College Of Engineering Trivandrum

Application Software Development Lab

Abhishek Manoharan
S5 CSE Roll No:2

TVE17CS002

Department of Computer Science

August 23, 2019

Contents

List of Figures

Cycle 2

Exp No 10

TRIGGER AND EXCEPTION HANDLING

1 Aim

To study PL/SQL trigger and exception handling.

2 Description

Triggers are procedures that are stored in the database and implicitly run, or fired, when something happens
Exceptions are used to handle run time errors in program

3 Questions

3.1 Trigger whenever data is inserted

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

3.1.1 Table Creation

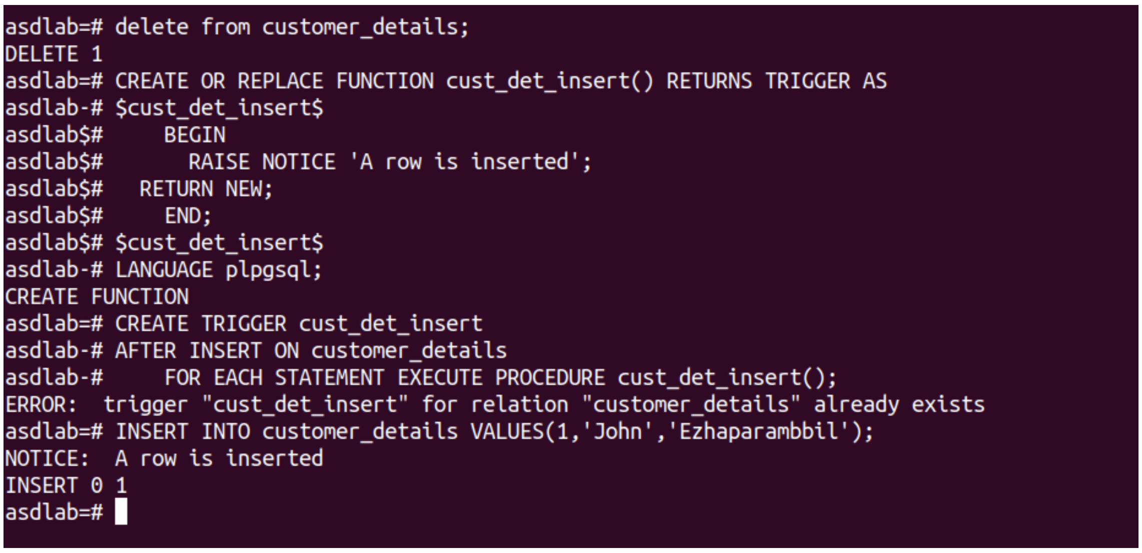
CREATE TABLE customer_details (cust_id int UNIQUE,cust_name varchar(25),address varchar(30));

3.1.2 Code

```
CREATE OR REPLACE FUNCTION cust_det_insert() RETURNS TRIGGER AS
$cust_det_insert$
    BEGIN
     RAISE NOTICE 'A row is inserted';
RETURN NEW;
    END;
$cust_det_insert$
LANGUAGE plpgsql;
CREATE TRIGGER cust_det_insert
AFTER INSERT ON customer_details
    FOR EACH STATEMENT EXECUTE PROCEDURE cust_det_insert();
```

3.1.3 Output

```
INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
```



```
asdlab=# delete from customer_details;
DELETE 1
asdlab=# CREATE OR REPLACE FUNCTION cust_det_insert() RETURNS TRIGGER AS
asdlab-# $cust_det_insert$
asdlab$#     BEGIN
asdlab$#       RAISE NOTICE 'A row is inserted';
asdlab$#   RETURN NEW;
asdlab$#     END;
asdlab$# $cust_det_insert$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# CREATE TRIGGER cust_det_insert
asdlab-# AFTER INSERT ON customer_details
asdlab-#     FOR EACH STATEMENT EXECUTE PROCEDURE cust_det_insert();
ERROR:  trigger "cust_det_insert" for relation "customer_details" already exists
asdlab=# INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
NOTICE:  A row is inserted
INSERT 0 1
asdlab=#
```

Figure 1: Data is inserted

3.2 Message when salary >20000

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

3.2.1 Table Creation

CREATE TABLE emp_details(empid INT UNIQUE,empname varchar(20),salary int);

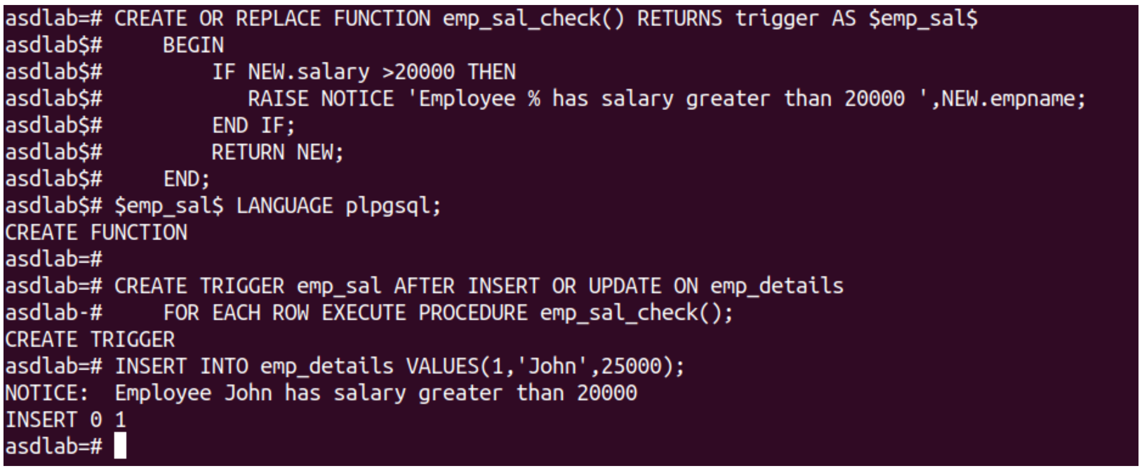
3.2.2 Code

```
CREATE OR REPLACE FUNCTION emp_sal_check() RETURNS trigger AS $emp_sal$
    BEGIN
        IF NEW.salary >20000 THEN
           RAISE NOTICE 'Employee % has salary greater than 20000 ',NEW.empname;
        END IF;
        RETURN NEW;
    END;
$emp_sal$ LANGUAGE plpgsql;

CREATE TRIGGER emp_sal AFTER INSERT OR UPDATE ON emp_details
    FOR EACH ROW EXECUTE PROCEDURE emp_sal_check();
```

3.2.3 Output

```
INSERT INTO emp_details VALUES(1,'John',25000);
```



```
asdlab=# CREATE OR REPLACE FUNCTION emp_sal_check() RETURNS trigger AS $emp_sal$
asdlab$#     BEGIN
asdlab$#         IF NEW.salary >20000 THEN
asdlab$#            RAISE NOTICE 'Employee % has salary greater than 20000 ',NEW.empname;
asdlab$#         END IF;
asdlab$#         RETURN NEW;
asdlab$#     END;
asdlab$# $emp_sal$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER emp_sal AFTER INSERT OR UPDATE ON emp_details
asdlab-#     FOR EACH ROW EXECUTE PROCEDURE emp_sal_check();
CREATE TRIGGER
asdlab=# INSERT INTO emp_details VALUES(1,'John',25000);
NOTICE:  Employee John has salary greater than 20000
INSERT 0 1
asdlab=#
```

Figure 2: Salary >20000

3.3 Row count

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.

3.3.1 Table creation

CREATE TABLE cust_count(count_row int);

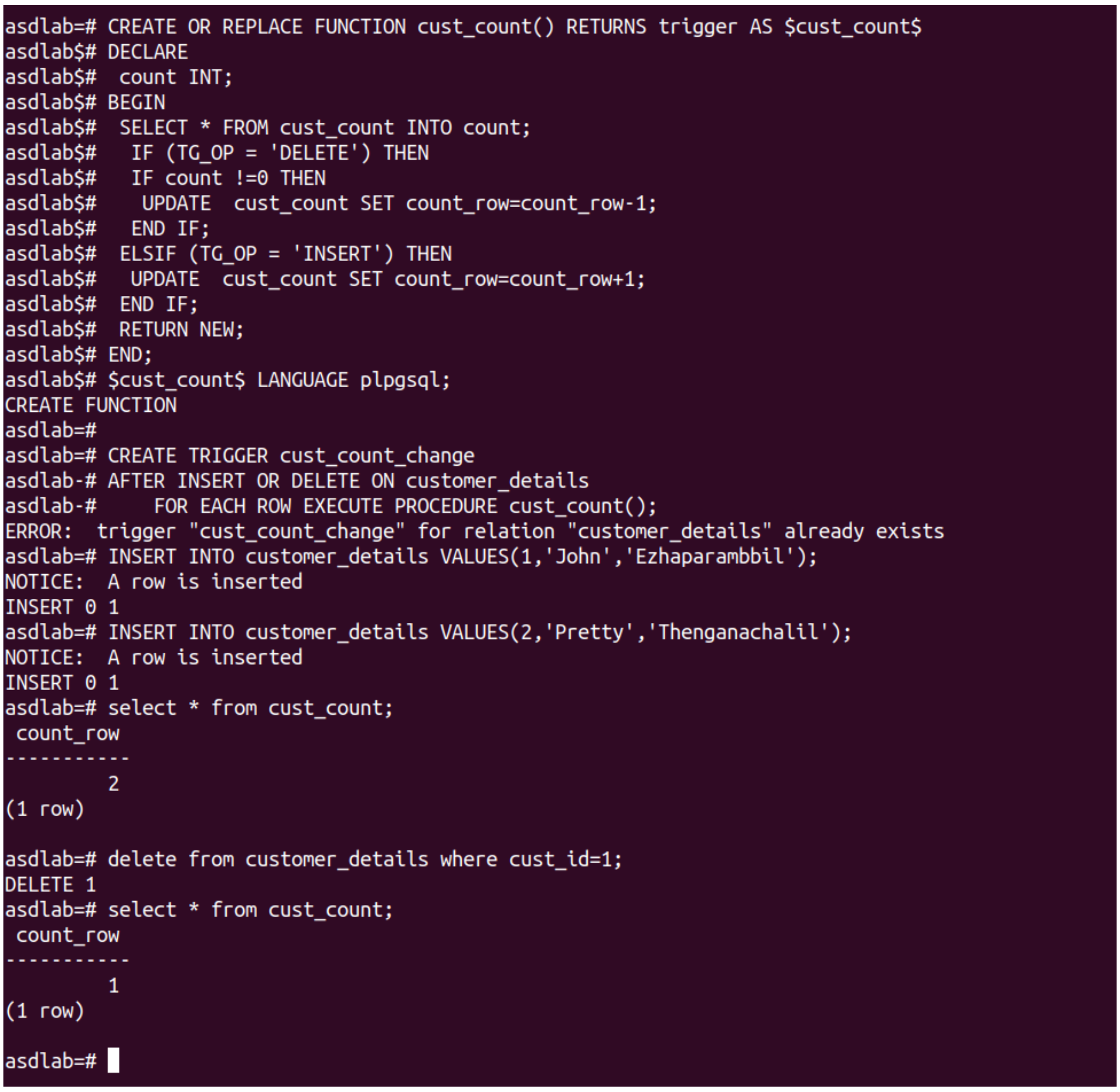
```
insert into cust_count VALUES(0);
```

3.3.2 Code

```
CREATE OR REPLACE FUNCTION cust_count() RETURNS trigger AS $cust_count$
DECLARE
count INT;
BEGIN
SELECT * FROM cust_count INTO count;
  IF (TG_OP = 'DELETE') THEN
IF count !=0 THEN
UPDATE  cust_count SET count_row=count_row-1;
END IF;
ELSIF (TG_OP = 'INSERT') THEN
UPDATE  cust_count SET count_row=count_row+1;
END IF;
RETURN NEW;
END;
$cust_count$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER cust_count_change
AFTER INSERT OR DELETE ON customer_details
    FOR EACH ROW EXECUTE PROCEDURE cust_count();
```

3.3.3 Output



```
asdlab=# CREATE OR REPLACE FUNCTION cust_count() RETURNS trigger AS $cust_count$
asdlab$# DECLARE
asdlab$#  count INT;
asdlab$# BEGIN
asdlab$#  SELECT * FROM cust_count INTO count;
asdlab$#   IF (TG_OP = 'DELETE') THEN
asdlab$#   IF count !=0 THEN
asdlab$#    UPDATE  cust_count SET count_row=count_row-1;
asdlab$#   END IF;
asdlab$#  ELSIF (TG_OP = 'INSERT') THEN
asdlab$#   UPDATE  cust_count SET count_row=count_row+1;
asdlab$#  END IF;
asdlab$#  RETURN NEW;
asdlab$# END;
asdlab$# $cust_count$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER cust_count_change
asdlab-# AFTER INSERT OR DELETE ON customer_details
asdlab-#     FOR EACH ROW EXECUTE PROCEDURE cust_count();
ERROR:  trigger "cust_count_change" for relation "customer_details" already exists
asdlab=# INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
NOTICE:  A row is inserted
INSERT 0 1
asdlab=# INSERT INTO customer_details VALUES(2,'Pretty','Thenganachalil');
NOTICE:  A row is inserted
INSERT 0 1
asdlab=# select * from cust_count;
 count_row
-----------
         2
(1 row)

asdlab=# delete from customer_details where cust_id=1;
DELETE 1
asdlab=# select * from cust_count;
 count_row
-----------
         1
(1 row)

asdlab=# 
```

Figure 3: Row count

3.4 Deletion and Updating

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 updatedT)

3.4.1 Table creation

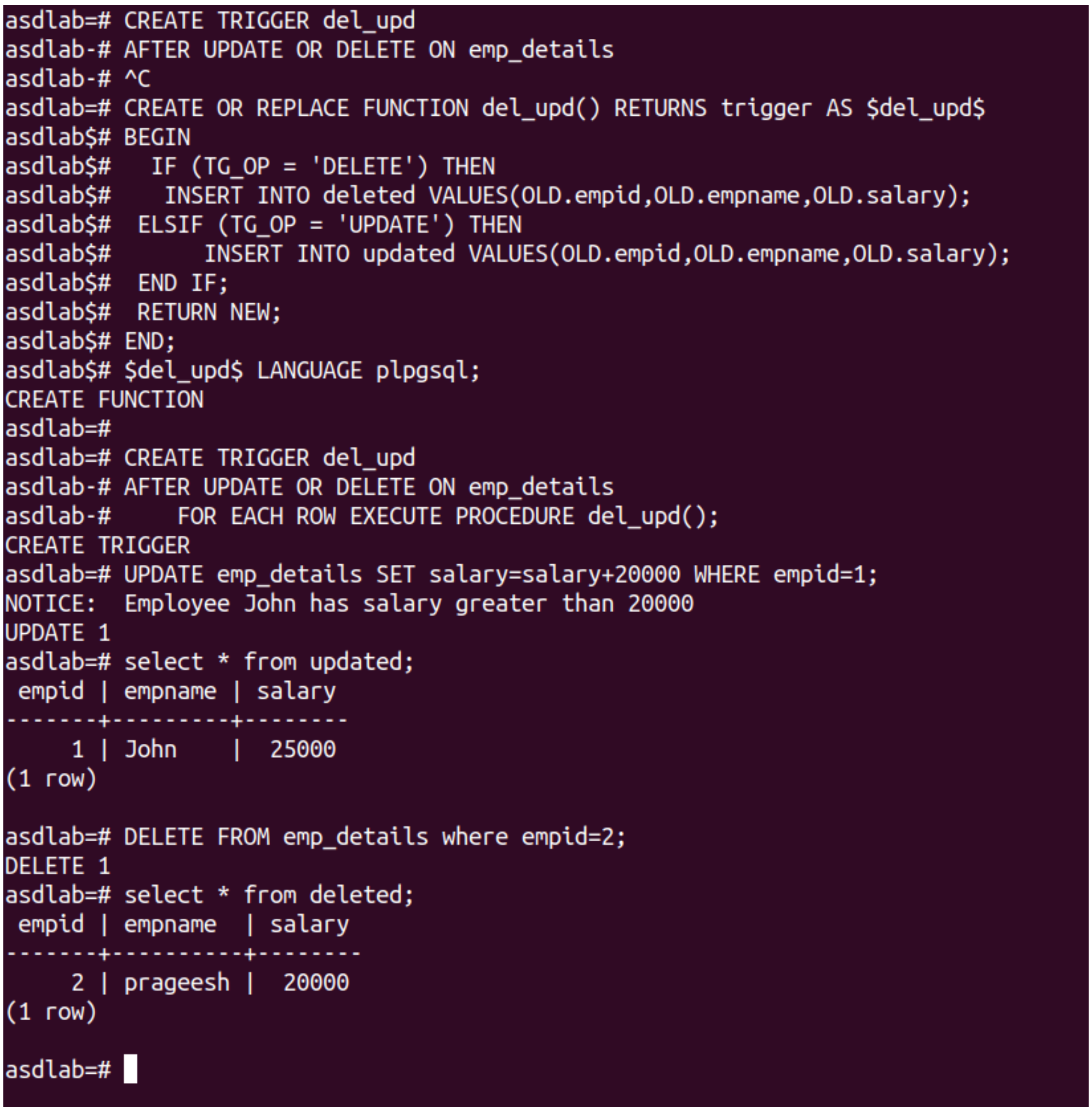
CREATE TABLE deleted(empid INT ,empname varchar(20),salary int);
CREATE TABLE updated(empid INT,empname varchar(20),salary int);

3.4.2 Code

```
CREATE OR REPLACE FUNCTION del_upd() RETURNS trigger AS $del_upd$
BEGIN
  IF (TG_OP = 'DELETE') THEN
INSERT INTO deleted VALUES(OLD.empid,OLD.empname,OLD.salary);
ELSIF (TG_OP = 'UPDATE') THEN
     INSERT INTO updated VALUES(OLD.empid,OLD.empname,OLD.salary);
END IF;
RETURN NEW;
END;
$del_upd$ LANGUAGE plpgsql;

CREATE TRIGGER del_upd
AFTER UPDATE OR DELETE ON emp_details
    FOR EACH ROW EXECUTE PROCEDURE del_upd();
```

3.4.3 Output



```
asdlab=# CREATE TRIGGER del_upd
asdlab-# AFTER UPDATE OR DELETE ON emp_details
asdlab-# ^C
asdlab=# CREATE OR REPLACE FUNCTION del_upd() RETURNS trigger AS $del_upd$
asdlab$# BEGIN
asdlab$#   IF (TG_OP = 'DELETE') THEN
asdlab$#    INSERT INTO deleted VALUES(OLD.empid,OLD.empname,OLD.salary);
asdlab$#  ELSIF (TG_OP = 'UPDATE') THEN
asdlab$#       INSERT INTO updated VALUES(OLD.empid,OLD.empname,OLD.salary);
asdlab$#  END IF;
asdlab$#  RETURN NEW;
asdlab$# END;
asdlab$# $del_upd$ LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER del_upd
asdlab-# AFTER UPDATE OR DELETE ON emp_details
asdlab-#     FOR EACH ROW EXECUTE PROCEDURE del_upd();
CREATE TRIGGER
asdlab=# UPDATE emp_details SET salary=salary+20000 WHERE empid=1;
NOTICE:  Employee John has salary greater than 20000
UPDATE 1
asdlab=# select * from updated;
 empid | empname | salary
-------+---------+--------
     1 | John    |  25000
(1 row)

asdlab=# DELETE FROM emp_details where empid=2;
DELETE 1
asdlab=# select * from deleted;
 empid | empname  | salary
-------+----------+--------
     2 | prageesh |  20000
(1 row)

asdlab=# 
```

Figure 4: Deleted and Updated table

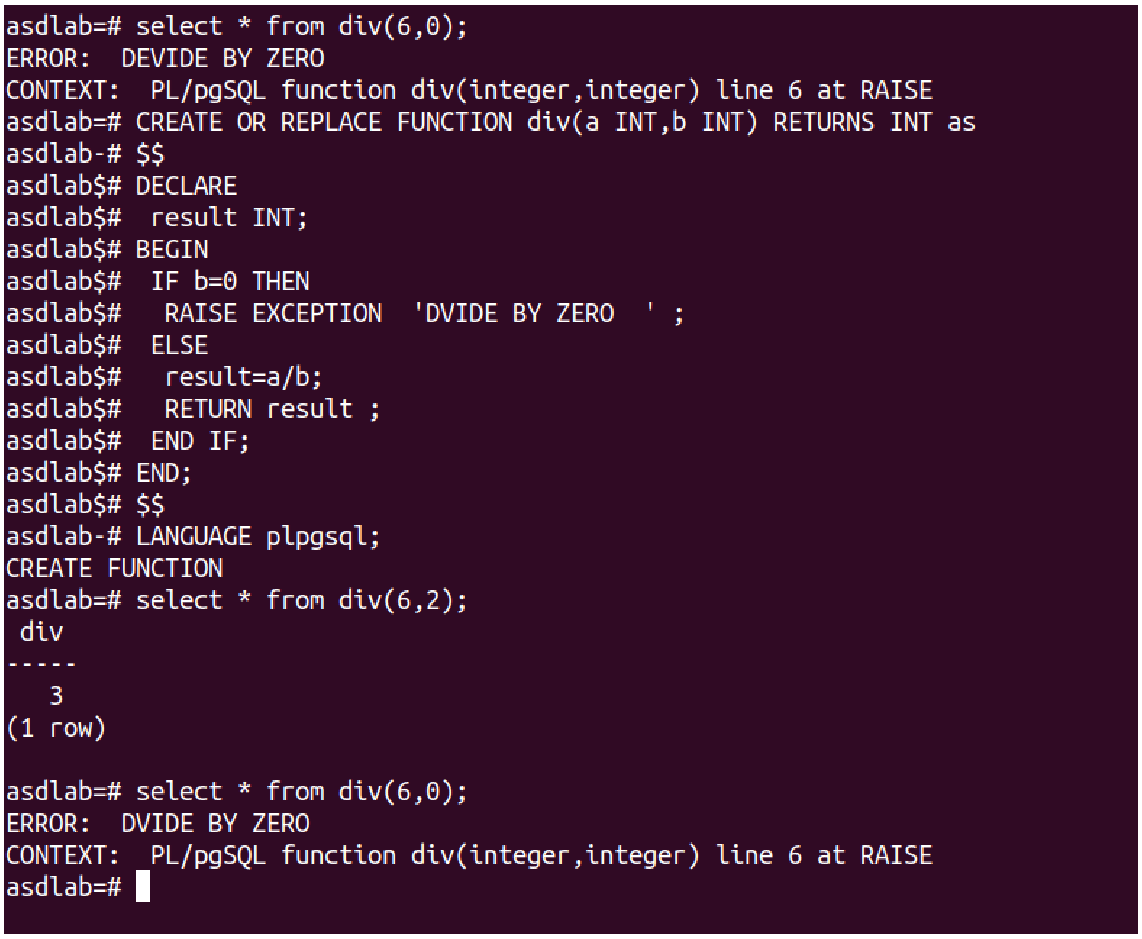
3.5 Divide zero Exception

Write a PL/SQL to show divide by zero exception

3.5.1 Code

```
CREATE OR REPLACE FUNCTION div(a INT,b INT) RETURNS INT as
$$
DECLARE
result INT;
BEGIN
IF b=0 THEN
RAISE EXCEPTION  'DVIDE BY ZERO  ' ;
ELSE
result=a/b;
RETURN result ;
END IF;
END;
$$
LANGUAGE plpgsql;
```

3.5.2 Output



```
asdlab=# select * from div(6,0);
ERROR:  DEVIDE BY ZERO
CONTEXT:  PL/pgSQL function div(integer,integer) line 6 at RAISE
asdlab=# CREATE OR REPLACE FUNCTION div(a INT,b INT) RETURNS INT as
asdlab-# $$
asdlab$# DECLARE
asdlab$#  result INT;
asdlab$# BEGIN
asdlab$#  IF b=0 THEN
asdlab$#   RAISE EXCEPTION  'DVIDE BY ZERO  ' ;
asdlab$#  ELSE
asdlab$#   result=a/b;
asdlab$#   RETURN result ;
asdlab$#  END IF;
asdlab$# END;
asdlab$# $$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=# select * from div(6,2);
 div
-----
   3
(1 row)

asdlab=# select * from div(6,0);
ERROR:  DVIDE BY ZERO
CONTEXT:  PL/pgSQL function div(integer,integer) line 6 at RAISE
asdlab=#
```

Figure 5: Divide By Zero

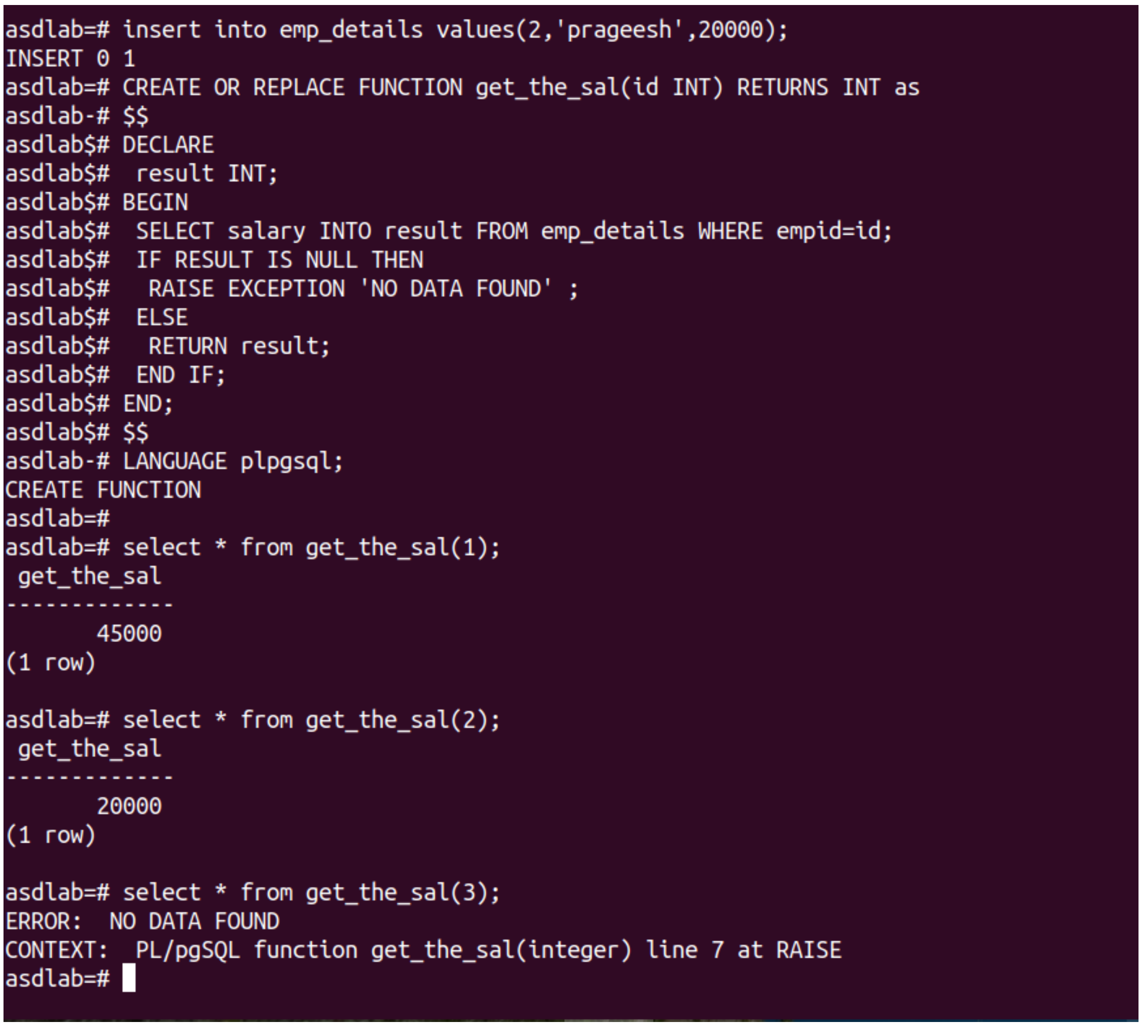
3.6 No Data Found Exception

Write a PL/SQL to show no data found exception

3.6.1 Code

```
CREATE OR REPLACE FUNCTION get_the_sal(id INT) RETURNS INT as
$$
DECLARE
result INT;
BEGIN
SELECT salary INTO result FROM emp_details WHERE empid=id;
IF RESULT IS NULL THEN
RAISE EXCEPTION 'NO DATA FOUND' ;
ELSE
RETURN result;
END IF;
END;
$$
LANGUAGE plpgsql;
```

3.6.2 Output



```
asdlab=# insert into emp_details values(2,'prageesh',20000);
INSERT 0 1
asdlab=# CREATE OR REPLACE FUNCTION get_the_sal(id INT) RETURNS INT as
asdlab-# $$
asdlab$# DECLARE
asdlab$#  result INT;
asdlab$# BEGIN
asdlab$#  SELECT salary INTO result FROM emp_details WHERE empid=id;
asdlab$#  IF RESULT IS NULL THEN
asdlab$#   RAISE EXCEPTION 'NO DATA FOUND' ;
asdlab$#  ELSE
asdlab$#   RETURN result;
asdlab$#  END IF;
asdlab$# END;
asdlab$# $$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# select * from get_the_sal(1);
 get_the_sal
-------------
       45000
(1 row)

asdlab=# select * from get_the_sal(2);
 get_the_sal
-------------
       20000
(1 row)

asdlab=# select * from get_the_sal(3);
ERROR:  NO DATA FOUND
CONTEXT:  PL/pgSQL function get_the_sal(integer) line 7 at RAISE
asdlab=#
```

Figure 6: No Data Found

3.7 Wrong Ebill

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

3.7.1 Table creation

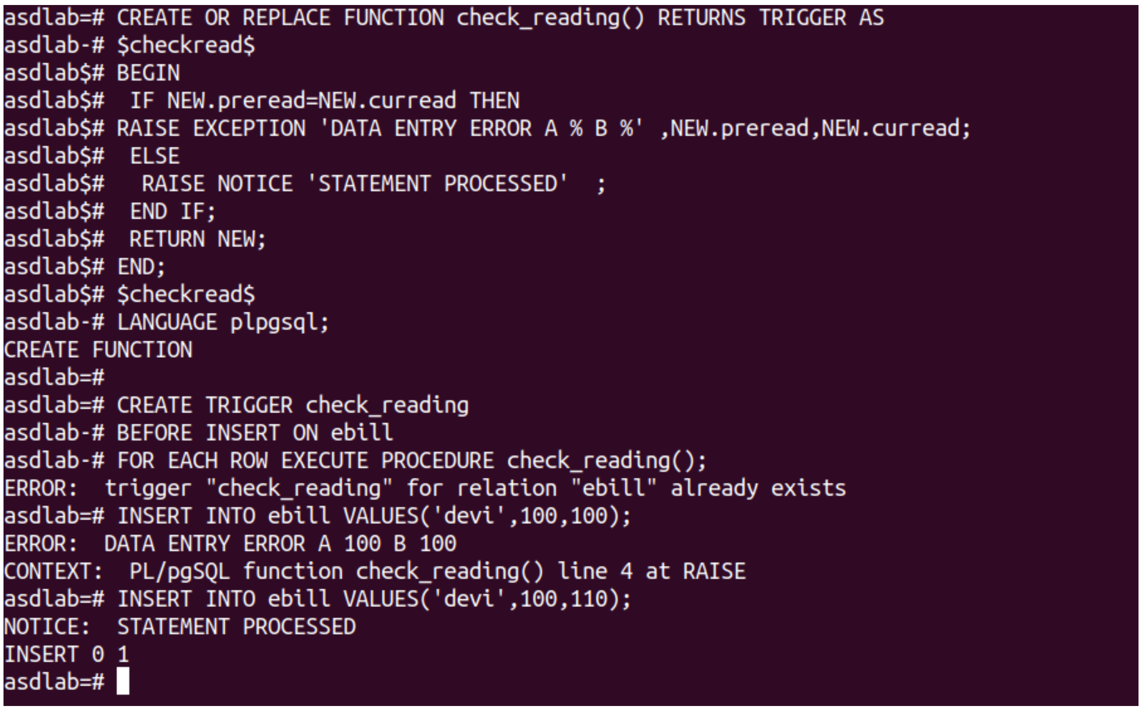
CREATE TABLE ebill(cname varchar(20),preread int,curread int);

3.7.2 Code

```
CREATE OR REPLACE FUNCTION check_reading() RETURNS TRIGGER AS
$checkread$
BEGIN
IF NEW.preread=NEW.curread THEN
RAISE EXCEPTION 'DATA ENTRY ERROR A % B %' ,NEW.preread,NEW.curread;
ELSE
RAISE NOTICE 'STATEMENT PROCESSED'  ;
END IF;
RETURN NEW;
END;
$checkread$
LANGUAGE plpgsql;

CREATE TRIGGER check_reading
BEFORE INSERT ON ebill
FOR EACH ROW EXECUTE PROCEDURE check_reading();
```

3.7.3 Output



```
asdlab=# CREATE OR REPLACE FUNCTION check_reading() RETURNS TRIGGER AS
asdlab-# $checkread$
asdlab$# BEGIN
asdlab$#  IF NEW.preread=NEW.curread THEN
asdlab$# RAISE EXCEPTION 'DATA ENTRY ERROR A % B %' ,NEW.preread,NEW.curread;
asdlab$#  ELSE
asdlab$#   RAISE NOTICE 'STATEMENT PROCESSED'  ;
asdlab$#  END IF;
asdlab$#  RETURN NEW;
asdlab$# END;
asdlab$# $checkread$
asdlab-# LANGUAGE plpgsql;
CREATE FUNCTION
asdlab=#
asdlab=# CREATE TRIGGER check_reading
asdlab-# BEFORE INSERT ON ebill
asdlab-# FOR EACH ROW EXECUTE PROCEDURE check_reading();
ERROR:  trigger "check_reading" for relation "ebill" already exists
asdlab=# INSERT INTO ebill VALUES('devi',100,100);
ERROR:  DATA ENTRY ERROR A 100 B 100
CONTEXT:  PL/pgSQL function check_reading() line 4 at RAISE
asdlab=# INSERT INTO ebill VALUES('devi',100,110);
NOTICE:  STATEMENT PROCESSED
INSERT 0 1
asdlab=#
```

Figure 7: Incorrect Reading

4 Result

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