数据库系统课程实验报告

实验名称:	实验八：触发器
实验日期:	2022/5/17
实验地点:	厦门大学德旺图书馆
提交日期:	2022/5/17

学号:	20420192201952
姓名:	庾晓萍
专业年级:	软工 2020 级
学年学期:	2021-2022 学年第二学期

1. 实验目的

- 理解数据库系统用户（user）、权限（privilege）和角色（role）的概念和作用
- 理解 openGauss 触发器的作用和工作原理
  - ■ AFTER/BEFORE 触发器
  - ■ 行级(row)触发器和语句级(statement)触发器
- 熟练掌握 openGauss 触发器的设计方法
- 熟练掌握 openGauss 触发器的定义、查看、禁止、启用和删除操作

2. 实验内容和步骤

一、创建 Teacher 表：Teacher(ID, job, Sal)，其中，

ID 为教工号，定长为 5 的字符型，主码

JOB 为职称，最大长度为 20 的变长字符型，非空

SAL 为工资，长度为 7 的数字型，其中保留两位小数

```
sale=> CREATE TABLE Teacher(
sale(>      ID CHAR(5) PRIMARY KEY,
sale(>      JOB VARCHAR(20) NOT NULL,
sale(>      SAL NUMBER(7,2)
sale(> );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "teacher_pkey" for table "teacher"
CREATE TABLE
```

二、 为 Teacher 表增加以下实验数据：（‘10001’,‘教授’,3800),(‘10002’,‘教授’,4100),(‘10003’,‘副教授’,3500),(‘10004’,‘助理教授’,3000)

```
sale=> INSERT INTO Teacher VALUES ('10001','教授',3800);
INSERT 0 1
sale=> INSERT INTO Teacher VALUES ('10002','教授',4100);
INSERT 0 1
sale=> INSERT INTO Teacher VALUES ('10003','副教授',3500);
INSERT 0 1
sale=> INSERT INTO Teacher VALUES ('10004','助理教授',3000);
INSERT 0 1
```

三、在 Teacher 表上创建一个 BEFORE 行级触发器（名称：INSERT_OR_UPDATE_SAL）以实现如下完整性规则：教授的工资不得低于 4000 元，如果低于 4000 元，自动改为 4000 元。

```
sale=> CREATE OR REPLACE FUNCTION tri_teacher_insert_func1()
sale-> RETURNS TRIGGER AS $$ DECLARE
sale$> BEGIN
sale$> IF (new.Job='教授') AND (new.Sal < 4000) THEN
sale$> new.Sal :=4000;
sale$> END IF;
sale$> RETURN NEW;
sale$> END;
sale$> $$LANGUAGE PLPGSQL;
CREATE FUNCTION
sale=>
sale=> CREATE TRIGGER Insert_Or_Update_Sal  BEFORE INSERT OR UPDATE ON Teacher
sale-> FOR EACH ROW EXECUTE PROCEDURE tri_teacher_insert_func1();
CREATE TRIGGER
sale=>
```

四、验证触发器是否正常工作：分别执行以下 A，B 两种操作，验证 INSERT_OR_UPDATE_SAL 触发器是否被触发？工作是否正确？如果正确，请观察 Teacher 表中数据的变化是否与预期一致。

A. 插入两条新数据(‘10005’,‘教授’,3999),(‘10006’,‘教授’,4000);

B. 更新数据(‘10002’,‘教授’,4100)为(‘10002’,‘教授’,3900)。

```
sale=> INSERT INTO Teacher VALUES ('10005','教授',3999);
INSERT 0 1
sale=> INSERT INTO Teacher VALUES ('10006','教授',4000);
INSERT 0 1
```

```
sale=> UPDATE Teacher SET SAL=3900 WHERE Teacher.ID='10002';
UPDATE 1
sale=> SELECT *FROM Teacher;
  id   |   job    |   sal
-------+----------+---------
 10001 | 教授     | 3800.00
 10003 | 副教授   | 3500.00
 10004 | 助理教授 | 3000.00
 10005 | 教授     | 4000.00
 10006 | 教授     | 4000.00
 10002 | 教授     | 4000.00
(6 rows)
```

五、完成教材【例 5.21】：当对表 SC 的 Grade 属性进行修改时，若分数增加了 10%及其以上，则将此次操作记录到下面表中：SC_U(Sno, Cno, Oldgrade, Newgrade)，其中，Oldgrade 是修改前的分数，Newgrade 是修改后的分数。

① 创建 SC_U 表：SC_U(Sno, Cno, Oldgrade, Newgrade)。

Sno：定长为 9 的字符型，外码，引用 Student 表中 Sno 的值。

Cno：定长为 4 的字符型，外码，引用 Course 表中 Cno 的值。

Oldgrade 的数据类型：长度为 3 的整型。

Newgrade 的数据类型：长度为 3 的整型

```
sale=> CREATE TABLE SC_U(
sale(>     Sno CHAR(9), FOREIGN KEY(Sno) REFERENCES Student(Sno),
sale(>     Cno CHAR(4), FOREIGN KEY(Cno) REFERENCES Course(Cno),
sale(>     Oldgrade INTeger(3),
sale(>     Newgrade INTeger(3)
sale(> );
CREATE TABLE
```

② 创建 SC 表上的 AFTER 行级触发器，触发器名为 SC_AFTER_UPDATE

```
sale=> CREATE OR REPLACE FUNCTION tri_func2()
sale-> RETURNS TRIGGER AS $$ DECLARE
sale$> BEGIN
sale$> IF (new.Grade>=1.1*Old.Grade) THEN
sale$> INSERT INTO SC_U(Sno,Cno,OldGrade,NewGrade)
sale$> Values(Old.Sno,Old.Cno,Old.Grade,New.Grade);
sale$> END IF;
sale$> RETURN NEW;
sale$> END;
sale$> $$LANGUAGE PLPGSQL;
CREATE FUNCTION
sale=>
sale=> CREATE TRIGGER SC_AFTER_UPDATE  AFTER UPDATE OF GRADE ON SC
sale-> FOR EACH ROW EXECUTE PROCEDURE tri_func2();
CREATE TRIGGER
```

③ 验证 SC_AFTER_UPDATE 触发器是否正常工作（测试数据同教材）。执行上述两种操作，验证 SC_AFTER_UPDATE 触发器是否被触发且是否正确工作？如果触发器正确工作，请观察 SC_U 表中数据的变化。

```
UPDATE SC
SET GRADE=100
WHERE SNO='201215122' AND CNO='2';
```

```
UPDATE SC
SET GRADE=90
WHERE SNO='201215121' AND CNO='2';
```

触发器正常工作，当修改的新成绩是旧成绩的 1.1 倍以上时，加入 SC_U 中。

```
sale=> UPDATE SC SET GRADE=100  WHERE SNO='201215122' AND CNO='2';
UPDATE 1
sale=> UPDATE SC SET GRADE=90  WHERE SNO='201215121' AND CNO='2';
UPDATE 1
sale=> SELECT *FROM SC_U;
    sno    | cno  | oldgrade | newgrade
-----------+------+----------+----------
 201215122 | 2    |       80 |      100
 201215121 | 2    |       70 |       90
(2 rows)
```

（6）查看触发器；

```
  16809 | sc_after_update           | 16887 |    17 | O       | f          |          0 |          0 |          0 | f          |
f          |      0 | 3     | \x    |       |   16576
(70 rows)
```

（7）验证触发器禁用后效果

① 修改 SC 表，使 SC_AFTER_UPDATE 触发器失效；//建议将数据还原到步骤（5）之前，以便对比

```
sale=> ALTER TABLE SC DISABLE TRIGGER SC_AFTER_UPDATE;
ALTER TABLE
```

② 执行上面的步骤③，验证触发器被禁用后是否还能正常工作。

```
sale=> Select *FROM SC;
    sno    | cno  | grade
-----------+------+-------
 201215122 | 1    |    80
 201215121 | 1    |    70
 201215122 | 2    |    80
 201215121 | 2    |    70
(4 rows)

sale=> UPDATE SC SET GRADE=100  WHERE SNO='201215122' AND CNO='2';
UPDATE 1
sale=> UPDATE SC SET GRADE=90  WHERE SNO='201215121' AND CNO='2';
UPDATE 1
sale=> Select *FROM SC;
    sno    | cno  | grade
-----------+------+-------
 201215122 | 1    |    80
 201215121 | 1    |    70
 201215122 | 2    |   100
 201215121 | 2    |    90
(4 rows)
```

触发器被禁用后无法正常工作。

```
sale=> SELECT *FROM SC_U;
 sno | cno | oldgrade | newgrade
-----+-----+----------+----------
(0 rows)
```

（8）删除所有创建的触发器。

```
sale=> DROP TRIGGER SC_AFTER_UPDATE ON SC;
DROP TRIGGER
sale=> DROP TRIGGER Insert_Or_Update_Sal ON Teacher;
DROP TRIGGER
sale=>
```

3. 实验总结

3.1 实验思考

· 简述 openGauss 触发器的作用及适用场景。

答:

① 触发器的主要作用就是其能够实现由主键和外键所不能保证的复杂的参照完整性和数据的一致性。除此之外，触发器还有其它许多不同的功能。(1) 强化约束：触发器能够实现比 CHECK 语句更为复杂的约束。(2) 跟踪变化：触发器可以侦测数据库内的操作，从而不允许数据库中未经许可的指定更新和变化。(3) 级联运行：触发器可以侦测数据库内的操作，并自动地级联影响整个数据库的各项内容。例如，某个表上的触发器中包含有对另外一个表的数据操作(如删除，更新，插入)而该操作又导致该表上触发器被触发。(4) 存储过程的调用。

② 适用场景：（1）复杂的安全性检查:比如:禁止在非工作时间插入新员工。（2）数据库的确认:比如:涨工资,工资应该越涨越多的,如果越长越少就不叫涨工资了。（3）数据库审计:比如:跟踪表上操作的记录,比如什么时间什么人操作了数据库,操作了表上的 记录是什么等。（4）数据库的备份和同步:比如有两个数据库一个是主数据库,一个是备用数据库,在主数据库中的数据被修改了以后可以通过触发器监听,如果被修改会将修改的数据传递给备份数据库,当主数据崩溃以后不

影响数据的使用。

3.2 对实验的认识

通过实验我对 openGauss 中的一些语句更熟悉了。如

如 SELECT * FROM customer_t1;可以用来查询表 customer_t1 的所有数据。gsql -d sale -p 26000 -U yuxiaoping -W yuxiaoping@123 -r 或者 gsql -d sale -p 26000 -U user1 -W user1@123 -r 可以用来将新用户连接到数据库。可以使用 gsql -d postgres -p 26000 -r 连接到 postgres。gs_om -t start 可以开启数据库。

3.3 遇到的困难及解决方法

要更改当前会话的默认 Schema，请使用 SET 命令。执行如下命令 SET SEARCH_PATH To icebear,public;将搜索路径设置为 myschema、public，首先搜索 myschema。

```
sale=> SET SEARCH_PATH TO icebear, public;
SET
```

高斯默认有 session 超时时间，若想要 session 一直保持，需要修改配置项：ALTER DATABASE sale SET session_timeout TO 0;

```
postgres=# ALTER DATABASE postgres SET session_timeout TO 0;
ALTER DATABASE
```