

# College of Engineering, Trivandrum

Department of Computer Science and Engineering



## CS333 APPLICATION SOFTWARE DEVELOPMENT LAB

---

### LABORATORY REPORT 11

Trigger and Exception Handling

---

**Student Name**

1. Justine Biju(S5)

**Student ID**

170445(Roll No:37)

Submission Date : 19/08/2019

# 1 Introduction

## 1. Trigger

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

The syntax is as follows:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

## 2. Exception

An exception is an error which disrupts the normal flow of program instructions. PL/SQL provides us the exception block which raises the exception thus helping the programmer to find out the fault and resolve it.

The syntax is as follows:

```
DECLARE
declarations section;

BEGIN
executable command(s);

EXCEPTION
WHEN exception1 THEN
statement1;
WHEN exception2 THEN
statement2;
[WHEN others THEN]
/* default exception handling code */

END;
```

All these are supported by PostgreSQL and can be implemented in a similar fashion.

## 2 Questions

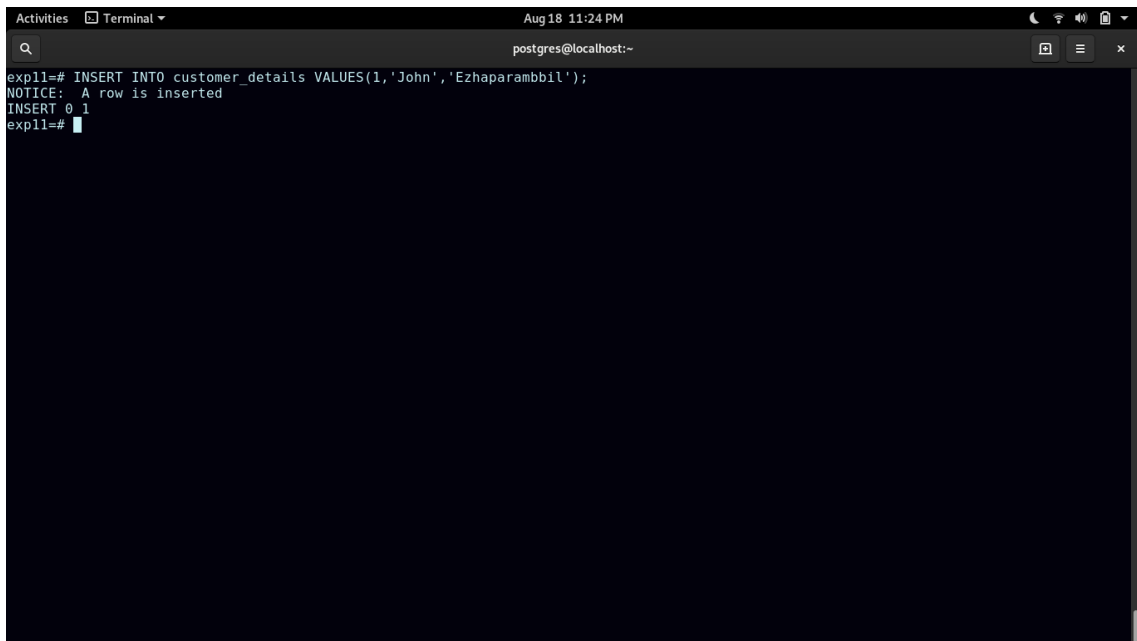
1. Create a trigger whenever a new record is inserted in the customer\_details table.

—function

```
CREATE OR REPLACE FUNCTION show_ins() RETURNS TRIGGER AS $$  
BEGIN  
    RAISE NOTICE 'A row is inserted';  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

—trigger

```
CREATE TRIGGER t1  
AFTER INSERT  
ON customer_details  
FOR EACH ROW  
EXECUTE PROCEDURE show_ins();
```



The screenshot shows a terminal window titled "Terminal" with the date and time "Aug 18 11:24 PM". The user is logged in as "postgres@localhost". The terminal shows the following commands and output:

```
exp11=# INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');  
NOTICE: A row is inserted  
INSERT 0 1  
exp11=#
```

Figure 1: Question 1

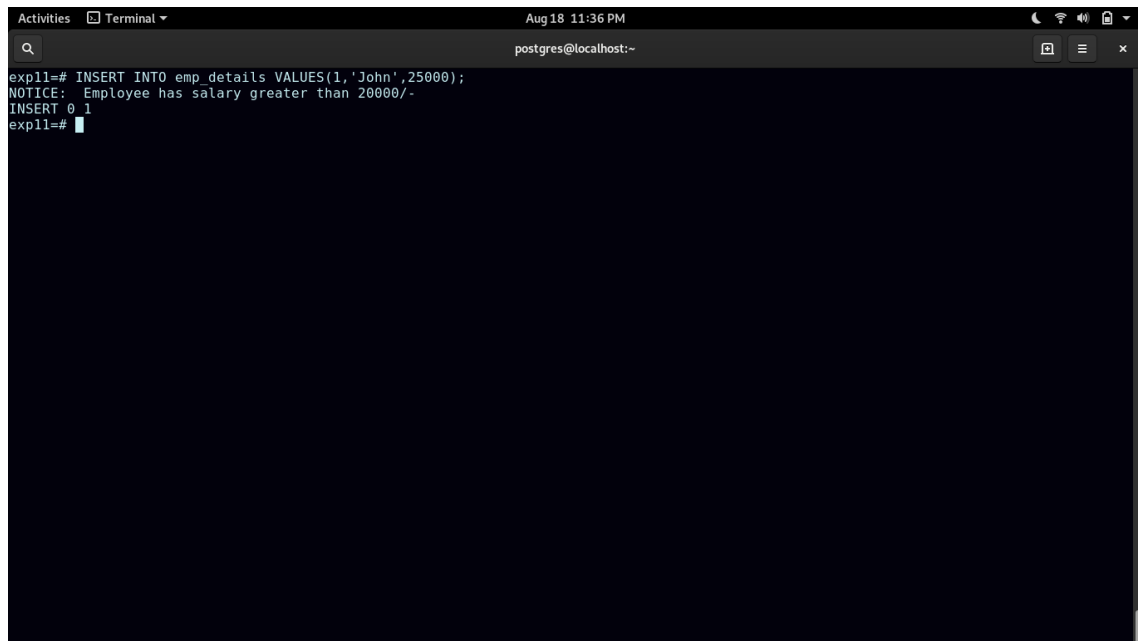
2. Create a trigger to display a message when a user enters a value  $\geq 20000$  in the salary.

—function

```
CREATE OR REPLACE FUNCTION sal_check() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.salary > 20000 THEN
        RAISE NOTICE 'Employee has salary greater than 20000/-';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

—trigger

```
CREATE TRIGGER t2
AFTER INSERT
ON emp_details
FOR EACH ROW
EXECUTE PROCEDURE sal_check();
```



```
exp11=# INSERT INTO emp_details VALUES(1,'John',25000);
NOTICE: Employee has salary greater than 20000/-
INSERT 0 1
exp11=#
```

Figure 2: Question 2

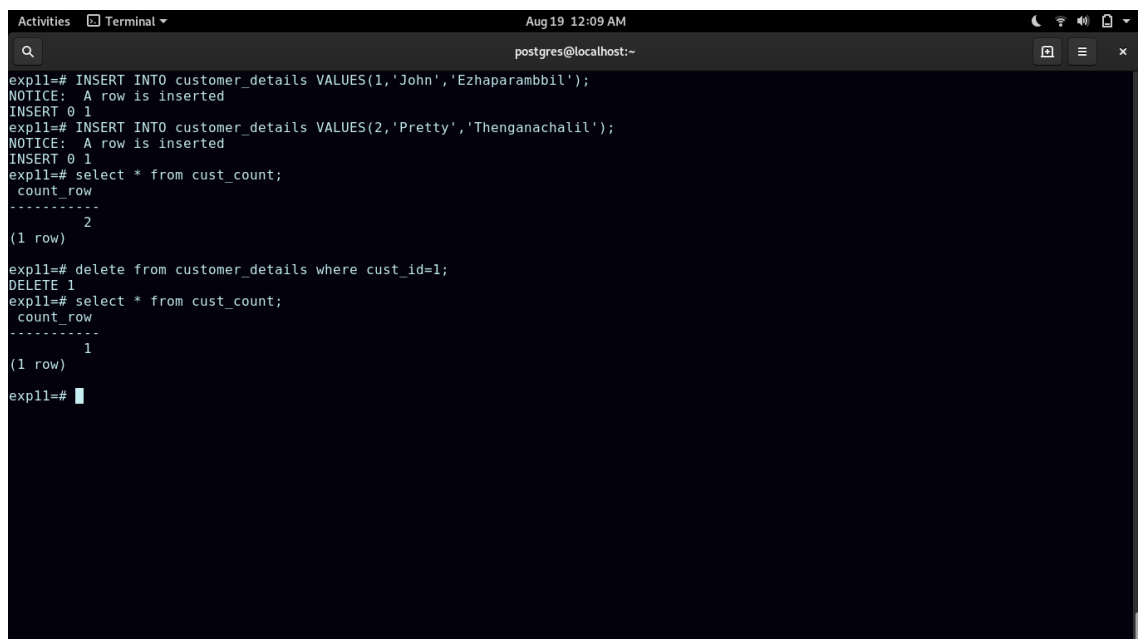
3. Create a trigger w.r.t customer\_detailstable. Increment the value of count\_row (in cust\_count table) whenever a new tuple is inserted and decrement the value of count\_row when a tuple is deleted. Initial value of the count\_row is set to 0.

—function

```
CREATE OR REPLACE FUNCTION inc_count() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        UPDATE cust_count
        SET count_row = count_row+1;
    ELSE
        UPDATE cust_count
        SET count_row = count_row-1;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

—trigger

```
CREATE TRIGGER t3
AFTER INSERT OR DELETE
ON customer_details
FOR EACH ROW
EXECUTE PROCEDURE inc_count();
```



```
Aug 19 12:09 AM
postgres@localhost:~
exp1=# INSERT INTO customer_details VALUES(1,'John','Ezhaparambbil');
NOTICE: A row is inserted
INSERT 0 1
exp1=# INSERT INTO customer_details VALUES(2,'Pretty','Thenganachalil');
NOTICE: A row is inserted
INSERT 0 1
exp1=# select * from cust_count;
 count_row
-----
         2
(1 row)

exp1=# delete from customer_details where cust_id=1;
DELETE 1
exp1=# select * from cust_count;
 count_row
-----
         1
(1 row)
exp1=#
```

Figure 3: Question 3

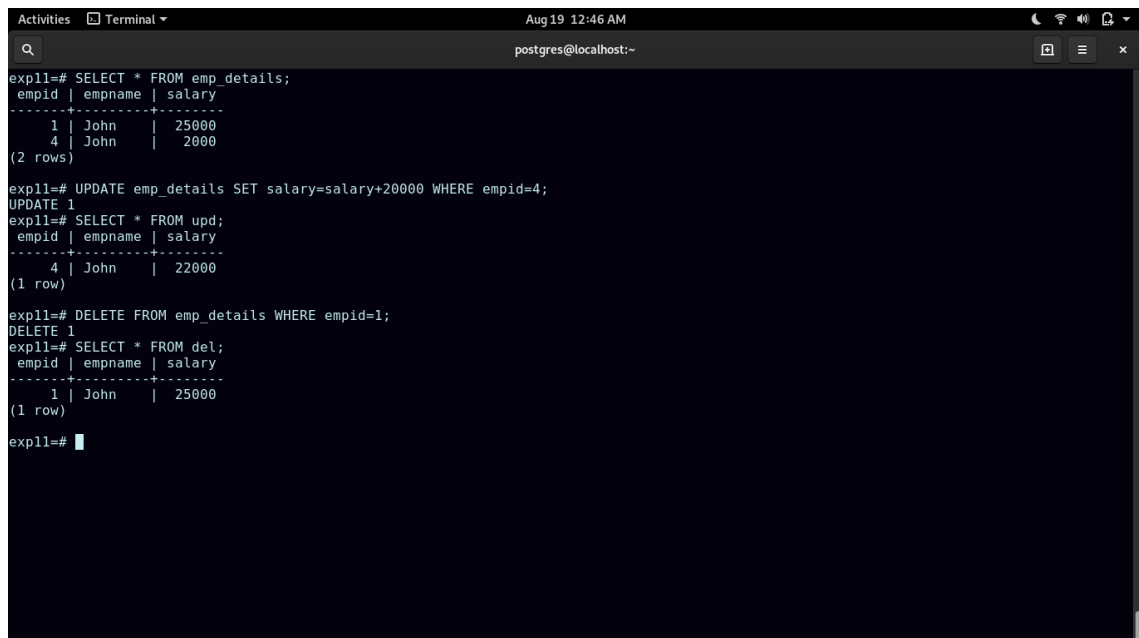
4. Create a trigger to insert the deleted rows from emp\_details to another table and updated rows to another table. ( Create the tables deleted and updated )

--function

```
CREATE OR REPLACE FUNCTION del_upd() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'DELETE' THEN
        INSERT INTO del VALUES(old.empid, old.empname, old.salary);
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO upd VALUES(new.empid, new.empname, new.salary);
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

--trigger

```
CREATE TRIGGER t4
AFTER UPDATE OR DELETE
ON emp_details
FOR EACH ROW
EXECUTE PROCEDURE del_upd();
```



The screenshot shows a terminal window with the following SQL commands and their outputs:

```
exp11=# SELECT * FROM emp_details;
 empid | empname | salary
-----+-----+-----
      1 | John    | 25000
      4 | John    | 2000
(2 rows)

exp11=# UPDATE emp_details SET salary=salary+20000 WHERE empid=4;
UPDATE 1
exp11=# SELECT * FROM upd;
 empid | empname | salary
-----+-----+-----
      4 | John    | 22000
(1 row)

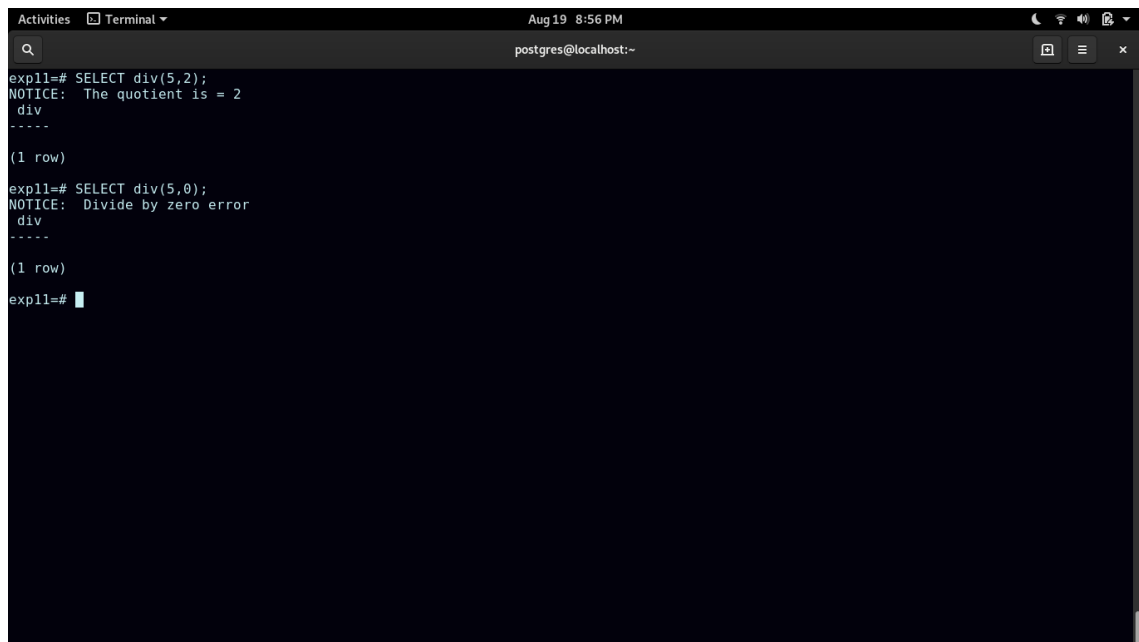
exp11=# DELETE FROM emp_details WHERE empid=1;
DELETE 1
exp11=# SELECT * FROM del;
 empid | empname | salary
-----+-----+-----
      1 | John    | 25000
(1 row)

exp11=#
```

Figure 4: Question 4

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

```
CREATE OR REPLACE FUNCTION div(a integer, b integer) RETURNS VOID AS
BEGIN
    IF b = 0 THEN
        RAISE EXCEPTION USING errcode = 22012;
    END IF;
    RAISE NOTICE 'The quotient is = %',a/b;
EXCEPTION
    WHEN SQLSTATE '22012' THEN
        RAISE NOTICE 'Divide by zero error';
END;
$$ LANGUAGE plpgsql;
```

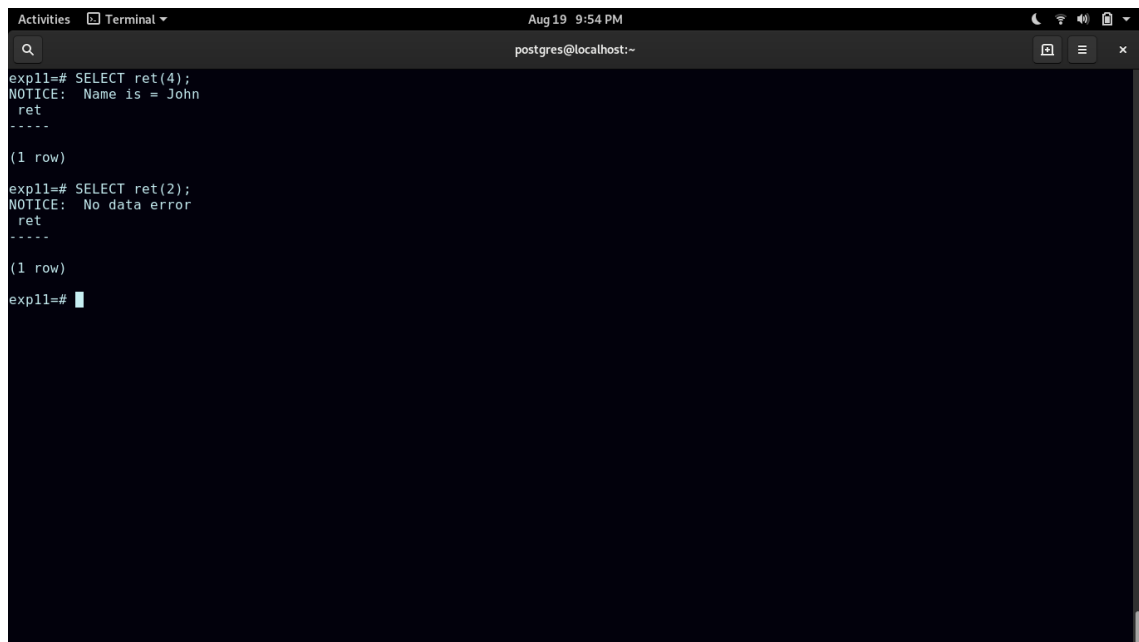
A terminal window titled 'Terminal' with a search bar and window controls. The terminal shows the execution of a PL/SQL function named 'div'. The first command is 'exp11=# SELECT div(5,2);', which results in a 'NOTICE: The quotient is = 2' and a single row of output. The second command is 'exp11=# SELECT div(5,0);', which results in a 'NOTICE: Divide by zero error' and a single row of output. The prompt 'exp11=#' is visible at the bottom of the terminal.

```
exp11=# SELECT div(5,2);
NOTICE: The quotient is = 2
div
-----
(1 row)
exp11=# SELECT div(5,0);
NOTICE: Divide by zero error
div
-----
(1 row)
exp11=#
```

Figure 5: Question 5

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

```
CREATE OR REPLACE FUNCTION ret(a integer) RETURNS VOID AS $$  
DECLARE  
    name VARCHAR(30);  
BEGIN  
    SELECT empname INTO STRICT name FROM emp_details WHERE empid =  
    RAISE NOTICE 'Name is = %',name;  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        RAISE NOTICE 'No data error';  
END;  
$$ LANGUAGE plpgsql;
```



The screenshot shows a terminal window titled 'Terminal' with the date and time 'Aug 19 9:54 PM'. The user is logged in as 'postgres@localhost'. The terminal displays the following commands and output:

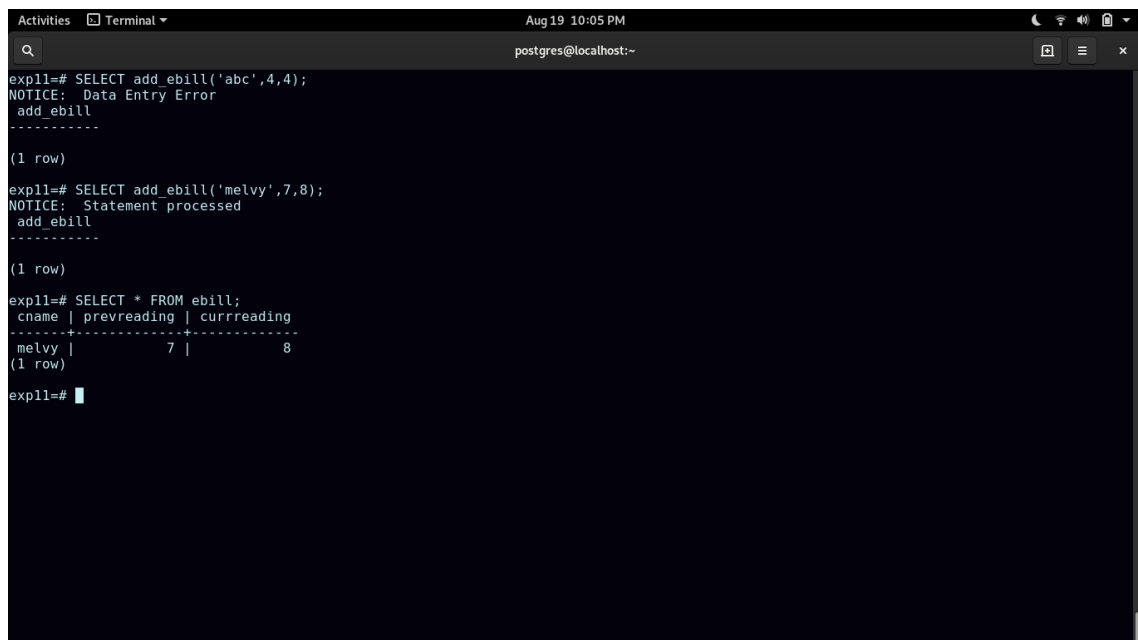
```
exp11=# SELECT ret(4);  
NOTICE: Name is = John  
ret  
-----  
(1 row)  
  
exp11=# SELECT ret(2);  
NOTICE: No data error  
ret  
-----  
(1 row)  
  
exp11=#
```

Figure 6: Question 6



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

```
CREATE OR REPLACE FUNCTION add_ebill(name VARCHAR(20), p integer, c integer)
BEGIN
    IF p = c THEN
        RAISE EXCEPTION USING errcode = '50001';
    END IF;
    INSERT INTO ebill VALUES (name,p,c);
    RAISE NOTICE 'Statement processed';
EXCEPTION
    WHEN SQLSTATE '50001' THEN
        RAISE NOTICE 'Data Entry Error';
END;
$$ LANGUAGE plpgsql;
```



The screenshot shows a terminal window titled 'Activities Terminal' with the date and time 'Aug 19 10:05 PM'. The user is logged in as 'postgres@localhost'. The terminal displays the following commands and output:

```
exp11=# SELECT add_ebill('abc',4,4);
NOTICE: Data Entry Error
add_ebill
-----
(1 row)

exp11=# SELECT add_ebill('melvy',7,8);
NOTICE: Statement processed
add_ebill
-----
(1 row)

exp11=# SELECT * FROM ebill;
  cname | prevreading | currreading
-----+-----+-----
 melvy |          7 |          8
(1 row)
```

Figure 7: Question 7

### 3 Result

- Successfully implemented Triggers and Exceptions in PostgreSQL.