实验八 数据库完整性

计算机科学与技术

2021160291 李景昊

一、实验目的和要求

1、掌握通过SQL对数据进行完整性控制；学习用户自定义约束，并实践用户完整性，利用 SQL 查询分析器用短语 NOT NULL、UNIQUE、CHECK 保证用户定义完整性
2、掌握使用SQL定义触发器。

二、实验内容和步骤

1、数据库的约束*The Constraints of Database*

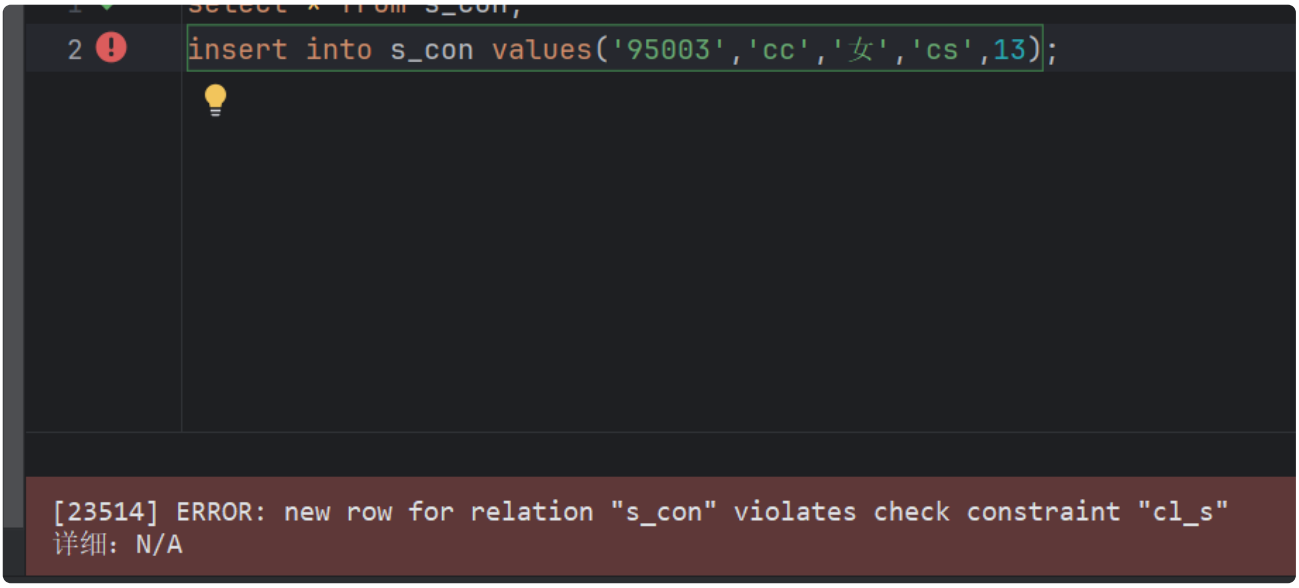
首先“新建查询”，在建表的同时定义各种约束，如图8-1所示。

```
create table s_con
(
  sno varchar(10) CONSTRAINT p_s primary key,
  sname varchar(20) CONSTRAINT u_s unique,
  ssex char(2) constraint c_s check(ssex in('男','女'))
  CONSTRAINT d_s default('男'),
  sdept char(10),
  sage int constraint cl_s check(sage>15 AND sage<30)
)
```

当执行插入语句时，执行结果如图8-2所示。

```
insert into s_con values('95004','dd','男','cs',17);
insert into s_con(sno,sname,sdept,sage)
values('95002','bb','cs',18);
select * from s_con
insert into s_con values('95003','cc','女','cs',13);
```

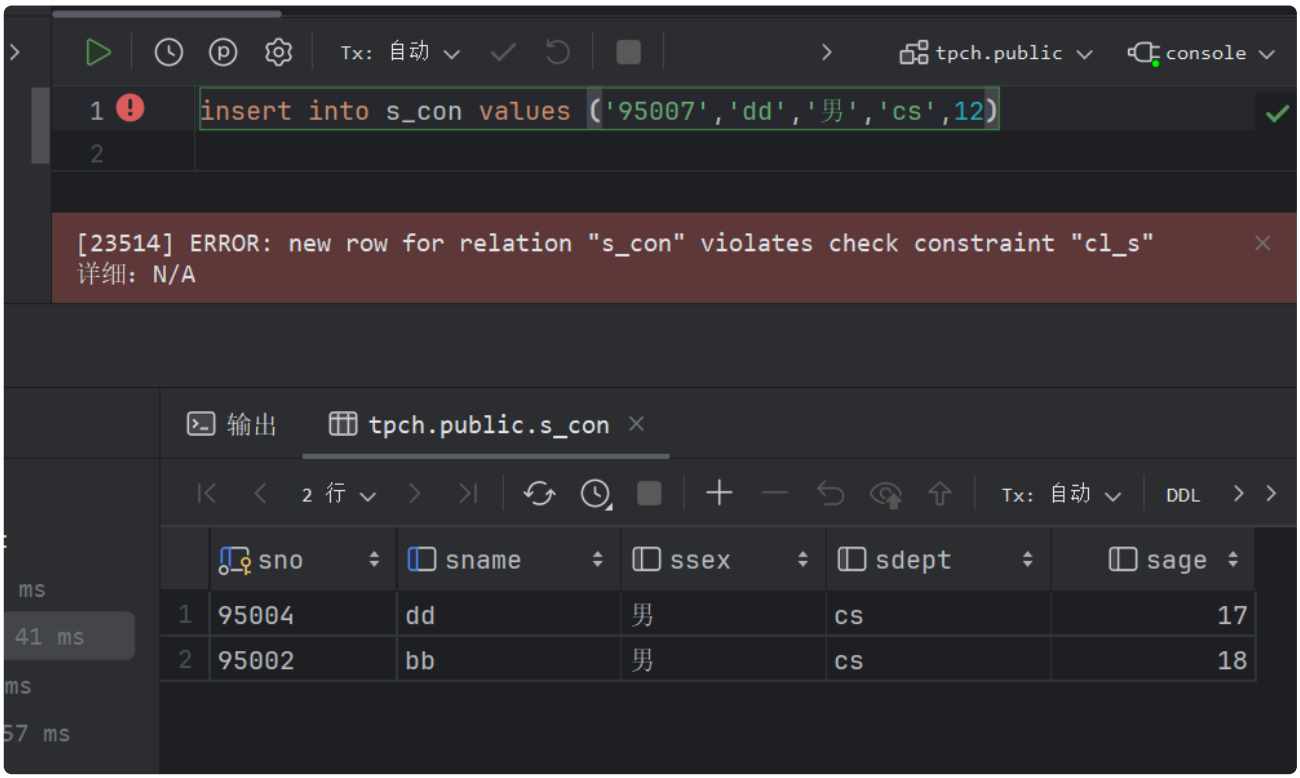
无法插入因为违反约束了



这时，在执行下列插入语句，查看一下效果：

```
insert into s_con values('95004','dd','男','cs',17);
insert into s_con values('905007','dd','男','cs',12);
```

违反约束条件而无法插入



2,触发器

触发器的工作机制是ECA规则，即当应用对一个对象发起DML操作时，就会产生一个触发事件（Event）。如果该对象上拥有该事件对应的触发器，那么就会检查触发器的触发条件（Condition）是否满足，如果满足触发条件，那么就会执行触发动作（Action）

（1）CREATE TRIGGER命令

CREATE TRIGGER创建触发器，指定触发器的事件、条件和动作。语法如图8-3所示。

```
CREATE TRIGGER <trigger name>                                    # 触发器名称
    <trigger action time><trigger event>                         # 触发器事件
ON <table name>                                                  # 触发器对象
    [REFERENCING <old or new values alias list>]                 # 触发器条件
    <triggered action>                                           # 触发器动作
```

（2）触发器语法提供触发时机

BEFORE：在触发器事件执行之前检查触发条件以及执行触发动作；

AFTER：触发事件之后检查触发条件以及执行触发动作。

例1：在UPDATE事件发生之前执行触发器，如图8-4所示。

```
CREATE TRIGGER before_update
    BEFORE UPDATE
ON ......
```

例2：在INSERT事件发生之后执行触发器，如图8-5所示。

```
CREATE TRIGGER after_insert
    AFTER INSERT
 ON ......
```

（3）元组级触发器或语句级触发器

元组级触发器是每一行元组都会触发一次，当事件涉及多个元组时，触发器将被触发多次；语句级触发器则是一条触发一次。默认情况下语句级的触发器。

SQL语句级的触发器的定义如图所8-6所示。

```
CREATE TRIGGER after_insert
    AFTER INSERT
ON warehouse
FOR EACH STATEMENT
......
```

元组级的触发器如图8-7所示。

```
CREATE TRIGGER after_insert
    AFTER INSERT
ON warehouse
FOR EACH ROW
......
```

(4)触发器中的系统临时变量

系统为每一个触发器建立临时变量NEW和OLD，column_name为定义触发器对象的属性。

New.column_name：update或insert事件对应“新”元组，column_name对应新元组上的对应的列值：

OLD.column_name：update或insert事件对应“老”元组，column_name对应老元组上的对应的列值：

（5）OpenGuass在创建触发器之前，需要先创建一个函数，如果返回值是Trigger，那么该函数就是触发器函数，否则是普通函数。同一个触发器可以指定多个触发事件，每个事件发生时都能激活触发器来执行触发器的动作。

（6）禁用触发器

（7）INSTEAD触发器

三、实验内容

练习一:创建表创建表 worker（no, name, sex, sage, salary），完成表上的约束

1:自定义 2 个约束 U1 以及 U2,其中 U1 规定 Name 字段唯一，U2 规定 sage（级别）字段的上限是 28。

```
create table worker (
    no int,
    name varchar(50) constraint w_name unique ,
    sex char(2) constraint w_sex check ( sex in ('男','女') ),
    sage varchar(3) constraint  w_age check ( sage<=28 ),
    salary int
)
```

2:在 worker 表中插入一条合法纪录。

```
insert into worker(no, name, sex, sage, salary)
values(
       '123',
       'Ben',
       '男',
       '21',
       '15000'
      )
```

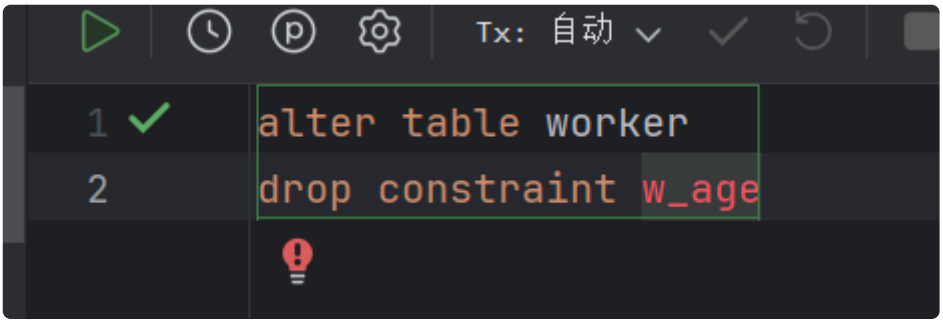
no	name	sex	sage	salary
123	Ben	男	21	15000

3:演示插入违反 U2 约束的例子，U2 规定元组的 sage 属性的值必须<=28。

```
insert into worker(no, name, sex, sage, salary)
values(
       '456',
       'Tom',
       '男',
       '30',
       '15000'
      )
```

```
[23514] ERROR: new row for relation "worker" violates check constraint "w_age"
详细：N/A
```

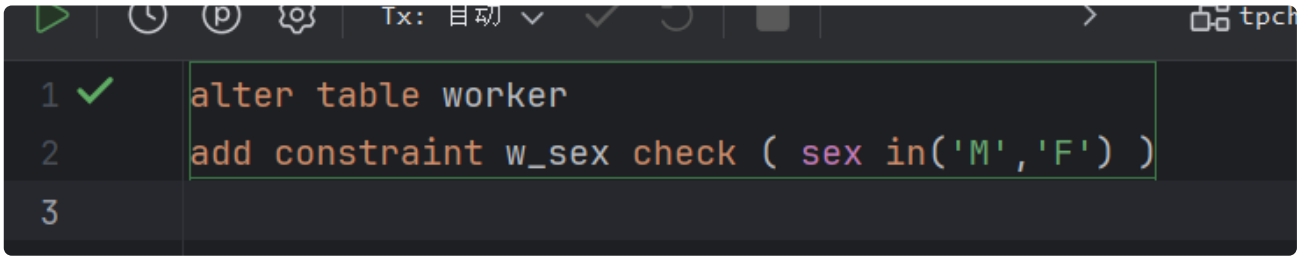
4:去除 U2 约束



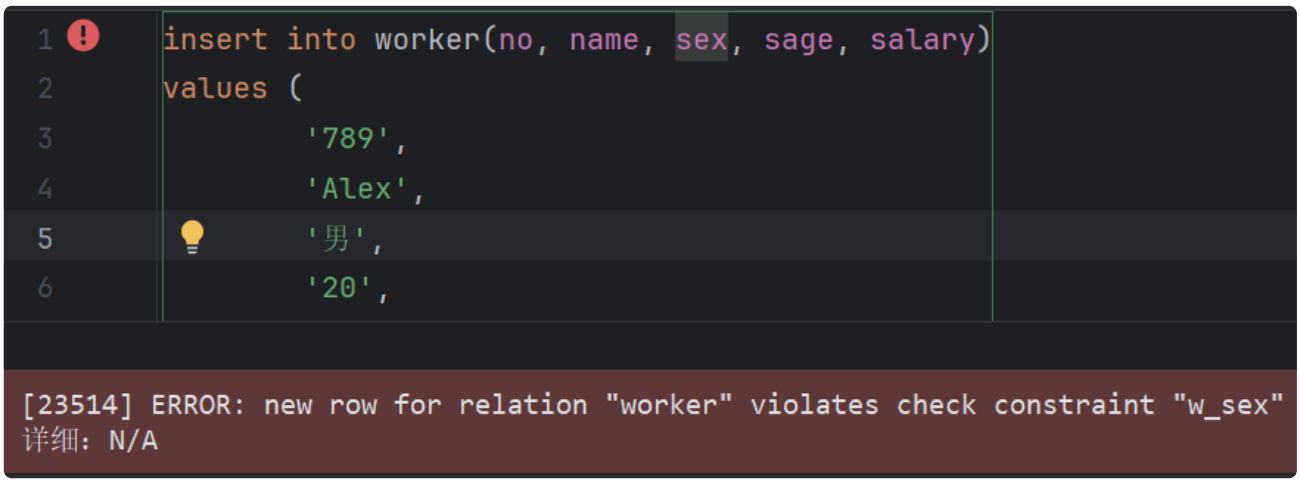
5:重新插入（3）中想要插入的数据，由于去除了 U2 约束，所以插入成功

	no	name	sex	sage	salary
1	456	Tom	男	30	15000
2	123	Ben	男	21	15000

6:创建约束 rule_sex，规定插入或更新的值只能是 M 或 F，并绑定到 worker 的 sex字段



7:演示违反约束 rule_sex 的插入操作



练习2：对PC机产品数据库编写触发器。如果不满足声明的约束则拒绝或撤销更新

product(maker,model,type)

PC(model,speed,ram,hd,price)

1:当修改PC的价格时，检查不存在速度与其相同但价格更低的PC机

创建一个例程

```
create or replace function check_pc_price()
returns trigger as $$
begin
    if new.price >0 then
        if exists (select 1 from pc where speed = new.speed and price < new.price) then
            raise exception '存在速度相同但是价格更低的pc机';
        end if;
    end if;
    return new;
end;
$$ language plpgsql;
```

添加一个trigger

```
create trigger check_price_trigger
after update of price on pc
for each row
    execute procedure check_pc_price()
```

选择一台pc尝试修改价格

	model	speed	ram	hd	price
1	PC2	3	8GB	1TB	800

触发trigger，并且成功运行提示

WHERE ORDER BY

	model	speed	ram	hd	price
1	PC2	3	8GB	1TB	2000
2	PC6	2	8GB	500GB	500
3	PC7	4	16GB	2TB	1500
4	PC8	3	4GB	1TB	900
5	PC9	3	16GB	2TB	1300
6	PC10	1	8GB	500GB	600

```
[P0001] ERROR: 存在速度相同但是价格更低的pc机
```

2:当修改任何PC机的RAM或hd时，要求被修改的PC机的hd至少是RAM 的100倍

创建一个例程

```
create function check_pc_ram() returns trigger
    language plpgsql
as
$$
begin
    IF NEW.RAM IS NOT NULL AND NEW.hd IS NOT NULL THEN
        IF NEW.hd < NEW.RAM * 100 THEN
            RAISE EXCEPTION 'hd的值太小了';
        end if;
    end if;
    return new;
end;
$$;

alter function check_pc_ram() owner to gaussdb;
```

常数修改hd的数值，成功提示错误

```
WHERE                                ORDER BY model
```

	model	speed	ram	hd	price
1	PC1	7	8	10	600
2	PC10	1	8	1000	600
3	PC2	3	8	500	800

```
[P0001] ERROR: hd的值太小了
```

练习3：对PC机数据库创建视图

```sql
CREATE VIEW NewPC AS
SELECT product.maker, pc.model, pc.speed, pc.ram, pc.hd, pc.price
FROM product
JOIN pc ON product.model = pc.model
WHERE product.type = 'pc';
```

（1）写一个替换触发器用于处理对视图插入操作。

```sql
CREATE OR REPLACE FUNCTION insert_newpc_trigger()
RETURNS TRIGGER AS
$$
BEGIN
    -- 检查插入的数据是否符合条件，根据实际需求进行修改
    IF NEW.type = 'pc' THEN
        INSERT INTO pc (model, speed, ram, hd, price)
        VALUES (NEW.model, NEW.speed, NEW.ram, NEW.hd, NEW.price);
    ELSE
        RAISE EXCEPTION '输入类型不符合要求';
    END IF;

    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER insert_newpc_trigger
INSTEAD OF INSERT ON NewPC
FOR EACH ROW
EXECUTE procedure insert_newpc_trigger();
```

对视图进行插入操作

```sql
insert into newpc(maker, model, speed, ram, hd, price)
values('ben',123,10,32,100000,500)
```

成功把数据插入到pc表中

WHERE ORDER BY model

	model	speed	ram	hd	price
1	123	10	32	100000	500

（2）写一个替换触发器用于处理对视图中属性price的修改操作。

```
create function update_newpc_price_trigger() returns trigger
    language plpgsql
as
$$
BEGIN
    -- 将新的价格乘以2
    NEW.price := NEW.price * 2;

    IF NEW.price < 0 THEN
        RAISE EXCEPTION '金额必须为正数';
    END IF;

    -- 执行实际的更新操作
    UPDATE pc
    SET price = NEW.price
    WHERE model = NEW.model;

    RETURN NEW;
END;
$$;

alter function update_newpc_price_trigger() owner to gaussdb;

```

update_newpc_price_trigger()

（3）写一个替换触发器用于处理从视图中删除一个特定的元组。

根据model的名称进行删除