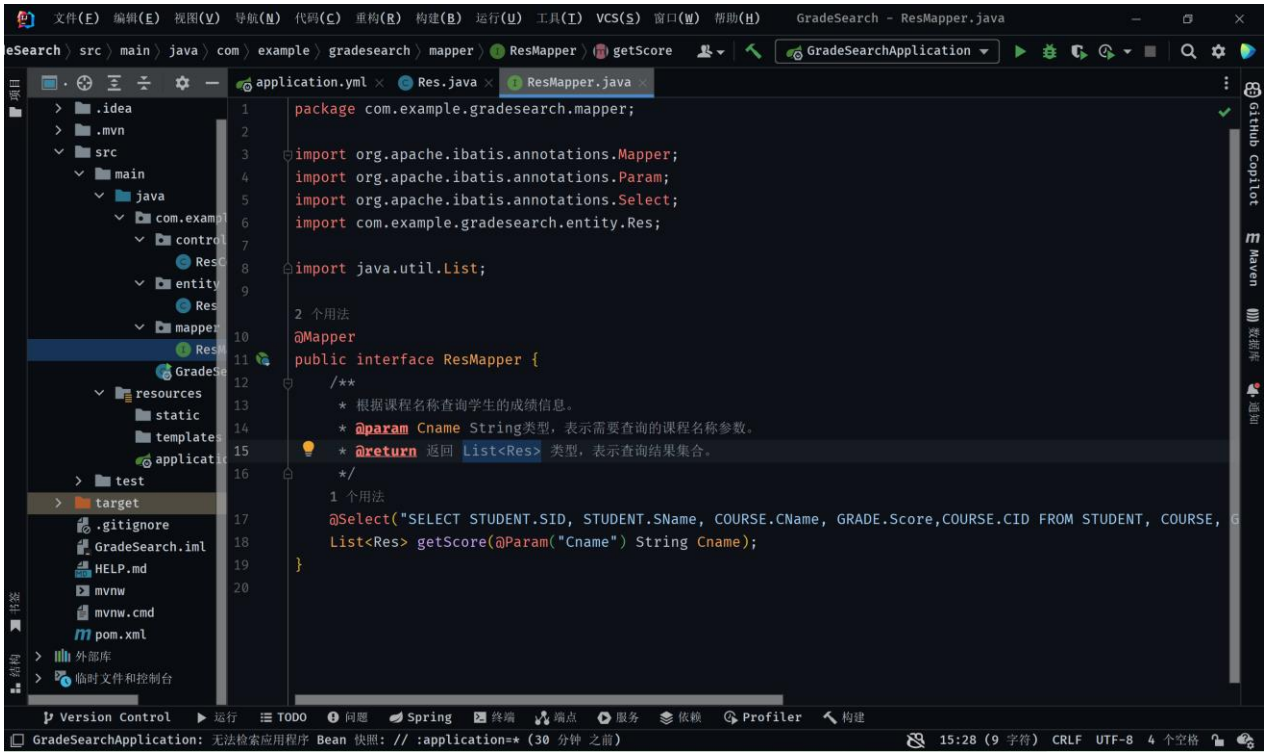

图 13：定义 ResMapper

ResMapper 是 Spring Boot 应用程序与数据库之间的数据交互层，负责封装了数据库的操作。在上述代码中，我们定义了一个 ResMapper 接口，并使用 MyBatis 提供的注解方式声明了一个名为 getScore 的查询方法。该方法接收一个参数 Cname，用于指定要查询的课程名称。通过 SQL 语句从数据库中获取成绩信息，并返回 List<Res> 对象。

其中，对数据库查询的语句为

```
SELECT STUDENT.SID, STUDENT.SName, COURSE.CName, GRADE.Score,COURSE.CID
  FROM STUDENT, COURSE, GRADE
  WHERE GRADE.CID = COURSE.CID
     AND GRADE.SID = STUDENT.SID
     AND COURSE.CName = #{Cname}
```

最后，在控制层定义 Rescontroller 类，主要负责 Web 请求的处理

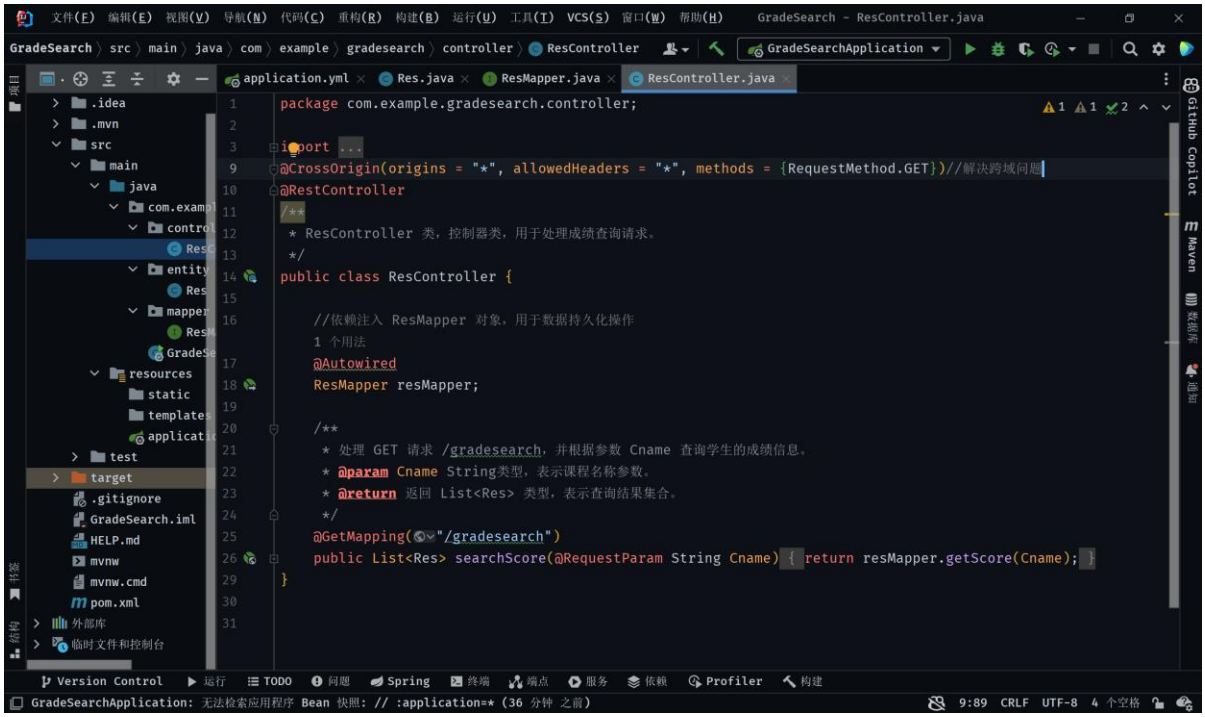

图 14：定义 ResController 类

这段代码定义了一个 ResController 类，使用 @RestController 注解将其标记为 RESTful Web 服务的控制器。该类中定义了一个方法 searchScore()，使用 @GetMapping 注解将其映射到 /gradesearch 路径上，以便能够响应 HTTP GET 请求。同时，该方法接收一个名为 Cname 的请求参数，并通过调用 ResMapper 接口中的 getScore() 方法从数据库中查询课程成绩信息并进行返回。

至此，我们完成了实体层、Mapper 层、控制层三个主要层级的搭建过程，后端基本成型。点击右上角构建按钮运行后端。

运行后截图如下页所示，无报错，项目成功运行：

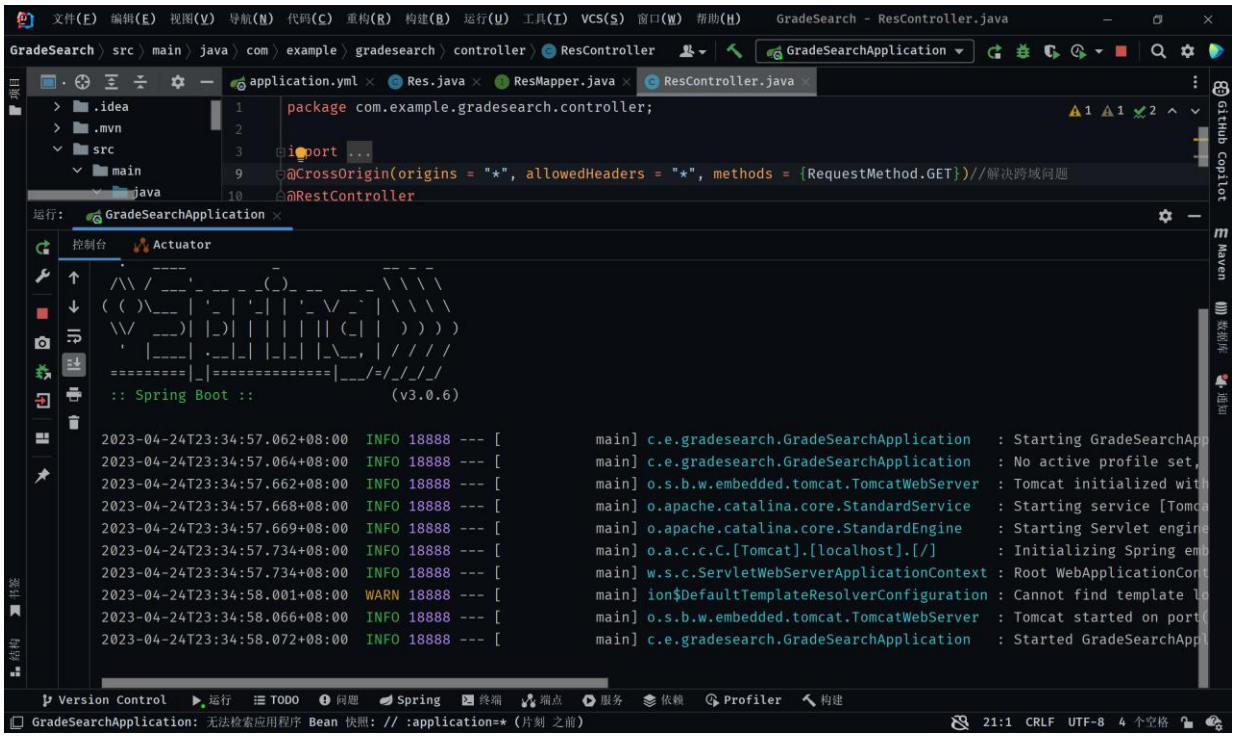

图 15：运行后端

使用 Apipost 内置的 Postman 对后端进行测试，对其发送 GET 请求：

```
GET http://192.168.137.1:8081/gradesearch?Cname=数据库原理及应用
```

运行结果如下图：

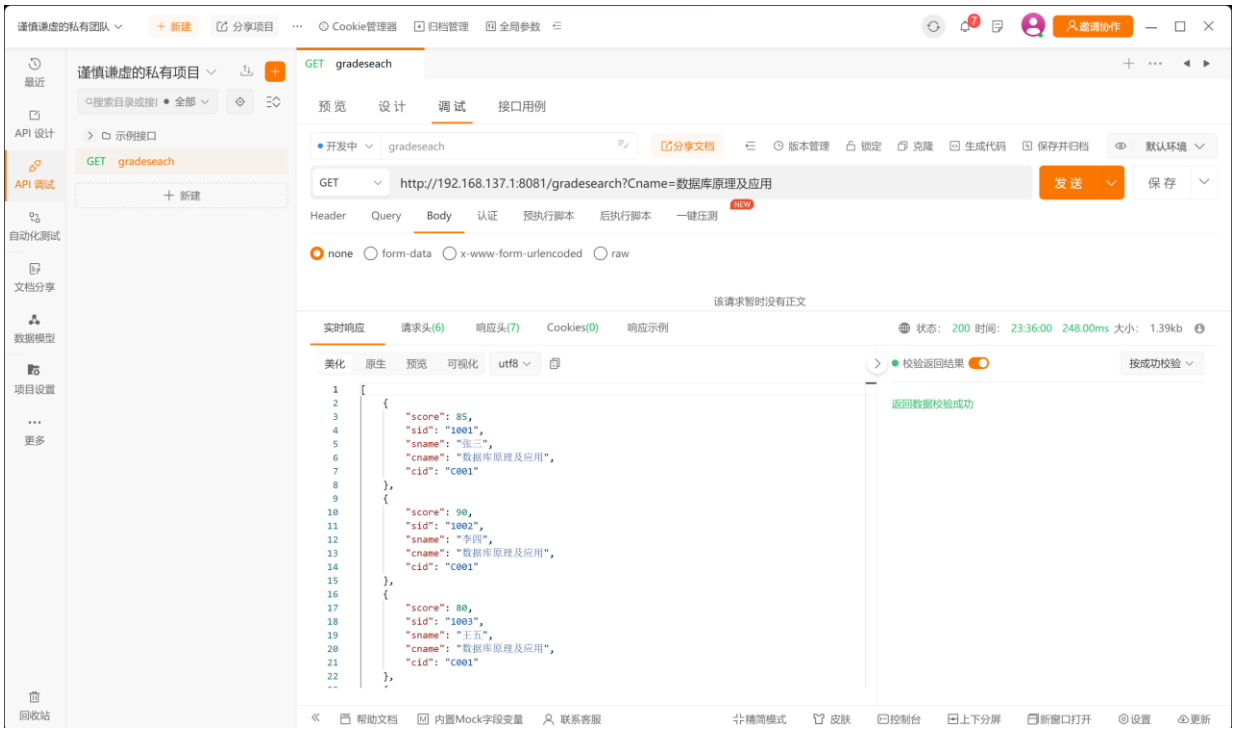

图 16：使用 Apipost 验证后端业务逻辑

其与我们预期的相符合，成功返回了 json 对象，说明后端构建成功。

3.3　前端系统实现——基于 Vue+ElementUI 框架

首先引入 Vue、axios 和 ElementUI 库，编写如下语句

```
<!-- 引入 vue -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<!-- 引入 axio -->
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<!-- 引入 ElementUI -->
<script src="https://unpkg.com/element-ui/lib/index.js"></script>
```

随后，使用 HTML 编写整体框架，其中使用了部分 ElementUI 内置的组件库，以美化网页。

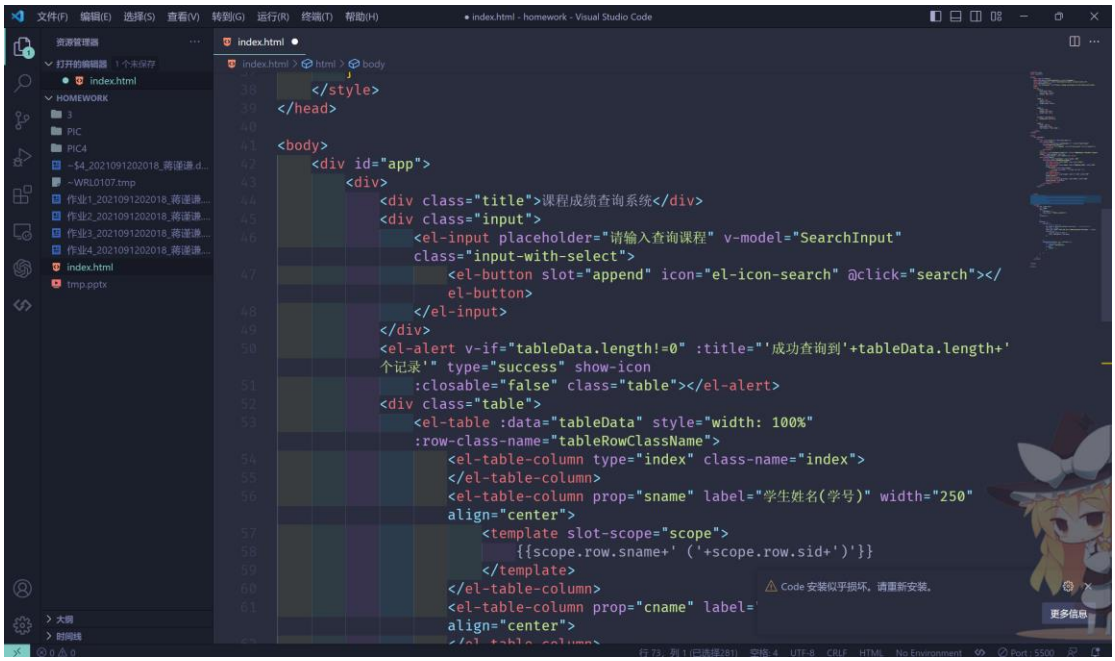

图 17：编写前端模板框架

在上述代码中，我们使用了三个主要 div 来布局，分别装入标题、搜索框以及查询结果表。查询结果表使用 ElementUI 内置的表格组件<el-table>进行组构。

随后编写 js 代码以实现网络请求。通过使用 axios 内置的 get 方法，向网络端发送数据查询请求，其中填写的课程名通过动态绑定(v-model)绑定到 data 对象中的 SearchInput 中，在请求的头部中进行携带

请求返回的结果保存到 tableData 中准备进行表格渲染和数据处理。

相关方法见下页：

```
<script>
    var app = new Vue({
        el: '#app',
        data: {
            tableData: [],
            SearchInput: '数据库原理及应用',
        },
        created() {


        },
        methods: {
            search() {
                let input = this.SearchInput.toString();  //获取输入框的值
                let this_ = this;
                axios.get('http://192.168.137.1:8081/gradesearch?Cname=' + input).
                then(res => {
                    console.info(res.data);
                    this_.tableData = res.data;
                })


            },
```

图 18：前端 js 代码

为了使得界面更加美观，我们设计了一些 CSS 代码进行布局和美化，使项目的输入框、表格居中显示。

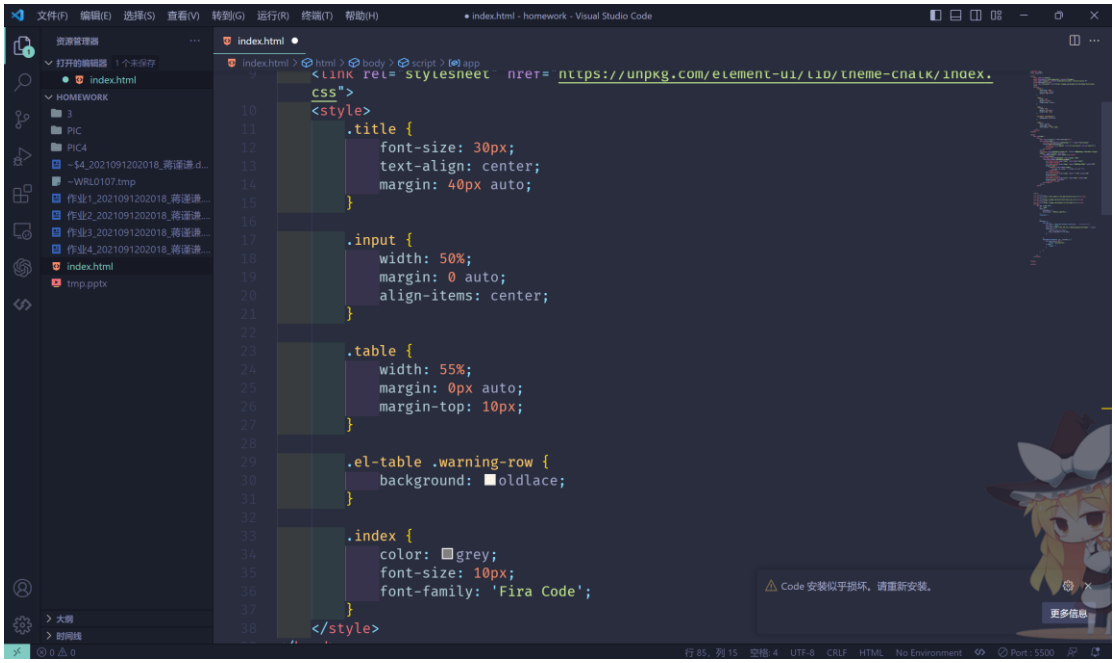

图 19：前端 CSS 代码

随后使用 live server 插件打开网站，在输入框输入“数据库原理及应用”，点击放大镜搜索按钮，相关结果见下图：

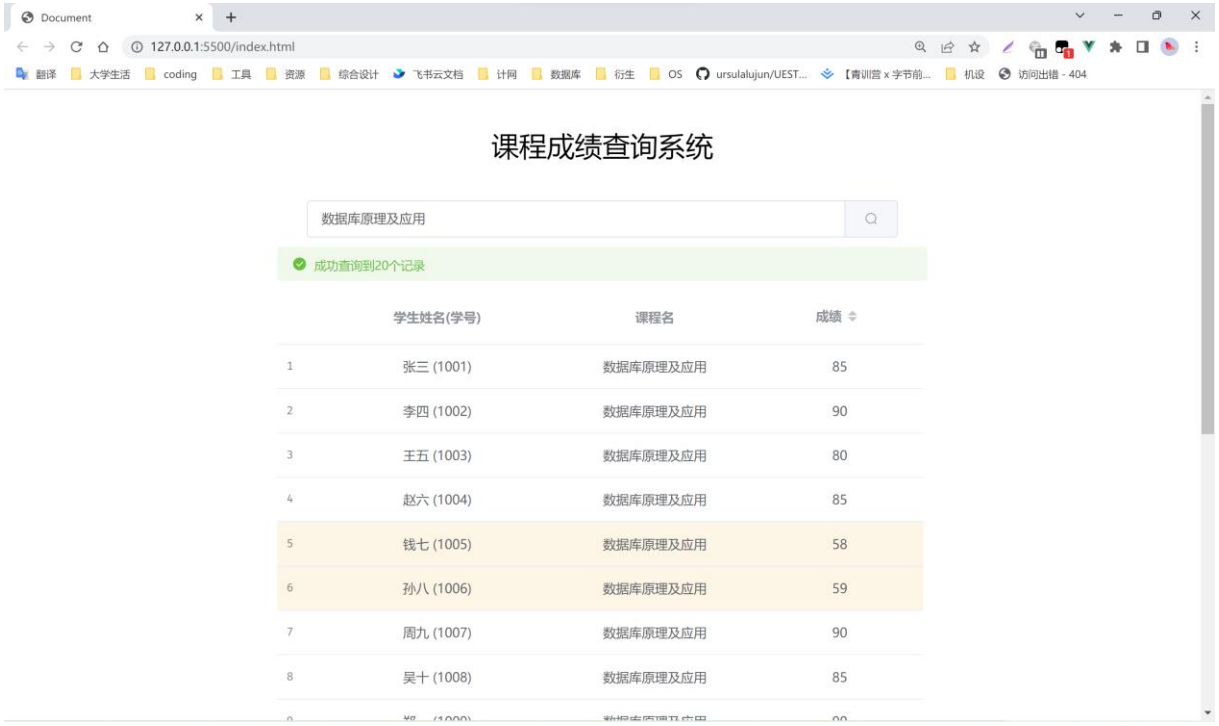

图 20：查询“数据库原理及应用”

可以看到，表格完成了正确的渲染。同时，考虑到实际功能需求，我们对页面进行功能优化，使得整个页面设计更加符合用户需求：

- ✓ 使用了橙色底来标识成绩不合格的学生行
- ✓ 搜索框下方使用绿色提醒框展示查询到的总计个数
- ✓ 点击表头的“成绩”栏的上下箭头可以进行排序（本图暂未排序）

随后，使用开发者工具进行网络请求抓包，验证前后端通信过程

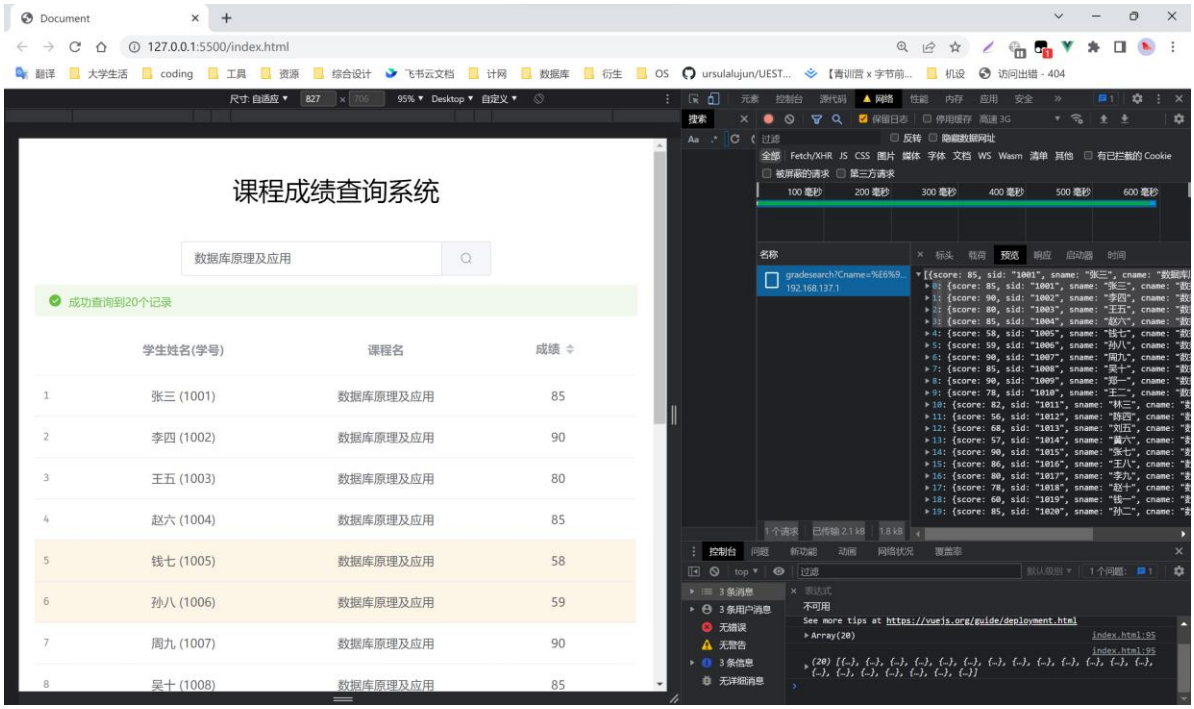

图 21：使用开发者工具验证前后端通信

可以看到上图右侧，前端发送了正确的请求，后端返回了正确的 json 结果数据，因此可以得出结论——前后端通信正确。

同理我们也可以查询“数据结构与算法”的成绩信息

图 22：查询数据结构与算法的成绩（排序后）

在上图的“成绩”列头位置，我们点击了升序排列的按钮（蓝色向上小三角），可以看到成绩也已经按照成绩升序排列，黄色底行均为不合格学生数据，共 6 位，输出结果完全符合预期。

综上，我们完成了整个系统的搭建。

本系统的相关代码会放在本作业文档的末尾。

四、 心得体会与感悟

通过本次大作业与实践研究，收获颇多。

首先对数据库的从业人员的道德约束要求和隐私数据加密技术进行进一步深入了解，感悟到了加强自身法规与职业道德意识的重要性，同时学会从各个技术层面对数据进行加密处理的基本方略。在当今信息化时代，数据越来越重要，保护隐私和数据安全对于个人和企业来说都至关重要。如何使用加密算法、访问控制等技术来保护数据隐私，实现数据安全…这些探索让我认识到自己作为一名程序员应该具有的社会责任感和职业道德意识。

随后对于触发器、存储过程、游标等一系列数据库后端编程内容进行实践实战，更加加深了对这几个概念的深刻理解。我发现触发器、存储过程、游标等技

术都是非常实用的，可以大幅度提高数据库的效率和可靠性。当然，在编程时部分地方还略显生疏，这几个知识点皆为重点，需要多加巩固，只有精益求精，才能写出高质量的代码。

紧接着，通过编程实践了 Java Web 对数据库进行查询的项目编程，通过对课程成绩查询系统的构造，掌握了前后端分离模式下使用 Mybatis 框架进行数据库快速开发的要点和核心技术。在项目开发中，我学习到了前后端分离的开发理念，通过 Mybatis 框架进行数据库快速开发，同时也学习到了使用 Vue.js 进行前端开发的方法，使得前后端工作高度协同，编程效率得到了极大的提升。的确，类似于 Mybatis、Vue、Spring 等框架的出现对于当代程序员的开发效率有极大幅度的提升，我们应该学会应用这些便捷的框架，提高编程效率和规范程度。同时，也感悟到了数据库编程技术的魅力。

综上所述，本次大作业和实践研究让我受益匪浅，不仅让我掌握了数据库编程技术和 Web 开发技术，还让我深刻认识到程序员应该具备的职业道德和社会责任感，对于未来的技术发展和自身职业规划也有很大的启示作用。

附：挑战性研讨程序核心代码段

仅展示核心的代码段，其余代码可以用默认配置。

1 后端部分——

Res.java

```
    package com.example.gradesearch.entity;
    import lombok.Data;

    @Data
    public class Res {
        String Sname; //学生姓名
        String SID;   //学生学号
        String Cname; //课程名称
        String CID;   //课程号
        int Score;    //课程成绩
}
```

ResMapper.java

```
    package com.example.gradesearch.mapper;

    import org.apache.ibatis.annotations.Mapper;
    import org.apache.ibatis.annotations.Param;
    import org.apache.ibatis.annotations.Select;
    import com.example.gradesearch.entity.Res;
    import java.util.List;

    @Mapper
```

```
    public interface ResMapper {
        /**
         * 根据课程名称查询学生的成绩信息。
         * @param Cname String 类型，表示需要查询的课程名称参数。
         * @return 返回 List<Res> 类型，表示查询结果集合。
         */
     @Select("SELECT    STUDENT.SID,    STUDENT.SName,    COURSE.CName,
GRADE.Score,COURSE.CID  FROM  STUDENT,  COURSE,  GRADE  WHERE  GRADE.CID  =
COURSE.CID AND GRADE.SID = STUDENT.SID AND COURSE.CName = #{Cname}")
        List<Res> getScore(@Param("Cname") String Cname);
    }
```

Rescontroller.java

```
    package com.example.gradesearch.controller;

    import com.example.gradesearch.entity.Res;
    import com.example.gradesearch.mapper.ResMapper;
    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.web.bind.annotation.*;
    import java.util.List;
    @CrossOrigin(origins  =  "*",  allowedHeaders  =  "*",  methods  =
{RequestMethod.GET})//解决跨域问题
    @RestController
    // ResController 类，控制器类，用于处理成绩查询请求。
    public class ResController {

        //依赖注入 ResMapper 对象，用于数据持久化操作
        @Autowired
        ResMapper resMapper;

        /**
         * 处理 GET 请求 /gradesearch，并根据参数 Cname 查询学生的成绩信息。
         * @param Cname String 类型，表示课程名称参数。
         * @return 返回 List<Res> 类型，表示查询结果集合。
         */
        @GetMapping("/gradesearch")
        public List<Res> searchScore(@RequestParam String Cname){
            return resMapper.getScore(Cname);
        }
    }
```

2 前端部分——

Template 部分

```
    <div id="app">
         <div>
             <div class="title">课程成绩查询系统</div>
             <div class="input">
                 <el-input  placeholder="请输入查询课程"  v-
model="SearchInput" class="input-with-select">
                 <el-button      slot="append"      icon="el-icon-search"
```

```
@click="search"></el-button>
                </el-input>
            </div>
            <el-alert  v-if="tableData.length!=0"  :title="'成功查询到
'+tableData.length+'个记录'" type="success" show-icon
                  :closable="false" class="table"></el-alert>
            <div class="table">
              <el-table :data="tableData" style="width: 100%" :row-class-
name="tableRowClassName">
                <el-table-column type="index" class-name="index">
                 </el-table-column>
                 <el-table-column  prop="sname"  label="学生姓名(学号)"
width="250" align="center">
                        <template slot-scope="scope">
                            {{scope.row.sname+' ('+scope.row.sid+')'}}
                        </template>
                    </el-table-column>
                    <el-table-column  prop="cname"  label="课程名"
width="220" align="center">
                    </el-table-column>
                    <el-table-column   prop="score"   label="成绩"
width="180" align="center" sortable>
                    </el-table-column>
                </el-table>
              </div>
            </div>
        </div>
```

script 部分

```
    <script>
            var app = new Vue({
                el: '#app',
                data: {
                    tableData: [],
                    SearchInput: '数据库原理及应用',
                },
                methods: {
                    search() {
                        let input = this.SearchInput.toString();
                        //获取输入框的值
                        let this_ = this;

    axios.get('http://192.168.137.1:8081/gradesearch?Cname=' +
    input).then(res => {
                        //发送网络请求
                            console.info(res.data);
                            this_.tableData = res.data;
                        })
                    },
                    tableRowClassName({ row, rowIndex }) {
```

```
                    if (row.score < 60) {
                        return 'warning-row';
                    } else {
                        return '';
                    }
                }
            }
        })
    </script>
```

CSS（style）部分仅对页面进行美化，不太重要且内容较多，因篇幅原因略去。