ASSIGNMENT NO.06
TITLE: CURSORS: IMPLICIT, EXPLICIT, CURSOR FOR LOOP, PARAMETERIZED CURSOR

PROBLEM STATEMENT: Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped. Also demonstrate working of all types of cursors.

OBJECTIVES:
- To learn manipulation of data using cursors
- To understand working of various types of cursors and their appropriate use

OUTCOMES:
- Students will be able to manipulate data using cursors.
- Students will be able to use various types of cursors identifying its need.

SOFTWARE & HARDWARE REQUIREMENTS:
1. 64-bit Open source Linux or its derivative
2. Oracle Server

THEORY:
When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.
Cursor is the work area which Oracle reserves for internal processing of SQL statements. This work area is private for oracles reserved are called cursor.
In PL/SQL block SELECT statement cannot return more than one row at a time. So Cursor is used to group these rows (more than one row) for implementing certain logic to get all the group of records. A PL/SQL cursor is a **pointer** that points to the result set of an SQL query against database tables.

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.

Implicit Cursor Attributes

Following are implicit cursor attributes,

Cursor Attribute	Cursor Variable	Description
%ISOPEN	SQL%ISOPEN	Oracle engine automatically open the cursor If cursor open **return TRUE** otherwise **return FALSE.**
%FOUND	SQL%FOUND	If SELECT statement return one or more rows or DML statement (INSERT, UPDATE, DELETE) affect one or more rows If affect **return TRUE** otherwise **return FALSE.** If not execute SELECT or DML statement **return NULL.**
%NOTFOUND	SQL%NOTFOUND	If SELECT INTO statement return no rows and fire no_data_found PL/SQL exception before you can check SQL%NOTFOUND. If not affect the row **return TRUE** otherwise **return FALSE.**
%ROWCOUNT	SQL%ROWCOUNT	Return the number of rows affected by a SELECT statement or DML statement (insert, update, delete). If not execute SELECT or DML statement **return NULL.**

Implicit Cursor Example:

```
SET SERVEROUTPUT ON;
DECLARE
  total_rows number(2);
```

```
BEGIN
  UPDATE customers
  SET ORDER_AMT = ORDER_AMT + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
```

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

Declaring the Cursor

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;

Example:

```
SET SERVEROUTPUT ON;
DECLARE
  c_id customers.cust_id%type;
  c_name customers.cust_name%type;
  c_addr customers.cust_address%type;
  CURSOR c_customers is
    SELECT cust_id, cust_name, cust_address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
  FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
```

Cursor For Loop

PL/SQL cursor FOR loop has one great advantage of loop continued until row not found. In sometime you require to use explicit cursor with FOR loop instead of use OPEN, FETCH, and CLOSE statement.

FOR loop iterate repeatedly and fetches rows of values from database until row not found.

The cursor FOR LOOP statement implicitly declares its loop index as a record variable of the row type that a specified cursor returns, and then opens a cursor. With each iteration, the cursor FOR LOOP statement fetches a row from the result set into the record. When there are no more rows to fetch, the cursor FOR LOOP statement closes the cursor. The cursor also closes if a statement inside the loop transfers control outside the loop or raises an exception.

The prototype for defining an explicit cursor FOR loop is shown below

```
FOR <Loop_index> IN (<Cursor_name>)

LOOP

<Executable_statements>;

END LOOP [<Loop_index>];
```

Example:

```
SET SERVEROUTPUT ON;
```

```
DECLARE
  c_id customers.cust_id%type;
  c_name customers.cust_name%type;
  c_addr customers.cust_address%type;
  CURSOR c_customers is
    SELECT cust_id, cust_name, cust_address FROM customers;
BEGIN
  FOR item in c_customers
    LOOP
        dbms_output.put_line(item.c_id || ' ' || item.c_name || ' ' || item.c_addr);
  END LOOP;
  CLOSE c_customers;
END;
```

Parameterized Cursor

PL/SQL Parameterized cursor pass the parameters into a cursor and use them in to query. PL/SQL Parameterized cursor define only datatype of parameter and there is not need to define it's length. Default values are assigned to the Cursor parameters and scope of the parameters is local.

Parameterized cursors are also called static cursors that can pass parameter value when cursor is opened.

Example:

```
DECLARE
        CURSOR PARAM_CURSOR (ROLL NUMBER) IS SELECT * FROM N_ROLLCALL
        WHERE ROLLNO = ROLL;
        TMP PARAM_CURSOR %ROWTYPE;
BEGIN
        FOR TMP IN PARAM_CURSOR (101) LOOP
        dbms_output.put_line('NAME   '||TMP.NAME);
        dbms_output.put_line('DATE '||TMP.ATT_DATE);
        dbms_output.put_line('ATTENDANCE:  '||TMP.ATTENDANCE);
        END Loop;
END;
```

CONCLUSION: