TATA CONSULTANCY SERVICES

Oracle PL/SQL – Cursors

Introduction

- Oracle creates a memory area known as **context area** for processing an SQL statement, which holds the following information needed for processing the statement
  - Rows returned by the query
  - Number of rows processed by the query
  - A pointer to the parsed query in the shared pool
- A **cursor** is a pointer to the context area. PL/SQL controls the context area using the cursor.
- A cursor holds the rows returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.
- The cursor points to the memory and not to the data directly.
- The cursor can be named 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.

Cursor Attributes

The following are the main attributes supported by the cursors

Attribute Name	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open and FALSE if the cursor is not open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a record.
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a record.
%ROWCOUNT	Boolean	Returns the number of records fetched from the cursor at that given time.

Types of Cursors

Two main types of cursors are :

- Implicit Cursors
- Explicit Cursors

Implicit Cursors:

- Cursors which are automatically created by Oracle when any DML or SELECT..INTO statement is executed are called **implicit cursors.**
- Developers cannot control the implicit cursors and the information in it.
- Implicit cursors are also referred as **SQL cursor**.
- Any implicit cursor attribute is referred as **sql%attribute_name.** It refers to the recently executed DML or SELECT statement.

Example:

In the below example, sql%notfound will return true if there is no record with employee ID 101.

```
SET SERVEROUTPUT ON;
DECLARE
        total_rows number(2);
BEGIN
      UPDATE employee SET salary = salary + 500 WHERE emp_id=101;
      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;
/
```

Output :

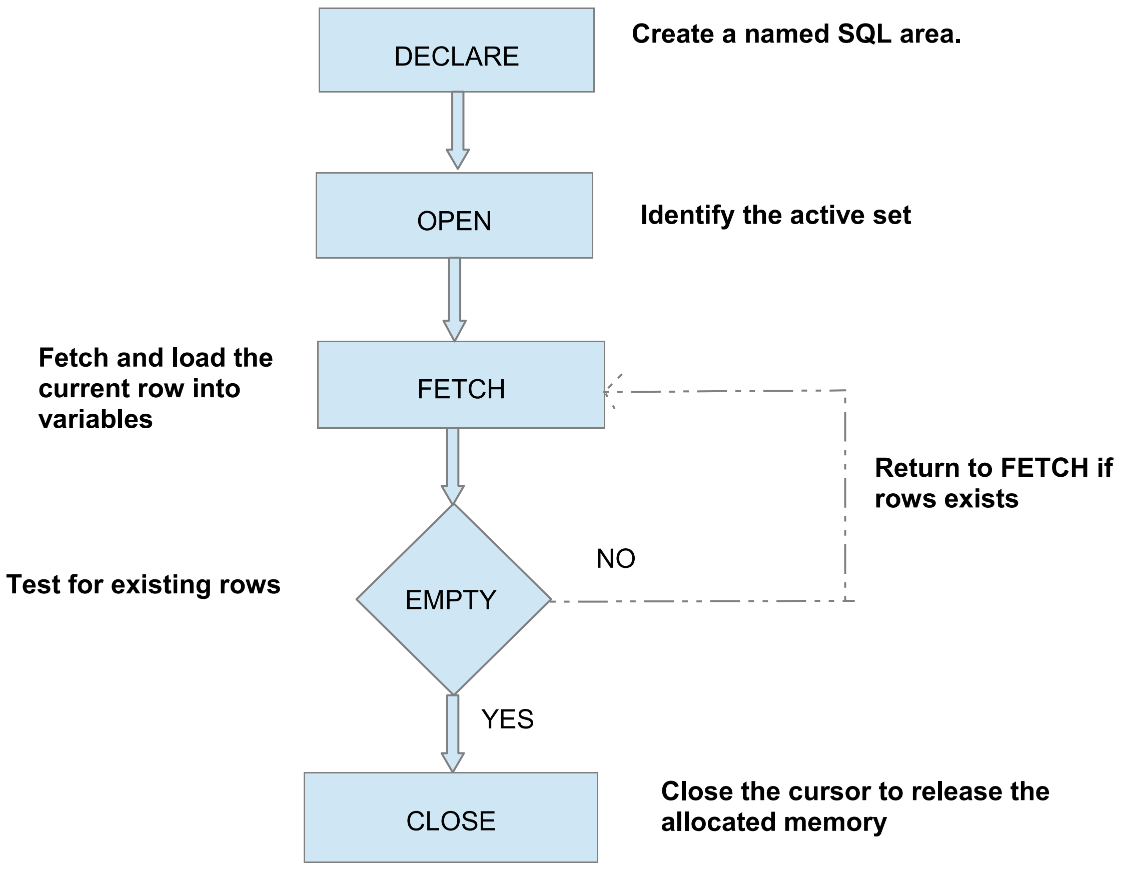
```
1 customers selected

PL/SQL procedure successfully completed
```

Explicit Cursors:

- Cursors which are explicitly defined by the programmers or developers are called explicit cursors
- These cursors are declared using a SELECT statement in the declaration section of the block.
- Explicit cursor provides the developer control over cursor processing.
- These cursors are meant to work with SELECT statements that return more than one record at a time.
- They are mainly used for processing or manipulating a result-set record wise, that is, record by record fetching and processing.
- Explicit cursors are also called as **named cursor.**

Working with an explicit cursor involves four steps:

- Declaring the cursor
- Opening the cursor which identifies the result set.
- Fetching the cursor for retrieving data
- Closing the cursor to release allocated memory


DECLARE
Create a named SQL area.
OPEN
Identify the active set
Fetch and load the current row into variables
FETCH
Return to FETCH if rows exists
NO
Test for existing rows
EMPTY
YES
CLOSE
Close the cursor to release the allocated memory

Declaring and Defining Explicit Cursors :

- Declares the cursor with a name and defines an associated SELECT statement.
- If rows need to be processed in a specific sequence then ORDER BY can be used in the SELECT statement

Syntax:

```
CURSOR cursor_name [ parameter_list ]
        IS select_statement;
```

Example

```
DECLARE
--declaring cursor
CURSOR C1 IS SELECT emp_id,emp_name,designation FROM employee;
BEGIN
NULL;
END;
/
```

Opening the cursor:

Opening the cursor does the following functions

- Allocates database resources to process the query
- Processes the query to identify the result set.
- Positions the cursor before the first row of the result set.

OPEN statement is included in the executable section of PL/SQL block.

Syntax:

```
OPEN <cursor_name>;
```

NOTE:

If data is added,deleted or modified after the cursor is opened , the new or changed data is not reflected in the cursor result set . Opening the cursor is literally like taking a snapshot of the data as it currently exists and not a dynamic pointer to live data.

Fetching Data from Cursor :

- FETCH statement retrieves the rows from the cursor to the variables
- It fetches the records one at a time
- It fetches the current row of the result set, stores the column values of that row into the variable and advances the cursor to the next row.
- FETCH statements are usually used inside a LOOP and the LOOP is terminated when the FETCH statement runs out of rows.

Syntax:

FETCH <*cursor_name*> into <*into_clause*>*;*

- into_clause is either a list of variables or a single record variable.
- For each column that the query returns, the variable list or record must have a corresponding type-compatible variable or field.

Example:

FETCH c1 into v_emp_id, v_emp_name;

where c1 is the cursor name and v_emp_id, v_emp_name are the variables which have the compatible data types like emp_id ,emp_name columns of Employee table respectively.

Closing the cursor:

- Closing the cursor means releasing the allocated memory.
- After closing a cursor, records cannot be fetched and the cursor attributes cannot be referenced.

Syntax:

CLOSE <*cursor_name*> ;

Example :

In the below example, c1 is the cursor name and the cursor points to the output of the SELECT query. Inside the loop, each row in the result set is fetched and the column values are printed. The control will come out of the loop when all the records in the cursor are processed.

```
SET SERVEROUTPUT ON;
DECLARE
       v_row Employee%ROWTYPE;
       --declaring cursor
       CURSOR c1 IS SELECT * FROM employee;
BEGIN
     OPEN c1; -- open the cursor
     LOOP
         FETCH c1 INTO v_row; -- fetch the records one by one to the variable
         EXIT WHEN c1%NOTFOUND;
             -- exit the loop when all the records in the cursor are processed
         dbms_output.put_line('Employee Id:'|| v_row.emp_id);
         dbms_output.put_line('Employee Name:'|| v_row.emp_name);
     END LOOP;
     CLOSE c1; -- closing the cursor
END;
/
```

Output :

```
Employee Id:101
Employee Name:Ryan
Employee Id:102
Employee Name:Brian

PL/SQL procedure successfully completed
```

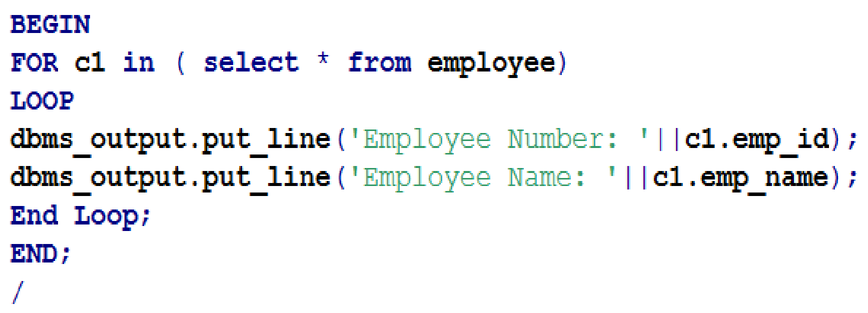
Cursor FOR Loops

- It is an easier method to process explicit cursors
- In Cursor FOR loop, open,fetch,exit and close happens implicitly and no need to explicitly mention them
- The record to which each row is fetched is implicitly declared

Syntax:

```
FOR <record_name> IN <cursor_name>
LOOP
         Statement 1;
        Statement 2;
        ..
END LOOP;
```

Example:



```
BEGIN
FOR c1 in ( select * from employee)
LOOP
dbms_output.put_line('Employee Number: '||c1.emp_id);
dbms_output.put_line('Employee Name: '||c1.emp_name);
End Loop;
END;
/
```

Output :

```
Employee Number: 101
Employee Name: Ryan
Employee Number: 102
Employee Name: Brian

PL/SQL procedure successfully completed
```

Cursor with Parameters

- Any number of parameters can be passed to explicit cursors.
- Parameter values can by dynamically used in the WHERE clause of the SELECT statement which is used for defining the cursor.

Declaring a Parameterized Cursor

```
CURSOR <cursor_name>(<variable_name datatype) IS <select_statement>;
```

Opening a Parameterized Cursor

```
OPEN <cursor_name>(Value/Variable/Expression);
```

Example:

Here, supervisor ID is the cursor parameter which is used in the WHERE clause. 3251 is passed as parameter to cursor and the cursor will point to the memory area holding the employee records with supervisor_ID as 3251.

```
DECLARE
        -- declaring cursor with parameter
        CURSOR c1(p_sid NUMBER) IS select * from employee where supervisor_id = p_sid;
        r_emp employee%rowtype;
        v_s_id number;
BEGIN
        v_s_id :=3251;
        OPEN c1(v_s_id);
        Loop
                FETCH c1 into r_emp;
                Exit when c1%notfound;
                dbms_output.put_line('Employee Number: '||r_emp.emp_id);
                dbms_output.put_line('Employee Name: '||r_emp.emp_name);
        End Loop;
        CLOSE c1;
END;
/
```

Output:

```
Employee Number: 1200
Employee Name: JOHN
Employee Number: 4562
Employee Name: Ryan

PL/SQL procedure successfully completed
```