College of Engineering, Trivandrum

Department of Computer Science and Engineering

CS333 APPLICATION SOFTWARE DEVELOPMENT LAB

LABORATORY REPORT 10

Cursor

Student Name	Student ID
1. Justine Biju(S5)	170445(Roll No:37)

Submission Date : 19/08/2019

1 Introduction

A cursor is a temporary work area created in system memory when a SQL statement is executed. A cursor is a set of rows together with a pointer that identifies a current row. It is a database object to retrieve data from a result set one row at a time. It is useful when we want to manipulate the record of a table in a singleton method, in other words one row at a time. In other words, a cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

There are the following two types of Cursors:

1. Implicit Cursor
2. Explicit Cursor

The implicit cursors are types of cursors are generated and used by the system during the manipulation of a DML query (INSERT, UPDATE and DELETE). An implicit cursor is also generated by the system when a single row is selected by a SELECT command.

The explicit cursor is a type of cursor that is generated by the user using a SELECT command. An explicit cursor contains more than one row, but only one row can be processed at a time. An explicit cursor moves one by one over the records. An explicit cursor uses a pointer that holds the record of a row. After fetching a row, the cursor pointer moves to the next row.

Each cursor contains the followings 5 parts:

- Declare Cursor: In this part we declare variables and return a set of values.
- Open: This is the entering part of the cursor.
- Fetch: Used to retrieve the data row by row from a cursor.
- Close: This is an exit part of the cursor and used to close a cursor.
- Deallocate: In this part we delete the cursor definition and release all the system resources associated with the cursor.

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

2 Questions

1. Create table student (id, name, m1, m2, m3, grade).Insert 5 tuples into it. Find the total, calculate grade and update the grade in the table.

```
CREATE OR REPLACE FUNCTION porf() RETURNS INTEGER AS $$
DECLARE
    sum INTEGER;
    c1 CURSOR FOR SELECT * FROM student;
    rec RECORD;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO rec;
        EXIT WHEN NOT FOUND;
        sum = ceil((rec.m1 + rec.m2 + rec.m3)/3);
        IF sum >= 80 THEN
            UPDATE student
            SET grade = 'A'
            WHERE CURRENT OF c1;
        ELSIF sum >= 70 AND sum < 80 THEN
            UPDATE student
            SET grade = 'B'
            WHERE CURRENT OF c1;
        ELSIF sum >= 60 AND sum < 70 THEN
            UPDATE student
            SET grade = 'C'
            WHERE CURRENT OF c1;
        ELSIF sum >= 50 AND sum < 60 THEN
            UPDATE student
            SET grade = 'D'
            WHERE CURRENT OF c1;
        ELSE
            UPDATE student
            SET grade = 'F'
            WHERE CURRENT OF c1;
        END IF;
    END LOOP;
    CLOSE c1;
    RETURN 0;
END;
$$ LANGUAGE plpgsql;
```

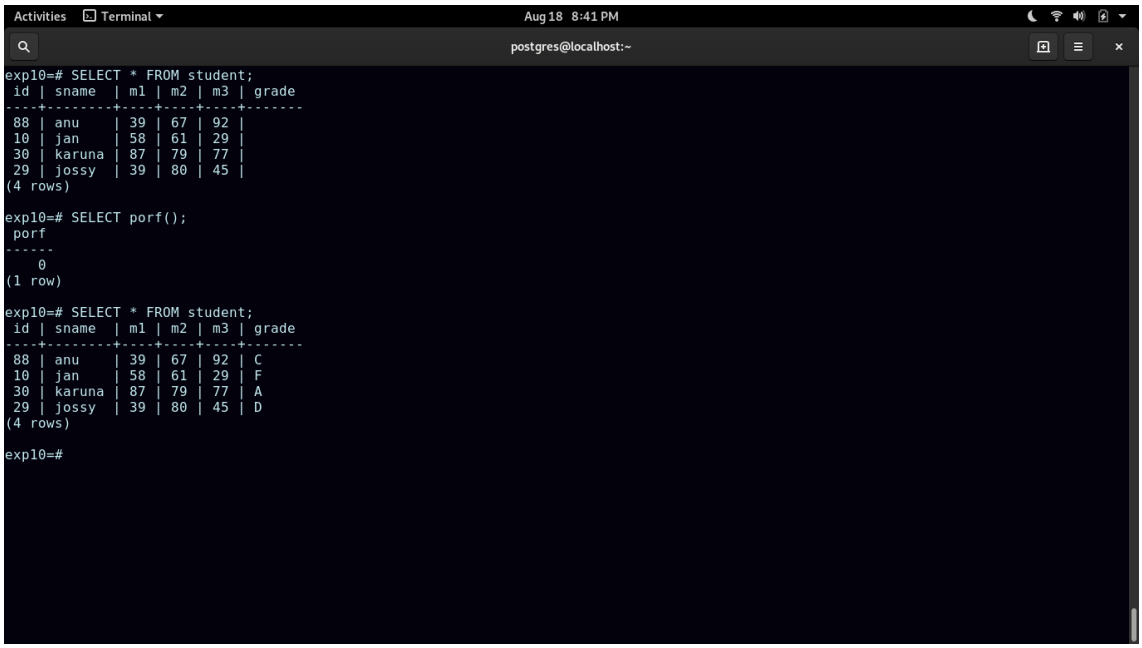

Figure 1: Question 1

2. Create bank_details (accno, name, balance, adate).Calculate the interest of the amount and insert into a new table with fields (accno, interest). Interest= 0.08*balance.

```
CREATE OR REPLACE FUNCTION cal_int() RETURNS INTEGER AS $$
DECLARE
    c1 CURSOR FOR SELECT * FROM bank_details;
    rec RECORD;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO rec;
        EXIT WHEN NOT FOUND;
        INSERT INTO bank_new VALUES (rec.accno, rec.balance*0.08);
    END LOOP;
    CLOSE c1;
    RETURN 0;
END;
$$ LANGUAGE plpgsql;
```

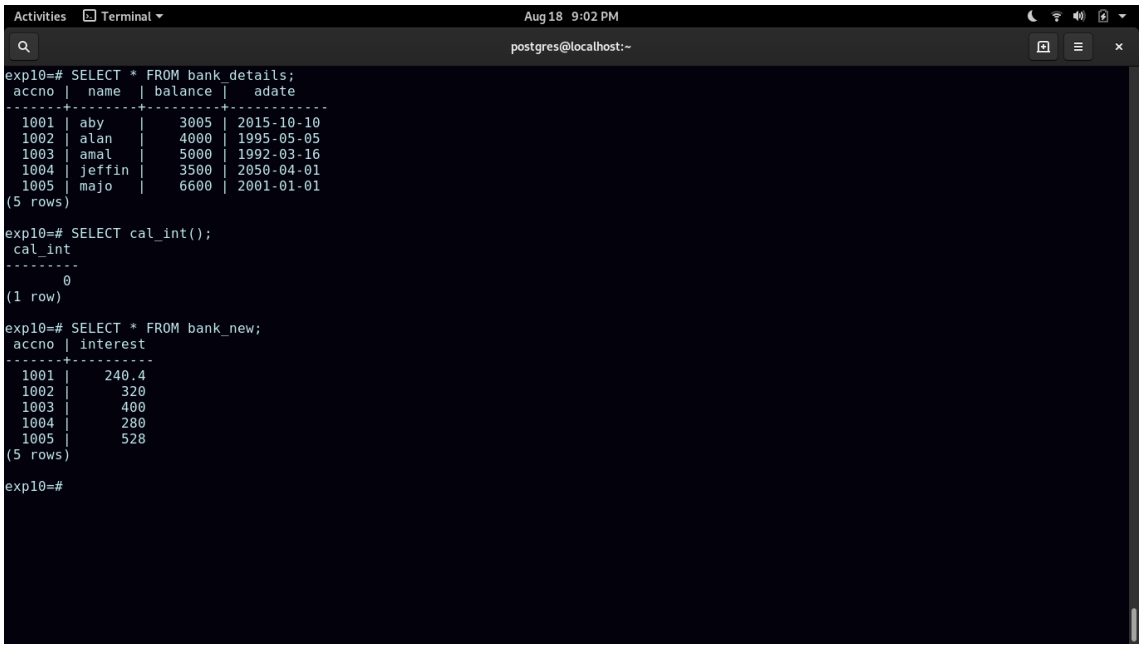

Figure 2: Question 2

3. Create table people_list (id, name, dt_joining, place).If person's experience is above 10 years, put the tuple in table exp_list (id, name, experience).

```
CREATE OR REPLACE FUNCTION cal_exp() RETURNS INTEGER AS
$$
DECLARE
    cd DATE := current_date;
    c1 CURSOR FOR SELECT * FROM people_list;
    rec RECORD;
    yd INT;
BEGIN
    OPEN c1;
    LOOP
        FETCH FROM c1 INTO rec;
        EXIT WHEN NOT FOUND;
        yd = date_part('year',age(rec.dt_joining));
        IF yd > 10 THEN
            INSERT INTO exp_list VALUES(rec.id,rec.name,yd);
        END IF;
    END LOOP;
    RETURN 0;
END;
$$ LANGUAGE plpgsql;
```

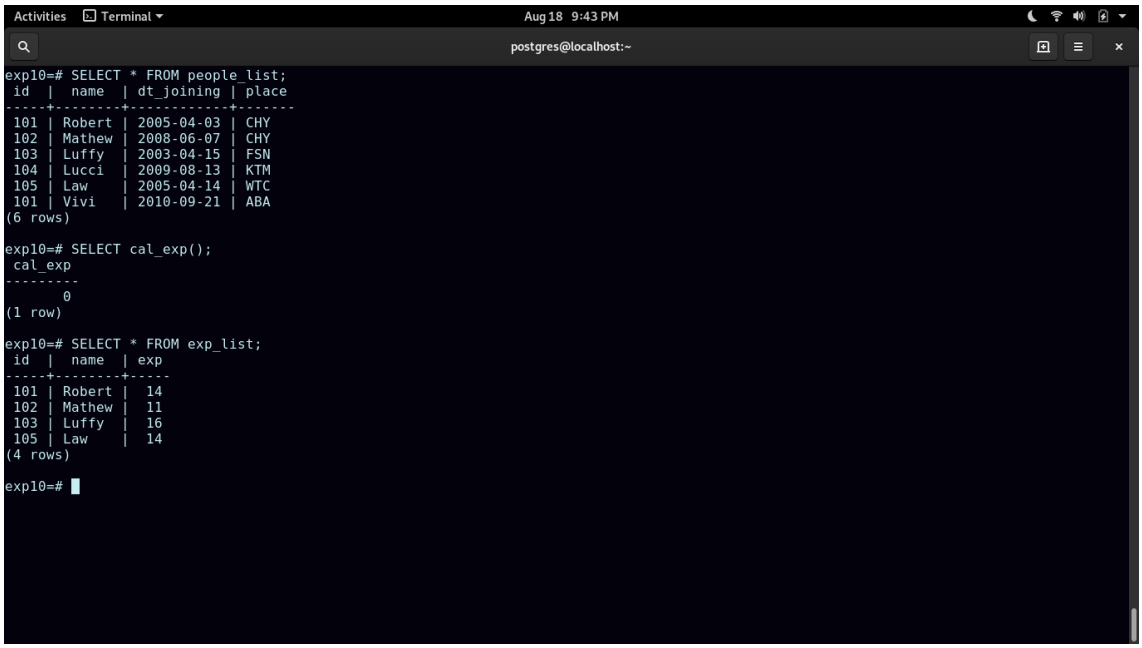

Figure 3: Question 3

4. Create table employee_list(id,name,monthly salary). If: annual salary¡60000, increment monthly salary by 25200000, increment by 20between 200000 and 500000, increment by 15annual salary¿500000, increment monthly salary by 10

```
CREATE OR REPLACE FUNCTION updt_sal() RETURNS INTEGER AS $$
DECLARE
    c1 CURSOR FOR SELECT * FROM emp_list;
    rec RECORD;
BEGIN
    OPEN c1;
    LOOP
        FETCH FROM c1 INTO rec;
        EXIT WHEN NOT FOUND;
        IF rec.m_sal*12 < 60000 THEN
            UPDATE emp_list
            SET m_sal = m_sal*1.25
            WHERE CURRENT OF c1;
        ELSIF rec.m_sal*12 >= 60000 AND rec.m_sal*12 < 200000
        THEN
            UPDATE emp_list
            SET m_sal = m_sal*1.20
            WHERE CURRENT OF c1;
        ELSIF rec.m_sal*12 >= 200000 AND rec.m_sal*12 < 500000
        THEN
            UPDATE emp_list
            SET m_sal = m_sal*1.15
            WHERE CURRENT OF c1;
        ELSIF rec.m_sal*12 >= 500000 THEN
```

```
                UPDATE emp_list
                SET m_sal = m_sal*1.10
                WHERE CURRENT OF c1;
            END IF;
        END LOOP;
        RETURN 0;
    END;
    $$ LANGUAGE plpgsql;
```

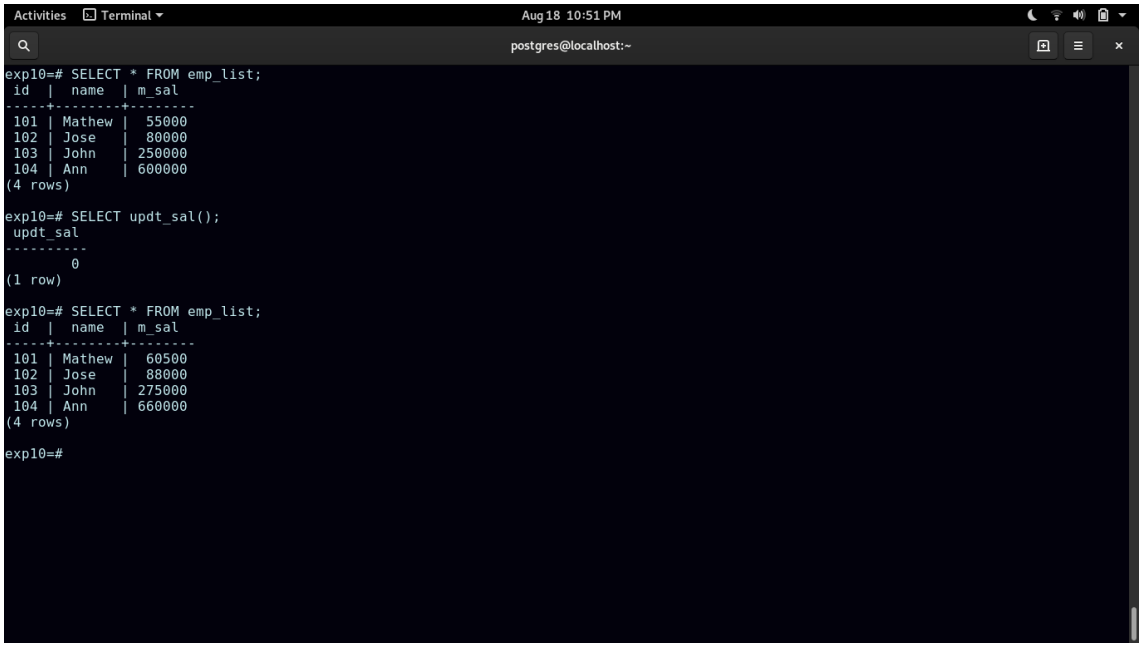

Figure 4: Question 4

3 Result

- Successfully Cursor using PL/SQL in PostgreSQL.