EXPERIMENT - IX
CURSOR

August 18, 2019

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

AIM

To study the use and implementation of cursors in PL/SQL.

CURSOR

Oracle uses work areas to execute SQL statements and store processing information. A PL/SQL construct called a cursor lets you name a work area and access its stored information. There are two kinds of cursors: implicit and explicit. PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row. For queries that return more than one row, you can explicitly declare a cursor to process the rows individually.

QUESTIONS

Write PL/SQL programs for the following(using Cursors):

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.

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION FINDGR()
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# temp CURSOR FOR SELECT * FROM STDNT;
postgres$# tempvar RECORD;
postgres$# total INT;
postgres$# BEGIN
postgres$# OPEN temp;
postgres$# LOOP
postgres$# FETCH temp INTO tempvar;
postgres$# EXIT WHEN NOT FOUND;
postgres$# total:=tempvar.M1+tempvar.M2+tempvar.M3;
postgres$# IF (total>270) THEN
postgres$# UPDATE STDNT SET GR='O'
postgres$# WHERE ID=tempvar.ID;
postgres$# ELSIF (total>240) THEN
postgres$# UPDATE STDNT SET GR='A'
postgres$# WHERE ID=tempvar.ID;
postgres$# ELSIF (total>210) THEN
postgres$# UPDATE STDNT SET GR='B'
postgres$# WHERE ID=tempvar.ID;
postgres$# ELSIF (total>180) THEN
postgres$# UPDATE STDNT SET GR='C'
postgres$# WHERE ID=tempvar.ID;
postgres$# ELSE
postgres$# UPDATE STDNT SET GR='D'
postgres$# WHERE ID=tempvar.ID;
postgres$# END IF;
postgres$# END LOOP;
postgres$# CLOSE temp;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM STDNT;
 id | sname  | m1 | m2 | m3 | gr
----+--------+----+----+----+----
 88 | Anu    | 39 | 67 | 92 |
 10 | Jan    | 58 | 61 | 29 |
 30 | Karuna | 87 | 79 | 77 |
 29 | Jossy  | 39 | 80 | 45 |
(4 rows)

postgres=# SELECT FINDGR();
 findgr
--------

(1 row)

postgres=# SELECT * FROM STDNT;
 id | sname  | m1 | m2 | m3 | gr
----+--------+----+----+----+----
 88 | Anu    | 39 | 67 | 92 | C
 10 | Jan    | 58 | 61 | 29 | D
 30 | Karuna | 87 | 79 | 77 | A
 29 | Jossy  | 39 | 80 | 45 | D
(4 rows)

postgres=#
```

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.

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION CALCINTR()
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# temp CURSOR FOR SELECT * FROM BANKDETAILS;
postgres$# tempvar RECORD;
postgres$# interest INT;
postgres$# BEGIN
postgres$# OPEN temp;
postgres$# LOOP
postgres$# FETCH temp INTO tempvar;
postgres$# EXIT WHEN NOT FOUND;
postgres$# interest:=0.08*tempvar.BALANCE;
postgres$# INSERT INTO BANKNEW
postgres$# VALUES(tempvar.ACCNO,interest);
postgres$# END LOOP;
postgres$# CLOSE temp;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM BANKDETAILS;
 accno |  name   | balance |   adate
-------+---------+---------+------------
  1001 | Aby     |    3005 | 2015-10-10
  1002 | Alan    |    4000 | 1995-05-05
  1003 | Amal    |    5000 | 1992-03-16
  1004 | Jeffin  |    3500 | 2050-04-01
  1005 | Majo    |    6600 | 2001-01-01
(5 rows)

postgres=# SELECT * FROM BANKNEW;
 accno | interest
-------+----------
(0 rows)

postgres=# SELECT CALCINTR();
 calcintr
----------

(1 row)

postgres=# SELECT * FROM BANKNEW;
 accno | interest
-------+----------
  1001 |      240
  1002 |      320
  1003 |      400
  1004 |      280
  1005 |      528
(5 rows)

postgres=#
```

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).

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION EXP()
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# temp CURSOR FOR SELECT * FROM PEOPLE_LIST;
postgres$# tempvar RECORD;
postgres$# year INT;
postgres$# exp INT;
postgres$# BEGIN
postgres$# OPEN temp;
postgres$# LOOP
postgres$# FETCH temp INTO tempvar;
postgres$# EXIT WHEN NOT FOUND;
postgres$# year:=EXTRACT(YEAR FROM tempvar.DT_JOINING);
postgres$# exp:=2019-year;
postgres$# IF (exp>10) THEN
postgres$# INSERT INTO EXP_LIST
postgres$# VALUES(tempvar.ID,tempvar.NAME,exp);
postgres$# END IF;
postgres$# END LOOP;
postgres$# CLOSE temp;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM PEOPLE_LIST;
 id  |  name  | dt_joining | place
-----+--------+------------+-------
 101 | Robert | 2005-04-03 | CHY
 102 | Mathew | 2008-06-07 | CHY
 103 | Luffy  | 2003-04-15 | FSN
 104 | Lucci  | 2009-08-13 | KTM
 105 | Law    | 2005-04-14 | WTC
 106 | Vivi   | 2010-09-21 | ABA
(6 rows)

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

postgres=# SELECT EXP();
 exp
-----

(1 row)

postgres=# SELECT * FROM EXP_LIST;
 id  |  name  | exp
-----+--------+-----
 101 | Robert |  14
 102 | Mathew |  11
 103 | Luffy  |  16
 105 | Law    |  14
(4 rows)

postgres=#
```

4. Create table employee_list(id,name,monthly salary). If: annual salary<60000, increment monthly salary by 25% between 60000 and 200000, increment by 20% between 200000 and 500000, increment by 15% annual salary>500000, increment monthly salary by 10%

SOURCE CODE

```
postgres=# CREATE OR REPLACE FUNCTION INCREMENT()
postgres-# RETURNS VOID AS
postgres-# $$
postgres$# DECLARE
postgres$# temp CURSOR FOR SELECT * FROM EMP_LIST;
postgres$# tempvar RECORD;
postgres$# BEGIN
postgres$# OPEN temp;
postgres$# LOOP
postgres$# FETCH temp INTO tempvar;
postgres$# EXIT WHEN NOT FOUND;
postgres$# IF (tempvar.M_SAL<60000) THEN
postgres$# UPDATE EMP_LIST SET M_SAL=1.25*tempvar.M_SAL
postgres$# WHERE ID=tempvar.ID;
postgres$# ELSIF (tempvar.M_SAL<200000) THEN
postgres$# UPDATE EMP_LIST SET M_SAL=1.2*tempvar.M_SAL
postgres$# WHERE ID=tempvar.ID;
postgres$# ELSIF (tempvar.M_SAL<500000) THEN
postgres$# UPDATE EMP_LIST SET M_SAL=1.15*tempvar.M_SAL
postgres$# WHERE ID=tempvar.ID;
postgres$# ELSE
postgres$# UPDATE EMP_LIST SET M_SAL=1.1*tempvar.M_SAL
postgres$# WHERE ID=tempvar.ID;
postgres$# END IF;
postgres$# END LOOP;
postgres$# CLOSE temp;
postgres$# END;
postgres$# $$
postgres-# LANGUAGE PLPGSQL;
CREATE FUNCTION
postgres=#
```

OUTPUT

```
postgres=# SELECT * FROM EMP_LIST;
 id  |  name  | m_sal
-----+--------+--------
 101 | Mathew |  55000
 102 | Jose   |  80000
 103 | John   | 250000
 104 | Ann    | 600000
(4 rows)

postgres=# SELECT INCREMENT();
 increment
-----------

(1 row)

postgres=# SELECT * FROM EMP_LIST;
 id  |  name  | m_sal
-----+--------+--------
 101 | Mathew |  68750
 102 | Jose   |  96000
 103 | John   | 287500
 104 | Ann    | 660000
(4 rows)

postgres=#
```

RESULT

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