#### 姓名：谢宝玛 学号：1120233506

# 实验一：建立数据库

## **（一）**实验目的

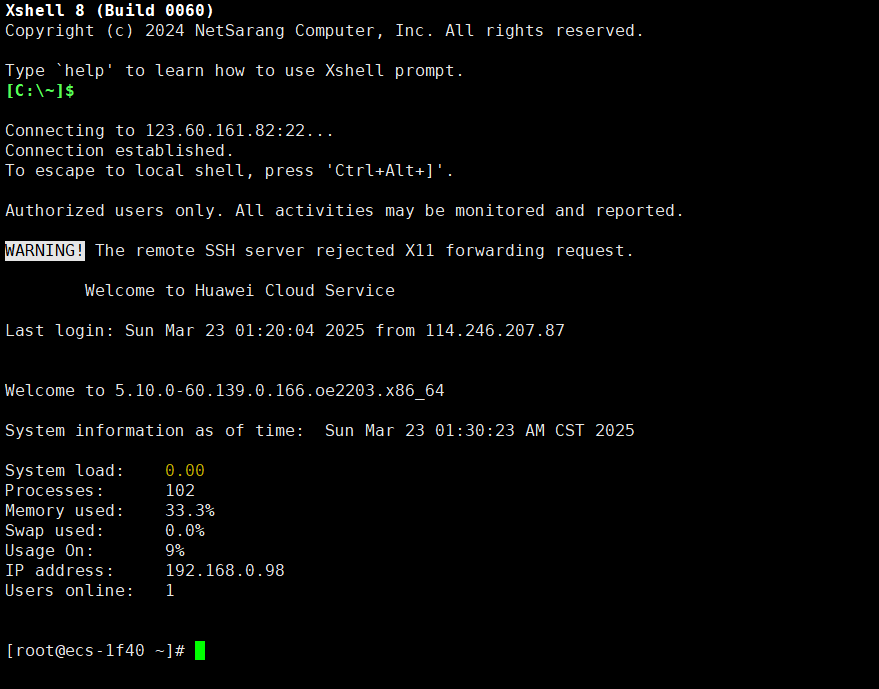
建立“学籍与成绩管理系统”表格；

• 建立表之间的参照关系

• 建立适当的索引

## （二）实验过程：

### 1，购买华为ECS服务器，选择openEuler操作系统，远程登录



### 购买Gauss数据库服务器，远程登录



### 建表

#### （1）学生表



##### SQL

CREATE TABLE xs (

xh CHAR(10) PRIMARY KEY, -- 学号，字符类型，长度10，主键

xm CHAR(8) NOT NULL, -- 姓名，字符类型，长度8，不能为空

ydh CHAR(2) NOT NULL, -- 所属学院代号，字符类型，长度2，不能为空

bj CHAR(8) NOT NULL, -- 班级，字符类型，长度8，不能为空

chrq DATE NOT NULL, -- 出生日期，日期类型，不能为空

xb CHAR(2) NOT NULL, -- 性别，字符类型，长度2，不能为空

FOREIGN KEY (ydh) REFERENCES xyb(ydh) -- 外键，关联学院表的ydh字段

);

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

VALUES ('20230001', '张三', '01', '01', '2000-05-15', 'M');

##### 结果

屏幕截图 2025-03-23 160620

#### （2）课程表



##### SQL

CREATE TABLE kc (

kcbh CHAR(3) PRIMARY KEY, -- 课程编号，字符类型，长度3，主键

kc CHAR(20) NOT NULL, -- 课程名称，字符类型，长度20，不能为空

lx CHAR(10), -- 课程类型，字符类型，长度10，允许为空

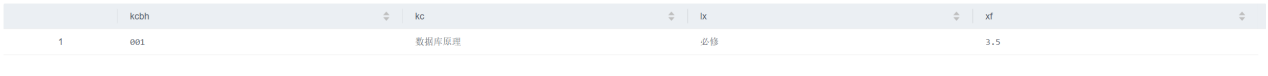
xf NUMERIC(5,1) -- 学分，数值类型，总长度5，小数位1，允许为空

);

INSERT INTO kc (kcbh, kc, lx, xf)

VALUES ('001', '数据库原理', '必修', 3.5);

##### 结果



#### （3）学院表



##### SQL

CREATE TABLE xyb (

ydh CHAR(2) PRIMARY KEY, -- 学院代号，字符类型，长度2，主键

ymc CHAR(30) NOT NULL -- 学院名称，字符类型，长度30，不能为空

);

INSERT INTO xyb (ydh, ymc)

VALUES ('01', '计算机学院');

##### 结果



#### （4）教师表



##### SQL

CREATE TABLE js (

xm CHAR(8) NOT NULL, -- 姓名，字符类型，长度8，不能为空

jsbh CHAR(10) PRIMARY KEY, -- 教师编号，字符类型，长度10，主键

zc CHAR(6) NOT NULL, -- 职称，字符类型，长度6，不能为空

ydh CHAR(2), -- 所属学院代号，字符类型，长度2，允许为空

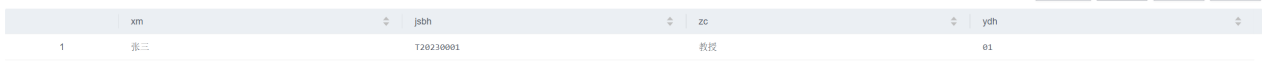
FOREIGN KEY (ydh) REFERENCES xyb(ydh) -- 外键，关联学院表的ydh字段

);

INSERT INTO js (xm, jsbh, zc, ydh)

VALUES ('张三', 'T20230001', '教授', '01');

##### 结果



#### （5）授课表



##### SQL

CREATE TABLE sk (

kcbh CHAR(3) NOT NULL, -- 课程编号，字符类型，长度3，不能为空

bh CHAR(10) NOT NULL, -- 教师编号，字符类型，长度10，不能为空

PRIMARY KEY (kcbh, bh), -- 联合主键，由课程编号和教师编号组成

FOREIGN KEY (kcbh) REFERENCES kc(kcbh), -- 外键，关联课程表的kcbh字段

FOREIGN KEY (bh) REFERENCES js(jsbh) -- 外键，关联教师表的jsbh字段

);

INSERT INTO sk (kcbh, bh)

VALUES ('001', 'T20230001');

##### 结果



#### （6）学生选课表



##### SQL

CREATE TABLE xk (

xh CHAR(10) NOT NULL, -- 学号，字符类型，长度10，不能为空

kcbh CHAR(3) NOT NULL, -- 课程编号，字符类型，长度3，不能为空

jsbh CHAR(10) NOT NULL, -- 教师编号，字符类型，长度10，不能为空

cj NUMERIC(5,1), -- 成绩，数值类型，总长度5，小数位1，允许为空

PRIMARY KEY (xh, kcbh, jsbh), -- 联合主键，由学号、课程编号和教师编号组成

FOREIGN KEY (xh) REFERENCES xs(xh), -- 外键，关联学生表的xh字段

FOREIGN KEY (kcbh) REFERENCES kc(kcbh), -- 外键，关联课程表的kcbh字段

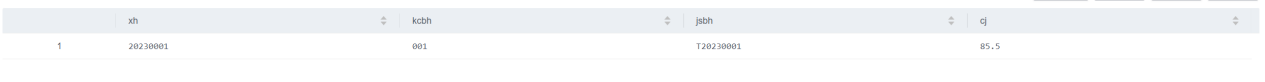
FOREIGN KEY (jsbh) REFERENCES js(jsbh) -- 外键，关联教师表的jsbh字段

);

INSERT INTO xk (xh, kcbh, jsbh, cj)

VALUES ('20230001', '001', 'T20230001', 85.5);

##### 结果



#### （7）结果



### 建立索引

#### 在xs表中将xh作为唯一索引

##### SQL

CREATE INDEX idx\_xh ON xs (xh);

#### 在js表中将jsbh作为唯一索引

##### SQL

CREATE INDEX idx\_jsbh ON js (jsbh);

#### 在js表中将jsbh作为唯一索引

##### SQL

CREATE INDEX idx\_xyh ON xyb (ydh);

## **（三）**实验结论

### 1，问题

#### （1）openGauss数据库难以安装（未解决）

本人已经在openEuler系统和CentOS 系统中尝试多次，都会遇到各种报错，最终放弃在服务器中安装openGauss数据库。

解决：购买了华为的GaussDB 数据库服务器。

#### （2）存在编码的问题

在Gauss数据库中，我设置的编码方式是UTF-8，由于中文和英文的编码不同，占据的位数也不同，于是会出现插入中文数据报错的情况，比如：

value too long for type character(8) Where: referenced column: xm

需要注意中文的长度。

## 实验体会

从安装开始openGauss数据库就透露出一股半成品的味道。

安装的过程及其繁琐，下载软件包，更改用户，更改系统配置，设置用户权限，使用脚本进行安装，只能使用single-node，这些东西都让openGauss不像是一个成熟的数据库软件。

连接数据库的客户端更是离谱，data studio根本不是一个发行的软件，更像一个开源的工具，需要自己配置Java环境，下载部署。最终我也没有成功使用data studio，而是用了华为提供的DAS。

个人感觉实验一的内容并不难，但是花了很多时间在安装，部署，启动这些琐碎的事情上，实在让人恼火，而且本人最终并没有成功，而是直接去买了台数据库服务器。与优秀，开源，成熟的关系型数据库相比（比如MySQL），openGauss明显还不完善。但在使用上感觉还可以，与传统的关系型数据库没有什么区别。

# 实验二

## 实验目的

执行数据库查询

## 实验过程

### 插入

#### 插入主表

##### 向学院表插入数据：

SQL：

INSERT INTO xyb (ydh, ymc) VALUES

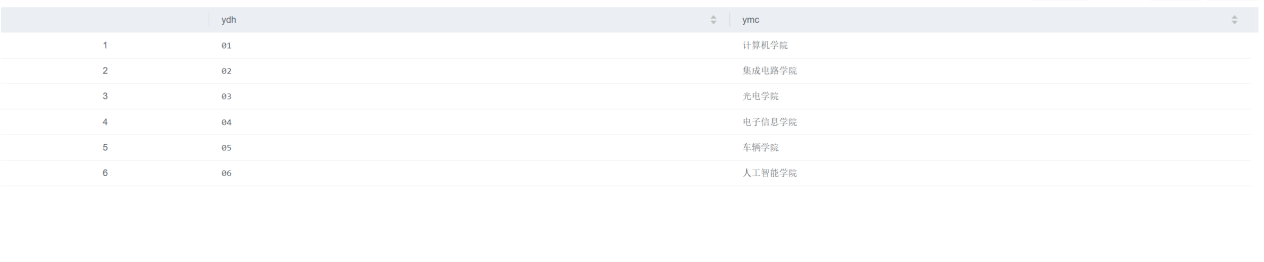
('02', '集成电路学院'),

('03', '光电学院'),

('04', '电子信息学院'),

('05', '车辆学院'),

('06', '人工智能学院');



##### 向课程表插入数据

SQL：

INSERT INTO kc (kcbh,kc,lx,xf) VALUES

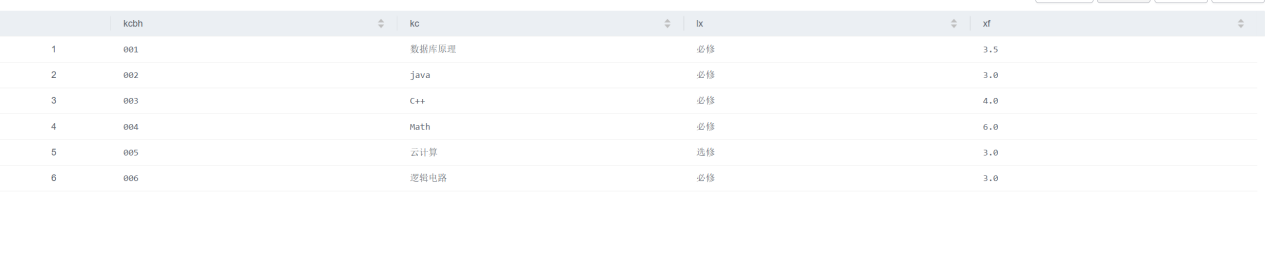
('002','java','必修','3'),

('003','C++','必修','4'),

('004','Math','必修','6'),

('005','云计算','选修','3'),

('006','逻辑电路','必修','3');



#### 插入子表

##### 向教师表插入数据

SQL：

INSERT INTO js (xm,jsbh,zc,ydh) VALUES

('Lisi','T20230002','教授','02'),

('LMK','T20230003','教授','03'),

('李四','T20230004','导师','04'),

('王五','T20230005','教授','05'),

('赵六','T20230006','老师','06');



##### 向学生表插入数据

SQl：

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

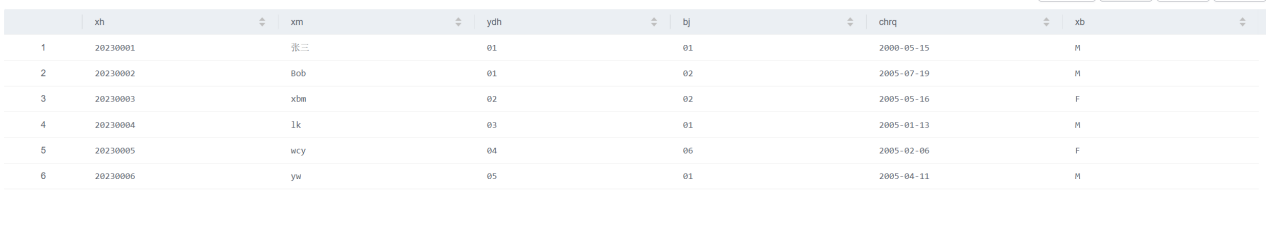
('20230002','Bob','01','02','2005-07-19','M'),

('20230003','xbm','02','02','2005-05-16','F'),

('20230004','lk','03','01','2005-01-13','M'),

('20230005','wcy','04','06','2005-02-06','F'),

('20230006','yw','05','01','2005-04-11','M');



##### 向授课表出入数据

SQL：

INSERT INTO sk (kcbh,bh) VALUES

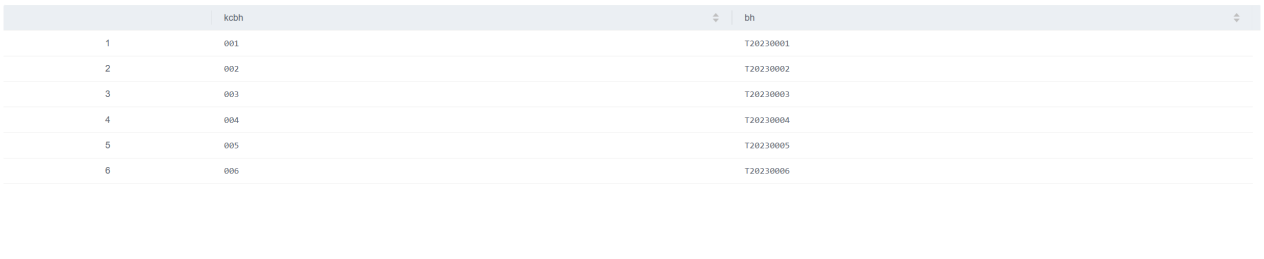
('002','T20230002'),

('003','T20230003'),

('004','T20230004'),

('005','T20230005'),

('006','T20230006');



##### 向选课表插入

SQL：

INSERT INTO xk (xh,kcbh,jsbh,cj) VALUES

('20230001','002','T20230002','90'),

('20230002','002','T20230002','92'),

('20230004','001','T20230001','70'),

('20230005','002','T20230002','90'),

('20230006','003','T20230003','50');



### 异常

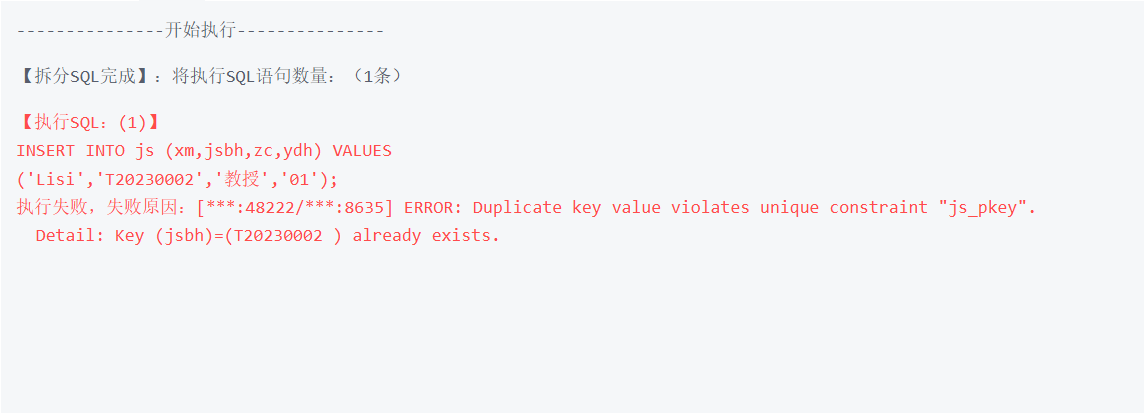
#### 新增教师异常

SQL：

INSERT INTO js (xm,jsbh,zc,ydh) VALUES

('Lisi','T20230002','教授','01'),

分析：jsbh是主键，目前已经存在编号为T20230002的教师，所以报错：



#### 新增授课表异常

SQL：

INSERT INTO sk (kcbh,bh) VALUES

('010','T20230002');

分析：kcbh是外键，没有课程编号为010的课，所以报错。



#### 向选课表插入异常

SQL：

INSERT INTO xk (xh,kcbh,jsbh,cj) VALUES

('20230010','002','T20230001','90');

分析：xh是外键，不存在学号为20230010的学生，所以报错。



### 查询

#### **（1）**学‘’数据库原理‘’的学生的所有信息

连接学生表找到选课的学生，根据课程名称筛选出“数据库原理”

SQL：

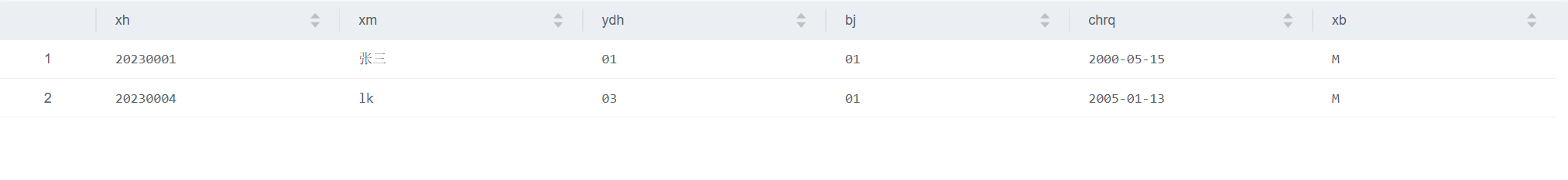
SELECT xs.\*

FROM xs

JOIN xk ON xs.xh = xk.xh

JOIN kc ON xk.kcbh = kc.kcbh

WHERE kc.kc = '数据库原理';



#### **（2）**没有授课的教师

从教师表 js 中找出那些 jsbh 不在 sk 授课表里的教师

SQL:

SELECT js.\*

FROM js

WHERE js.jsbh NOT IN (SELECT bh FROM sk);

结果为空：



#### （3）学生张三所选课程的总学分

先根据 xs 表找到名字为“张三”的学生，和 xk 表连接，找到他选的课程；和 kc 表连接，获取这些课程的学分；然后 SUM(kc.xf) 计算总学分

SQL：

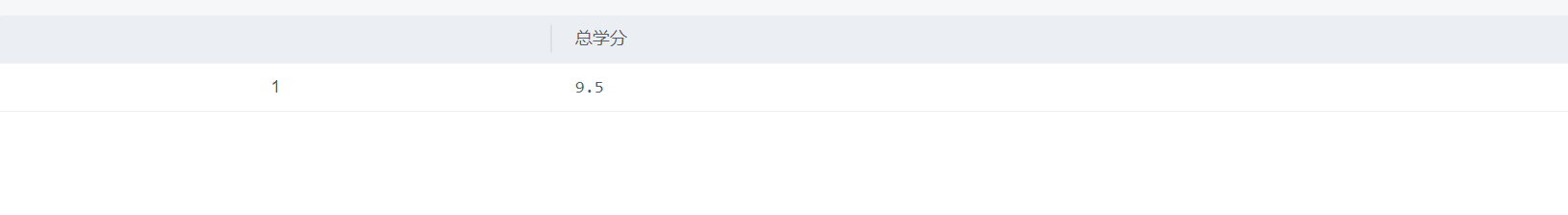
SELECT SUM(kc.xf) AS 总学分

FROM xs

JOIN xk ON xs.xh = xk.xh

JOIN kc ON xk.kcbh = kc.kcbh

WHERE xs.xm = '张三';



#### 教授学生张三必修课程的老师情况

SQL：

SELECT DISTINCT js.\*

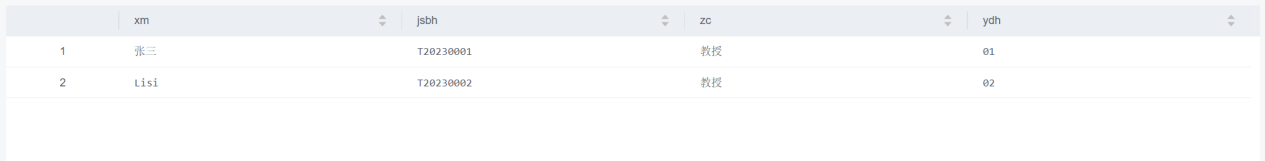
FROM xs

JOIN xk ON xs.xh = xk.xh

JOIN kc ON xk.kcbh = kc.kcbh

JOIN js ON xk.jsbh = js.jsbh

WHERE xs.xm = '张三'

AND kc.lx = '必修';  


#### 学生张三选修了哪些系的课程

学生选课表 xk 中有教师编号 jsbh，教师表 js 中通过 ydh 可以找到教师所在的学院，学院表 xyb 中 ymc 是学院名称， 用 DISTINCT 去重，得到“张三”选修过的所有学院名称

SQL：  
SELECT DISTINCT xyb.ymc

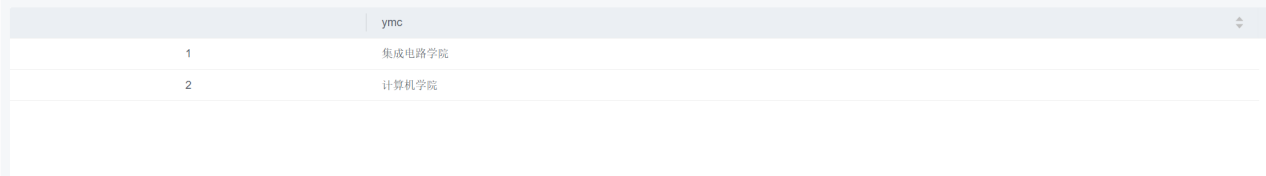
FROM xs

JOIN xk ON xs.xh = xk.xh

JOIN js ON xk.jsbh = js.jsbh

JOIN xyb ON js.ydh = xyb.ydh

WHERE xs.xm = '张三';



#### 计算机系学生所学的所有课程

通过学生表 xs 找到 ydh，连接学院表 xyb，筛选出“计算机学院”；通过学生学号 xh 与选课表 xk 连接，得到这些学生选的课程；与课程表 kc 连接，查询课程名称；使用 DISTINCT 去重，列出所有课程名。

SQL：

SELECT DISTINCT kc.kc

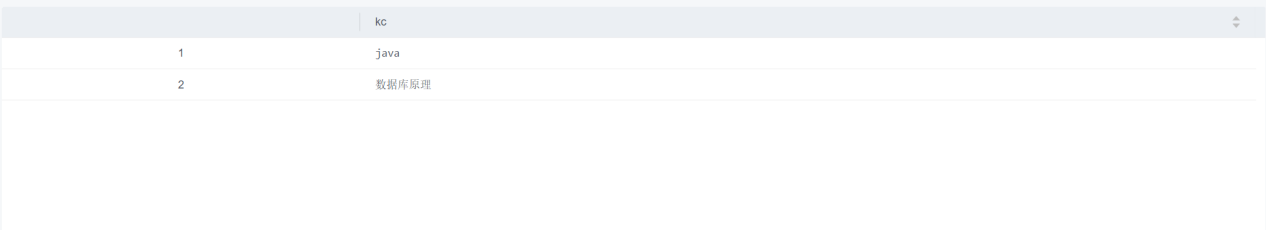
FROM xs

JOIN xyb ON xs.ydh = xyb.ydh

JOIN xk ON xs.xh = xk.xh

JOIN kc ON xk.kcbh = kc.kcbh

WHERE xyb.ymc = '计算机学院';



### 更新

#### 更新“张三”的出生日期为2007-09-20

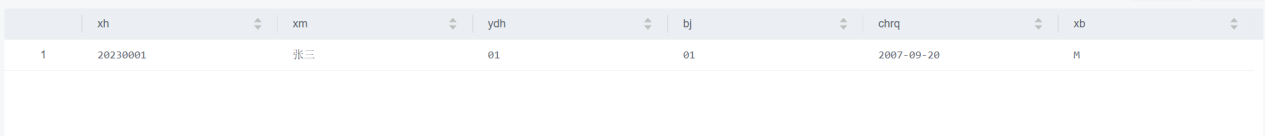
SQL：

UPDATE xs

SET chrq = '2007-09-20'

WHERE xm = '张三';

select \* from xs where xm ='张三';



#### 更新成绩

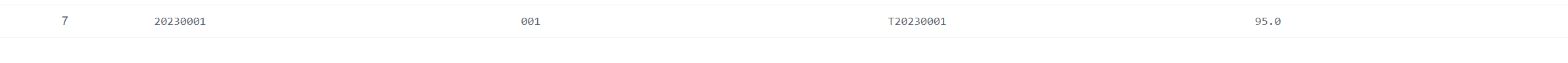
子查询先根据名字查出“张三”的学号 xh；再根据课程名查出 kcbh；然后在选课表 xk 中更新成绩。

SQL：  
UPDATE xk

SET cj = 95

WHERE xh = (SELECT xh FROM xs WHERE xm = '张三')

AND kcbh = (SELECT kcbh FROM kc WHERE kc = '数据库原理');



### 删除

删除“wcy”的学籍信息

1. 删除学生在选课表中的记录；
2. 然后再删除学生的学籍信息

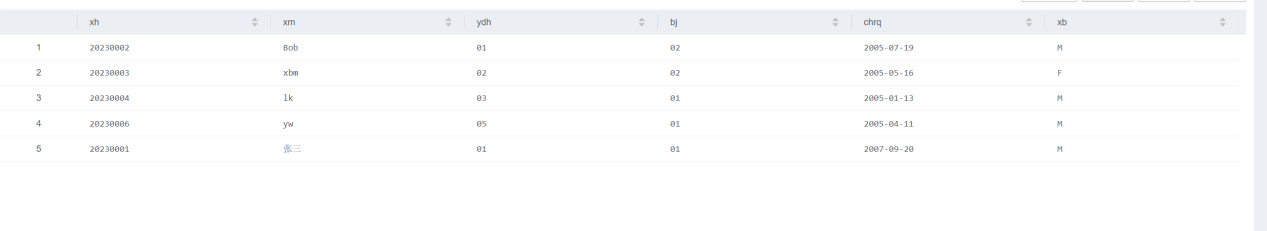
SQL：

DELETE FROM xk

WHERE xh = '20230005'; -- 使用需要删除的学生学号

DELETE FROM xs

WHERE xm = 'wcy';



## 实验体会

在对数据库做增，删，改的过程一定要注意表中是否存在外键，是否存在主键。

插入操作：外键需要在指定范围内；

删除操作：如果有外键，需要先删除相关联的表的数据，再删除该数据；

更新操作：同删除操作。

# 实验三：数据库物理设计

## 实验目的

创建数据库分区表

体会主键，外键约束

体会索引

权限管理

## 实验过程

### 练习更新、删除主表数据（针对主键属性且子表中可能有参照外键数据）

#### 更新

将学院编号为‘01’的学院改为‘10’，注意教师表和学生表都有学院编号

因为学院编号可能在教师表和学生表里作为外键或关联字段直接使用，直接改可能会导致关联性错误或约束冲突，所以一般采取“新增-修改-删除”三步走策略：

SQL：

-- 第一步：先在学院表新增编号为‘10’的学院，名称和原来的一样

INSERT INTO xyb (ydh, ymc)

SELECT '10', ymc

FROM xyb

WHERE ydh = '01';

-- 第二步：将教师表和学生表中原‘01’编号改为‘10’

UPDATE js

SET ydh = '10'

WHERE ydh = '01';

UPDATE xs

SET ydh = '10'

WHERE ydh = '01';

-- 第三步：删除原来的‘01’学院

DELETE FROM xyb

WHERE ydh = '01';



#### 删除

删除“张三”这名同学

SQL：

-- 先删除该学生在选课表中的记录

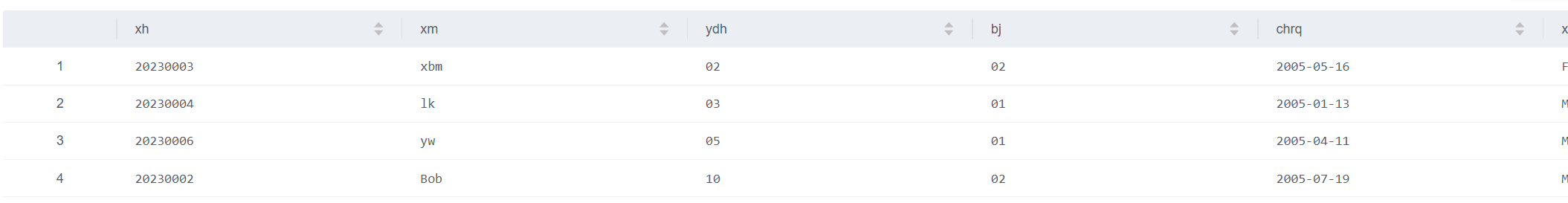
DELETE FROM xk

WHERE xh IN (SELECT xh FROM xs WHERE xm = '张三');

-- 再删除学生表中的记录

DELETE FROM xs

WHERE xm = '张三';



### 练习更新、删除主表数据（针对非主键 属性）

#### 更新

将“车辆学院”改名为“机械学院”

SQL：  
UPDATE xyb

SET ymc = '机械学院'

WHERE ymc = '车辆学院';



### 练习先删除子表数据，再删除主表数据

删除“张三”这名同学

SQL：

-- 先删除该学生在选课表中的记录

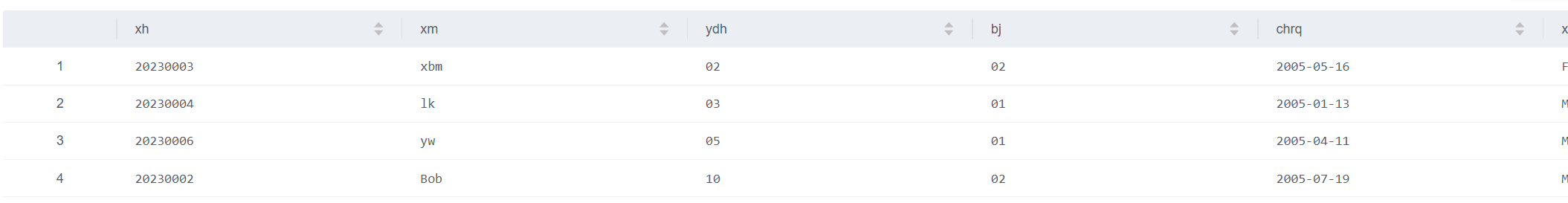
DELETE FROM xk

WHERE xh IN (SELECT xh FROM xs WHERE xm = '张三');

-- 再删除学生表中的记录

DELETE FROM xs

WHERE xm = '张三';



### 使用子查询方式更新、删除数据

将“教授”职称的教师的课程（授课表 sk 中对应 jsbh）在 选课表 xk 里他们授课的所有学生成绩（cj）加 5 分（如果超过 100 则设为 100），要求用**子查询**完成更新。

sk 表中有课程编号（kcbh）和教师编号（bh）js 表中有教师编号（jsbh）和职称（zc）xk 表是选课表，包含学生成绩（cj）、kcbh、jsbh子查询部分需要找出“教授”的教师编号，再根据这个编号去更新 xk 表中的成绩。

SQL：  
UPDATE xk

SET cj = CASE

WHEN cj + 5 > 100 THEN 100

ELSE cj + 5

END

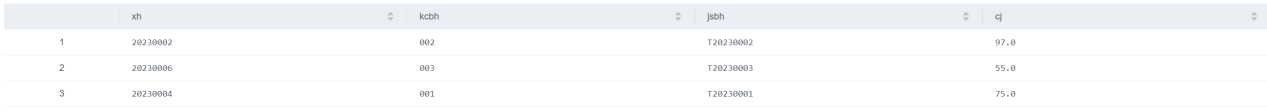
WHERE jsbh IN (

SELECT jsbh

FROM js

WHERE zc = '教授'

);



### 权限管理

#### 以不同身份用户登录数据库建立表，表名一样

##### 1.新增用户xbm

SQL:  
CREATE ROLE "xbm" LOGIN VALID BEGIN '2025-03-23 23:22:11+08:00' VALID UNTIL '2025-07-31 23:22:38+08:00' CREATEDB CREATEROLE INHERIT NOREPLICATION PASSWORD '\*\*\*\*\*\*';

GRANT root TO "xbm" WITH ADMIN OPTION;

/\* Switch to database postgres \*/

GRANT SELECT ON TABLE "pg\_catalog"."pg\_roles" TO "xbm" WITH GRANT OPTION;

我没有采用SQL语句，而是直接在华为的DAS中新增。

##### **2.**建立学院表

先建立一个新的schema “xbm\_student\_info”

再创建表

SQL：  
CREATE TABLE `xyb` (

`ydh` VARCHAR(10) NOT NULL,

`ymc` VARCHAR(50) DEFAULT NULL,

PRIMARY KEY (`ydh`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

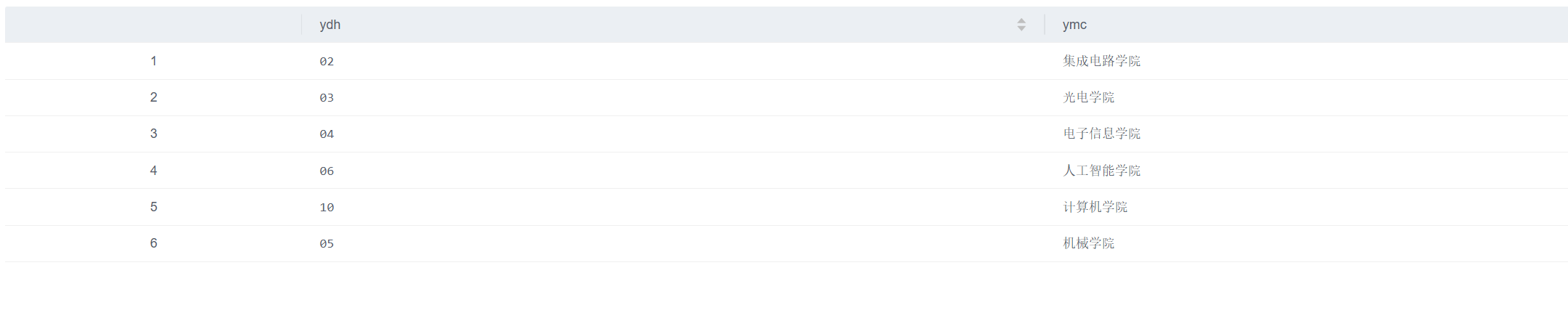
#### 以不同身份用户查询自己与其他用户建立 的表

##### 查询root用户建立的学院表

需要切换到‘student\_info’

SQL：

SELECT \* FROM xyb;



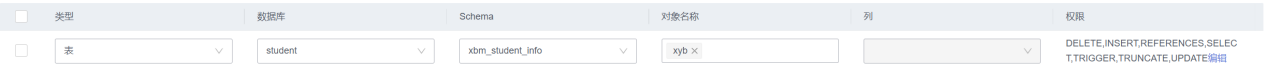
##### **2.**查询xbm用户建立的学院表

需要切换到‘xbm\_student\_info’

SQL:

SELECT \* FROM xyb;

#### 定义授权方案并进行验证

在以上的任务中，xbm用户具有增删改查的权限，并继承了root用户的权限

DELETE,INSERT,REFERENCES,SELECT,TRIGGER,TRUNCATE,UPDATE

修改xbm的权限，只保留select，让xbm用户对xbm\_student\_info只有可读权限。



验证

以xbm用户登录，执行插入操作

SQL:  
INSERT INTO xyb(ydh ,ymc )VALUES

('01','计算机学院');

结果会报错，无权限



但是xbm用户可以执行creat table建表的操作。

## 实验体会

1. 在执行更新，删除操作的时候一定要注意表是否有外键，如果有外键需要对子表先进行更新，删除操作。
2. 共同设计一个数据库的时候需要注意设置非root用户的权限。

# 实验四

## 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。