

北京理工大学

本科生实验报告

数据库设计与开发实验四 数据库开发

学 院：	计算机学院
专 业：	软件工程
班 级：	08012301 班
学生姓名：	蒋浩天
学 号：	1120231337
指导教师：	

2025 年 4 月 7 日

目 录

第 1 章 实验任务	1
第 2 章 实验过程	1
2.1 建立没有表之间参照关系的表格	1
2.2 建立视图	3
2.2.1 建立适当的视图, 使可直接单表查询学生的总学分与总成绩	3
2.2.2 建立适当的视图, 将所有的表连接起来	5
2.2.3 建立单表的视图, 通过视图来更新、删除数据	6
2.2.4 建立多表的视图, 通过视图来更新、删除数据	8
2.3 存储过程	11
2.3.1 输入不符合系统要求的数据	11
2.3.2 建立存储过程查找和删除不合法的数据	11
2.3.3 建立存储过程计算学生总学分总成绩, 保存在另一张表中	12
2.3.4 查询总成绩表并进行排序	14
2.4 触发器	15
2.4.1 在表上建立触发器实现主外键功能	15
2.4.2 讨论触发器与主外键的异同	22
2.4.3 在表上建立触发器实现对数据录入修改的限制	23
2.5 讨论视图、存储过程、触发器的使用范围及优缺点	24
2.5.1 视图	24
2.5.2 存储过程	25
2.5.3 触发器	26
第 3 章 实验结论	27
第 4 章 实验体会	27

第 1 章 实验任务

- 建立“学籍与成绩管理系统”表格：
 - 不建立表之间的参照关系
 - 输入数据，以便在表上进行各种操作
- 视图：
 - 计算学生的总学分、总成绩：
 - 建立适当的视图，使得可以直接单表查询就可以知道学生的总学分、总成绩；
 - 建立适当的视图，将所有的表连接起来，观察数据，体会建立多个表的好处；
 - 建立单表的视图，练习通过视图来更新、删除数据；
 - 建立多表的视图，练习通过视图来更新、删除数据；
- 存储过程：
 - 在“学籍与成绩管理系统”表格中输入不符合系统要求的数据（如学生学籍表中学号重复），建立适当的存储过程，分别查找和删除这些不合法的数据；
 - 建立适当的存储过程，计算学生的总学分、总成绩，并保存在另外一张表中；
 - 查询总成绩表，并进行排序。
- 触发器
 - 在相关的表上建立触发器，实现主外键的功能；
 - 讨论触发器与主外键的异同；
 - 在表上建立触发器实现对数据录入、修改的限制
 - 如出生日期不能大于今天、性别只能为“男、女”等
 - 题目自拟
- 讨论视图、存储过程、触发器的使用范围及优缺点

第 2 章 实验过程

2.1 建立没有表之间参照关系的表格

参照实验二的数据库设计，但去除所有的外键约束，主键和索引，通过以下 SQL 语句建立表格。

```
CREATE TABLE IF NOT EXISTS xyb (  
    ydh CHAR(2) NOT NULL,  
    ymc CHAR(30) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS xs (  
    xm CHAR(8) NOT NULL,  
    xh CHAR(10) NOT NULL,  
    ydh CHAR(2),  
    bj CHAR(8),  
    chrq DATE,  
    xb CHAR(2)  
);
```

```
CREATE TABLE IF NOT EXISTS js (  
    xm CHAR(8) NOT NULL,  
    jsbh CHAR(10) NOT NULL,  
    zc CHAR(6),  
    ydh CHAR(2)  
);
```

```
CREATE TABLE IF NOT EXISTS kc (  
    kcbh CHAR(3) NOT NULL,  
    kc CHAR(20) NOT NULL,  
    lx CHAR(10),  
    xf NUMERIC(5, 1)  
);
```

```
CREATE TABLE IF NOT EXISTS sk (  
    kcbh CHAR(3),  
    bh CHAR(10)  
);
```

```
CREATE TABLE IF NOT EXISTS xk (  
    xh CHAR(10),  
    kcbh CHAR(3),  
    jsbh CHAR(10),  
    cj NUMERIC(5, 1)  
);
```

删除原数据表, 创建新数据表后, 重新插入数据.

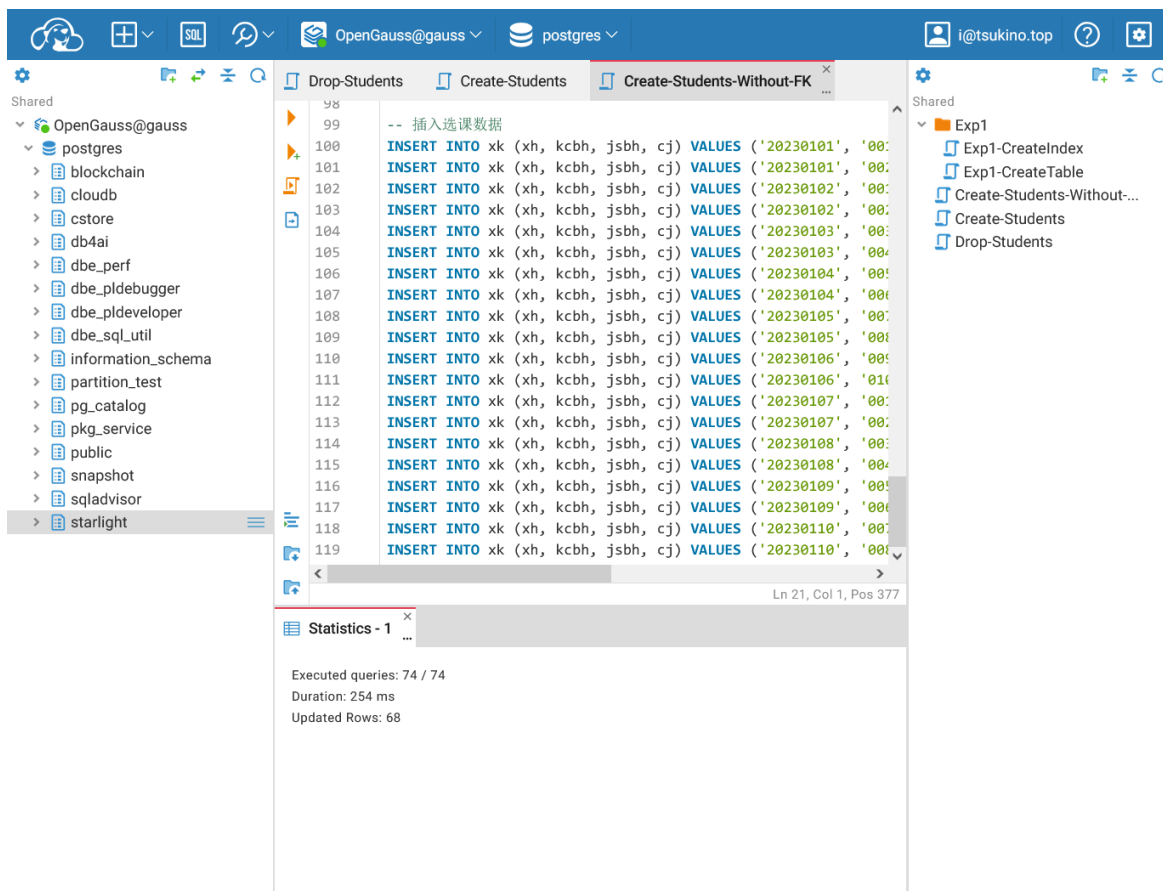


图 2-1 创建没有表之间参照关系的表格

2.2 建立视图

2.2.1 建立适当的视图, 使可直接单表查询学生的总学分与总成绩

链接逻辑: `xs` (学生表) 左联接 `xk` (选课表) → 包含未选课的学生; `xk` 左联接 `kc` (课程表) → 确保查询到对应课程学分。

总成绩的计算: 分子 `SUM(xk.cj * kc.xf)` 为所有选课成绩 × 课程学分之和。分母 `SUM(kc.xf)` 为所有选课的总学分。若总学分为 0, 直接返回 0 (避免除以零错误)。

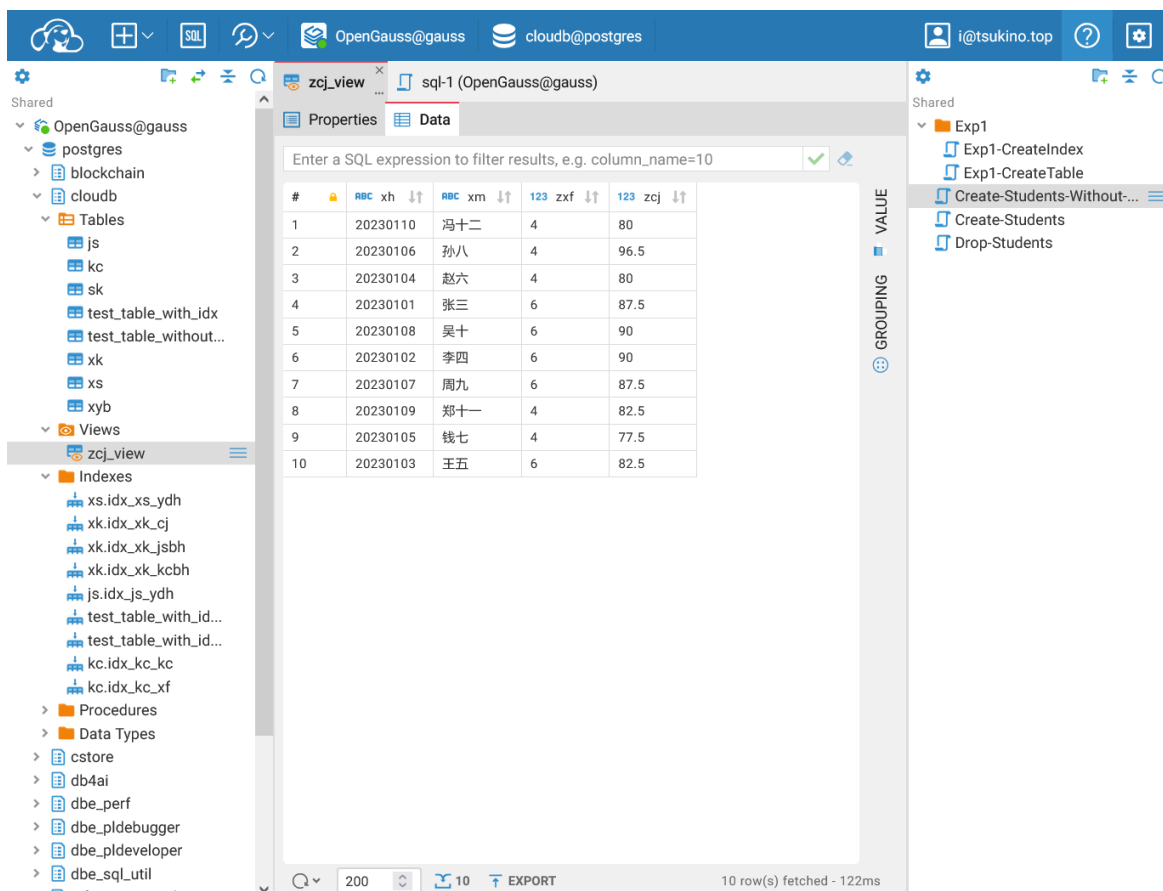
分组与聚合: 按学号 (`xs.xh`) 和姓名 (`xs.xm`) 分组, 确保结果按学生聚合。

```
DROP VIEW IF EXISTS zcj_view; -- 删除视图 (如果存在)
CREATE VIEW zcj_view AS
SELECT
    xs.xh AS xh, -- 学号
```

```

xs.xm AS xm,      -- 姓名
COALESCE(SUM(kc.xf), 0) AS zxf,    -- 总学分
CASE
    WHEN COALESCE(SUM(kc.xf), 0) = 0 THEN 0    -- 总学分0则返回0（避免除0）
    ELSE (SUM(xk.cj * kc.xf) / SUM(kc.xf))    -- 加权平均分
END AS zcj        -- 加权平均成绩
FROM xs
LEFT JOIN xk ON xs.xh = xk.xh    -- 左联接选课表（包含未选课的学生）
LEFT JOIN kc ON xk.kcbh = kc.kcbh -- 左联接课程表（获取学分信息）
GROUP BY xs.xh, xs.xm;
    
```

成功创建视图之后,可以看到视图中的所有数据. 视图中包含了所有学生的学号、姓名、总学分和加权平均成绩. 其中, 学号和姓名来自 `xs` 表格, 总学分和加权平均成绩是通过联接 `xk` 和 `kc` 表格计算得出的. 视图中的数据是动态的, 当基础表格中的数据发生变化时, 视图中的数据也会随之更新.



#	xh	xm	zxf	zcj
1	20230110	冯十二	4	80
2	20230106	孙八	4	96.5
3	20230104	赵六	4	80
4	20230101	张三	6	87.5
5	20230108	吴十	6	90
6	20230102	李四	6	90
7	20230107	周九	6	87.5
8	20230109	郑十一	4	82.5
9	20230105	钱七	4	77.5
10	20230103	王五	6	82.5

图 2-2 查看视图中的数据

2.2.2 建立适当的视图，将所有的表连接起来

假设以同学的选课表为主体，所有的表连接起来的结果为：（学生姓名，学生学号，学生学院名称，学生学院代号，班级，出生日期，性别，课程，课程编号，授课教师姓名，教师编号，教师学院名称，教师学院代号，教师职称，课程类型，课程学分，学生成绩）。

表 2-1 关键连接逻辑

连接表	连接条件	作用
xs → xyb_1	xs.ydh = xyb_1.ydh	获取学生所属院系名称（ymc）
xs → xk	xs.xh = xk.xh	通过学生学号关联其选课记录
xk → kc	xk.kcbh = kc.kcbh	获取课程的基本信息（名称、类型、学分等）
xk → js	xk.jsbh = js.jsbh	获取授课教师的信息（姓名、职称等）
js → xyb_2	js.ydh = xyb_2.ydh	获取教师所属的院系名称

创建视图的 SQL 语句如下：

```
DROP VIEW IF EXISTS all_info_view;
CREATE VIEW all_info_view AS (
  SELECT
    xs.xm AS xs_xm,
    xs.xh AS xs_xh,
    xyb_1.ymc AS xs_ymc,
    xs.ydh AS xs_ydh,
    xs.bj,
    xs.chrq,
    xs.xb,
    kc.kc,
    kc.kcbh,
    js.xm AS js_xm,
    js.jsbh,
    xyb_2.ymc AS js_ymc,
    js.ydh AS js_ydh,
    js.zc,
    kc.lx,
    kc.xf,
```

```
xk.cj
FROM (
  xs JOIN xyb AS xyb_1 ON (xs.ydh = xyb_1.ydh)
  JOIN xk ON (xs.xh = xk.xh)
  JOIN kc ON (xk.kcbh = kc.kcbh)
  JOIN js ON (js.jsbh = xk.jsbh)
  JOIN xyb AS xyb_2 ON (js.ydh = xyb_2.ydh)
)
ORDER BY (xs_xm, js_xm)
);
```

The screenshot shows a database management interface with a view named 'all_info_view' selected. The view displays 20 rows of student data. The columns are: #, abc, xs_xm, xs_xh, xs_ymc, xs_ydh, and bj. The data is as follows:

#	abc	xs_xm	xs_xh	xs_ymc	xs_ydh	bj
1		冯十二	20230110	机械工程学院	04	机械工
2		冯十二	20230110	机械工程学院	04	机械工
3		吴十	20230108	电子信息学院	02	电子信
4		吴十	20230108	电子信息学院	02	电子信
5		周九	20230107	计算机学院	01	软件工
6		周九	20230107	计算机学院	01	软件工
7		孙八	20230106	土木工程学院	05	土木工
8		孙八	20230106	土木工程学院	05	土木工
9		张三	20230101	计算机学院	01	软件工
10		张三	20230101	计算机学院	01	软件工
11		李四	20230102	计算机学院	01	软件工
12		李四	20230102	计算机学院	01	软件工
13		王五	20230103	电子信息学院	02	电子信
14		王五	20230103	电子信息学院	02	电子信
15		赵六	20230104	自动化学院	03	自动化
16		赵六	20230104	自动化学院	03	自动化
17		郑十一	20230109	自动化学院	03	自动化
18		郑十一	20230109	自动化学院	03	自动化
19		钱七	20230105	机械工程学院	04	机械工
20		钱七	20230105	机械工程学院	04	机械工

图 2-3 成功创建连接全部表的视图

2.2.3 建立单表的视图，通过视图来更新、删除数据

假设需要查询所有选课记录的成绩,将低于 90 分的成绩都加上 5 分,之后删除所有小于等于 80 分的选课记录.

由于我们只关心选课表中的成绩,因此我们可以创建一个单表视图,该视图只包含成绩,学生学号,课程编号.创建视图的语句如下:


```
CREATE OR REPLACE VIEW xk_cj_view("学生学号", "课程编号", "课程成绩") AS (
    SELECT xh, kcbh, cj FROM xk
);
```

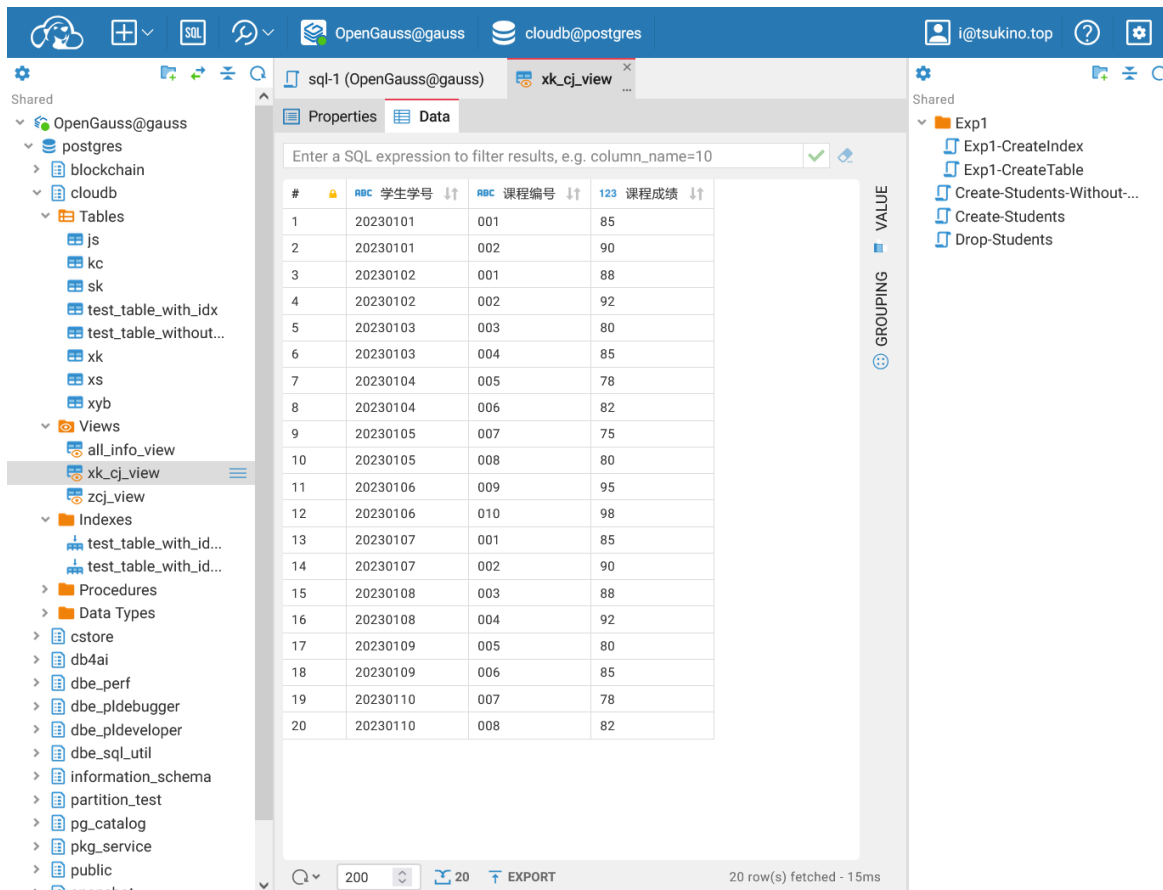


图 2-4 成功创建单表视图

由于 OpenGauss 中数据库视图默认为只读模式，默认不支持更新和删除操作，因此要想使用视图来更新、删除数据，必须通过 Rule 实现。

因此为 `xk_cj_view` 视图添加 Rule, 使其支持更新和删除操作。

```
CREATE OR REPLACE RULE xk_cj_view_update AS ON UPDATE TO xk_cj_view DO INSTEAD (
    UPDATE xk SET cj = NEW."课程成绩"
    WHERE xh = NEW."学生学号" AND kcbh = NEW."课程编号"
);
CREATE OR REPLACE RULE xk_cj_view_delete AS ON DELETE TO xk_cj_view DO INSTEAD (
    DELETE FROM xk
```

```
WHERE xh = OLD."学生学号" AND kcbh = OLD."课程编号"  
);
```

创建 Rules 成功后,我们可以直接使用 SQL 语句对视图进行更新和删除操作.

```
UPDATE xk_cj_view SET "课程成绩" = "课程成绩" + 5 WHERE "课程成绩" < 90;  
DELETE FROM xk_cj_view WHERE "课程成绩" ≤ 80;
```

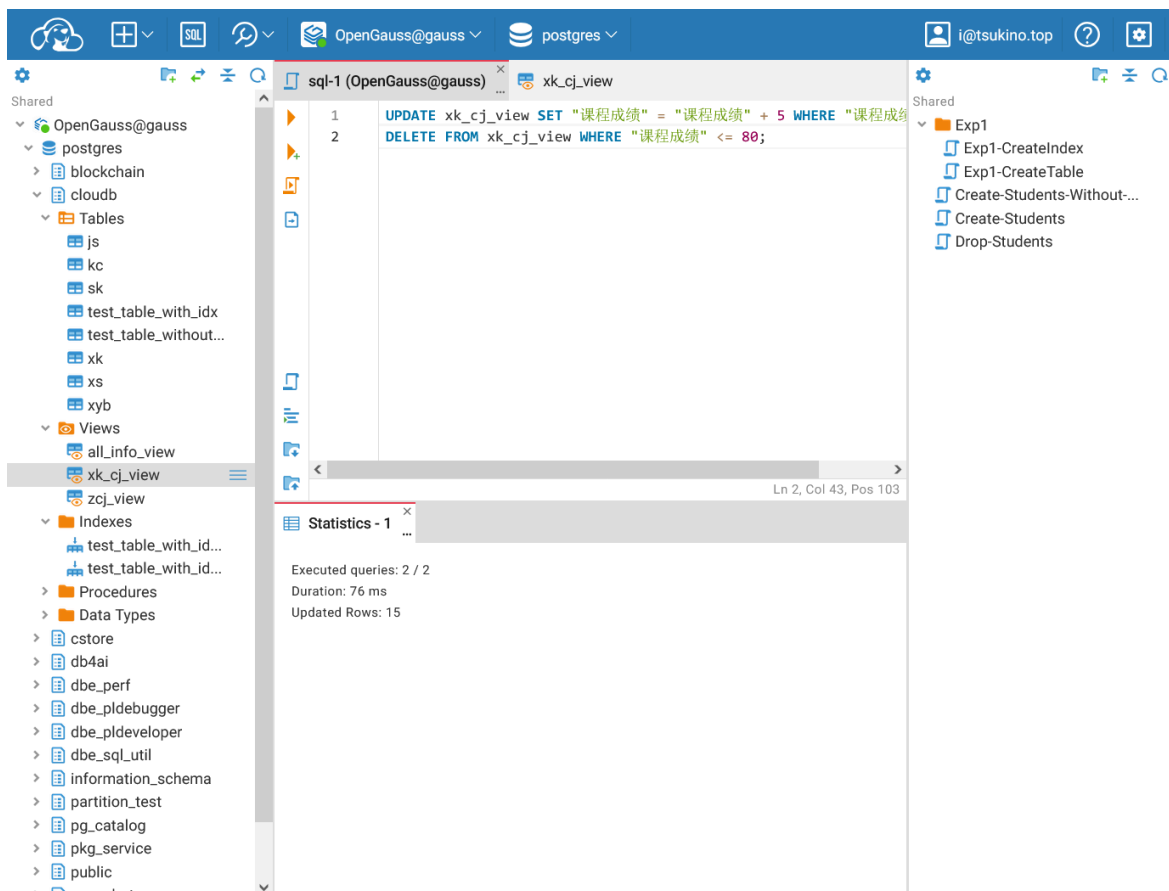


图 2-5 成功更新视图中的数据

可以看到,数据库返回更新和删除成功的信息. 查询数据库也可以确定,数据表中的数据已经被更新或删除.

2.2.4 建立多表的视图,通过视图来更新、删除数据

使用 **SELECT** 和 **WHERE** 语句可以同时多个表中查询信息,从而建立多表视图,比如每位同学的姓名,学号,选择课程的编号和对应成绩。

例如,我们可以建立一个视图,包含学生的姓名、学号、课程编号和对应成绩。

数据库设计与开发实验四 数据库开发

```
CREATE OR REPLACE VIEW xs_cj_view("姓名", "学号", "课程编号", "对应成绩") AS
SELECT xs.xm, xs.xh, xk.kcbh, xk.cj
FROM xs
JOIN xk ON xs.xh = xk.xh;
```

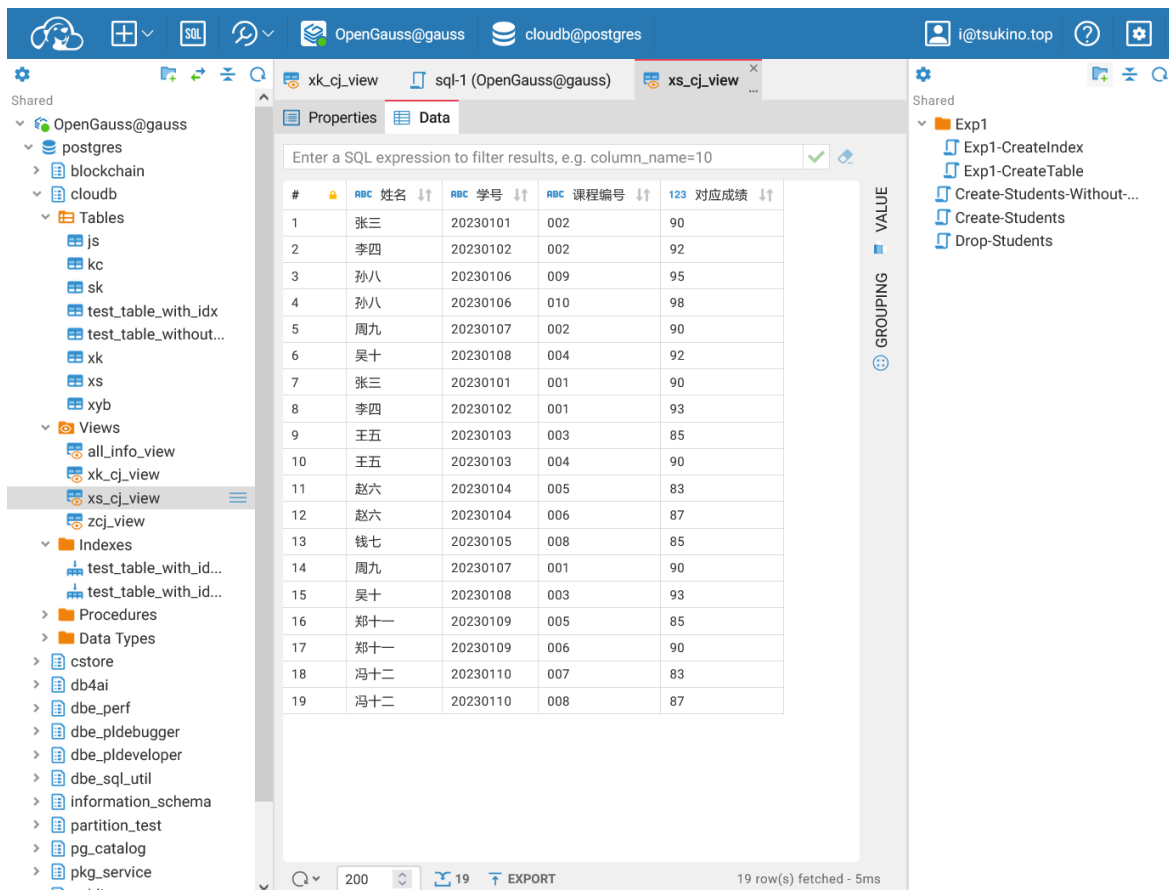


图 2-6 成功创建多表视图

可以看到视图已经创建成功, 通过 **SELECT** 和 **WHERE** 语句可以查询视图中的数据.

假设现在需要查询所有选课记录的成绩, 将低于 80 分的成绩设置为 **NULL**, 并为所有与这些记录相关的学生姓名添加后缀 **-非全优良**.

同样因为 OpenGauss 中数据库视图默认为只读模式, 默认不支持更新和删除操作, 因此必须通过 Rule 实现。

```
CREATE OR REPLACE RULE xs_cj_view_update AS ON UPDATE TO xs_cj_view DO INSTEAD
(
    UPDATE xk SET cj = NULL
    WHERE xh = OLD."学号" AND kcbh = OLD."课程编号" AND cj < 80;
    UPDATE xs SET xm = xm || '-非全优良'
```

数据库设计与开发实验四 数据库开发

```
WHERE xh = OLD."学号";  
);
```

创建 Rules 成功后,我们可以直接使用 SQL 语句对视图进行更新和删除操作.

```
UPDATE xs_cj_view SET "对应成绩" = NULL WHERE "对应成绩" < 80;  
UPDATE xs_cj_view SET "姓名" = "姓名" || '-非全优良' WHERE "对应成绩" IS NULL;
```

#	姓名	学号	课程编号	对应成绩
1	张三	20230101	001	85
2	张三	20230101	002	90
3	李四	20230102	001	88
4	李四	20230102	002	92
5	王五	20230103	003	80
6	王五	20230103	004	85
7	赵六-非全优良	20230104	006	82
8	钱七-非全优良	20230105	008	80
9	孙八	20230106	009	95
10	孙八	20230106	010	98
11	周九	20230107	001	85
12	周九	20230107	002	90
13	吴十	20230108	003	88
14	吴十	20230108	004	92
15	郑十一	20230109	005	80
16	郑十一	20230109	006	85
17	冯十二-非全优良	20230110	008	82
18	赵六-非全优良	20230104	005	[NULL]
19	钱七-非全优良	20230105	007	[NULL]
20	冯十二-非全优良	20230110	007	[NULL]

图 2-7 成功更新视图中的数据

查询数据库可以确定,数据表中的数据已经被更新,部分成绩被设置为 **NULL**,对应的学生姓名也添加了后缀 **-非全优良**.这说明我们成功地使用视图对数据进行了更新和删除操作.

2.3 存储过程

2.3.1 输入不符合系统要求的数据

首先删除原有的表,重建没有索引和主键外键约束的表.根据系统设计要求,向表中插入一些不符合要求的数据,例如学生表中有重复的学号.

```
INSERT INTO xs (xm, xh, ydh, bj, chrq, xb) VALUES ('吕十三', '20230111', '04',  
'机械工程', '2003-10-01', '女');  
INSERT INTO xs (xm, xh, ydh, bj, chrq, xb) VALUES ('艾十四', '20230111', '04',  
'机械工程', '2003-10-08', '男');
```

执行 SQL 语句后,由于我们的数据表中没有唯一约束等,因此可以成功插入重复的学号.

2.3.2 建立存储过程查找和删除不合法的数据

为了删除不合法的数据,我们可以创建一个存储过程,该存储过程会删除所有重复的学号.该存储过程的代码如下:

```
CREATE OR REPLACE PROCEDURE delete_duplicate() AS  
BEGIN  
    DELETE FROM xs WHERE xh IN (  
        SELECT xh FROM xs GROUP BY xh HAVING COUNT(*) > 1  
    );  
END;
```

成功创建存储过程之后,我们可以使用 `CALL delete_duplicate();` 来调用该存储过程,删除所有重复的学号.

但是需要注意,这一存储过程会删除所有重复的学号,而不是只删除其中一个.如果需要只保留一个,最好在创建表时就添加唯一约束 `UNIQUE`.

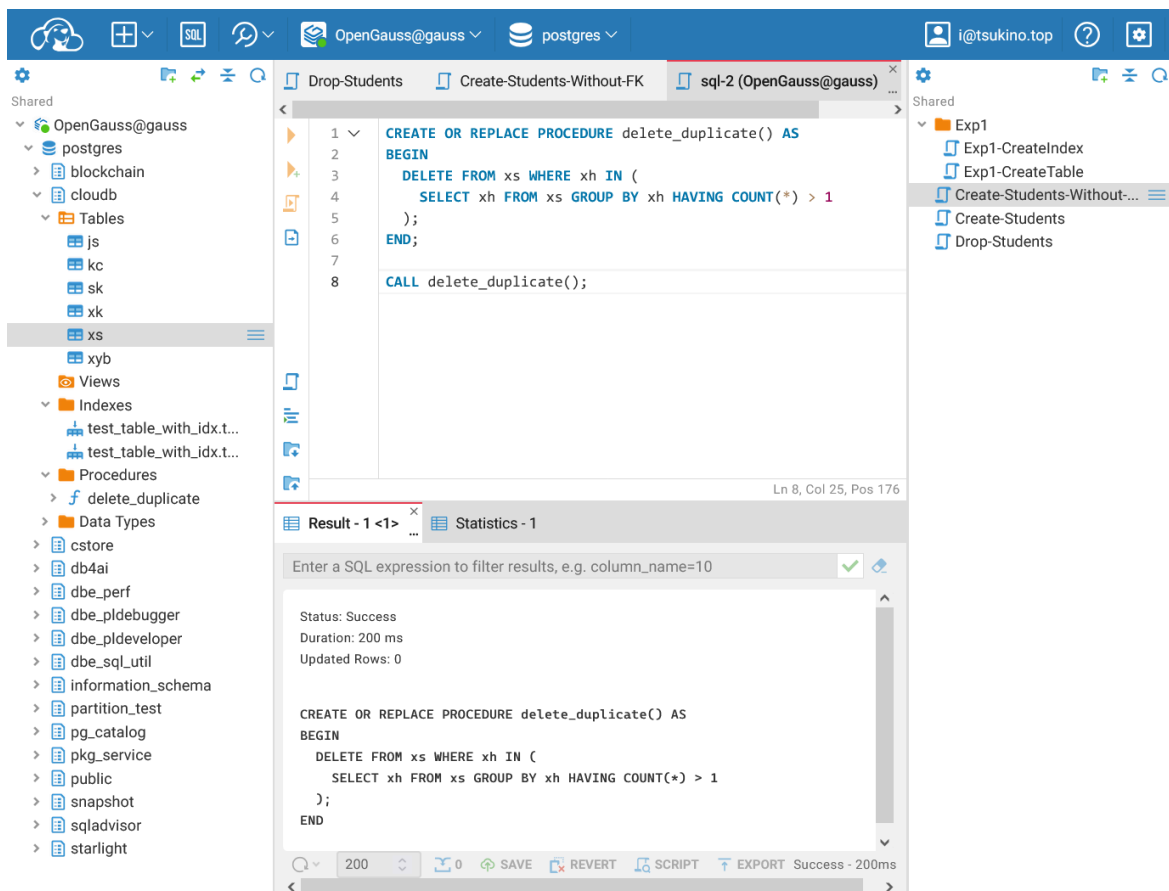


图 2-8 成功删除重复的学号

2.3.3 建立存储过程计算学生总学分总成绩，保存在另一张表中

通过关联以下三张表，获取计算所需的字段：

- **xk**（选课表）：核心表，包含学生的选课记录和成绩。
- **kc**（课程表）：提供课程对应的学分信息。
- **xs**（学生表）：提供学生的姓名信息。

通过 **INNER JOIN** 操作，按以下条件关联表：

- **xk.kcbh = kc.kcbh**：通过课程编号关联选课表和课程表，获取课程学分。
- **xk.xh = xs.xh**：通过学号关联选课表和学生表，确保每个成绩对应的学生姓名。

- 总学分（**zxf**）：通过 **SUM(kc.xf)** 汇总学生所有课程的学分总和。
- 加权平均成绩（**zcyj**）：
 1. 计算 加权总分：各课程成绩（**xk.cj**）× 对应学分（**kc.xf**）的总和。
 2. 除以总学分（**SUM(kc.xf)**）得到加权平均值。

3. 通过 `NULLIF` 避免除以零错误（当总学分为 0 时返回 `NULL`）。
 4. 使用 `ROUND(..., 1)` 格式化结果，保留一位小数。
- 由于同一学号 `xh` 对应唯一姓名，但需满足数据库的聚合规则，使用 `MAX(xs.xm)` 从多个相同记录中选择姓名。

```
CREATE TABLE IF NOT EXISTS student_score(  
    xh CHAR(10) NOT NULL PRIMARY KEY,  
    xm CHAR(8) NOT NULL,  
    zxf DECIMAL(5,1),  
    zcj DECIMAL(5,1)  
);  
  
CREATE OR REPLACE PROCEDURE calc_total_score() AS  
BEGIN  
    -- 清空表格  
    TRUNCATE TABLE student_score;  
  
    INSERT INTO student_score (xm, xh, zxf, zcj)  
    SELECT  
        MAX(xs.xm) AS xm,  
        xk.xh,  
        SUM(kc.xf) AS zxf,  
        ROUND(  
            (SUM(xk.cj * kc.xf) /  
             NULLIF(SUM(kc.xf), 0::NUMERIC))  
            , 1  
        ) AS zcj  
    FROM  
        xk  
    INNER JOIN kc ON xk.kcbh = kc.kcbh  
    INNER JOIN xs ON xk.xh = xs.xh  
    WHERE  
        xk.cj IS NOT NULL  
    GROUP BY  
        xk.xh;  
END;
```

```
-- 执行存储过程
CALL calc_total_score();
```

#	ABC xh	ABC xm	123 zxf	123 zcj
1	20230107	周九	6	87.5
2	20230101	张三	6	87.5
3	20230103	王五	6	82.5
4	20230105	钱七	4	77.5
5	20230104	赵六	4	80
6	20230110	冯十二	4	80
7	20230108	吴十	6	90
8	20230102	李四	6	90
9	20230106	孙八	4	96.5
10	20230109	郑十一	4	82.5

图 2-9 成功计算学生总学分和总成绩

从结果中可以看到, 通过存储过程成功计算了学生的总学分和加权平均成绩, 并将结果存储在了 `student_score` 表中.

2.3.4 查询总成绩表并进行排序

由于我们已经将学生的总学分和加权平均成绩 `student_score` 表, 因此可以直接使用 `SELECT` 语句查询该表. 通过 `ORDER BY` 子句对总学分和总成绩进行排序.

```
SELECT * FROM student_score ORDER BY zcj DESC;
```

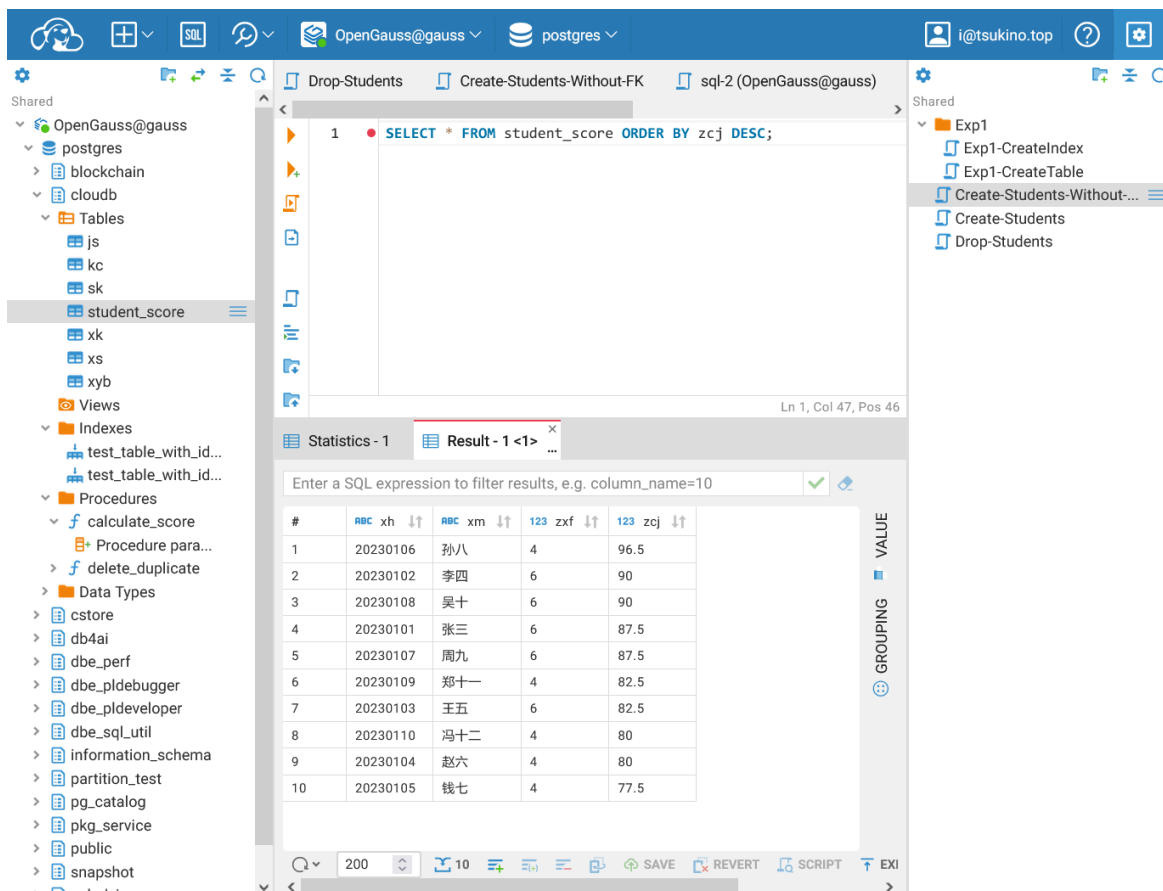



图 2-10 查询总学分和总成绩表

可以看到, 通过 `ORDER BY` 子句, 我们可以对总学分和总成绩进行排序. 通过 `DESC` 可以实现降序排列, 通过 `ASC` 可以实现升序排列.

2.4 触发器

2.4.1 在表上建立触发器实现主外键功能

由于 OpenGauss 的限制, 触发器的主体只能是对函数或者存储过程的调用, 因此我们将触发器的主体剥离为函数, 包含对主键和外键的检查逻辑。

```
-- 通用检查函数
CREATE OR REPLACE FUNCTION check_not_null(value anyelement, field_name text)
RETURNS void AS $$
BEGIN
    IF value IS NULL THEN
        RAISE EXCEPTION '%s 不能为空', field_name;
    END IF;
END;
```

```
END;
$$ LANGUAGE plpgsql;

-- 检查单字段唯一性
CREATE OR REPLACE FUNCTION check_unique_field(
    table_name text,
    field_name text,
    field_value text,
    OUT found boolean
) RETURNS void AS $$
BEGIN
    EXECUTE format('SELECT EXISTS(SELECT 1 FROM %I WHERE %I = $1)', table_name,
field_name)
    INTO found
    USING field_value;

    IF found THEN
        RAISE EXCEPTION '%s 已存在', field_value;
    END IF;
END;
$$ LANGUAGE plpgsql;

-- 检查外键引用
CREATE OR REPLACE FUNCTION check_foreign_key(
    parent_table text,
    parent_field text,
    field_value anyelement,
    error_message text
) RETURNS void AS $$
DECLARE
    query text;
    exists_check boolean;
BEGIN
    query := format('SELECT EXISTS(SELECT 1 FROM %I WHERE %I = $1)',
parent_table, parent_field);
    EXECUTE query INTO exists_check USING field_value;

    IF field_value IS NOT NULL AND NOT exists_check THEN
        RAISE EXCEPTION '%s', error_message;
```

```
END IF;
END;
$$ LANGUAGE plpgsql;

-- xyb 表触发器函数
CREATE OR REPLACE FUNCTION xyb_pkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    PERFORM check_not_null(NEW.ydh, 'ydh');
    PERFORM check_unique_field('xyb', 'ydh', NEW.ydh);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- xs 表触发器函数
CREATE OR REPLACE FUNCTION xs_pkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    PERFORM check_not_null(NEW.xh, '学号');
    PERFORM check_unique_field('xs', 'xh', NEW.xh);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION xs_fkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    PERFORM check_foreign_key('xyb', 'ydh', NEW.ydh, '所属学院代号 (ydh) 在 xyb 表中不存在');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- js 表触发器函数
CREATE OR REPLACE FUNCTION js_pkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    PERFORM check_not_null(NEW.jsbh, 'jsbh');
```

```
PERFORM check_unique_field('js', 'jsbh', NEW.jsbh);
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- kc 表触发器函数
CREATE OR REPLACE FUNCTION kc_pkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    PERFORM check_not_null(NEW.kcbh, 'kcbh');
    PERFORM check_unique_field('kc', 'kcbh', NEW.kcbh);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- sk 表触发器函数
CREATE OR REPLACE FUNCTION sk_pkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.kcbh IS NULL OR NEW.bh IS NULL THEN
        RAISE EXCEPTION 'kcbh 和 bh 不能为空';
    END IF;

    IF EXISTS(SELECT 1 FROM sk WHERE kcbh = NEW.kcbh AND bh = NEW.bh) THEN
        RAISE EXCEPTION 'kcbh 和 bh 的组合已存在';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION sk_fkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    PERFORM check_foreign_key('kc', 'kcbh', NEW.kcbh, '课程编号(kcbh)在 kc 表中不存在');
    PERFORM check_foreign_key('js', 'jsbh', NEW.bh, '教师编号(bh)在 js 表中不存在');
    RETURN NEW;
```

```
END;
$$ LANGUAGE plpgsql;

-- xk 表触发器函数
CREATE OR REPLACE FUNCTION xk_pkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.xh IS NULL OR NEW.kcbh IS NULL OR NEW.jsbh IS NULL THEN
        RAISE EXCEPTION '学号、课程编号、教师编号均不能为 NULL';
    END IF;

    IF EXISTS(
        SELECT 1 FROM xk
        WHERE xh = NEW.xh AND kcbh = NEW.kcbh AND jsbh = NEW.jsbh
    ) THEN
        RAISE EXCEPTION '选课记录已存在';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION xk_fkey_trigger_func()
RETURNS TRIGGER AS $$
BEGIN
    IF NOT EXISTS(
        SELECT 1 FROM sk
        WHERE sk.kcbh = NEW.kcbh AND sk.bh = NEW.jsbh
    ) THEN
        RAISE EXCEPTION '(kcbh, jsbh) 组合在 sk 表中不存在';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- 删除原有触发器
DROP TRIGGER IF EXISTS xyb_before_insert_pkey ON xyb;
DROP TRIGGER IF EXISTS xyb_before_update_pkey ON xyb;
```

```
DROP TRIGGER IF EXISTS xs_before_insert_pkey ON xs;
DROP TRIGGER IF EXISTS xs_before_update_pkey ON xs;
DROP TRIGGER IF EXISTS xs_before_insert_fkey ON xs;
DROP TRIGGER IF EXISTS xs_before_update_fkey ON xs;

DROP TRIGGER IF EXISTS js_before_insert_pkey ON js;
DROP TRIGGER IF EXISTS js_before_update_pkey ON js;
DROP TRIGGER IF EXISTS js_before_insert_fkey ON js;
DROP TRIGGER IF EXISTS js_before_update_fkey ON js;

DROP TRIGGER IF EXISTS kc_before_insert_pkey ON kc;
DROP TRIGGER IF EXISTS kc_before_update_pkey ON kc;

DROP TRIGGER IF EXISTS sk_before_insert_pkey ON sk;
DROP TRIGGER IF EXISTS sk_before_update_pkey ON sk;
DROP TRIGGER IF EXISTS sk_before_insert_fkey ON sk;
DROP TRIGGER IF EXISTS sk_before_update_fkey ON sk;

DROP TRIGGER IF EXISTS xk_before_insert_pkey ON xk;
DROP TRIGGER IF EXISTS xk_before_update_pkey ON xk;
DROP TRIGGER IF EXISTS xk_before_insert_fkey ON xk;
DROP TRIGGER IF EXISTS xk_before_update_fkey ON xk;

-- 创建触发器
CREATE TRIGGER xyb_before_insert_pkey BEFORE INSERT ON xyb
    FOR EACH ROW EXECUTE PROCEDURE xyb_pkey_trigger_func();
CREATE TRIGGER xyb_before_update_pkey BEFORE UPDATE ON xyb
    FOR EACH ROW EXECUTE PROCEDURE xyb_pkey_trigger_func();

CREATE TRIGGER xs_before_insert_pkey BEFORE INSERT ON xs
    FOR EACH ROW EXECUTE PROCEDURE xs_pkey_trigger_func();
CREATE TRIGGER xs_before_update_pkey BEFORE UPDATE ON xs
    FOR EACH ROW EXECUTE PROCEDURE xs_pkey_trigger_func();
CREATE TRIGGER xs_before_insert_fkey BEFORE INSERT ON xs
    FOR EACH ROW EXECUTE PROCEDURE xs_fkey_trigger_func();
CREATE TRIGGER xs_before_update_fkey BEFORE UPDATE ON xs
    FOR EACH ROW EXECUTE PROCEDURE xs_fkey_trigger_func();

CREATE TRIGGER js_before_insert_pkey BEFORE INSERT ON js
```

```

FOR EACH ROW EXECUTE PROCEDURE js_pkey_trigger_func();
CREATE TRIGGER js_before_update_pkey BEFORE UPDATE ON js
FOR EACH ROW EXECUTE PROCEDURE js_pkey_trigger_func();
CREATE TRIGGER js_before_insert_fkey BEFORE INSERT ON js
FOR EACH ROW EXECUTE PROCEDURE xs_fkey_trigger_func();
CREATE TRIGGER js_before_update_fkey BEFORE UPDATE ON js
FOR EACH ROW EXECUTE PROCEDURE xs_fkey_trigger_func();

CREATE TRIGGER kc_before_insert_pkey BEFORE INSERT ON kc
FOR EACH ROW EXECUTE PROCEDURE kc_pkey_trigger_func();
CREATE TRIGGER kc_before_update_pkey BEFORE UPDATE ON kc
FOR EACH ROW EXECUTE PROCEDURE kc_pkey_trigger_func();

CREATE TRIGGER sk_before_insert_pkey BEFORE INSERT ON sk
FOR EACH ROW EXECUTE PROCEDURE sk_pkey_trigger_func();
CREATE TRIGGER sk_before_update_pkey BEFORE UPDATE ON sk
FOR EACH ROW EXECUTE PROCEDURE sk_pkey_trigger_func();
CREATE TRIGGER sk_before_insert_fkey BEFORE INSERT ON sk
FOR EACH ROW EXECUTE PROCEDURE sk_fkey_trigger_func();
CREATE TRIGGER sk_before_update_fkey BEFORE UPDATE ON sk
FOR EACH ROW EXECUTE PROCEDURE sk_fkey_trigger_func();

CREATE TRIGGER xk_before_insert_pkey BEFORE INSERT ON xk
FOR EACH ROW EXECUTE PROCEDURE xk_pkey_trigger_func();
CREATE TRIGGER xk_before_update_pkey BEFORE UPDATE ON xk
FOR EACH ROW EXECUTE PROCEDURE xk_pkey_trigger_func();
CREATE TRIGGER xk_before_insert_fkey BEFORE INSERT ON xk
FOR EACH ROW EXECUTE PROCEDURE xk_fkey_trigger_func();
CREATE TRIGGER xk_before_update_fkey BEFORE UPDATE ON xk
FOR EACH ROW EXECUTE PROCEDURE xk_fkey_trigger_func();

```

将上述 SQL 语句保存为 `Create-Trigger.sql` 文件, 然后使用 `gsql` 工具连接到数据库, 执行该 SQL 文件. 可以通过尝试添加不合法数据来验证触发器的创建成功与否. 例如:

```

-- 插入不合法数据
INSERT INTO xyb (ydh, ymc) VALUES ('06', '车辆工程学院'); -- 合法数据
INSERT INTO xyb (ydh, ymc) VALUES ('06', '对外关系学院'); -- 重复的 ydh

```

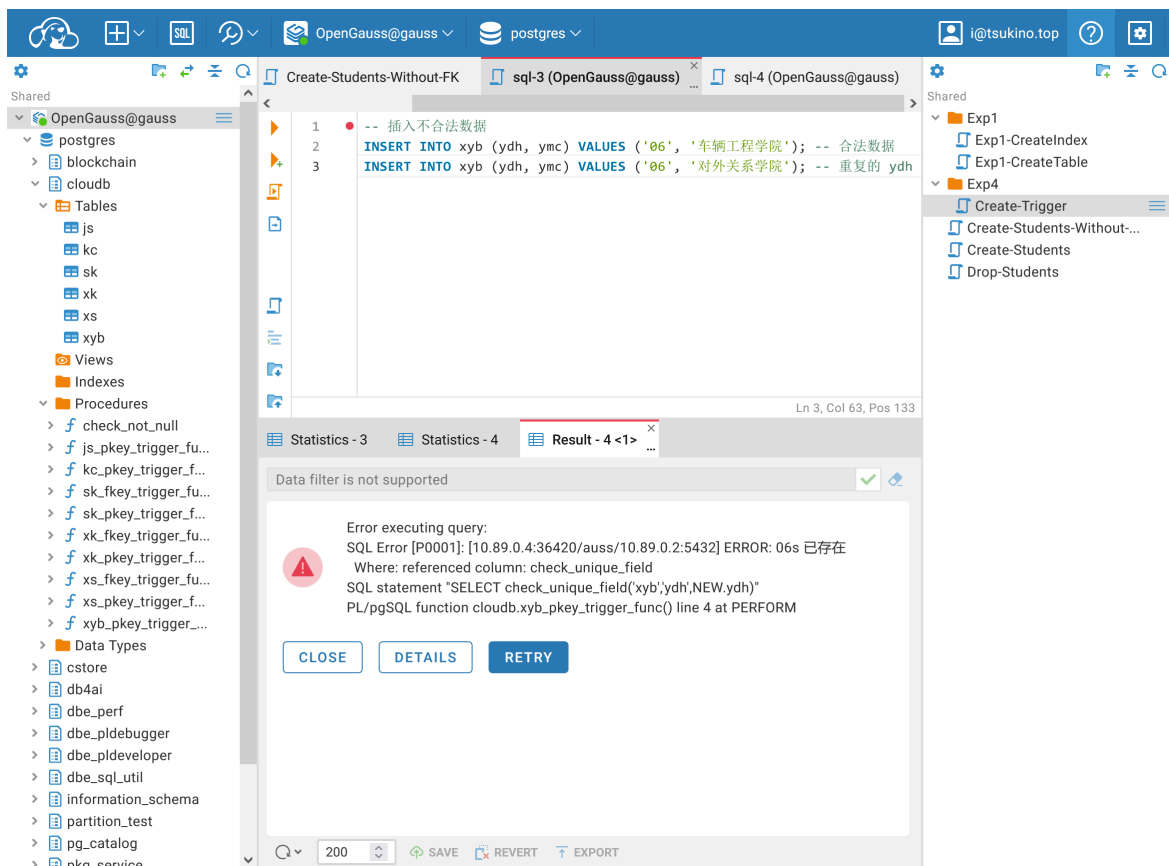


图 2-11 触发器功能验证

可以看见，我们创建的触发器成功地阻止了有相同主键的不合法数据的插入，这表明触发器的创建成功。

2.4.2 讨论触发器与主外键的异同

触发器和主外键约束都是用于维护数据完整性和一致性的机制，但它们在实现方式和应用场景上都有较大区别。

触发器的优点：

1. 灵活性: 触发器可以执行任意复杂的逻辑，不仅限于简单的约束检查。
2. 可定制性: 可以根据具体业务需求自定义错误信息和处理逻辑。
3. 功能扩展: 除了约束检查，还可以实现审计日志、数据同步等功能。

主外键的优点：

1. 性能优势: 数据库系统对主外键约束进行了优化，执行效率更高。
2. 维护简单: 无需编写和维护额外的代码。
3. 标准化: 是数据库标准的一部分，可移植性好。

- 如果只需要简单的引用完整性约束，优先使用主外键约束。
- 当需要复杂的业务逻辑或特殊的错误处理时，选择触发器实现。
- 在某些情况下，可以同时使用两者来获得最佳效果。

2.4.3 在表上建立触发器实现对数据录入修改的限制

假设需要限制学生表中学号的长度为 8 位，学生出生日期不得晚于当前日期，这些数据限制可以通过触发器来实现。

```
CREATE OR REPLACE FUNCTION check_student_data()
RETURNS TRIGGER AS $$
BEGIN
    -- 检查学号长度
    IF LENGTH(NEW.xh) ≠ 8 THEN
        RAISE EXCEPTION '学号长度必须为 8 位';
    END IF;

    -- 检查出生日期
    IF NEW.chrq > CURRENT_DATE THEN
        RAISE EXCEPTION '出生日期不能晚于当前日期';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- 删除旧的触发器
DROP TRIGGER check_student_data_trigger ON xs;

-- 创建触发器
CREATE TRIGGER check_student_data_trigger
BEFORE INSERT OR UPDATE ON xs
FOR EACH ROW
EXECUTE PROCEDURE check_student_data();
```

将上述 SQL 语句保存为 `Create-Student-Trigger.sql` 文件，然后使用 `gsql` 工具连接到数据库，执行该 SQL 文件。

可以通过尝试添加不合法数据来验证触发器的创建成功与否。例如：

-- 插入不合法数据

```
INSERT INTO xs (xh, xm, chrq) VALUES ('12345678', '张三', '2025-08-01'); -- 出生日期晚于当前日期
```

```
INSERT INTO xs (xh, xm, chrq) VALUES ('1234567', '李四', '2000-01-01'); -- 学号长度不合法
```

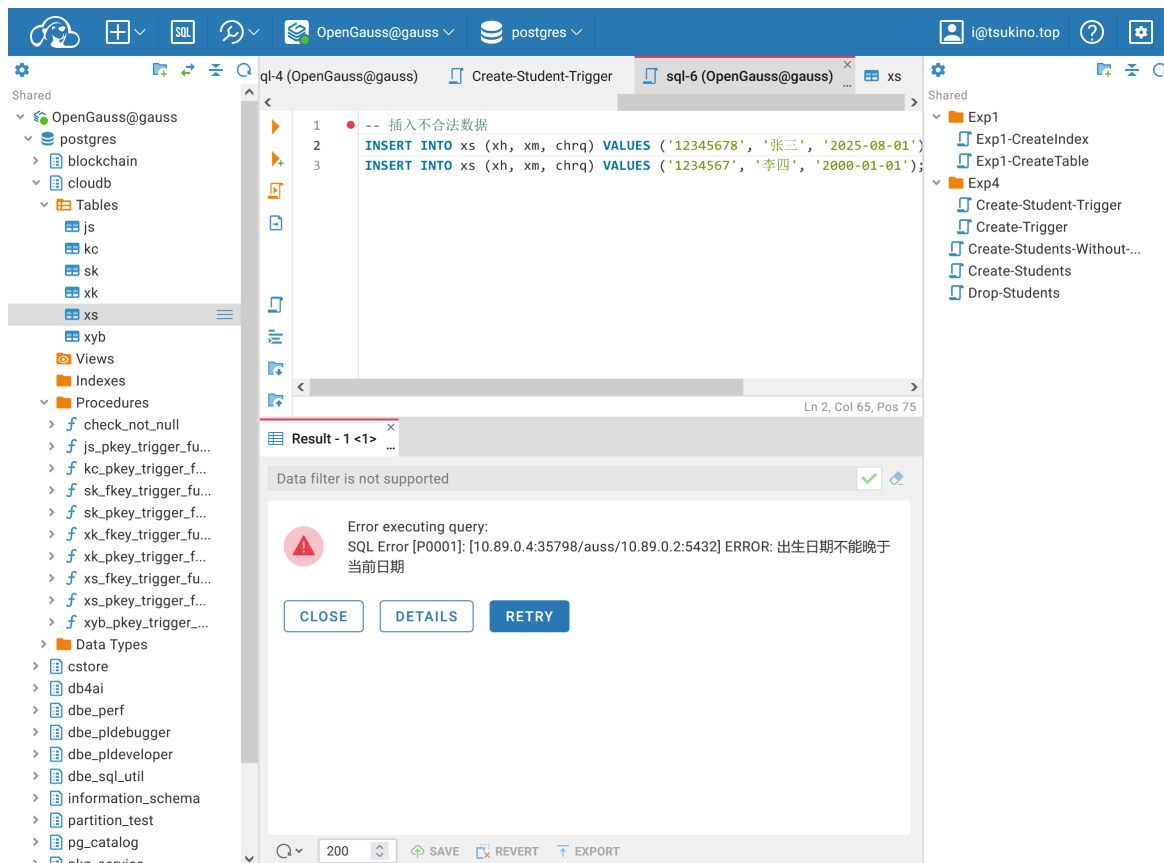


图 2-12 触发器功能验证

2.5 讨论视图、存储过程、触发器的使用范围及优缺点

2.5.1 视图

• 使用范围：

- ▶ 简化复杂查询：将复杂的联合、连接或计算逻辑封装成一个虚拟表，使用户直接查询视图，而无需编写复杂 SQL。
- ▶ 安全控制：通过视图限制用户只能访问特定数据（例如，只允许查看部分字段或特定条件的数据），隐藏底层表结构。

- 统一接口：提供统一的查询入口，即使底层表结构发生变化，也可以通过修改视图保持查询语句不变。
- 逻辑数据独立性：将不同的物理表抽象为逻辑上的视图，适应不同的业务需求。
- 优点：
 - 简化查询：用户直接访问视图时无需理解底层复杂逻辑。
 - 安全性：通过视图控制数据访问权限，隐藏敏感数据。
 - 维护性：修改视图定义即可改变查询逻辑，而无需修改应用程序中的 SQL 语句。
 - 逻辑独立性：底层表结构变化时，可调整视图定义以保持前端应用兼容。
- 缺点：
 - 性能问题：复杂视图可能降低查询速度，尤其是涉及大量计算或连接时。
 - 不可更新：某些复杂视图（如包含聚合函数、`DISTINCT`、`GROUP BY` 等）无法直接更新。
 - 依赖底层表：如果底层表被删除或结构改变，视图可能失效。
 - 存储开销：虽然视图本身不存储数据，但需要存储定义，且频繁使用可能增加系统负担。

2.5.2 存储过程

- 使用范围：
 - 复杂业务逻辑：执行多步骤的业务流程（如订单处理、事务操作），减少客户端与服务器的交互。
 - 提高性能：预编译执行计划，减少网络传输和 SQL 解析时间，尤其适合重复性操作。
 - 安全性：通过权限控制存储过程的调用，避免直接暴露底层表结构。
 - 代码重用：将常用逻辑封装为存储过程，供多个应用程序调用。
- 优点：
 - 性能优化：预编译 SQL 语句，减少解析和编译时间。
 - 减少网络流量：客户端只需调用存储过程名，避免传输大量 SQL 语句。
 - 事务控制：可在存储过程中直接管理事务（如 `BEGIN TRANSACTION`），确保数据一致性。
 - 代码重用和集中管理：逻辑集中于存储过程，便于统一维护。
 - 安全性：可通过角色或权限限制存储过程的执行权限。

- **缺点：**

- 可移植性差：不同数据库的存储过程语法和功能差异较大，切换数据库可能需要重构。
- 调试难度：执行流程在数据库层，调试复杂存储过程较为困难。
- 开发维护成本：编写和维护复杂存储过程需要较高技能，尤其是涉及多步骤事务或错误处理时。
- 代码重用局限：存储过程与具体数据库紧密绑定，难以跨平台复用。

2.5.3 触发器

- **使用范围：**

- 数据完整性约束：自动执行检查约束（如更新时间戳、确保外键存在性）。
- 审计日志：自动记录对表的修改操作（如插入、更新、删除）。
- 级联操作：当某一操作发生时，自动执行相关操作（如删除主表记录时同步删除从表记录）。
- 业务规则强制：如余额不足时阻止资金转账操作。

- **优点：**

- 自动化：无需客户端干预，自动执行业务逻辑。
- 实时性：与数据修改操作紧密绑定，能即时响应，确保规则的强约束。
- 集中管理：将关键业务规则放在数据库层，避免在多个应用中重复实现。
- 简化客户端代码：客户端只需执行基本操作，无需处理复杂的业务规则判断。

- **缺点：**

- 性能开销：触发器会增加数据操作的执行时间，尤其在频繁操作或复杂逻辑时。
- 调试复杂度：触发器问题可能导致难以察觉的错误（如死锁、逻辑冲突）。
- 副作用风险：不当设计可能引发级联问题（如触发器调用其他触发器导致无限循环）。
- 透明性不足：触发器会隐式修改数据，可能导致应用程序开发人员未意识到潜在逻辑。
- 依赖性风险：复杂触发器可能绑定底层表结构，影响系统的灵活性。

第 3 章 实验结论

通过本次实验，我对数据库视图和存储过程的实际应用有了深入理解：

1. 视图的应用场景：

- 单表视图适合简化查询操作，如计算学生的总学分和总成绩
- 多表连接视图可以整合分散在不同表中的相关数据，便于综合分析
- 视图可以用于数据更新操作，但存在一定限制条件

2. 存储过程的功能：

- 可以实现复杂的业务逻辑处理，如查找和删除不合法数据
- 能够进行批量数据计算并存储结果，如计算学生总学分和总成绩
- 执行效率高，适合频繁执行的操作

3. 数据完整性：

- 通过存储过程可以检测和处理违反约束的数据
- 在没有表之间参照关系的情况下，需要额外的程序逻辑来维护数据一致性

4. 数据分析：

- 通过存储过程生成的聚合数据可以支持后续的排序和分析操作
- 视图和存储过程共同为数据分析提供了灵活且高效的方式

5. 触发器的应用：

- 触发器可以有效实现数据完整性约束，包括主键和外键约束
- 通过触发器可以自动化数据验证和业务规则的执行
- 触发器能够在数据变更时自动执行相关的业务逻辑

第 4 章 实验体会

本次实验给我带来了丰富的实践经验和深刻的思考：

1. 视图与存储过程的区别与联系：

- 视图更侧重于数据的展示和简化查询，而存储过程更侧重于业务逻辑的封装
- 两者可以结合使用，发挥各自优势，提高数据库应用的效率和可维护性

2. 数据库设计的重要性：

- 合理的表结构设计是高效数据操作的基础

- 虽然可以不建立表之间的参照关系，但这会增加维护数据一致性的难度

3. 数据操作的灵活性:

- OpenGauss 数据库提供了丰富的函数和操作符，使得复杂计算变得简单
- 通过存储过程可以实现更加复杂的数据处理逻辑

4. 实践与理论的结合:

- 理论知识在实际应用中得到验证和深化
- 解决实际问题的过程促进了对数据库概念的理解

5. 数据库开发的价值:

- 熟练掌握视图和存储过程等数据库开发技术，可以显著提高应用系统的性能和可维护性
- 数据库开发不仅是技术实现，更是业务需求与技术解决方案的桥梁

6. 触发器开发的经验:

- 触发器的设计需要考虑性能影响，避免过度使用导致系统性能下降
- 在 OpenGauss 中实现触发器时需要注意其特殊限制，如触发器主体必须是函数调用
- 通过触发器实现数据完整性约束比直接使用数据库约束更灵活，但也增加了维护的复杂度

通过本次实验，我不仅掌握了视图和存储过程的基本操作，更加深了对数据库系统整体架构和应用模式的理解。这些知识和经验将对我未来的数据库应用开发提供宝贵的指导。