NAME:AVINASH BHIWALE   ROLL NO.:T1851050   PRN NO.:71901284J   DIV:A

Assignment No. 10

Aim: Write a PL/SQL block to implement all types of cursor.

Objective:

- To study and implement PL/SQL cursors.

Theory :

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and **%BULK_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes −

S.No	Attribute & Description

1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

DeclaringtheCursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS

  SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Conclusion:-

We have studied and implemented cursors in PL/SQL.

CODE:

1) First Example

```
mysql> create procedure CONNAMSAL()
-> begin
-> declare ename varchar(100);
-> declare esalary integer(10);
-> declare v_finished integer default 0;
-> declare c1 cursor for select NAME, SALARY from EMPLOYEE;
-> declare continue handler for NOT FOUND set v_finished=1;
-> open c1;
-> get_emp:LOOP
-> fetch c1 into ename,esalary; if v_finished=1 then
-> leave get_emp; end if;
-> select concat(ename,esalary); end loop get_emp;
-> close c1;
-> end $
Query OK, 0 rows affected (0.05 sec)
mysql> call CONNAMSAL()$
+-----------------------+
| concat(ename,esalary) |
+-----------------------+
| AMLAN10000 |
+-----------------------+
1 row in set (0.00 sec)
+-----------------------+
| concat(ename,esalary) |
+-----------------------+
| PIYUSH20000 |
+-----------------------+
1 row in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

2) Second Example

```
mysql> SELECT * FROM EMPLOYEE$
+........+------+...........+...........+..........+
| ID | MARKS1 | MARKS2 | MARKS3 | GRADE |
+........+------+...........+...........+..........+
| 1 | 50 | 50 | 50 | NULL |
| 2 | 60 | 62 | 63 | NULL |
| 1 | 65 | 70 | 75 | NULL |
| 3 | 90 | 90 | 95 | NULL |
| 4 | 80 | 80 | 85 | NULL |
+........+------+...........+...........+..........+
5 rows in set (0.00 sec)
mysql> create procedure grade2()
-> begin
-> declare
-> eid,m1,m2,m3 int;
-> declare avgl float;
-> declare finished int default 0;
-> declare c1 cursor for select ID,MARKS1,MARKS2,MARKS3 from EMPLOYEE;
-> declare continue handler for not found set finished=1;
-> open c1;
-> l4:loop fetch c1 into eid,m1,m2,m3;
-> if(finished=1)
-> then
-> leave l4;
-> end if;
-> set avgl = (m1+m2+m3)/3;
-> if(avgl>=80)
-> then
-> update EMPLOYEE set grade='a' where eid=id;
-> else if(avgl>=60 and avgl<80)
-> then
-> update EMPLOYEE set grade='b' where eid=id;
-> else if(avgl>=40 and avgl<60)
-> then
-> update EMPLOYEE set grade='c' where eid=id;
-> else
-> update EMPLOYEE set grade='f' where eid=id;
-> end if;
-> end if;
-> end if;
-> end loop;
-> close c1;
-> end;
-> $
Query OK, 0 rows affected (0.00 sec)
mysql> call grade2()$
Query OK, 0 rows affected (0.20 sec)
mysql> select * from EMPLOYEE$
+.........+- - - - - -+............+............+..........+
| ID | MARKS1 | MARKS2 | MARKS3 | GRADE |
+.........+- - - - - -+............+............+..........+
| 1 | 50 | 50 | 50 | b |
| 2 | 60 | 62 | 63 | b |
| 1 | 65 | 70 | 75 | b |
| 3 | 90 | 90 | 95 | a |
| 4 | 80 | 80 | 85 | a |
+.........+- - - - - -+............+............+..........+
5 rows in set (0.00 sec)
```

3) Third Example

```
mysql> create procedure stat()
-> begin
-> declare id,sal int;
-> declare finished int default 0;
-> declare c2 cursor for select cid,salary from customer;
-> declare continue handler for not found set finished=1;
-> open c2;
-> l1:loop fetch c2 into id,sal;
-> if(finished=1)
-> then leave l1;
-> end if;
-> if(sal>=60000)
-> then update customer set status='platinum' where cid=id;
-> else if(sal>=40000 and sal<60000)
-> then update customer set status='gold' where cid=id;
-> else if(sal<40000 and sal>=10000)
-> then update customer set status='silver' where cid=id;
-> end if;
-> end if;
-> end if;
-> end loop;
-> close c2;
-> end;
-> $$
Query OK, 0 rows affected (0.17 sec)
mysql> select * from customer;
-> $$
+......+..................+- - - - - -+...........+
| cid | cname | salary | status |
+......+..................+..........-+...........+
| 1 | shweta | 98000 | NULL |
| 2 | mahi | 10000 | NULL |
| 3 | bhagyashree | 75000 | NULL |
| 4 | mitesh | 67000 | NULL |
| 5 | sarika | 15000 | NULL |
| 6 | sushila | 45000 | NULL |
| 7 | reshma | 65000 | NULL |
| 8 | pratibha | 99000 | NULL |
+......+..................+...........+...........+
8 rows in set (0.00 sec)
mysql> call stat$$
Query OK, 0 rows affected, 1 warning (0.24 sec)
mysql> select * from customer;
-> $$
```

```
+-------+----------------+---------+------------+
| cid | cname | salary | status |
+-------+----------------+---------+------------+
| 1 | shweta | 98000 | platinum |
| 2 | mahi | 10000 | silver |
| 3 | bhagyashree | 75000 | platinum |
| 4 | mitesh | 67000 | platinum |
| 5 | sarika | 15000 | silver |
| 6 | sushila | 45000 | gold |
| 7 | reshma | 65000 | platinum |
| 8 | pratibha | 99000 | platinum |
+-------+----------------+---------+------------+
8 rows in set (0.00 sec
```