Exp No. 11 Date : 23-08-2019

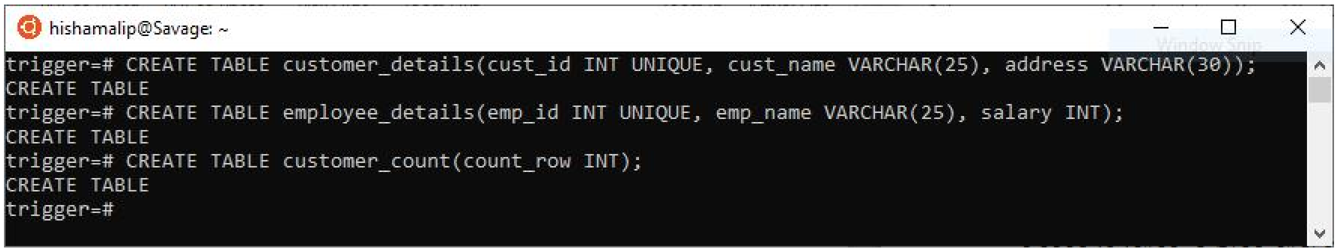
TRIGGER AND EXCEPTION HANDLING

AIM:

To study PL/SQL trigger and exception handling.

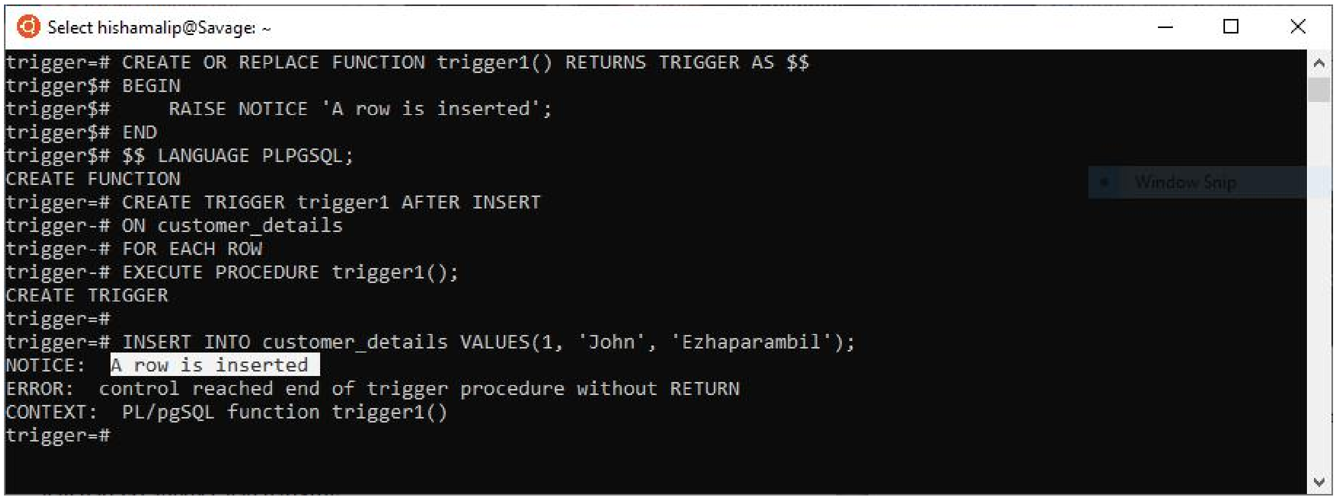
QUESTIONS:

Create a table customer_details (cust_id (unique) , cust_name, address).
Create a table employee_details (emp_id(unique), emp_name, salary)
Create table cust_count (count_row)



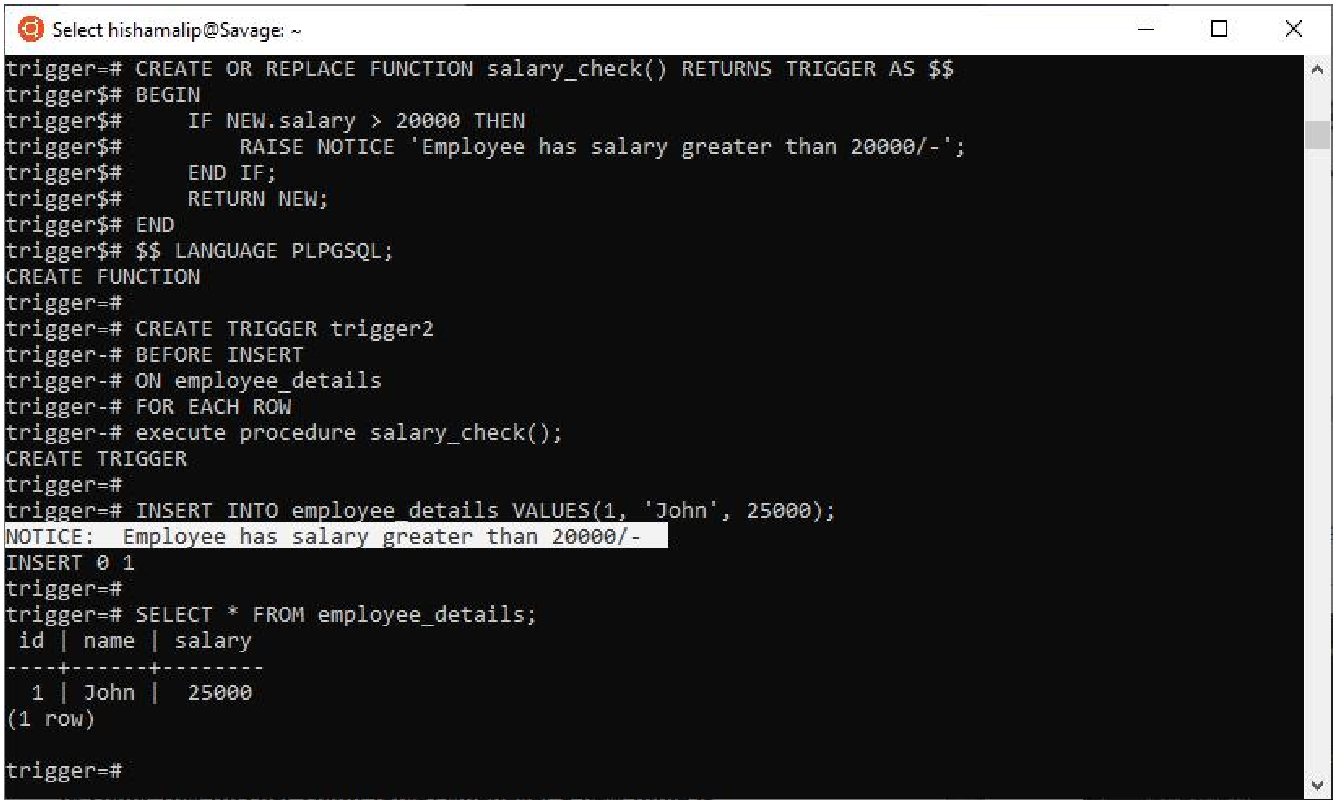
```
trigger=# CREATE TABLE customer_details(cust_id INT UNIQUE, cust_name VARCHAR(25), address VARCHAR(30));
CREATE TABLE
trigger=# CREATE TABLE employee_details(emp_id INT UNIQUE, emp_name VARCHAR(25), salary INT);
CREATE TABLE
trigger=# CREATE TABLE customer_count(count_row INT);
CREATE TABLE
trigger=#
```

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



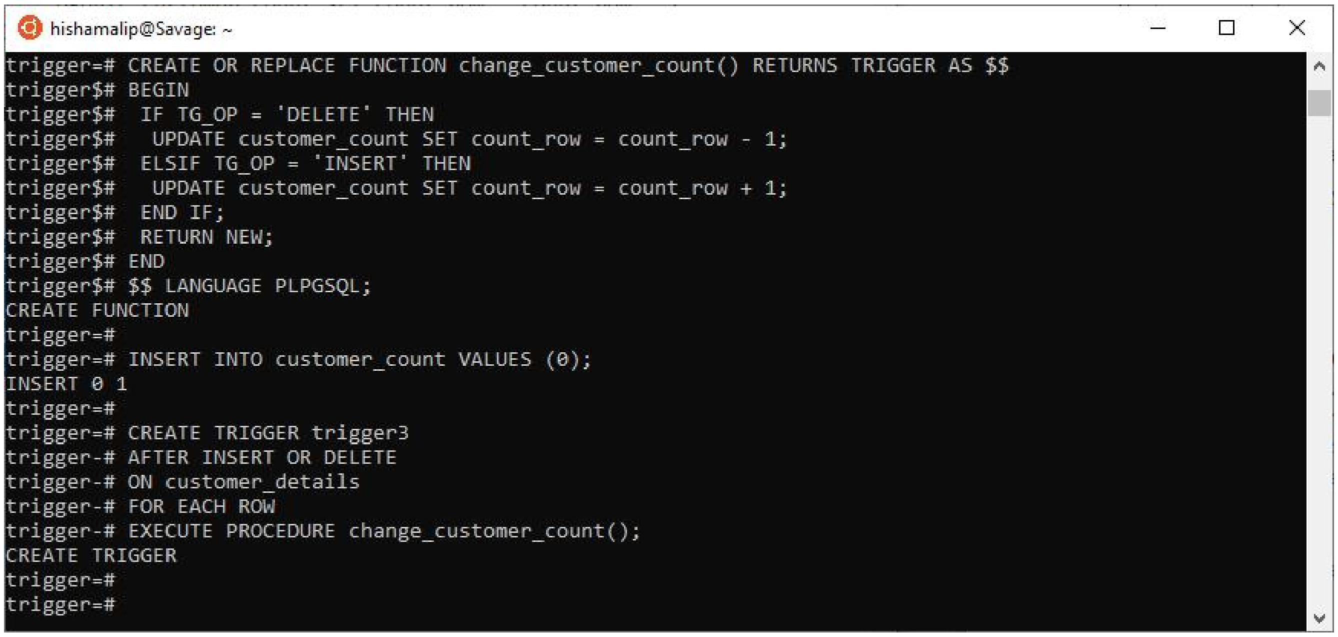
```
trigger=# CREATE OR REPLACE FUNCTION trigger1() RETURNS TRIGGER AS $$
trigger$# BEGIN
trigger$#     RAISE NOTICE 'A row is inserted';
trigger$# END
trigger$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
trigger=# CREATE TRIGGER trigger1 AFTER INSERT
trigger-# ON customer_details
trigger-# FOR EACH ROW
trigger-# EXECUTE PROCEDURE trigger1();
CREATE TRIGGER
trigger=#
trigger=# INSERT INTO customer_details VALUES(1, 'John', 'Ezhaparambil');
NOTICE:  A row is inserted
ERROR:  control reached end of trigger procedure without RETURN
CONTEXT:  PL/pgSQL function trigger1()
trigger=#
```

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

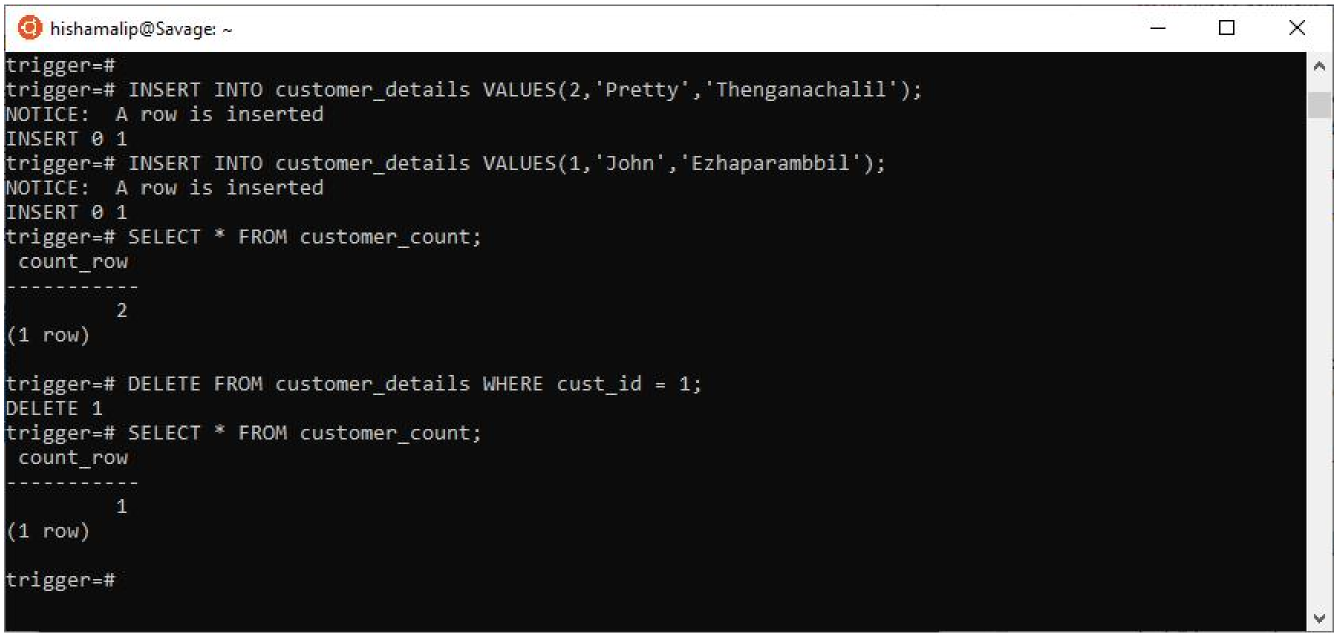


```
trigger=# CREATE OR REPLACE FUNCTION salary_check() RETURNS TRIGGER AS $$
trigger$# BEGIN
trigger$#     IF NEW.salary > 20000 THEN
trigger$#         RAISE NOTICE 'Employee has salary greater than 20000/-';
trigger$#     END IF;
trigger$#     RETURN NEW;
trigger$# END
trigger$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
trigger=#
trigger=# CREATE TRIGGER trigger2
trigger-# BEFORE INSERT
trigger-# ON employee_details
trigger-# FOR EACH ROW
trigger-# execute procedure salary_check();
CREATE TRIGGER
trigger=#
trigger=# INSERT INTO employee_details VALUES(1, 'John', 25000);
NOTICE:  Employee has salary greater than 20000/-
INSERT 0 1
trigger=#
trigger=# SELECT * FROM employee_details;
 id | name | salary
----+------+--------
  1 | John |  25000
(1 row)

trigger=#
```

3. Create a trigger with respect to customer_details table. Increment the value of count_row (in customer_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.

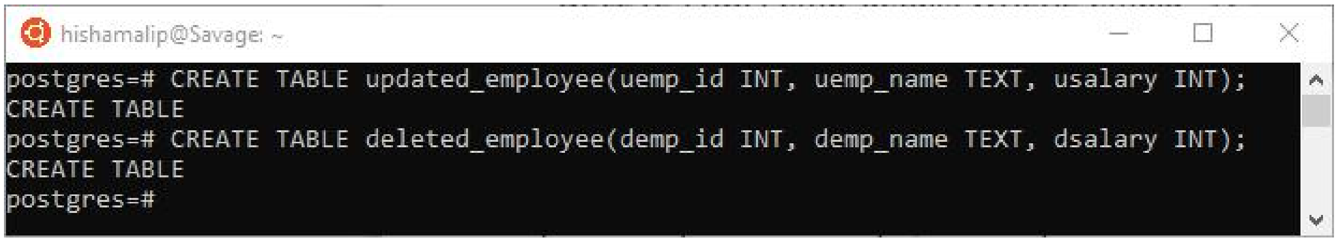


```
trigger=# CREATE OR REPLACE FUNCTION change_customer_count() RETURNS TRIGGER AS $$
trigger$# BEGIN
trigger$#  IF TG_OP = 'DELETE' THEN
trigger$#   UPDATE customer_count SET count_row = count_row - 1;
trigger$#  ELSIF TG_OP = 'INSERT' THEN
trigger$#   UPDATE customer_count SET count_row = count_row + 1;
trigger$#  END IF;
trigger$#  RETURN NEW;
trigger$# END
trigger$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
trigger=#
trigger=# INSERT INTO customer_count VALUES (0);
INSERT 0 1
trigger=#
trigger=# CREATE TRIGGER trigger3
trigger-# AFTER INSERT OR DELETE
trigger-# ON customer_details
trigger-# FOR EACH ROW
trigger-# EXECUTE PROCEDURE change_customer_count();
CREATE TRIGGER
trigger=#
trigger=#
```



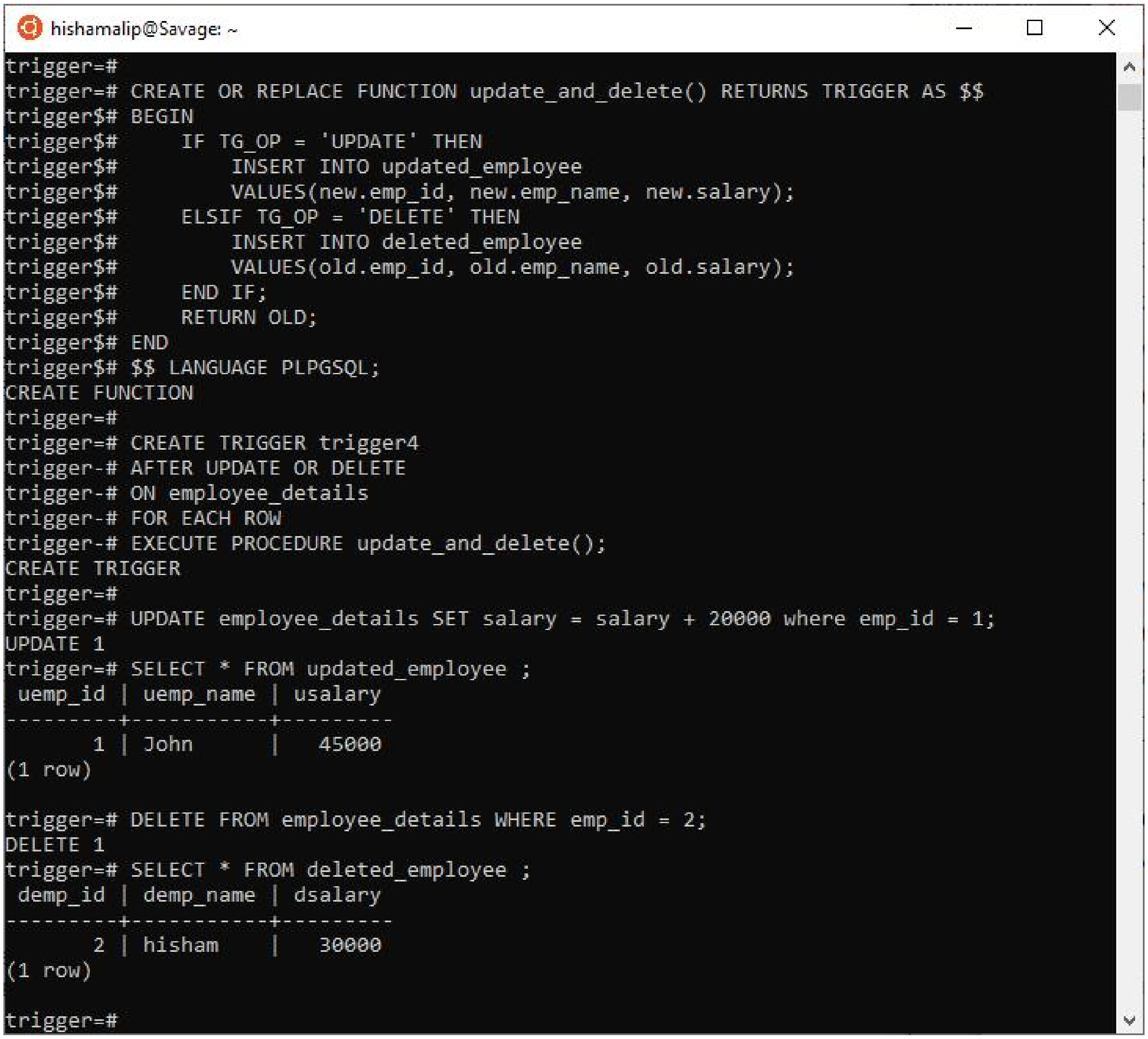
```
trigger=#
trigger=# INSERT INTO customer_details VALUES(2,'Pretty','Thenganachalil');
NOTICE:  A row is inserted
INSERT 0 1
trigger=# INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
NOTICE:  A row is inserted
INSERT 0 1
trigger=# SELECT * FROM customer_count;
 count_row
-----------
         2
(1 row)

trigger=# DELETE FROM customer_details WHERE cust_id = 1;
DELETE 1
trigger=# SELECT * FROM customer_count;
 count_row
-----------
         1
(1 row)

trigger=#
```

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



hishamalip@Savage: ~

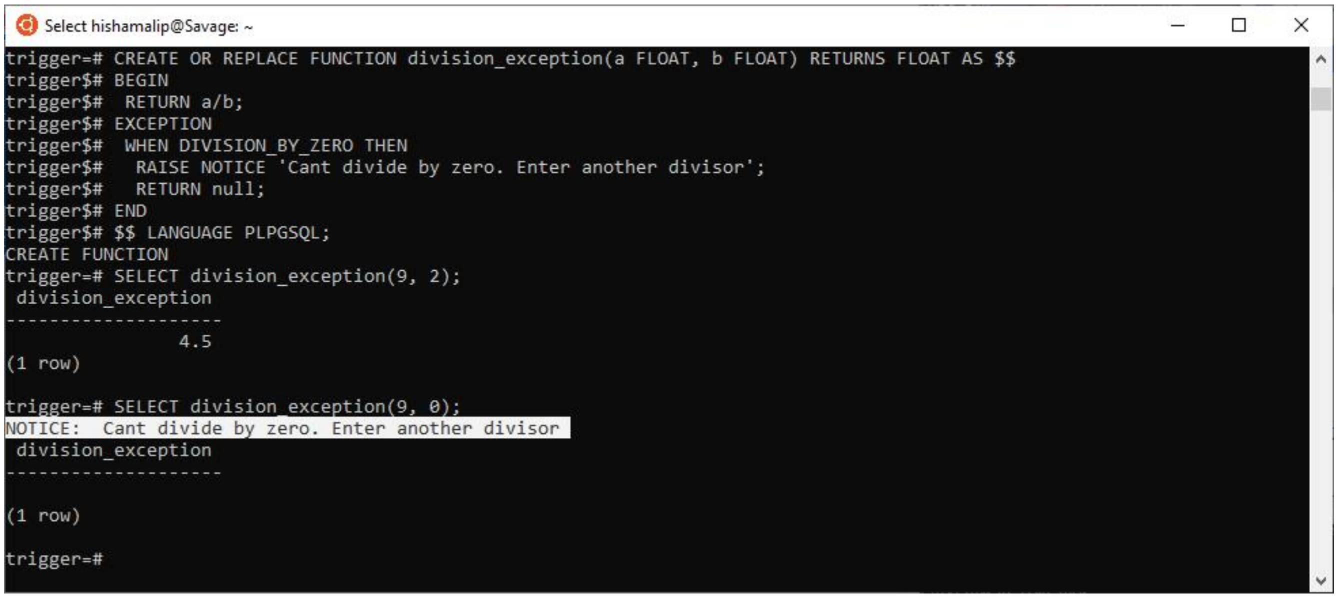
```
postgres=# CREATE TABLE updated_employee(uemp_id INT, uemp_name TEXT, usalary INT);
CREATE TABLE
postgres=# CREATE TABLE deleted_employee(demp_id INT, demp_name TEXT, dsalary INT);
CREATE TABLE
postgres=#
```



hishamalip@Savage: ~

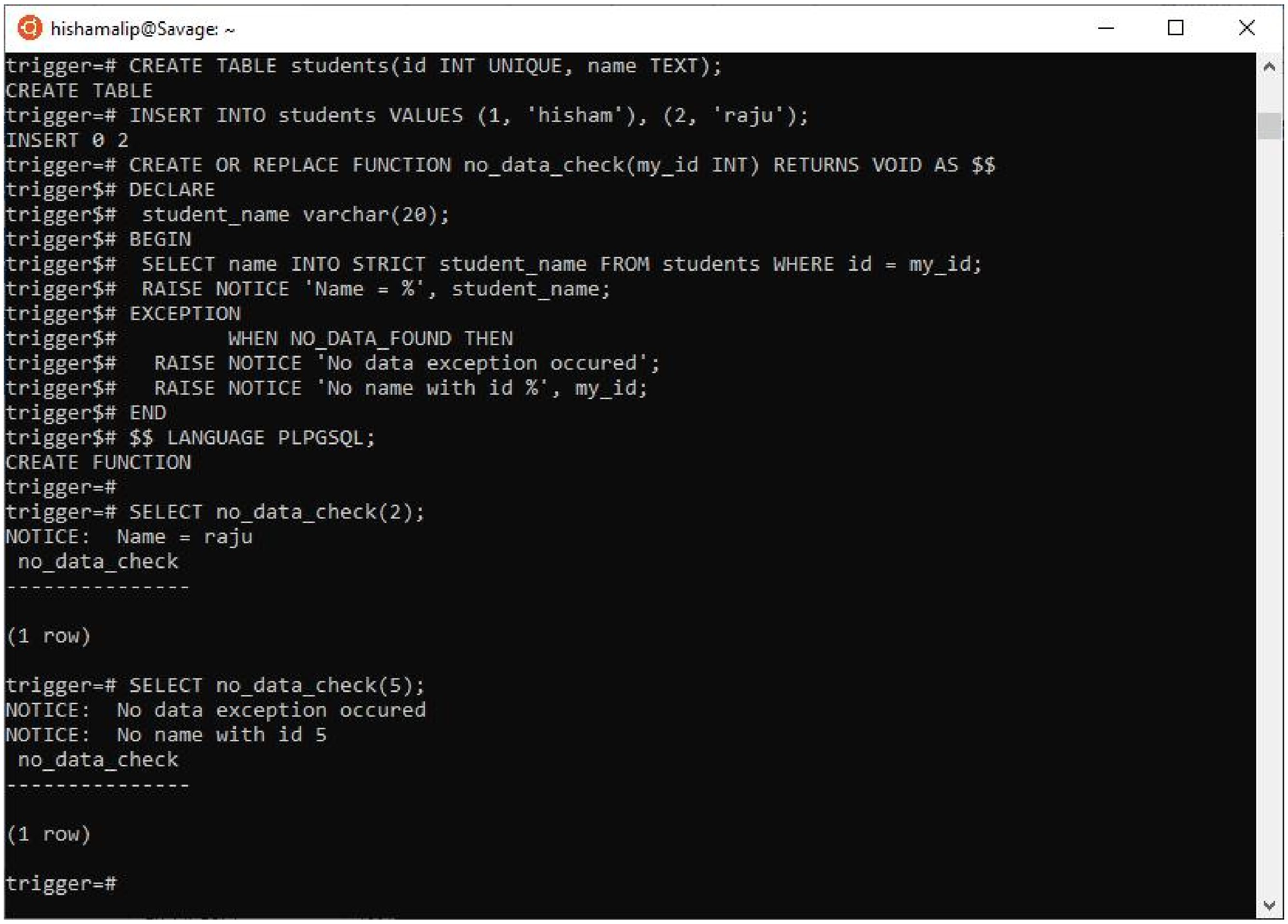
```
trigger=#
trigger=# CREATE OR REPLACE FUNCTION update_and_delete() RETURNS TRIGGER AS $$
trigger$# BEGIN
trigger$#     IF TG_OP = 'UPDATE' THEN
trigger$#         INSERT INTO updated_employee
trigger$#         VALUES(new.emp_id, new.emp_name, new.salary);
trigger$#     ELSIF TG_OP = 'DELETE' THEN
trigger$#         INSERT INTO deleted_employee
trigger$#         VALUES(old.emp_id, old.emp_name, old.salary);
trigger$#     END IF;
trigger$#     RETURN OLD;
trigger$# END
trigger$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
trigger=#
trigger=# CREATE TRIGGER trigger4
trigger-# AFTER UPDATE OR DELETE
trigger-# ON employee_details
trigger-# FOR EACH ROW
trigger-# EXECUTE PROCEDURE update_and_delete();
CREATE TRIGGER
trigger=#
trigger=# UPDATE employee_details SET salary = salary + 20000 where emp_id = 1;
UPDATE 1
trigger=# SELECT * FROM updated_employee ;
 uemp_id | uemp_name | usalary
---------+-----------+---------
       1 | John      |   45000
(1 row)

trigger=# DELETE FROM employee_details WHERE emp_id = 2;
DELETE 1
trigger=# SELECT * FROM deleted_employee ;
 demp_id | demp_name | dsalary
---------+-----------+---------
       2 | hisham    |   30000
(1 row)

trigger=#
```

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



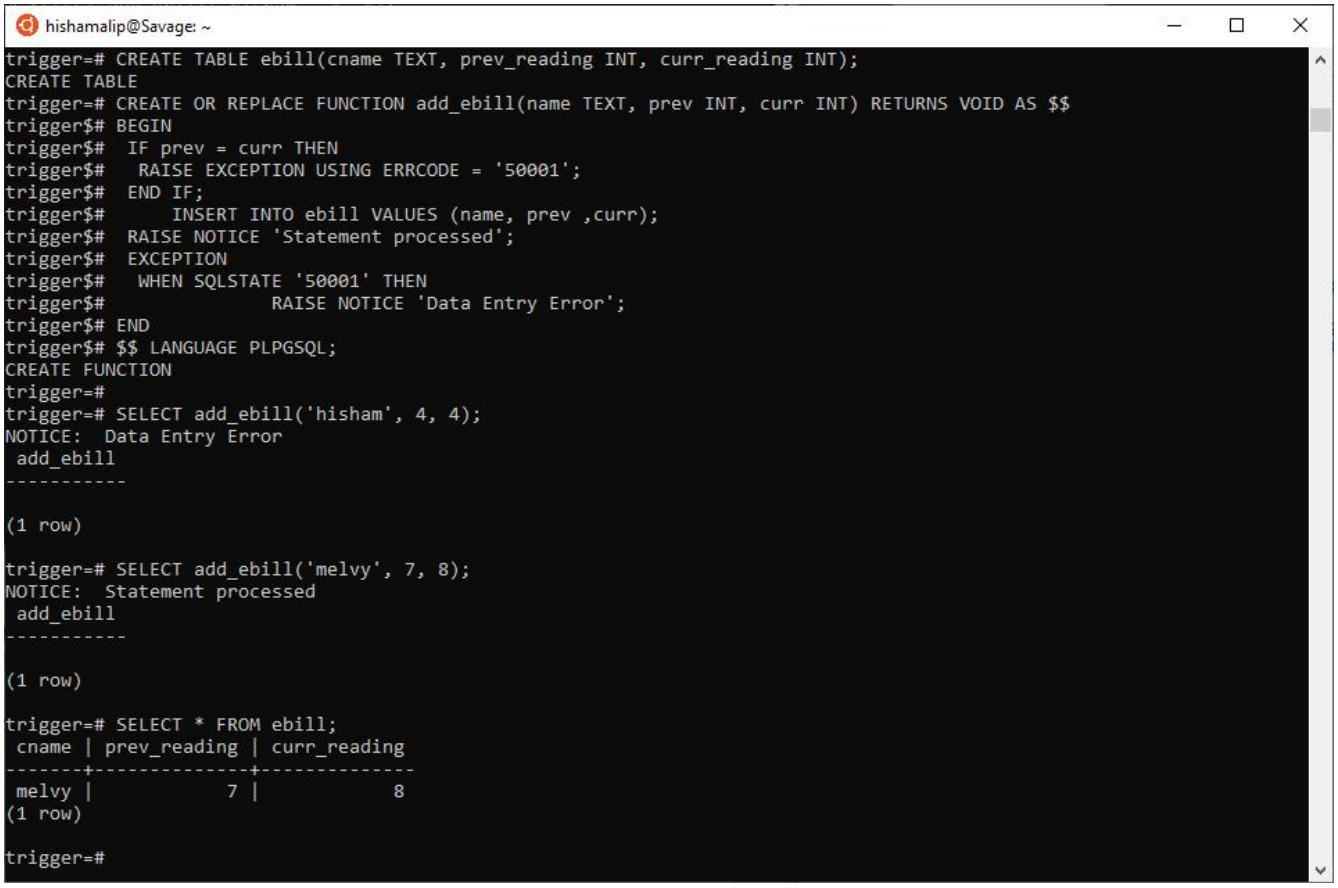
```
trigger=# CREATE OR REPLACE FUNCTION division_exception(a FLOAT, b FLOAT) RETURNS FLOAT AS $$
trigger$# BEGIN
trigger$#  RETURN a/b;
trigger$# EXCEPTION
trigger$#  WHEN DIVISION_BY_ZERO THEN
trigger$#   RAISE NOTICE 'Cant divide by zero. Enter another divisor';
trigger$#   RETURN null;
trigger$# END
trigger$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
trigger=# SELECT division_exception(9, 2);
 division_exception
-------------------
                4.5
(1 row)

trigger=# SELECT division_exception(9, 0);
NOTICE:  Cant divide by zero. Enter another divisor
 division_exception
-------------------

(1 row)

trigger=#
```

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



```
trigger=# CREATE TABLE students(id INT UNIQUE, name TEXT);
CREATE TABLE
trigger=# INSERT INTO students VALUES (1, 'hisham'), (2, 'raju');
INSERT 0 2
trigger=# CREATE OR REPLACE FUNCTION no_data_check(my_id INT) RETURNS VOID AS $$
trigger$# DECLARE
trigger$#  student_name varchar(20);
trigger$# BEGIN
trigger$#  SELECT name INTO STRICT student_name FROM students WHERE id = my_id;
trigger$#  RAISE NOTICE 'Name = %', student_name;
trigger$# EXCEPTION
trigger$#         WHEN NO_DATA_FOUND THEN
trigger$#   RAISE NOTICE 'No data exception occured';
trigger$#   RAISE NOTICE 'No name with id %', my_id;
trigger$# END
trigger$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
trigger=#
trigger=# SELECT no_data_check(2);
NOTICE:  Name = raju
 no_data_check
---------------

(1 row)

trigger=# SELECT no_data_check(5);
NOTICE:  No data exception occured
NOTICE:  No name with id 5
 no_data_check
---------------

(1 row)

trigger=#
```

7. Create a table with ebill(cname, prev_eading, curr_reading). If prev_reading = curr_reading then raise an exception ‘Data Entry Error’.


hishamalip@Savage: ~

```
trigger=# CREATE TABLE ebill(cname TEXT, prev_reading INT, curr_reading INT);
CREATE TABLE
trigger=# CREATE OR REPLACE FUNCTION add_ebill(name TEXT, prev INT, curr INT) RETURNS VOID AS $$
trigger$# BEGIN
trigger$#  IF prev = curr THEN
trigger$#   RAISE EXCEPTION USING ERRCODE = '50001';
trigger$#  END IF;
trigger$#      INSERT INTO ebill VALUES (name, prev ,curr);
trigger$#  RAISE NOTICE 'Statement processed';
trigger$#  EXCEPTION
trigger$#   WHEN SQLSTATE '50001' THEN
trigger$#                RAISE NOTICE 'Data Entry Error';
trigger$# END
trigger$# $$ LANGUAGE PLPGSQL;
CREATE FUNCTION
trigger=#
trigger=# SELECT add_ebill('hisham', 4, 4);
NOTICE:  Data Entry Error
 add_ebill
-----------

(1 row)

trigger=# SELECT add_ebill('melvy', 7, 8);
NOTICE:  Statement processed
 add_ebill
-----------

(1 row)

trigger=# SELECT * FROM ebill;
 cname | prev_reading | curr_reading
-------+--------------+-------------
 melvy |            7 |            8
(1 row)

trigger=#
```

RESULT:

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