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:	2023/5/19
实验地点:	文宣楼 A402
提交日期:	2023/5/24

学号:	22920212204392
姓名:	黄勖
专业年级:	软工 2021 级
学年学期:	2022-2023 学年第二学期

1.实验目的

- 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

（0）登录 ECS 服务器，以 omm 操作系统管理员身份登录数据库，使用 gsql 连接到数据库。

su - omm

gs_om -t start

gsql -d postgres -p 26000 -r

（gsql -d sales -p 26000 -U hx -W HX@123pass -r）

```
[root@ecs-hxnb ~]# su - omm
Last login: Fri May 19 16:53:17 CST 2023 on pts/0


Welcome to 4.19.90-2110.8.0.0119.oe1.aarch64

System information as of time:  Wed May 24 22:49:38 CST 2023

System load:    0.41
Processes:      148
Memory used:    10.2%
Swap used:      0.0%
Usage On:       14%
IP address:     192.168.0.99
Users online:   1



[omm@ecs-hxnb ~]$ gs_om -t start
Starting cluster.
=========================================
[SUCCESS] ecs-hxnb
2023-05-24 22:49:40.099 646e2404.1 [unknown] 281472024313872 [unknown] 0
D] WARNING:  could not create any HA TCP/IP sockets
=========================================
Successfully started.
```

（1）创建部门表 dept(deptno, deptname)，其中，

- deptno 为部门号，定长为 2 的字符型，主码
- deptname 为部门名，最大长度为 20 的变长字符型，非空

```
CREATE TABLE dept(deptno char(2), deptname VARCHAR(20) NOT NULL, PRIMARY KEY(deptno));
```

```
sales=> CREATE TABLE dept(deptno char(2), deptname VARCHAR(20) NOT NULL, PRIMARY KEY(deptno));
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "dept_pkey" for table "dept"
CREATE TABLE
```

（2）创建 Teacher 表：Teacher(ID, job, Sal, deptno)，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
- deptno 为部门号，定长为 2 的字符型，外码，引用 dept 表中的主码 deptno

```
11  CREATE TABLE Teacher(
12      ID char(5),
13      JOB VARCHAR(20) NOT NULL,
14      SAL NUMERIC(7,2),
15      deptno char(2),
16      PRIMARY KEY(ID),
17      FOREIGN KEY(deptno) REFERENCES dept(deptno)
18  );
```

```
sales=> CREATE TABLE Teacher(
sales(>     ID char(5),
sales(>     JOB VARCHAR(20) NOT NULL,
sales(>     SAL NUMERIC(7,2),
sales(>     deptno char(2),
sales(>     PRIMARY KEY(ID),
sales(>     FOREIGN KEY(deptno) REFERENCES dept(deptno)
sales(> );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "teacher_pkey" for table "teacher"
CREATE TABLE
```

（3）为 dept 表增加实验数据：（‘01’,’ CS’),(‘02’,’ SW’),(‘03’,’ MA’)；为 Teacher 表增加实验数据：（‘10001’,‘教授’,3800,’ 01’),(‘10002’,‘教授’,4100,’ 02’),(‘10003’,‘副教授’,3500,’ 01’),(‘10004’,‘助理教授’,3000,’ 03’)

```sql
INSERT INTO dept VALUES ('01','CS'), ('02','SW'), ('03','MA');
INSERT INTO Teacher VALUES
    ('10001','教授',3800,'01'),
    ('10002','教授',4100,'02'),
    ('10003','副教授',3500,'01'),
    ('10004','助理教授',3000,'03');
```

```
sales=> INSERT INTO dept VALUES ('01','CS'), ('02','SW'), ('03','MA');
INSERT 0 3
sales=> INSERT INTO Teacher VALUES
sales->     ('10001','教授',3800,'01'),
sales->     ('10002','教授',4100,'02'),
sales->     ('10003','副教授',3500,'01'),
sales->     ('10004','助理教授',3000,'03');
INSERT 0 4
```

（4）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。

```sql
CREATE FUNCTION RECORD_INSERT_OR_UPDATE_SAL()
RETURNS TRIGGER AS $INSERT_OR_UPDATE_SAL$ DECLARE
BEGIN
    IF(new.JOB='教授' AND new.sal<4000) THEN new.sal=4000;
    END IF;
RETURN NEW;
END
$INSERT_OR_UPDATE_SAL$ LANGUAGE PLPGSQL;

CREATE TRIGGER INSERT_OR_UPDATE_SAL
BEFORE INSERT OR UPDATE ON Teacher
FOR EACH ROW
EXECUTE PROCEDURE RECORD_INSERT_OR_UPDATE_SAL();
```

```
sales=> CREATE FUNCTION RECORD_INSERT_OR_UPDATE_SAL()
sales-> RETURNS TRIGGER AS $INSERT_OR_UPDATE_SAL$ DECLARE
sales$> BEGIN
sales$>     IF(new.JOB='教授' AND new.sal<4000) THEN new.sal=4000;
sales$>     END IF;
sales$> RETURN NEW;
sales$> END
sales$> $INSERT_OR_UPDATE_SAL$ LANGUAGE PLPGSQL;
CREATE FUNCTION
sales=>
sales=> CREATE TRIGGER INSERT_OR_UPDATE_SAL
sales-> BEFORE INSERT OR UPDATE ON Teacher
sales-> FOR EACH ROW
sales-> EXECUTE PROCEDURE RECORD_INSERT_OR_UPDATE_SAL();
CREATE TRIGGER
```

（5）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,'02'),('10006','教授',4000,'03');

```
INSERT INTO Teacher VALUES ('10005','教授',3999,'02'), ('10006','教授',4000,'03');
```

```
sales=> INSERT INTO Teacher VALUES ('10005','教授',3999,'02'), ('10006','教授',4000,'03');
INSERT 0 2
```

B. 更新数据:将 id 为 10002 的教授工资改为 3900。

```
UPDATE Teacher SET Sal=3900 WHERE ID='10002';
```

```
sales=> UPDATE Teacher SET Sal=3900 WHERE ID='10002';
UPDATE 1
```

可以看到，10002 的教授工资还是 4000 元，触发器被触发。

```
sales=> select * from Teacher;
  id   |    job     |   sal    | deptno
-------+------------+----------+--------
 10001 | 教授       | 3800.00 | 01
 10003 | 副教授     | 3500.00 | 01
 10004 | 助理教授   | 3000.00 | 03
 10005 | 教授       | 4000.00 | 02
 10006 | 教授       | 4000.00 | 03
 10002 | 教授       | 4000.00 | 02
(6 rows)
```

（6）查看触发器（名称和代码）；

```
SELECT tgname, tgtype, tgenabled, tgconstrrelid FROM PG_TRIGGER WHERE tgname='insert_or_update_sal';
```

```
sales=> SELECT tgname, tgtype, tgenabled, tgconstrrelid FROM PG_TRIGGER WHERE tgname='insert_or_u
pdate_sal';
        tgname         | tgtype | tgenabled | tgconstrrelid
-----------------------+--------+-----------+---------------
 insert_or_update_sal |     23 | O         |             0
(1 row)
```

（7）设计触发器自动维持表间的外码约束：删除 dept 表中 deptno 为 03 的数据后，teacher 表上引用该数据的记录也被自动删除。

```
CREATE FUNCTION RECORD_DELETE_DEPT_TEACHER()
RETURNS TRIGGER AS $DELETE_DEPT_TEACHER$ DECLARE
BEGIN
    IF(old.deptno='03') THEN DELETE FROM Teacher WHERE deptno='03';
    END IF;
RETURN OLD;
END
$DELETE_DEPT_TEACHER$ LANGUAGE PLPGSQL;

CREATE TRIGGER DELETE_DEPT_TEACHER
BEFORE DELETE ON dept
FOR EACH ROW
EXECUTE PROCEDURE RECORD_DELETE_DEPT_TEACHER();
```

```
sales=> CREATE FUNCTION RECORD_DELETE_DEPT_TEACHER()
sales-> RETURNS TRIGGER AS $DELETE_DEPT_TEACHER$ DECLARE
sales$> BEGIN
sales$>     IF(old.deptno='03') THEN DELETE FROM Teacher WHERE deptno='03';
sales$>     END IF;
sales$> RETURN OLD;
sales$> END
sales$> $DELETE_DEPT_TEACHER$ LANGUAGE PLPGSQL;
CREATE FUNCTION
sales=>
sales=> CREATE TRIGGER DELETE_DEPT_TEACHER
sales-> BEFORE DELETE ON dept
sales-> FOR EACH ROW
sales-> EXECUTE PROCEDURE RECORD_DELETE_DEPT_TEACHER();
CREATE TRIGGER
```

查看原数据：

```
sales=> SELECT * FROM dept;
 deptno | deptname
--------+----------
 01     | CS
 02     | SW
 03     | MA
(3 rows)

sales=> SELECT * FROM Teacher;
  id   |   job    |   sal   | deptno
-------+----------+---------+--------
 10001 | 教授     | 3800.00 | 01
 10003 | 副教授   | 3500.00 | 01
 10004 | 助理教授 | 3000.00 | 03
 10005 | 教授     | 4000.00 | 02
 10006 | 教授     | 4000.00 | 03
 10002 | 教授     | 4000.00 | 02
(6 rows)
```

在 dept 表删除相应数据，可见 teacher 中的相关数据被同步修改了

```
sales=> DELETE FROM dept WHERE deptno='03';
DELETE 1
sales=> SELECT * FROM dept;
 deptno | deptname
--------+----------
 01     | CS
 02     | SW
(2 rows)

sales=> SELECT * FROM Teacher;
  id    |   job    |   sal    | deptno
--------+--------+---------+--------
 10001 | 教授    | 3800.00 | 01
 10003 | 副教授  | 3500.00 | 01
 10005 | 教授    | 4000.00 | 02
 10002 | 教授    | 4000.00 | 02
(4 rows)
```

（8）设计触发器实现审计日志记录（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。

完成教材工作：

```
sales=> CREATE TABLE Student(
sales(>     Sno CHAR(9) PRIMARY KEY,
sales(>     Sname CHAR(20) UNIQUE,
sales(>     Ssex CHAR(2),
sales(>     Sage SMALLINT,
sales(>     Sdept CHAR(20)
sales(> );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "student_pkey" for table "student"
NOTICE:  CREATE TABLE / UNIQUE will create implicit index "student_sname_key" for table "student"
CREATE TABLE
```

```
sales=> CREATE TABLE Course(
sales(>     Cno CHAR(4) PRIMARY KEY,
sales(>     Cname CHAR(40) NOT NULL,
sales(>     Cpno CHAR(4),
sales(>     Ccredit SMALLINT,
sales(>     FOREIGN KEY(Cpno) REFERENCES Course(Cno)
sales(> );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "course_pkey" for table "course"
CREATE TABLE
sales=> CREATE TABLE SC(
sales(>     Sno CHAR(9),
sales(>     Cno CHAR(4),
sales(>     Grade SMALLINT,
sales(>     PRIMARY KEY(Sno,Cno),
sales(>     FOREIGN KEY(Sno) REFERENCES Student(Sno),
sales(>     FOREIGN KEY(Cno) REFERENCES Course(Cno)
sales(> );
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "sc_pkey" for table "sc"
CREATE TABLE
```

```
sales=> INSERT INTO Student(Sno, Sname, Sage, Sdept) VALUES
sales->     ('201215121', '李勇',  20, 'CS'),
sales->     ('201215122', '刘晨', 19, 'CS'),
sales->     ('201215123', '王敏', 18, 'MA'),
sales->     ('201215125', '张立', 19, 'IS');
INSERT 0 4
sales=> INSERT INTO Course(Cno, Cname, Cpno, Ccredit) VALUES
sales->     ('1', '数据库', '5', 4),
sales->     ('2', '数学', NULL, 2),
sales->     ('3', '信息系统', '1', 4),
sales->     ('4', '操作系统', '6', 3),
sales->     ('5', '数据结构', '7', 4),
sales->     ('6', '数据处理', NULL, 2),
sales->     ('7', 'PASCAL语言', '6', 4);
INSERT 0 7
sales=> INSERT INTO SC(Sno, Cno, Grade) VALUES
sales->     ('201215121', '1', 92),
sales->     ('201215121', '2', 85),
sales->     ('201215121', '3', 88),
sales->     ('201215122', '2', 90),
sales->     ('201215122', '3', 80);
INSERT 0 5
```

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

- Sno 的数据类型：定长为 9 的字符型，外码，引用 Student 表中 Sno 的值
- Cno 的数据类型：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

```
CREATE TABLE SC_U(
    Sno CHAR(9),
    Cno CHAR(4),
    Oldgrade SMALLINT,
    Newgrade SMALLINT,
    FOREIGN KEY(Sno) REFERENCES Student(Sno),
    FOREIGN KEY(Cno) REFERENCES Course(Cno)
);
```

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

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

```
CREATE FUNCTION RECORD_tri_update_sc()
RETURNS TRIGGER AS $tri_update_sc$ DECLARE
BEGIN
    IF(new.Grade>=1.1*old.Grade)
    THEN INSERT INTO SC_U(Sno, Cno, Oldgrade, Newgrade) VALUES(old.Sno, old.Cno, old.Grade, new.Grade);
    END IF;
RETURN NEW;
END
$tri_update_sc$ LANGUAGE PLPGSQL;

CREATE TRIGGER tri_update_sc
AFTER UPDATE OF Grade ON SC
FOR EACH ROW
EXECUTE PROCEDURE RECORD_tri_update_sc();
```

```
sales=> CREATE FUNCTION RECORD_tri_update_sc()
sales-> RETURNS TRIGGER AS $tri_update_sc$ DECLARE
sales$> BEGIN
sales$>     IF(new.Grade>=1.1*old.Grade)
sales$>     THEN INSERT INTO SC_U(Sno, Cno, Oldgrade, Newgrade) VALUES(old.Sno, old.Cno, old.Grad
e, new.Grade);
sales$>     END IF;
sales$> RETURN NEW;
sales$> END
sales$> $tri_update_sc$ LANGUAGE PLPGSQL;
CREATE FUNCTION
sales=>
sales=> CREATE TRIGGER tri_update_sc
sales-> AFTER UPDATE OF Grade ON SC
sales-> FOR EACH ROW
sales-> EXECUTE PROCEDURE RECORD_tri_update_sc();
CREATE TRIGGER
```

③ 验证 tri_update_sc 触发器是否正常工作（测试数据同教材）。

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

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

要求：执行上述两种操作，如果触发器正确工作，请观察 SC_U 表中

数据的变化。

```
sales=> UPDATE SC SET Grade=100 WHERE Sno='201215122' AND Cno='2';
UPDATE 1
sales=> UPDATE SC SET Grade=90 WHERE Sno='201215121' AND Cno='2';
UPDATE 1
sales=> SELECT * FROM SC_U;
    sno     | cno  | oldgrade | newgrade
------------+------+----------+----------
 201215122 | 2    |       90 |      100
(1 row)
```

可以看到从 85->90 未被记录，而从 90->100 的数据则被记录到 sc_u 表里，触发器正常工作

（9）将触发器 tri_update_sc 改名为 update_sc_tri;

```
ALTER TRIGGER tri_update_sc ON SC RENAME TO update_sc_tri;
```

```
sales=> ALTER TRIGGER tri_update_sc ON SC RENAME TO update_sc_tri;
ALTER TRIGGER
```

（10）验证触发器禁用后效果

① 将数据还原到步骤(5)之前，即触发器工作前的原数据；

```
UPDATE SC SET Grade=90 WHERE Sno='201215122' AND Cno='2';
UPDATE SC SET Grade=85 WHERE Sno='201215121' AND Cno='2';
DELETE FROM SC_U;
```

```
sales=> UPDATE SC SET Grade=90 WHERE Sno='201215122' AND Cno='2';
UPDATE 1
sales=> UPDATE SC SET Grade=85 WHERE Sno='201215121' AND Cno='2';
UPDATE 1
sales=> DELETE FROM SC_U;
DELETE 1
```

② 修改 SC 表使 AFTER_UPDATE_SC 触发器失效；

```
ALTER TABLE SC DISABLE TRIGGER update_sc_tri;
```

```
sales=> ALTER TABLE SC DISABLE TRIGGER update_sc_tri;
ALTER TABLE
```

③ 再次执行上面的步骤③，验证触发器被禁用后是否还能正常工作？

```
UPDATE SC SET Grade=100 WHERE Sno='201215122' AND Cno='2';
UPDATE SC SET Grade=90 WHERE Sno='201215121' AND Cno='2';
```

```
sales=> UPDATE SC SET Grade=100 WHERE Sno='201215122' AND Cno='2';
UPDATE 1
sales=> UPDATE SC SET Grade=90 WHERE Sno='201215121' AND Cno='2';
UPDATE 1
```

```
sales=> select * from sc_u;
 sno | cno | oldgrade | newgrade
-----+-----+----------+----------
(0 rows)
```

sc_u 表中没有任何数据，可见禁用掉触发器后不会再起作用。

（11）删除所创建的触发器。

```
DROP TRIGGER INSERT_OR_UPDATE_SAL ON Teacher;
DROP FUNCTION RECORD_INSERT_OR_UPDATE_SAL();
DROP TRIGGER DELETE_DEPT_TEACHER ON dept;
DROP FUNCTION RECORD_DELETE_DEPT_TEACHER();
DROP TRIGGER update_sc_tri ON SC;
DROP FUNCTION RECORD_tri_update_sc();
```

```
sales=> DROP TRIGGER INSERT_OR_UPDATE_SAL ON Teacher;
DROP TRIGGER
sales=> DROP FUNCTION RECORD_INSERT_OR_UPDATE_SAL();
DROP FUNCTION
sales=> DROP TRIGGER DELETE_DEPT_TEACHER ON dept;
DROP TRIGGER
sales=> DROP FUNCTION RECORD_DELETE_DEPT_TEACHER();
DROP FUNCTION
sales=> DROP TRIGGER update_sc_tri ON SC;
DROP TRIGGER
sales=> DROP FUNCTION RECORD_tri_update_sc();
DROP FUNCTION
```

3.实验总结

3.1 完成的工作

设计正确的 openGauss SQL 语句并完成了所有实验要求。

3.2 对实验的认识

（1）简述 openGauss 触发器的作用及适用场景。

openGauss 是一种开源的关系型数据库管理系统，它是在 PostgreSQL 基础上开发而成的。在 openGauss 中，触发器（Triggers）是一种数据库对象，用于在特定事件发生时自动执

行一系列定义好的操作。

触发器的主要作用是在数据库中的表发生特定的数据操作时（如插入、更新或删除数据），触发器会自动被激活并执行相应的操作。触发器可以用于以下几个方面：

1. 数据完整性维护：通过在数据操作前后执行一些验证操作，触发器可以帮助确保数据的完整性。例如，可以使用触发器在插入或更新数据之前验证数据的有效性或完整性。

2. 日志记录和审计：触发器可以用于在数据库表的数据发生变化时记录相关的日志信息。这对于追踪和审计数据库操作非常有用，可以记录谁、什么时间和做了什么样的数据操作。

3. 数据复制和同步：通过在触发器中执行适当的逻辑，可以实现数据库的数据复制和同步。触发器可以在主数据库上捕获数据变更事件，并将这些事件传递给从数据库，从而确保数据的一致性。

4. 数据转换和处理：触发器可以用于对数据库中的数据进行转换和处理。例如，在数据插入之前，可以使用触发器对数据进行预处理或格式化，或者在数据更新之后，根据特定的规则对数据进行调整或计算。

适用场景包括但不限于：

1. 数据约束和验证：当需要在数据操作前后进行验证、强制执行一些约束规则时，可以使用触发器。例如，在某个表中有金额字段，可以通过触发器确保金额不会超出特定范围。

2. 日志记录和审计：当需要记录和追踪数据变更事件时，可以使用触发器。例如，可以在触发器中将相关信息写入日志表，用于后续审计或分析。

3. 数据复制和同步：当需要在多个数据库之间进行数据复制和同步时，可以使用触发器。触发器可以捕获主数据库上的数据变更事件，并将其传递给从数据库，以确保数据的一致性。

4. 数据转换和处理：当需要在数据操作前后对数据进行转换或处理时，可以使用触发器。例如，可以在触发器中对特定字段进行格式化、计算或更新。

总之，openGauss 触发器提供了一种灵活且强大的机制，可以在数据库中定义和管理各种数据操作的自动化行为。通过使用触发器，可以实现数据完整性维护、日志记录、数据复制、数据转换等功能，以满足不同的业务需求。

（2）收获

openGauss 数据库触发器是一项非常有用的功能，它需要与函数结合使用。尽管这可能看起来比较复杂，但它提供了一种处理各种事务的强大语法。在本次实验中，我熟悉了触发器的相关知识，这将对我以后处理更多更复杂的数据提供帮助。

3.3 遇到的困难及解决方法

在理解 openGauss 数据库触发器的使用过程中，我花费了很多时间，因为其语法与课本上的不一致。不过，通过查阅 openGauss 触发器的参考文档和老师提供的链接，我最终对 openGauss 数据库触

发器的使用有了初步的认识和理解。这帮助我解决了相关问题。