2024-2025学年度第一学期

华南师范大学

计算机学院

《大型数据库技术及应用课程设计》报告

题目： 基于PL/SQL的学生成绩管理系统

班级： 计算机科学与技术2班 计算机科学与技术3班

学号： 20212821020 20212821028

姓名： 林泽勋 林江荣

指导老师： 贺超波

目录

1 背景介绍

教育质量的高低，决定着一个国家的富强程度和发展水平，尤其是在高等教育领域，培养国家的脊梁和精英是高校的重要使命。随着科学技术的迅猛发展，计算机科学技术的进步不仅深刻改变了人们的生活方式，也在社会各领域发挥着越来越不可替代的作用。在高校教育领域，因特网技术的高速发展已经改变了传统的教学管理模式，并为教学模式提出了新的变革。特别是在高校学生成绩管理方面，计算机应用技术的使用已成为一个显著的趋势，其展现出的优势是传统手工管理无法比拟的[1]。

在这样的背景下，学生成绩管理系统（Student Grade Management System，简称 SGMS）的应用，为高校教师提供了一个强大的工具，使他们能够更加高效地管理并分析所教授各个班级的成绩情况。通过这样的系统，教师可以轻松录入、查询、统计和分析学生成绩，从而更准确地把握学生的学习情况，及时调整教学策略，提高教学质量。

学生成绩管理系统主要包含课程模块、班级模块和学生模块。课程模块具备新增课程和删除课程功能，还能根据关键词查询课程。班级模块可以新增班级和删除班级，同时支持通过关键词查询班级信息。学生模块能够查询班级学生信息，导入学生到班级，修改学生成绩以及统计学生成绩分布。这些模块相互协作，构成了一个完整的学生成绩管理系统。

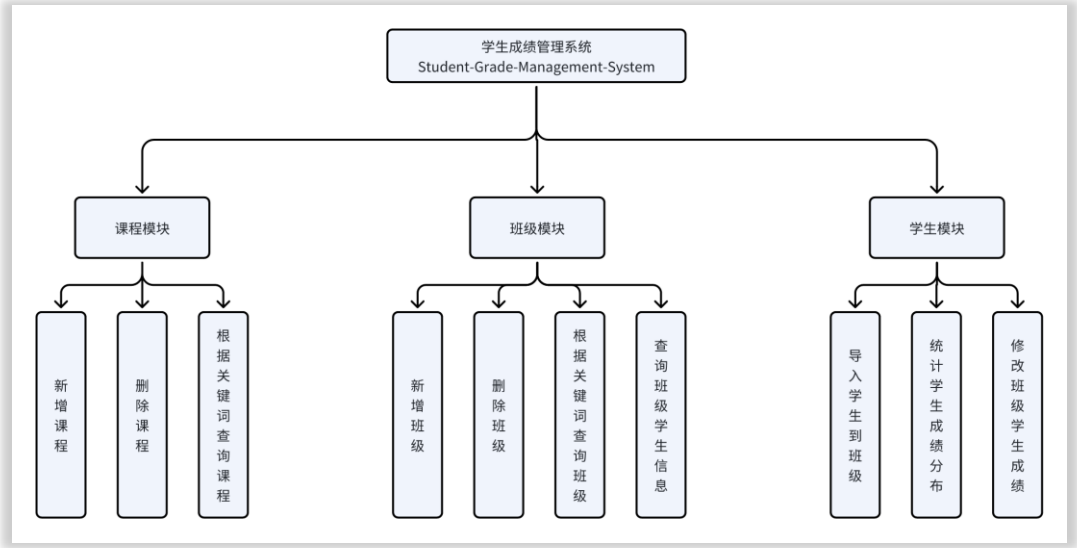

图 1 系统功能模块图

2 数据库设计

2.1 概念设计

概念设计的核心目的是为了建立概念数据模型。概念数据模型（Conceptual Data Model），简称概念模型，是面向数据库用户的现实世界的模型，主要用来描述世界的概念化结构，它使得我们在设计数据库的初始阶段，摆脱计算机系统及 DBMS 的具体技术问题，集中精力分析数据以及数据之间的联系等。

概念数据模型通常采用实体-关系图（E-R 图）作为其标准描述方式。下图展示了本系统的 E-R 图，详细描绘了系统中的实体及其相互关系。

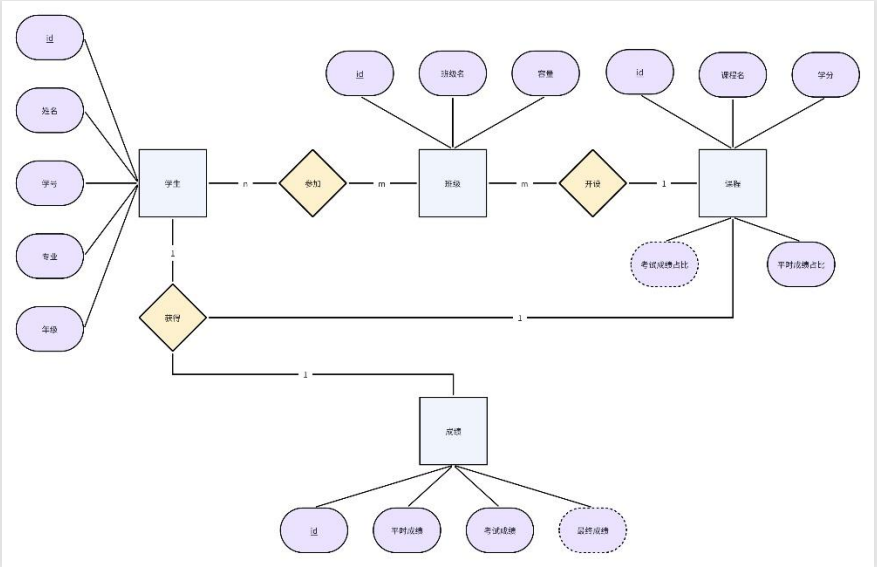

图 2 系统概念模型 E-R 图表示

2.2 逻辑设计

继概念设计阶段成功建立概念数据模型之后，即可进入逻辑设计的阶段。逻辑设计的主要任务是将概念模型转化为逻辑数据模型，这一模型将更加贴近于特定数据库管理系统的实现。这一阶段将详细定义数据结构、数据类型、属性、键以及实体之间的关系，同时考虑数据的完整性、一致性和安全性。

为了实现这一目标，我们选择采用关系数据模型作为逻辑数据模型的表现形式，依据 E-R 图建立本系统的关系数据模型，具体描述如下：

(1) 学生表（stu_tb）

a) 记录 ID（id）：表主键，整型，自动增长

b) 学生姓名（name）：存储学生姓名，最大长度为 100 个字符

c) 学号（campus_id）：存储学生学号，最大长度为 20 个字符

d) 专业（major）：存储学生所属专业名称，最大长度为 100 个字符

e) 年级（grade）：存储学生的年级信息，最大长度为 4 个字符

(2) 课程表（course_tb）

a) 课程 ID（id）：表主键，整型，自动增长

b) 课程名（name）：存储课程名称，最大长度为 100 个字符，不允许为空

c) 学分（credit）：存储课程的学分数，类型为数字，保留一位小数

d) 平时成绩占比（daily_ratio）：存储平时成绩在总成绩中的占比，类型为数字，保留两位小数，取值范围为 0 到 1

(3) 班级表（class_tb）

a) 班级 ID（id）：表主键，整型，自动增长

b) 班级名（name）：存储班级名称，最大长度为 100 个字符，不允许为空

c) 班级容量（capacity）：存储班级的最大容纳学生数，必须为正整数

(4) 学生-班级表（stu_class_tb）

a) 记录 ID（id）：表主键，整型，自动增长

b) 学生记录 ID（stu_id）：外键，存储关联的学生 ID，不允许为空

c) 班级 ID（class_id）：外键，存储关联的班级 ID，不允许为空

(5) 课程-班级表（cou_class_tb）

a) 记录 ID（id）：表主键，整型，自动增长

b) 班级 ID（class_id）：外键，存储关联的班级 ID，不允许为空

c) 课程 ID（course_id）：外键，存储关联的课程 ID，不允许为空

(6) 学生-课程-成绩表（stu_cou_score_tb）

a) 记录 ID（id）：表主键，整型，自动增长

b) 学生记录 ID（stu_id）：外键，存储关联的学生 ID，不允许为空

c) 课程 ID（course_id）：外键存储关联的课程 ID，不允许为空

d) 平时成绩（daily_score）：存储学生的平时成绩，取值范围为 0 到 100

e) 考试成绩（exam_score）：存储学生的考试成绩，取值范围为 0 到 100

2.3 物理设计

在本系统的物理设计阶段，将采用 Oracle 数据库和 PL/SQL 进行实现，以下是详细的物理设计步骤。

(1) 确定存储结构和存取方法：本系统将在 Oracle 数据库中使用表空间来管理数据存储，为不同类型的数据选择合适的表空间，如用户数据、临时数据等；

(2) 存储空间优化：利用 Oracle 的自动扩展表空间功能，以适应数据量的增长；除此之外，通过调整 PCTFREE 和 PCTUSED 参数，以优化行的插入和更新操作，以减少行链接、行迁移等情况；

(3) 索引建立：针对查询频繁且数据量大的表，我们将实施索引策略以加速查询处理。例如，对于班级表（class_tb），可在班级名称（name）等关键字段上创建索引，以支持快速的数据检索和连接操作。这将显著提高查询响应时间，尤其是在执行联表查询和数据聚合时；

3 关键技术实现

3.1 数据库端技术实现

对于数据库端的相关实现，本系统采用了基于 Oracle 的解决方案，并充分利用了 PL/SQL 的强大功能。PL/SQL 是 Oracle 数据库对 SQL 语句的扩展，通过在 SQL 命令语言中增加过程处理语句（如分支、循环等）使得 SQL 语言具备过程处理能力。

3.1.1 表空间与用户的创建

为了实现成绩管理系统的高效运行和数据安全性，我们为本系统在 Oracle 数据库中创建了专用的表空间和用户，这样做可以确保系统数据的隔离性，优化性能，简化备份和恢复流程，并提高整体的安全性和维护便利性。具体的 SQL 语句如下所示。

```
-- 创建表空间
CREATE TABLESPACE SGMS
DATAFILE 'D:\ora\oradata\ORCL2\SGMS.dbf' SIZE 100M
AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;

-- 创建用户
```

```
CREATE USER c##sgms IDENTIFIED BY sgms;

-- 将表空间设置为用户的默认表空间
ALTER USER c##sgms DEFAULT TABLESPACE SGMS;

-- 授予相关权限
GRANT UNLIMITED TABLESPACE TO c##sgms;
GRANT SELECT ANY TABLE TO c##sgms;
GRANT INSERT ANY TABLE TO c##sgms;
GRANT UPDATE ANY TABLE TO c##sgms;
GRANT DELETE ANY TABLE TO c##sgms;
GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, CREATE VIEW,
CREATE PROCEDURE, CREATE TYPE, CREATE TRIGGER TO c##sgms;

-- 使得更改生效
ALTER USER c##sgms ACCOUNT LOCK;
ALTER USER c##sgms ACCOUNT UNLOCK;
```

3.1.2 数据库表的创建

针对前文建立的关系模型，以 Oracle 数据库为基础，我们进行了数据库表的创建。这一步骤是将概念模型转化为物理存储结构的关键环节。在 Oracle 中，我们根据设计的 ER 图和业务需求，定义了各个表的结构和字段，包括数据类型、长度、约束等，以确保数据的完整性和准确性。我们还为每个表配置了适当的索引，以优化查询性能，使得数据检索更加高效。具体建表语句可参考附录 A。

3.1.3 数据增删改查操作的实现

在本系统中，我们采用 PL/SQL 函数来实现数据的增删改查操作，这种实现方式使得后端与数据库的交互变得简洁而高效。通过在数据库端定义这些函数，我们能够直接处理数据操作，从而减少了网络传输的开销，并提高了数据处理的速度。此外，这种方法还增强了数据操作的安全性和一致性，因为所有的数据验证和业务逻辑都在数据库层面得到了集中管理。下面是对本系统各个 PL/SQL 函数的流程说明：

(1) **insert_course 函数**：用于向 course_tb 表中插入新的课程记录。函数接收课程名称、学分和平时成绩占比作为参数，然后执行插入操作，并将新插入的课 ID 返回给调用者。在插入过程中，如果遇到任何异常，函数会回滚事务并返回-1。成功插入后，会提交事务并输出一条消息，显示插入的课程 ID。

```
CREATE OR REPLACE FUNCTION insert_course(
    p_name IN course_tb.name%TYPE,
    p_credit IN course_tb.credit%TYPE,
    p_daily_ratio IN course_tb.daily_ratio%TYPE
) RETURN course_tb.id%TYPE
IS v_id course_tb.id%TYPE;
BEGIN
    INSERT INTO course_tb (name, credit, daily_ratio)
    VALUES (p_name, p_credit, p_daily_ratio)
    RETURNING id INTO v_id;

    DBMS_OUTPUT.PUT_LINE('插入课程成功，课程 ID 为: '  || v_id);
    COMMIT;
    RETURN v_id;  -- 返回插入记录的主键值
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;  -- 如果出现异常，回滚事务，并返回-1
        DBMS_OUTPUT.PUT_LINE('插入课程失败，具体错误信息: ' || SQLERRM);
        RETURN -1;
END;
/
```

(2) **delete_course_by_id 函数：**用于根据给定的课程 ID 删除 course_tb 表中的课程记录。首先，函数检查要删除的课程是否存在。如果存在，函数执行删除操作并返回 1 表示删除成功。如果课程不存在，则返回-1 表示未找到对应课程。整个过程中，任何异常都会导致事务回滚。

```
CREATE OR REPLACE FUNCTION delete_course_by_id(p_course_id NUMBER)
RETURN NUMBER IS
    v_count NUMBER;
BEGIN
    -- 先检查要删除的课程 id 是否存在
    SELECT COUNT(*)
    INTO v_count
    FROM course_tb
    WHERE id = p_course_id;

    IF v_count > 0 THEN
        -- 如果存在则执行删除操作
        DELETE FROM course_tb
        WHERE id = p_course_id;
        COMMIT;
        -- 返回表示成功删除的代码（这里可以自定义合适的值，比如 1 表示成功）
        RETURN 1;
```

```
    ELSE
        -- 如果不存在，返回表示未找到对应课程的代码（这里可以自定义合适的值，
比如 -1 表示没找到）
        RETURN -1;
    END IF;
END;
/
```

(3) **get_course_class_count 函数：**返回一个游标，用于查询课程及其关联的班级数量。函数根据输入参数是否为空来构建不同的查询语句。如果没有提供课程名，函数将返回所有课程及其班级数量；如果提供了课程名，函数将返回与课程名匹配的课程及其班级数量。查询结果通过游标返回，调用者可以遍历游标获取所有结果。

```
CREATE OR REPLACE FUNCTION get_course_class_count(p_course_name IN
VARCHAR2 DEFAULT NULL) RETURN SYS_REFCURSOR IS
    v_cursor SYS_REFCURSOR;
    v_sql VARCHAR2(4000);
BEGIN
    -- 根据参数是否为 NULL 来构建不同的查询语句
    IF p_course_name IS NULL THEN
        v_sql := 'SELECT c.id AS id, c.name AS name, c.credit AS credit, c.daily_ratio AS
daily_ratio, COUNT(cc.class_id) AS count
                  FROM course_tb c
                  LEFT JOIN cou_class_tb cc ON c.id = cc.course_id
                  GROUP BY c.id, c.name, c.credit, c.daily_ratio';
    ELSE
        v_sql := 'SELECT c.id AS id, c.name AS name, c.credit AS credit, c.daily_ratio AS
daily_ratio, COUNT(cc.class_id) AS count
                  FROM course_tb c
                  LEFT JOIN cou_class_tb cc ON c.id = cc.course_id
                  WHERE c.name LIKE "%' || p_course_name || '%"
                  GROUP BY c.id, c.name, c.credit, c.daily_ratio';
    END IF;

    -- 打开游标执行构建好的查询语句
    OPEN v_cursor FOR v_sql;
    RETURN v_cursor;
END;
/
```

(4) **insert_class 函数：**用于创建新班级并将其与课程关联。函数接收班级名称、班级容量和课 ID 作为输入参数。首先，函数将班级信息插入到 class_tb 表

中，然后插入关联信息到 cou_class_tb 表。成功插入后，函数提交事务并返回新插入的班级 ID。如果过程中出现异常，函数会回滚事务并返回-1。

```
CREATE OR REPLACE FUNCTION insert_class(
    c_name IN class_tb.name%TYPE,
    c_capacity IN class_tb.capacity%TYPE,
    p_id IN course_tb.id%TYPE
) RETURN class_tb.id%TYPE
IS v_id class_tb.id%TYPE;
BEGIN
    -- 插入数据到班级表
    INSERT INTO class_tb (name, capacity)
    VALUES (c_name, c_capacity)
    RETURNING id INTO v_id;  -- 获取自增主键的值

    -- 插入关联到课程-班级表
    INSERT INTO cou_class_tb (class_id, course_id)
    VALUES (v_id, p_id);

    DBMS_OUTPUT.PUT_LINE('插入班级成功，班级 ID 为: '  || v_id);
    COMMIT;  -- 提交事务，确保数据持久化
    RETURN v_id;  -- 返回插入记录的主键值
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;  -- 如果出现异常，回滚事务，并返回-1
        DBMS_OUTPUT.PUT_LINE('插入班级失败，具体错误信息: ' || SQLERRM);
        RETURN -1;
END;
/
```

(5) **delete_class_by_id 函数：**用于根据给定的班级 ID 删除 class_tb 表中的班级记录。函数首先检查要删除的班级是否存在。如果存在，函数执行删除操作并返回 1 表示删除成功。如果班级不存在，则返回-1 表示未找到对应班级。整个过程中，任何异常都会导致事务回滚。

```
CREATE OR REPLACE FUNCTION delete_class_by_id(p_class_id NUMBER)
RETURN NUMBER IS
    v_count NUMBER;
BEGIN
    -- 先检查要删除的班级 id 是否存在
    SELECT COUNT(*)
    INTO v_count
    FROM class_tb
    WHERE id = p_class_id;
```

```
    IF v_count > 0 THEN
        -- 如果存在则执行删除操作
        DELETE FROM class_tb
        WHERE id = p_class_id;
        COMMIT;
        RETURN 1;
    ELSE
        RETURN -1;
    END IF;
END;
/
```

(6) **get_class 函数**：返回一个游标，用于查询班级及其关联的课程信息。函数根据输入参数是否为空来构建不同的查询语句。如果没有提供班级名，函数将返回所有班级及其关联的课程信息；如果提供了班级名，函数将返回与班级名匹配的班级及其关联的课程信息。查询结果通过游标返回，调用者可以遍历游标获取所有结果。

```
CREATE OR REPLACE FUNCTION get_class(p_class_name IN VARCHAR2 DEFAULT
NULL) RETURN SYS_REFCURSOR IS
    v_cursor SYS_REFCURSOR;
    v_sql VARCHAR2(4000);
BEGIN
    -- 根据参数是否为 NULL 来构建不同的查询语句
    IF p_class_name IS NULL THEN
        v_sql := 'SELECT c1.id AS id, c1.name AS name, c1.capacity AS capacity, c2.id AS
course_id, c2.name AS course_name
                  FROM class_tb c1
                  LEFT JOIN cou_class_tb cc ON c1.id = cc.class_id
                  LEFT JOIN course_tb c2 ON c2.id = cc.course_id
                  GROUP BY c1.id, c1.name, c1.capacity, c2.id, c2.name';
    ELSE
        v_sql := 'SELECT c1.id AS id, c1.name AS name, c1.capacity AS capacity, c2.id AS
course_id, c2.name AS course_name
                  FROM class_tb c1
                  LEFT JOIN cou_class_tb cc ON c1.id = cc.class_id
                  LEFT JOIN course_tb c2 ON c2.id = cc.course_id
                  WHERE c1.name LIKE '%' || p_class_name || '%'
                  GROUP BY c1.id, c1.name, c1.capacity, c2.id, c2.name';
    END IF;

    -- 打开游标执行构建好的查询语句
```

```
    OPEN v_cursor FOR v_sql;
    RETURN v_cursor;
END;
/
```

(7) **get_class_student_info 函数：**返回一个游标，用于查询特定班级中所有学生的详细信息，包括学生基本信息和成绩。函数构建一个查询语句，根据班级 ID 检索学生信息，并返回一个包含学生 ID、姓名、学号、专业、年级以及成绩信息的游标。调用者可以遍历游标获取所有学生的具体信息。

```
CREATE OR REPLACE FUNCTION get_class_student_info(p_class_id NUMBER)
RETURN SYS_REFCURSOR IS
    v_cursor SYS_REFCURSOR;
    v_sql VARCHAR2(4000);
BEGIN
    v_sql := 'SELECT st.id, st.name, st.campus_id, st.major, st.grade, scsv.daily_score,
scsv.daily_ratio, scsv.exam_score, scsv.exam_ratio, scsv.final_score
                FROM stu_tb st
                JOIN stu_class_tb sct ON st.id = sct.stu_id
                JOIN cou_class_tb cct ON sct.class_id = cct.class_id
                LEFT JOIN stu_cou_score_view scsv ON st.id = scsv.stu_id AND
cct.course_id = scsv.course_id
                WHERE cct.class_id = ' || p_class_id || '
                GROUP BY st.id, st.name, st.campus_id, st.major, st.grade, scsv.daily_score,
scsv.daily_ratio, scsv.exam_score, scsv.exam_ratio, scsv.final_score';

    -- 打开游标执行构建好的查询语句
    OPEN v_cursor FOR v_sql;
    RETURN v_cursor;
END;
/
```

(8) **insert_student 函数：**用于将新学生插入到学生表 stu_tb 并关联到班级。函数首先检查学生是否已存在，如果存在则直接将学生 ID 和班级 ID 关联到 stu_class_tb 表。如果学生不存在，则插入新学生记录并进行关联。成功插入后，函数提交事务并返回新插入的学生 ID。如果过程中出现异常，函数会回滚事务并返回-1。

```
CREATE OR REPLACE FUNCTION insert_student(
    s_name IN stu_tb.name%TYPE,
    s_campus_id IN stu_tb.campus_id%TYPE,
    s_major IN stu_tb.major%TYPE,
    s_grade IN stu_tb.grade%TYPE,
```

```
    c_id IN class_tb.id%TYPE
) RETURN stu_tb.id%TYPE IS
    s_id stu_tb.id%TYPE;
BEGIN
    -- 先查询学生是否已存在
    SELECT id
    INTO s_id
    FROM stu_tb
    WHERE name = s_name
      AND campus_id = s_campus_id
      AND major = s_major
      AND grade = s_grade;

    -- 插入关联到学生-班级表
    INSERT INTO stu_class_tb (stu_id, class_id)
    VALUES (s_id, c_id);

    DBMS_OUTPUT.PUT_LINE('插入学生成功，学生 ID 为: ' || s_id);
    COMMIT;
    RETURN s_id;  -- 返回插入记录的主键值

    -- 如果没有找到（即 NO_DATA_FOUND 异常被触发），则执行插入学生操作
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            BEGIN
                -- 插入数据到学生表
                INSERT INTO stu_tb (name, campus_id, major, grade)
                VALUES (s_name, s_campus_id, s_major, s_grade)
                RETURNING id INTO s_id;
                -- 插入关联到学生-班级表
                INSERT INTO stu_class_tb (stu_id, class_id)
                VALUES (s_id, c_id);
                DBMS_OUTPUT.PUT_LINE('插入学生成功，学生 ID 为: ' || s_id);
                COMMIT;
                RETURN s_id;  -- 返回插入记录的主键值
            EXCEPTION
                WHEN OTHERS THEN
                    ROLLBACK;
                    DBMS_OUTPUT.PUT_LINE('插入学生失败，具体错误信息: ' ||
SQLERRM);
                    RETURN -1;
            END;
END;
/
```

(9) **delete_stu_class_by_id 函数：**用于删除学生和班级的关联关系。函数首先检查要删除的关联记录是否存在。如果存在，函数执行删除操作并返回 1 表示删除成功。如果关联记录不存在，则返回-1 表示未找到对应关联。整个过程中，任何异常都会导致事务回滚。

```
CREATE OR REPLACE FUNCTION delete_stu_class_by_id(
    p_stu_id NUMBER,
    p_class_id NUMBER
)
RETURN NUMBER IS
    v_count NUMBER;
BEGIN
    -- 先检查要删除的班级 id 是否存在
    SELECT COUNT(*)
    INTO v_count
    FROM stu_class_tb
    WHERE stu_id = p_stu_id AND class_id = p_class_id;

    IF v_count > 0 THEN
        -- 如果存在则执行删除操作
        DELETE FROM stu_class_tb
        WHERE stu_id = p_stu_id AND class_id = p_class_id;
        COMMIT;
        RETURN 1;
    ELSE
        RETURN -1;
    END IF;
END;
/
```

(10) **add_or_update_stu_cou_score 函数：**用于添加或更新学生的成绩记录。函数首先检查是否存在对应学生和课程的成绩记录。如果存在，则更新成绩；如果不存在，则插入新的成绩记录。函数提交事务并返回受影响的行数，这通常是 1。如果过程中出现异常，函数会回滚事务并返回 0。

```
CREATE OR REPLACE FUNCTION add_or_update_stu_cou_score(
    p_stu_id NUMBER,
    p_course_id NUMBER,
    p_daily_score NUMBER,
    p_exam_score NUMBER
) RETURN NUMBER IS
    v_count NUMBER;
BEGIN
```

```
    -- 先检查是否已存在对应学生和课程的成绩记录
    SELECT COUNT(*)
    INTO v_count
    FROM stu_cou_score_tb
    WHERE stu_id = p_stu_id
      AND course_id = p_course_id;

    IF v_count > 0 THEN
        -- 如果存在，更新成绩
        UPDATE stu_cou_score_tb
        SET daily_score = p_daily_score,
            exam_score = p_exam_score
        WHERE stu_id = p_stu_id
          AND course_id = p_course_id;
    ELSE
        -- 如果不存在，插入新记录
        INSERT INTO stu_cou_score_tb (stu_id, course_id, daily_score, exam_score)
        VALUES (p_stu_id, p_course_id, p_daily_score, p_exam_score);
    END IF;

    COMMIT;
    -- 返回受影响的行数（插入为 1，更新则是更新的行数，通常为 1，若有异常则为
0）
    RETURN SQL%ROWCOUNT;
END;
/
```

3.1.4 触发器的创建

为了确保添加到班级的学生数量不会超过该班级的容量限制，我们设置了一个在向 stu_class_tb 表插入或更新数据前进行检查的触发器。触发器在每次插入或更新操作之前被激活，对每一行影响的数据执行以下操作：

1. 首先，触发器会计算出当前班级（即新插入或更新的班级）在 stu_class_tb 表中已有的学生数量，并将这个值存储在变量 current_count 中。
2. 然后，触发器会从 class_tb 表中获取对应班级的容量限制，并将这个值存储在变量 class_capacity 中。
3. 如果 current_count + 1（即插入或更新后的学生数量）超过了 class_capacity（班级的容量限制），触发器会通过 RAISE_APPLICATION_ERROR 抛出一个自定义异常，错误信息为“班级学生数量超过容量限制”。这个异常会阻止当前

的插入或更新操作，确保班级的学生数量不会超过设定的容量。

```
-- 创建触发器，在向 stu_class_tb 表插入或更新数据前进行检查
CREATE OR REPLACE TRIGGER check_student_count
BEFORE INSERT OR UPDATE ON stu_class_tb
FOR EACH ROW
DECLARE
    current_count NUMBER;
    class_capacity NUMBER;
BEGIN
    -- 获取当前班级已有的学生数量
    SELECT COUNT(*) INTO current_count
    FROM stu_class_tb
    WHERE class_id = :NEW.class_id;

    -- 获取当前班级的容量限制
    SELECT capacity INTO class_capacity
    FROM class_tb
    WHERE id = :NEW.class_id;

    -- 如果插入或更新后学生数量超过容量限制，则抛出异常
    IF current_count + 1 > class_capacity THEN
        RAISE_APPLICATION_ERROR(-20001, '班级学生数量超过容量限制');
    END IF;
END;
/
```

3.2 系统开发技术实现

为了确保成绩管理系统的广泛适用性和便于测试，同时避免频繁在命令行终端手动执行相关数据库操作语句，我们构建了基于 B/S 架构的系统，这种架构允许用户通过浏览器界面与系统交互，而无需直接处理后端数据库，从而简化了操作流程，并提高了用户体验。

具体来说，本系统采用 Vue3 作为前端框架，结合 Express 作为后端框架，致力于构建一个现代化、高效、可靠的学生成绩管理系统。这种技术组合使得 SGMS 在提供强大的数据存储和快速处理能力的同时，也能够实现灵活的用户交互，满足高校教师在成绩管理方面的多样化需求。

3.2.1 前端部分

Vue 3 为学生成绩管理系统的前端开发提供了强大的支持，其响应式和组件

化的特性使得系统界面直观且易于操作。成绩的更新可以实时渲染，同时，系统的组件化设计让各个功能模块如成绩录入、查询和统计分析等可以独立开发和维护，提高了开发效率和系统的可扩展性。此外，Vue 3 的虚拟 DOM 技术优化了界面渲染，确保了在处理大量数据时系统的流畅性和响应速度，从而提升了整个成绩管理系统的用户体验和操作效率。

图 3 系统整体效果图

3.2.2 后端部分

后端选用基于 Nodejs 的 Express 开发框架来构建整个技术体系。它能够高效地处理各类复杂数据请求，凭借其出色的异步非阻塞 I/O 模型，让系统在应对大量并发数据处理任务时依然游刃有余，有力地确保了后台运行的高效性。

不仅如此，为实现与 Oracle 数据库的无缝对接与高效交互，后端采用了官方提供的 node-oracledb 库。通过该库，系统能够很好地完成与 Oracle 数据库的实时交互工作，无论是简单数据的插入，还是复杂数据结构的批量存储，都能精准无误地执行。

3.3 数据库优化

在本系统的设计与实现过程中，为了提高查询效率和数据访问数据速度，我们采取了下述优化措施：

(1) **索引优化**：为了提升查询性能，我们针对高频访问和大量数据的表建立了索引，特别是在课程表的课程名以及班级表的班级名等关键字段上创建了索引，

以此来提高了数据检索的速度。

(2) **视图应用**：在本系统中，我们采用了视图来简化成绩的查询和计算过程。具体来说，通过存储了学生的平时成绩、考试成绩以及平时成绩占比这三个基本数据项，之后构建一个视图，它根据每门课程的平时成绩占比自动计算学生的最终成绩。这个视图将成绩计算逻辑封装在数据库层面，避免了在应用层面进行重复和复杂的计算，从而提高了数据处理的效率和准确性。通过这种方式，我们不仅优化了数据库存储，还提升了系统性能，使得成绩查询变得更加直观和迅速。

(3) **查询优化**：利用了 PL/SQL 的程序化特性，如分支控制和游标，来优化查询性能。通过这些扩展，我们能够根据具体的查询需求动态构建和执行 SQL 语句，有效避免了全表扫描，减少了数据的冗余传输，进而提高了查询效率和系统响应速度。

4 测试

4.1 基础增删改查操作的测试

(1) 新增“高等数学”课程的测试

图 4 新增“高等数学”课程操作

图 5 新增“高等数学”课程成功

(2) 删除“高等数学”课程的测试

图 6 删除“高等数学”课程成功

(3) 根据关键词查询课程的测试

图 7 根据关键词查询课程成功

(4) 修改学生平时成绩和考试成绩的测试

修改该班级中小黄这名学生的平时成绩为 80。

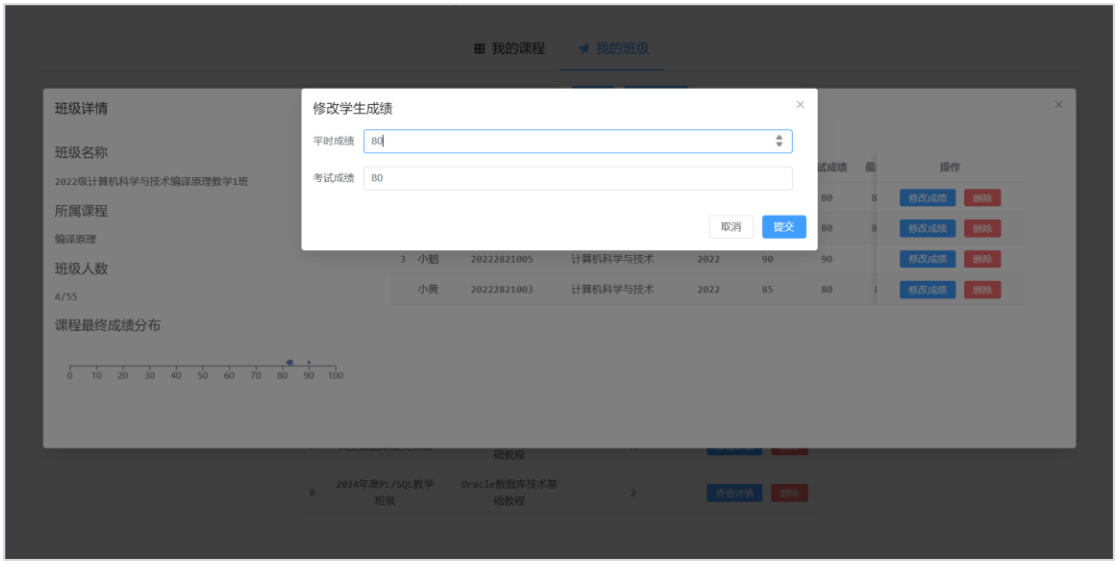

图 8 执行修改学生成绩操作

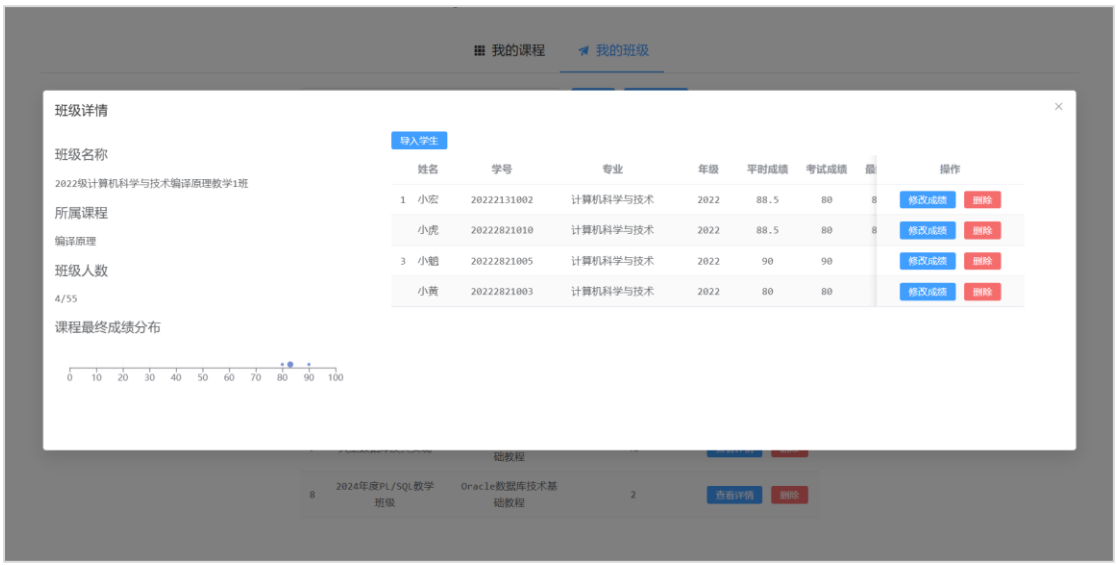

图 9 修改学生成绩操作成功

4.2 视图的测试

系统能够按照学生的平时成绩、考试成绩以及课程的平时成绩占比计算出计算出最终成绩。

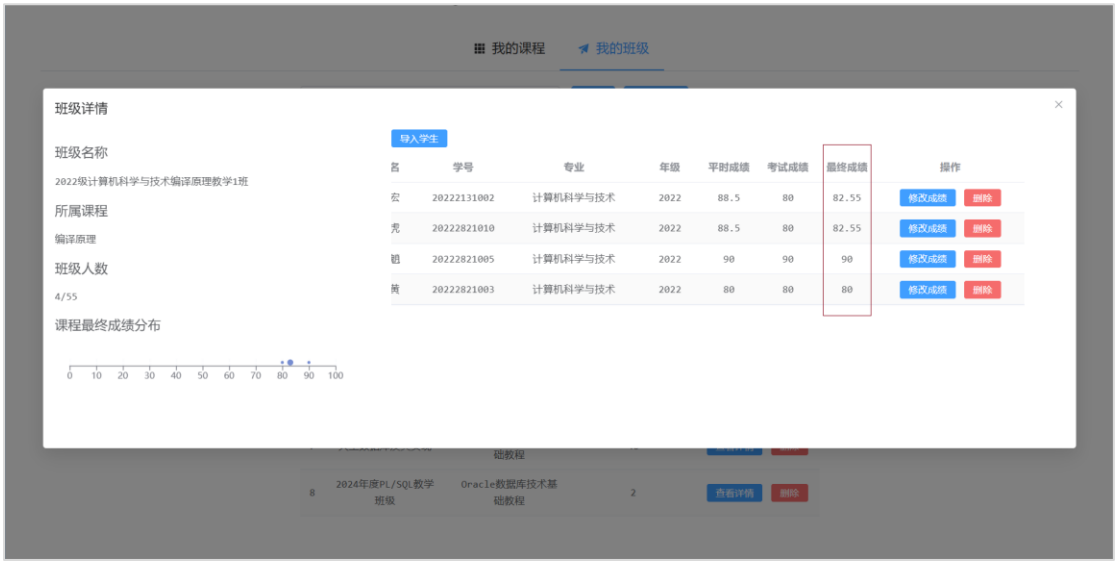

图 10 视图的测试

4.3 触发器的测试

在课程容量为 2 且已有两个学生的班级插入新学生提示插入失败。

图 11 触发器的测试

5 总结

在本次《大型数据库技术及应用课程设计》项目中，我们成功构建了一个基于 PL/SQL 的学生成绩管理系统。该系统旨在提高高校学生成绩管理的效率和准确性，通过现代化的数据库技术和应用实现，为教师提供了一个强大的成绩管理工具。系统的开发过程涉及了数据库设计、关键技术实现、系统开发以及数据库

优化等多个方面，最终实现了一个功能全面、操作便捷、性能优异的学生成绩管理系统。

在数据库设计阶段，我们采用了概念设计、逻辑设计和物理设计三个步骤，确保了数据库结构的合理性和高效性。概念设计阶段，我们通过实体-关系图（E-R 图）建立了概念数据模型，为后续的设计打下了坚实的基础。逻辑设计阶段，我们将概念模型转化为关系数据模型，详细定义了数据结构和关系。物理设计阶段，我们针对 Oracle 数据库进行了具体的存储结构和存取方法的确定，包括索引的建立和存储空间的优化。

在关键技术实现方面，我们充分利用了 PL/SQL 的强大功能，实现了数据库端的数据操作和业务逻辑。通过创建触发器，我们确保了班级学生数量不会超过容量限制，保证了数据的一致性和完整性。同时，我们还实现了数据增删改查操作的 PL/SQL 函数，提高了数据处理的速度和安全性。

系统开发方面，我们采用了 B/S 架构，使用 Vue3 作为前端框架，Express 作为后端框架，构建了一个现代化的用户界面。这种架构不仅简化了操作流程，还提高了用户体验。前端部分利用 Vue3 的响应式和组件化特性，实现了直观且易于操作的界面。后端部分则通过 Express 框架高效处理数据请求，并通过 node-oracledb 库与 Oracle 数据库实现了无缝对接。

在数据库优化方面，我们采取了索引优化、视图应用和查询优化等措施，显著提高了查询效率和数据访问速度。通过建立索引，我们加速了关键字段的数据检索。视图的应用简化了成绩的查询和计算过程，提高了数据处理的效率和准确性。PL/SQL 的程序化特性使我们能够根据具体需求动态构建和执行 SQL 语句，有效避免了全表扫描，减少了数据的冗余传输。

总的来说，本项目不仅加深了我们对大型数据库技术及应用的理解，还提高了我们的实践能力和问题解决能力。通过这个项目，我们学会了如何设计和实现一个完整的数据库系统，从概念设计到物理实现，再到系统的开发和优化，每一步都是对我们专业知识的一次挑战和提升。我们相信，这个学生成绩管理系统将在实际应用中发挥重要作用，为高校教育领域带来积极的影响。

参考文献

[1] 陈亮.基于 Web 的高校学生成绩管理系统的设计和实现[D].湖北工业大学,2017.

[2] 刘博.Oracle 数据库性能调整与优化[D].大连理工大学,2007.

[3] 张灵钰.基于 web 平台的高校学生工作管理系统的设计与实现[D].电子科技大学,2013.

[4] 王利明.基于 B/S 结合 C/S 结构的高校教务学生管理系统设计与实现[D].吉林大学,2012.

[5] 陈少云.基于 Web 的高职学院教务管理系统的设计与实现[D].四川大学,2005.

[6] 邹俊.基于 Oracle 数据库系统性能调整与优化研究[D].江西财经大学,2006.

附录 A Oracle 建表（视图及索引）语句

```
-- 创建学生表
DROP TABLE stu_tb;
CREATE TABLE stu_tb (
   id NUMBER GENERATED BY DEFAULT AS IDENTITY START WITH 1 INCREMENT
BY 1,
   name VARCHAR2(100),
   campus_id VARCHAR2(20),
   major VARCHAR2(100),
   grade VARCHAR2(4),
   CONSTRAINT pk_stu PRIMARY KEY (id)
);

-- 创建课程表
DROP TABLE course_tb;
CREATE TABLE course_tb (
   id NUMBER GENERATED BY DEFAULT AS IDENTITY START WITH 1 INCREMENT
BY 1,
   name VARCHAR2(100) NOT NULL,
   credit NUMBER(3,1),
   daily_ratio NUMBER(3,2) CHECK (daily_ratio >= 0 AND daily_ratio <= 1),
   CONSTRAINT pk_course PRIMARY KEY (id)
);

-- 课程名称索引
CREATE INDEX idx_course_name
ON course_tb (name);

-- 创建班级表
DROP TABLE class_tb;
CREATE TABLE class_tb (
   id NUMBER GENERATED BY DEFAULT AS IDENTITY START WITH 1 INCREMENT
BY 1,
   name VARCHAR2(100) NOT NULL,
   capacity NUMBER CHECK (capacity > 0),
   CONSTRAINT pk_class PRIMARY KEY (id)
);

-- 班级名称索引
CREATE INDEX idx_class_name
ON class_tb (name);
```

```
-- 创建学生-班级关联表
DROP TABLE stu_class_tb;
CREATE TABLE stu_class_tb (
   id NUMBER GENERATED BY DEFAULT AS IDENTITY START WITH 1 INCREMENT
BY 1,
   stu_id NUMBER NOT NULL,
   class_id NUMBER NOT NULL,
   CONSTRAINT pk_stu_class PRIMARY KEY (id),
   CONSTRAINT fk_stu_id FOREIGN KEY (stu_id) REFERENCES stu_tb(id) ON DELETE
CASCADE,
   CONSTRAINT fk_class_id FOREIGN KEY (class_id) REFERENCES class_tb(id) ON
DELETE CASCADE
);

-- 创建课程-班级关联表
DROP TABLE cou_class_tb;
CREATE TABLE cou_class_tb (
   id NUMBER GENERATED BY DEFAULT AS IDENTITY START WITH 1 INCREMENT
BY 1,
   class_id NUMBER NOT NULL,
   course_id NUMBER NOT NULL,
   CONSTRAINT pk_cou_class PRIMARY KEY (id),
   CONSTRAINT fk_cou_class_class_id FOREIGN KEY (class_id) REFERENCES
class_tb(id) ON DELETE CASCADE,
   CONSTRAINT fk_cou_class_course_id FOREIGN KEY (course_id) REFERENCES
course_tb(id) ON DELETE CASCADE
);

-- 创建学生-课程-成绩表
DROP TABLE stu_cou_score_tb;
CREATE TABLE stu_cou_score_tb (
   id NUMBER GENERATED BY DEFAULT AS IDENTITY START WITH 1 INCREMENT
BY 1,
   stu_id NUMBER NOT NULL,
   course_id NUMBER NOT NULL,
   daily_score NUMBER CHECK (daily_score >= 0 AND daily_score <= 100),
   exam_score NUMBER CHECK (exam_score >= 0 AND exam_score <= 100),
   CONSTRAINT pk_stu_cou_score PRIMARY KEY (id),
   CONSTRAINT fk_stu_cou_score_stu_id FOREIGN KEY (stu_id) REFERENCES stu_tb(id)
ON DELETE CASCADE,
   CONSTRAINT fk_stu_cou_score_course_id FOREIGN KEY (course_id) REFERENCES
course_tb(id) ON DELETE CASCADE
);
```

```
-- 创建学生-课程-成绩查询视图
CREATE OR REPLACE VIEW stu_cou_score_view AS
SELECT
    scs.stu_id,
    scs.course_id,
    scs.daily_score,
    c.daily_ratio,
    scs.exam_score,
    1 - c.daily_ratio AS exam_ratio,    -- 计算考试成绩占比
    scs.daily_score * c.daily_ratio + scs.exam_score * (1 - c.daily_ratio) AS final_score
-- 按照上述规则计算最终成绩
FROM stu_cou_score_tb scs
JOIN course_tb c ON scs.course_id = c.id;
```