Name :- Rushikesh Kazbhari Palve
Roll No. 31258

: Date :	Page No :
/ /20	

Assignment No. A6

DOP :- 28-09-2021 **DOS :-** 30-11-2021

Title :- Cursors : (All types : Implicit, Explicit, Cursor FOR loop, Parameterized Cursor)

Problem Definition :-

Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_EmpId with the data available in the table O_EmpId. If the data in the first table already exist in the second table then the data should be skipped.

Objectives :-

(i) Understand the types of cursors.
(ii) Understand how to use cursors with PL/SQL block.

Outcomes :-

After completion of the assignment students will be able to
(i) Understand the concept of cursors
(ii) Use cursors with PL/SQL block.

Theory :-

CURSOR :-

For the processing of any SQL statement, database needs to allocate memory. This memory is called context area. The context area is a part of PGA (Process global area) and is allocated on the Oracle server. A cursor is associated with this work area used by ORACLE, for multi row queries. A cursor is a handle or pointer to the context area. The cursor allows to process contents in the context area row by row.
There are two types of cursors :-

1] Implicit Cursor :- Implicit cursors are defined by ORACLE implicitly. ORACLE defines implicit cursor for every DML statements.

2] Explicit Cursor :- There are user-defined cursors are which are defined in the declaration section of the PL/SQL block. There are four steps in which the explicit cursor is processed.

1) Declaring a cursor
2) Opening a cursor
3) Fetching rows from an opened cursor
4) closing cursor.

General syntax for CURSOR :-

```
DECLARE
    Cursor cursor_name IS select_statement or query
BEGIN
    Open cursor_name;
Fetch cursor_name into list_of_variables;
    close cursor_name;
END;
```

Where,

⇒ (i) cursor_name :- is the name of the cursor

⇒ (ii) select_statement :- is the query that defines the set of rows to be processed by the cursor.

⇒ (iii) Open cursor_name :- Open the cursor that has been previously declared.

When cursor is opened following things happen

ⓐ The active set pointer is set to the first row

ⓑ The value of the binding variables are examined.

⇒ (iv) Fetch statement is used to retrieve a row from the selected row, one at a time, into PL/SQL variables.

⇒ (v) close cursor_name :- When all of cursor rows have been retrieved, the cursor should be closed.

Explicit cursor attributes :-

Following are the cursor attributes

1.] %FOUND :- this is Boolean attribute. It returns TRUE if the previous fetch returns a row and false if it doesn't

2.] %NOTFOUND :- If fetch returns a row it returns FALSE and TRUE if doesn't. This is often used as the exit condition for the fetch loop.

3.] %ISOPEN :- This attribute is used to determine whether or not the associated cursor is open. If so it return TRUE otherwise FALSE.

4.] %ROWCOUNT :- This numeric attribute returns a number of rows fetched by the cursor.

Cursor Fetch Loops :-

1] Simple loop :

Syntax :

```
LOOP
    Fetch cursorname into list of variables;
    EXIT WHEN cursorname %NOTFOUND
        sequence_of_statements;
END LOOP;
```

2] WHILE LOOP :

Syntax :

```
FETCH cursorname INTO list of variables;
WHILE cursorname%FOUND LOOP
        sequence_of_statements;
        FETCH cursorname INTO list of variables;
END LOOP;
```

3] Cursor FOR Loop :

syntax :

```
FOR variable_name IN cursorname LOOP
        -- an implicit fetch is done here.
        -- cursorname%NOTFOUND is also implicitly checked
        -- process the fetch records
        sequence_of_statements;
END LOOP;
```

There are two important things to note about :-

i) variable_name is not declared in the DECLARE section. This variable is implicitly declared by the PL/SQL compiler.

ii) Type of this variable is cursorname%ROWTYPE.

Implicit Cursors :-

PL/SQL issues an implicit cursor whenever you execute a SQL statement directly in your code, as long as that code does not employ an explicit cursor. It is called an "implicit" cursor because you, the developer, do not explicitly declare a cursor for the SQL statement.

If you use an implicit cursor, Oracle performs the open, fetches and close for you automatically; these actions are outside of your programmatic control. You can, however, obtain information about the most recently executed SQL statement by examining the values in the implicit SQL cursor attributes. PL/SQL employs an implicity cursor for each UPDATE, DELETE or INSERT statement you execute in a program. You cannot, in other words, execute these statements within an explicit cursor, even if you want to. You have a choice between using an implicit or explicit cursor only when you execute a single-row SELECT statement (a SELECT that returns only one row).

In the following UPDATE statement, which gives everyone in the company a 10% raise, PL/SQL creates an implicit cursor to identify the set of rows in the table which would be affected by the update:

```
UPDATE employee
    SET salary = salary * 1.1;
```

The following single-row query calculates and returns the total salary for a department. Once again, PL/SQL creates an implicit cursor for this statement:

```
SELECT SUM(salary) INTO department_total
  FROM employee
 WHERE department_number = 10;
```

If you have a SELECT statement that returns more than one row, you must use an explicit cursor for that query and then process the rows returned one at a time. PL/SQL does not yet support any kind of array interface between a database table and a composite PL/SQL datatype such as a PL/SQL table.

Drawbacks of Implicit Cursors :-

Even if your query returns only a single row, you might still deride to use an explicit cursor. The implicit cursor has the following drawbacks :

- It is less efficient than an explicit cursor
- It is more vulnerable to data errors
- It gives you less programmatic control.

The following sections explore each of these limitations to the implicit cursor.

Inefficiencies of implicit cursors :-

An explicit cursor is, at least theoretically, more efficient than an implicit cursor. An implicit cursor executes a SQL statement and Oracle's SQL is ANSI-standard. ANSI dictates that a single-row query must not only fetch the first record but must also perform a second fetch to determine if too many rows will be returned by that query (such a

situation will RAISE the TOO_MANY_ROWS PL/SQL exception). Thus, an implicit query always performs a minimum of two fetches, while an explicit cursor only needs to perform a single fetch.

This additional fetch is usually not noticeable, and you shouldn't be neurotic about using an implicit cursor for a single-row query (it takes less coding, so the temptation is always there). Look out for indiscriminate use of the implicit cursor in the parts of your application where the cursor will be executed repeatedly. A good example is the post-Query trigger in the Oracle Forms.

post-Query fires once for each record retrieved by the query (created from the base table block and the criteria entered by the user). If a query retrieves ten rows, then an additional ten fetches are needed with an implicit query. If you have 25 users on your system all performing a similar query, your server must process 250 additional (unnecessary) fetches against the database. So, while it might be easier to write an implicit query, there are some places in your code where you will want to make that extra effort and go with the explicit cursor.

~~Truth~~

Vulnerability of data errors :-

If an implicit SELECT statement returns more than one row, it raise the TOO_MANY_ROWS

exception. When this happens, execution in the current block terminates and control is passed to the exception section. Unless you deliberately plan to handle this srenario, use of the implicit cursor is a declaration of faith. You are saying, "I trust that query to always return a single row!"

It may well be that today, with the current data, the query will only return a single row. If the nature of the data ever changes, however, you may find the SELECT statement which formerly identified a single row now returns several. Your program will raise an exception. Perhaps this is what you will want. On the other hand, perhaps the presence of additional records is inconsequential and should be ignored.

With the implicit query you cannot easily handle these different possibilities. With an explicit query, your program will be protected against changes in data and will continue to fetch rows without raising exceptions.

Conclusion :-

We successfully understood the concept of Cursors, cursor types. Also, used cursors with PL/SQL block.

OUTPUT :-

```
/*
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.
*/

mysql> CREATE TABLE O_Employee(
    -> EmpId INT PRIMARY KEY,
    -> EName VARCHAR(20),
    -> DId VARCHAR(20),
    -> DName VARCHAR(15)
    -> );

mysql> INSERT INTO O_Employee VALUES
    -> (1, 'Vidyut', 10, 'Computer'),
    -> (2, 'Pratap', 20,'IT'),
    -> (3, 'Kailash', 30, 'E&TC'),
    -> (4, 'Mukund', 40,'Mechanical'),
    -> (5, 'Girish', 50,'Civil'),
    -> (6, 'Neeraj', 60, 'Electrical'),
    -> (7, 'Prashant', 30, 'E&TC'),
    -> (8, 'Raj', 10, 'Computer'),
    -> (9, 'Hari', 20,'IT'),
    -> (10, 'Aditya', 40,'Mechanical');

mysql> CREATE TABLE N_Employee(
    -> EmpId INT PRIMARY KEY,
    -> EName VARCHAR(20),
    -> DId VARCHAR(20),
    -> DName VARCHAR(15)
    -> );
```

mysql> # Procedure Without Cursor

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE Copy(
    -> IN eID INT
    -> )
    -> BEGIN
    -> DECLARE flag BOOLEAN;
    -> DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'ENTRY NOT FOUND' AS EXCEPTION;
-> SET flag = FALSE;
-> IF NOT EXISTS(SELECT * FROM  O_Employee WHERE EmpId = eID) THEN
-> SIGNAL SQLSTATE '45000';
-> END IF;
-> IF NOT EXISTS(SELECT * FROM N_Employee WHERE EmpId = eID ) THEN
-> SET flag = TRUE;
-> INSERT INTO N_Employee
-> SELECT * FROM O_Employee
-> WHERE EmpId = eID;
-> END IF;
-> IF NOT flag THEN
-> SELECT 'Record Already Exists' AS MESSAGE;
-> ELSE
-> SELECT * FROM N_Employee;
```

```
    -> END IF;
    -> END $$
```

mysql> # Procedure With Cursor

```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE NewCopy(
    -> IN eID INT
-> )
-> BEGIN
-> DECLARE flag BOOLEAN;
-> DECLARE C1 CURSOR FOR SELECT EmpId FROM O_Employee WHERE EmpId = eID;
-> DECLARE EXIT HANDLER FOR NOT FOUND SELECT 'ENTRY NOT FOUND' AS EXCEPTION;
-> OPEN C1;
-> FETCH C1 INTO eID;
-> SET flag = FALSE;
-> IF NOT EXISTS(SELECT * FROM N_Employee WHERE EmpId = eID ) THEN
-> SET flag = TRUE;
-> INSERT INTO N_Employee
-> SELECT * FROM O_Employee
-> WHERE EmpId = eID;
-> END IF;
-> IF NOT flag THEN
-> SELECT 'Record Already Exists' AS MESSAGE;
-> ELSE
-> SELECT * FROM N_Employee;
-> END IF;
-> CLOSE C1;
-> END $$
```

```
mysql> DELIMITER ;
mysql> CALL Copy(1);
mysql> CALL Copy(2);
mysql> CALL Copy(3);
mysql> CALL Copy(4);
mysql> CALL Copy(5);
```

EmpId	EName	DId	DName
1	Vidyut	10	Computer
2	Pratap	20	IT
3	Kailash	30	E&TC
4	Mukund	40	Mechanical
5	Girish	50	Civil

```
mysql> CALL Copy(11);
```

EXCEPTION
ENTRY NOT FOUND

```
mysql> CALL NewCopy(1);
```

```
+-----------------------+
| MESSAGE               |
+-----------------------+
| Record Already Exists |
+-----------------------+
```

```
mysql> CALL NewCopy(6);
mysql> CALL NewCopy(7);
mysql> CALL NewCopy(8);
mysql> CALL NewCopy(9);
mysql> CALL NewCopy(10);
```

```
+-------+----------+------+------------+
| EmpId | EName    | DId  | DName      |
+-------+----------+------+------------+
|     1 | Vidyut   | 10   | Computer   |
|     2 | Pratap   | 20   | IT         |
|     3 | Kailash  | 30   | E&TC       |
|     4 | Mukund   | 40   | Mechanical |
|     5 | Girish   | 50   | Civil      |
|     6 | Neeraj   | 60   | Electrical |
|     7 | Prashant | 30   | E&TC       |
|     8 | Raj      | 10   | Computer   |
|     9 | Hari     | 20   | IT         |
|    10 | Aditya   | 40   | Mechanical |
+-------+----------+------+------------+
```

```
mysql> CALL NewCopy(11);
```

```
+------------------+
| EXCEPTION        |
+------------------+
| ENTRY NOT FOUND  |
+------------------+
```