# 实验四

#### 姓名：谢宝玛 学号：1120233506

## 实验目的

学习视图，存储过程，触发器

## 实验过程

### **1，**查询学生总学分，总成绩

SQL：

SELECT

xs.xh AS 学号,

xs.xm AS 姓名,

SUM(kc.xf) AS 总学分,

SUM(xk.cj) AS 总成绩

FROM

xs

JOIN

xk ON xs.xh = xk.xh

JOIN

kc ON xk.kcbh = kc.kcbh

GROUP BY

xs.xh, xs.xm;



### **2，**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

SQL：  
CREATE VIEW student\_score\_summary AS

SELECT

xs.xh AS 学号,

xs.xm AS 姓名,

SUM(kc.xf) AS 总学分,

SUM(xk.cj) AS 总成绩

FROM

xs

JOIN

xk ON xs.xh = xk.xh

JOIN

kc ON xk.kcbh = kc.kcbh

GROUP BY

xs.xh, xs.xm;



查询SQL：

SELECT \* FROM student\_score\_summary;

### **3，**建立视图，将所有的表连起来

SQL：  
CREATE VIEW all\_info\_view AS

SELECT

xs.xh AS 学号,

xs.xm AS 学生姓名,

xs.bj AS 班级,

xs.chrq AS 出生日期,

xs.xb AS 性别,

xyb.ymc AS 学院名称,

kc.kcbh AS 课程编号,

kc.kc AS 课程名称,

kc.lx AS 课程类型,

kc.xf AS 学分,

js.jsbh AS 教师编号,

js.xm AS 教师姓名,

js.zc AS 教师职称,

xk.cj AS 成绩

FROM

xs

JOIN

xyb ON xs.ydh = xyb.ydh

JOIN

xk ON xs.xh = xk.xh

JOIN

kc ON xk.kcbh = kc.kcbh

JOIN

js ON xk.jsbh = js.jsbh;



### **4，**建立单表（以学生表为例）的视图

SQL：

CREATE VIEW student\_view AS

SELECT

xs.xh AS 学号,

xs.xm AS 姓名,

xs.bj AS 班级,

xs.chrq AS 出生日期,

xs.xb AS 性别,

xs.ydh AS 学院编号,

xyb.ymc AS 学院名称

FROM

xs

JOIN

xyb ON xs.ydh = xyb.ydh;



### **5，**建立多表（以教师表和学院表为例）的视图

SQL：  
CREATE VIEW teacher\_college\_view AS

SELECT

js.jsbh AS 教师编号,

js.xm AS 教师姓名,

js.zc AS 职称,

js.ydh AS 学院编号,

xyb.ymc AS 学院名称

FROM

js

JOIN

xyb ON js.ydh = xyb.ydh;



### **6，**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

SQL：  
CREATE OR REPLACE PROCEDURE student\_info.clean\_invalid\_students()

AS DECLARE

BEGIN

-- Routine body goes here...

-- 删除学号重复的多余记录，只保留xh、xm组合下第一条

DELETE xs

FROM xs

INNER JOIN (

SELECT xh, MIN(chrq) AS min\_chrq

FROM xs

GROUP BY xh

HAVING COUNT(\*) > 1

) dup

ON xs.xh = dup.xh AND xs.chrq > dup.min\_chrq; -- 假设用出生日期早的为准

-- （如果表有主键自增id字段，则可以根据id删除）

-- 例如：

-- DELETE FROM xs

-- WHERE id NOT IN (

-- SELECT MIN(id)

-- FROM xs

-- GROUP BY xh

-- );

END

;

/

### 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

SQL：

CREATE OR REPLACE PROCEDURE student\_info.calc\_student\_summary()

AS DECLARE

BEGIN

-- Routine body goes here...

-- 清空原有数据，避免重复插入

TRUNCATE TABLE student\_summary;

-- 将统计结果插入到 student\_summary 表

INSERT INTO student\_summary (xh, xm, total\_xf, total\_cj)

SELECT

xs.xh,

xs.xm,

SUM(kc.xf) AS total\_xf,

SUM(xk.cj) AS total\_cj

FROM

xs

JOIN

xk ON xs.xh = xk.xh

JOIN

kc ON xk.kcbh = kc.kcbh

GROUP BY

xs.xh, xs.xm;

END

;

/



### **8，**查询总成绩表，并进行排序

SQL：

SELECT

xh AS 学号,

xm AS 姓名,

total\_xf AS 总学分,

total\_cj AS 总成绩

FROM

student\_summary

ORDER BY

total\_cj DESC;



### 数据库中，触发器和主外键的异同

#### 相同点

##### （1）数据完整性：

触发器和主外键都可以用于维护数据的完整性。主外键通过约束关系来确保数据的引用完整性（Referential Integrity）。触发器可以通过自定义逻辑来检查或强制执行某些业务规则，从而维护数据的完整性。

##### （2）自动执行：

触发器在特定事件（如插入、更新、删除）发生时自动执行。

主外键的约束在数据操作（如插入、更新、删除）时自动生效。

#### 不同点

##### （1）定义和用途：

主外键：

主键（Primary Key）用于唯一标识表中的每一行数据。

外键（Foreign Key）用于建立表与表之间的关联关系，确保数据的一致性。

主外键是数据库设计中的基本约束，用于维护数据的引用完整性。

触发器：

触发器是一段存储在数据库中的程序代码，在特定事件（如插入、更新、删除）发生时自动执行。

触发器可以用于实现复杂的业务逻辑、审计、日志记录等功能。

##### （2）实现方式：

主外键：

主外键是数据库的约束机制，通常通过 CREATE TABLE 或 ALTER TABLE 语句定义。

主外键的约束由数据库引擎自动维护。

触发器：

触发器是数据库中的存储过程，通过 CREATE TRIGGER 语句定义。

触发器的逻辑需要用户手动编写。

##### （3）执行时机：

主外键：

主外键的约束在数据操作（如插入、更新、删除）时立即生效。

触发器：

触发器可以在数据操作之前（BEFORE）或之后（AFTER）执行，具体取决于触发器的定义。

##### （4）灵活性和复杂性：

主外键：

主外键的功能相对固定，主要用于维护数据的唯一性和引用完整性。

触发器：

触发器可以编写复杂的逻辑，灵活性强，但过度使用可能导致性能问题。

##### （5）性能影响：

主外键：

主外键的约束通常由数据库引擎优化，性能影响较小。

触发器：

触发器的执行可能会增加额外的开销，尤其是在复杂的逻辑或高频操作时。

### 建立触发器：

在学生表 xs 上建立的触发器，目的是在插入新数据时，检查 chrq（出生日期）是否大于今天，如果大于今天，则阻止操作并抛出错误：检查xb（性别）是否是’M’或’F’。

SQL：

CREATE OR REPLACE FUNCTION check\_xs\_before\_insert\_or\_update()

RETURNS trigger AS $$

BEGIN

-- 出生日期不能大于今天

IF NEW.chrq > CURRENT\_DATE THEN

RAISE EXCEPTION '出生日期不能大于今天！';

END IF;

-- 性别只能是 M 或 F

IF NEW.xb NOT IN ('M', 'F') THEN

RAISE EXCEPTION '性别必须是 M 或 F！';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_insert\_xs

BEFORE INSERT ON xs

FOR EACH ROW

EXECUTE PROCEDURE check\_xs\_before\_insert\_or\_update();

CREATE TRIGGER before\_update\_xs

BEFORE UPDATE ON xs

FOR EACH ROW

EXECUTE PROCEDURE check\_xs\_before\_insert\_or\_update();

#### 测试：

SQL：

INSERT INTO xs(xh,xm,ydh,bj,chrq,xb) VALUES

('20230007','xbm','02','03','2027-03-12','M');

触发器触发：  


SQL：

INSERT INTO xs(xh,xm,ydh,bj,chrq,xb) VALUES

('20230007','xbm','02','03','2002-03-12','A');

触发器触发：  


### 讨论视图、存储过程、触发器的使用范 围及优缺点

#### （1）视图 (View)

使用范围：

用于封装复杂查询，让用户像查询表一样方便调用

数据权限控制（只展示部分字段或部分数据）

将多表连接结果或者聚合结果作为一个逻辑表

优点：

简化复杂SQL查询

提高可读性和复用性

可以控制用户权限，只暴露部分数据

缺点：

视图本身不存储数据（除非是物化视图）

复杂嵌套视图可能影响性能

有些数据库不支持在视图中进行 DML 操作

#### （2）存储过程 (Stored Procedure)

使用范围：

封装一组SQL逻辑，可以批量处理数据

实现定期计算、统计、批量修改等操作

减少客户端和服务器之间的交互

优点：

封装逻辑，提高代码复用性

执行效率高，减少网络传输

代码集中易于管理

缺点：

不如应用层逻辑灵活（修改需要重新部署）

不同数据库语法差异较大，可移植性差

不方便版本控制

#### （3）触发器 (Trigger)

使用范围：

自动处理一些数据校验、日志记录、审计操作

保证数据完整性、自动更新相关表

限制非法操作或自动生成字段

优点：

自动触发，无需手动干预

可在表数据变化时强制执行规则

便于做审计、历史记录、数据校验

缺点：

不容易排查问题（执行过程在后台）

大量触发器会影响性能，容易形成“隐式逻辑”

容易导致复杂依赖，维护困难

## 实验结论

遇到的问题：  
1. Gauss数据库的语法大部分与SQL语句一样，但是有细微的不同，比如建立触发器的语法，需要先建立一个函数，再调用如下SQl：

CREATE [ CONSTRAINT ] TRIGGER trigger\_name

{ BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] } ON table\_name

[ FROM referenced\_table\_name ]

{ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }

[ FOR [ EACH ] { ROW | STATEMENT } ]

[ WHEN ( condition ) ]

EXECUTE PROCEDURE function\_name ( arguments );

## 实验体会

在设计数据库的时候，应该适当设置主外键，建立触发器，让数据库更稳定，更合理。主外键 是数据库中的约束机制，主要用于维护数据的唯一性和引用完整性。触发器 是数据库中的程序代码，用于在特定事件发生时执行自定义逻辑。两者都可以用于维护数据完整性，但触发器的功能更灵活，而主外键的性能更高效。在实际应用中，通常优先使用主外键来维护数据完整性，而触发器则用于实现复杂的业务逻辑或特殊需求。