EXPERIMENT - X
TRIGGER AND EXCEPTION HANDLING

August 18, 2019

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

AIM

To study PL/SQL trigger and exception handling.

TRIGGER

Triggers are procedures that are stored in the database and implicitly run, or fired, when something happens.

Syntax

TRIGGER <trigger name>
BEFORE/AFTER/INSTEAD OF INSERT/DELETE/UPDATE
OF <column name> ON <table name>
FOR EACH ROW WHEN <condition>
<pl/sql block with DECLARE–BEGIN–END constructs>

EXCEPTION

Exceptions are used to handle run time errors in program.

Syntax

BEGIN
<executable statements>
EXCEPTION
<exception handling>
END;

QUESTIONS

Create a table customer_details (cust_id (unique) ,cust_name, address). Create a table emp_details(empid(unique), empname, salary). Create table cust_count(count_row).
Write PL/SQL programs for the following:

1. Create a trigger whenever a new record is inserted in the customer _details table.

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION TRIG1()
postgres-# RETURNS TRIGGER AS
postgres-# $trigger1$
postgres$# BEGIN
postgres$# RAISE NOTICE 'A Row is Inserted';
postgres$# RETURN NEW;
postgres$# END;
postgres$# $trigger1$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE TRIGGER trigger1
postgres-# AFTER INSERT ON CUSTOMER_DETAILS
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE TRIG1();
CREATE TRIGGER
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM CUSTOMER_DETAILS;
 cust_id | cust_name | address
---------+-----------+---------
(0 rows)

postgres=# INSERT INTO CUSTOMER_DETAILS
postgres-# VALUES(1,'John','Ezhaparambbil');
NOTICE:  A Row is Inserted
INSERT 0 1
postgres=# SELECT * FROM CUSTOMER_DETAILS;
 cust_id | cust_name |    address
---------+-----------+---------------
       1 | John      | Ezhaparambbil
(1 row)

postgres=#
```

2. Create a trigger to display a message when a user enters a value > 20000 in the salary field of emp_details table.

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION TRIG2()
postgres-# RETURNS TRIGGER AS
postgres-# $trigger2$
postgres$# BEGIN
postgres$# IF(NEW.SALARY>20000) THEN
postgres$# RAISE NOTICE 'Employee has salary greater than 20000/-';
postgres$# END IF;
postgres$# RETURN NEW;
postgres$# END;
postgres$# $trigger2$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE TRIGGER trigger2
postgres-# AFTER INSERT ON EMP_DETAILS
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE TRIG2();
CREATE TRIGGER
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM EMP_DETAILS;
 empid | empname | salary
-------+---------+--------
(0 rows)

postgres=# INSERT INTO EMP_DETAILS
postgres-# VALUES(1,'John',25000);
NOTICE:  Employee has salary greater than 20000/-
INSERT 0 1
postgres=# SELECT * FROM EMP_DETAILS;
 empid | empname | salary
-------+---------+--------
     1 | John    |  25000
(1 row)

postgres=#
```

3. Create a trigger w.r.tcustomer_detailstable.Increment the value of count_row (in cust_count table) whenever a new tuple is inserted and decrement the value of count_row when a tuple is deleted. Initial value of the count_row is set to 0.

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION TRIG3()
postgres-# RETURNS TRIGGER AS
postgres-# $trigger3$
postgres$# BEGIN
postgres$# IF (TG_OP = 'DELETE') THEN
postgres$# UPDATE CUST_COUNT
postgres$# SET COUNT_ROW=COUNT_ROW-1;
postgres$# ELSE
postgres$# UPDATE CUST_COUNT
postgres$# SET COUNT_ROW=COUNT_ROW+1;
postgres$# END IF;
postgres$# RETURN NULL;
postgres$# END;
postgres$# $trigger3$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE TRIGGER trigger3
postgres-# BEFORE INSERT OR DELETE ON CUSTOMER_DETAILS
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE TRIG3();
CREATE TRIGGER
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM CUST_COUNT;
 count_row
-----------
         0
(1 row)

postgres=# INSERT INTO CUSTOMER_DETAILS
VALUES(1,'John','Ezhaparambbil');
INSERT 0 0
postgres=# INSERT INTO CUSTOMER_DETAILS
VALUES(2,'Pretty','Thenganachalil');
INSERT 0 0
postgres=# SELECT * FROM CUST_COUNT;
 count_row
-----------
         2
(1 row)

postgres=# DELETE FROM CUSTOMER_DETAILS
postgres-# WHERE CUST_ID=1;
DELETE 0
postgres=# SELECT * FROM CUST_COUNT;
 count_row
-----------
         1
(1 row)

postgres=#
```

4. Create a trigger to insert the deleted rows from emp_details to another table and updated rows to another table. (Create the tables deleted and updated)

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION TRIG4()
postgres-# RETURNS TRIGGER AS
postgres-# $trigger4$
postgres$# BEGIN
postgres$# IF (TG_OP = 'DELETE') THEN
postgres$# INSERT INTO DELETED
postgres$# VALUES(OLD.EMPID,OLD.EMPNAME,OLD.SALARY);
postgres$# ELSE
postgres$# INSERT INTO UPDATED
postgres$# VALUES(NEW.EMPID,NEW.EMPNAME,NEW.SALARY);
postgres$# END IF;
postgres$# RETURN NULL;
postgres$# END;
postgres$# $trigger4$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE TRIGGER trigger4
postgres-# BEFORE DELETE OR UPDATE ON EMP_DETAILS
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE TRIG4();
CREATE TRIGGER
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM EMP_DETAILS;
 empid | empname | salary
-------+---------+--------
     1 | John    |  25000
     4 | John    |   2000
(2 rows)

postgres=# UPDATE EMP_DETAILS
postgres-# SET SALARY=SALARY+20000
postgres-# WHERE EMPID=4;
UPDATE 0
postgres=# SELECT * FROM UPDATED;
 empid | empname | salary
-------+---------+--------
     4 | John    |  22000
(1 row)

postgres=# DELETE FROM EMP_DETAILS
postgres-# WHERE EMPID=1;
DELETE 0
postgres=# SELECT * FROM DELETED;
 empid | empname | salary
-------+---------+--------
     1 | John    |  25000
(1 row)

postgres=#
```

5. Write a PL/SQL to show divide by zero exception

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION DIVIDE(A INT,B INT)
postgres-# RETURNS NUMERIC AS
postgres-# $$
postgres$# BEGIN
postgres$# IF (b=0) THEN
postgres$# RAISE EXCEPTION 'Enter another divisor';
postgres$# ELSE
postgres$# RETURN a/b;
postgres$# END IF;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT

```
postgres=# SELECT DIVIDE(9,3);
 divide
--------
      3
(1 row)

postgres=# SELECT DIVIDE(9,0);
ERROR:  Enter another divisor
```

6. Write a PL/SQL to show no data found exception

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION FINDDATA()
postgres-# RETURNS INT AS
postgres-# $$
postgres$# DECLARE
postgres$# temp CURSOR FOR SELECT * FROM DETAILS;
postgres$# tempvar RECORD;
postgres$# BEGIN
postgres$# OPEN temp;
postgres$# LOOP
postgres$# FETCH temp INTO tempvar;
postgres$# IF NOT FOUND THEN
postgres$# RAISE EXCEPTION 'Data Not Found';
postgres$# ELSE
postgres$# RETURN tempvar.id;
postgres$# END IF;
postgres$# END LOOP;
postgres$# CLOSE temp;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM DETAILS;
 id | name
----+------
(0 rows)

postgres=# SELECT FINDDATA();
ERROR:  Data Not Found
CONTEXT:  PL/pgSQL function finddata() line 10 at RAISE
postgres=# INSERT INTO DETAILS VALUES(1,'Hello');
INSERT 0 1
postgres=# SELECT FINDDATA();
 finddata
----------
        1
(1 row)

postgres=#
```

7. Create a table with ebill(cname,prevreading,currreading). If prevreading = currreading then raise an exception 'Data Entry Error'.

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION TRIG7()
postgres-# RETURNS TRIGGER AS
postgres-# $trigger7$
postgres$# BEGIN
postgres$# IF (NEW.PREVREADING=NEW.CURRREADING) THEN
postgres$# RAISE EXCEPTION 'Data Entry Error';
postgres$# ELSE
postgres$# RETURN NEW;
postgres$# END IF;
postgres$# END;
postgres$# $trigger7$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=# CREATE TRIGGER trigger7
postgres-# BEFORE INSERT ON EBILL
postgres-# FOR EACH ROW
postgres-# EXECUTE PROCEDURE TRIG7();
CREATE TRIGGER
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM EBILL;
 cname | prevreading | currreading
-------+-------------+-------------
(0 rows)

postgres=# INSERT INTO EBILL
postgres-# VALUES('Melvy',4,4);
ERROR:  Data Entry Error
CONTEXT:  PL/pgSQL function trig7() line 4 at RAISE
postgres=# INSERT INTO EBILL
postgres-# VALUES('Melvy',7,8);
INSERT 0 1
postgres=# SELECT * FROM EBILL;
 cname | prevreading | currreading
-------+-------------+-------------
 Melvy |           7 |           8
(1 row)

postgres=#
```

RESULT

The PL/SQL program was executed successfully and the output was obtained.