# Oracle (PL/SQL)

## Lesson 02 : Cursors

Capgemini

# Lesson Objectives

- To understand the following topics:
  - Introduction to Cursors
  - Implicit and Explicit Cursors
  - Cursor attributes
  - Processing Implicit Cursors and Explicit Cursors
  - Cursor with Parameters
  - Difference between Cursors and Cursor Variables
  - Use of Cursor Variables

# Concept

- A cursor is a “handle” or “name” for a private SQL area.
  - An SQL area (context area) is an area in the memory in which a parsed statement and other information for processing the statement are kept.
  - 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 must declare an explicit cursor.
  - Thus the two types of cursors are:
    - implicit
    - explicit

# Concept

- Implicit Cursor:
  - The PL/SQL engine takes care of automatic processing.
  - PL/SQL implicitly declares cursors for all DML statements.
  - They are simple SELECT statements and are written in the BEGIN block (executable section) of the PL/SQL.
  - They are easy to code, and they retrieve exactly one row

# Implicit Cursors

- Processing Implicit Cursors:
  - Oracle implicitly opens a cursor to process each SQL statement that is not associated with an explicitly declared cursor.
  - This implicit cursor is known as SQL cursor.
    - Program cannot use the OPEN, FETCH, and CLOSE statements to control the SQL cursor. PL/SQL implicitly does those operations .
    - You can use cursor attributes to get information about the most recently executed SQL statement.
    - Implicit Cursor is used to process INSERT, UPDATE, DELETE, and single row SELECT INTO statements.

# Implicit Cursors - Example

```
BEGIN
    UPDATE dept SET dname =`Production' WHERE deptno= 50;
    IF SQL%NOTFOUND THEN
      INSERT into department_master VALUES ( 50, `Production');
    END IF;
END;
```

```
BEGIN
   UPDATE dept SET dname =`Production' WHERE deptno = 50;
   IF SQL%ROWCOUNT = 0 THEN
      INSERT into  department_master VALUES ( 50, `Production');
END IF;
END;
```

# Explicit Cursors

- Explicit Cursor:
  - The set of rows returned by a query can consist of zero, one, or multiple rows, depending on how many rows meet your search criteria.
  - When a query returns multiple rows, you can explicitly declare a cursor to process the rows.
  - You can declare a cursor in the declarative part of any PL/SQL block, subprogram, or package.
  - Processing has to be done by the user.

# Processing Explicit Cursors

- While processing Explicit Cursors you have to perform the following four steps:
  - Declare the cursor
  - Open the cursor for a query
  - Fetch the results into PL/SQL variables
  - Close the cursor

# Processing Explicit Cursors

- Declaring a Cursor:
  - Syntax:

```
CURSOR Cursor_Name IS Select_Statement;
```

  - Any SELECT statements are legal including JOINS, UNION, and MINUS clauses.
    - SELECT statement should not have an INTO clause.
  - Cursor declaration can reference PL/SQL variables in the WHERE clause.
    - The variables (bind variables) used in the WHERE clause must be visible at the point of the cursor.

# Processing Explicit Cursors

- Opening a Cursor
  - Syntax:

```
OPEN  Cursor_Name;
```

  - When a cursor is opened, the following events occur:
    1. The values of bind variables are examined.
    2. The active result set is determined.
    3. The active result set pointer is set to the first row.

# Processing Explicit Cursors

- Fetching from a Cursor
  - Syntax:

```
FETCH Cursor_Name  INTO List_Of_Variables;
FETCH  Cursor_Name INTO PL/SQL _Record;
```

  - The "list of variables" in the INTO clause should match the "column names list" in the SELECT clause of the CURSOR declaration, both in terms of count as well as in datatype.
  - After each FETCH, the active set pointer is increased to point to the next row.
    - The end of the active set can be found out by using %NOTFOUND attribute of the cursor.

# Processing Explicit Cursors

## Closing a Cursor

- Syntax

```
CLOSE  Cursor_Name;
```

- Closing a Cursor frees the resources associated with the Cursor.
  - You cannot FETCH from a closed Cursor.
  - You cannot close an already closed Cursor.

# Attributes

- Cursor Attributes:
  - Explicit cursor attributes return information about the execution of a multi-row query.
  - When an “Explicit cursor” or a “cursor variable” is opened, the rows that satisfy the associated query are identified and form the result set.
  - Rows are fetched from the result set.
  - Examples: %ISOPEN, %FOUND, %NOTFOUND, %ROWCOUNT, etc.

# Types of Cursor Attributes

- The different types of cursor attributes are described in brief, as follows:
  - %ISOPEN
    - %ISOPEN returns TRUE if its cursor or cursor variable is open. Otherwise it returns FALSE.
    - Syntax:

```
Cur_Name%ISOPEN
```

# Types of Cursor Attributes

- Example:

```
DECLARE
        cursor c1 is
    select_statement ;
 BEGIN
    IF c1%ISOPEN THEN
        pl/sql_statements ;
    END IF;
  END ;
```

# Types of Cursor Attributes

- %FOUND
  - %FOUND yields NULL after a cursor or cursor variable is opened but before the first fetch.
  - Thereafter, it yields:
    - TRUE if the last fetch has returned a row, or
    - FALSE if the last fetch has failed to return a row
  - Syntax:

```
cur_Name%FOUND
```

# Types of Cursor Attributes

- Example:

```
DECLARE section;
    open c1 ;
    fetch c1 into var_list ;
IF c1%FOUND THEN
     pl/sql_statements ;
END IF ;
```

# Types of Cursor Attributes

- %NOTFOUND
  - %NOTFOUND is the logical opposite of %FOUND.
  - %NOTFOUND yields:
    - FALSE if the last fetch has returned a row, or
    - TRUE if the last fetch has failed to return a row
  - It is mostly used as an exit condition.
  - Syntax:

```
cur_Name%NOTFOUND
```

# Types of Cursor Attributes

- %ROWCOUNT
  - %ROWCOUNT returns number of rows fetched from the cursor area using FETCH command.
  - %ROWCOUNT is zeroed when its cursor or cursor variable is opened.
    - Before the first fetch, %ROWCOUNT yields 0.
    - Thereafter, it yields the number of rows fetched at that point of time.
  - The number is incremented if the last FETCH has returned a row.
  - Syntax:

```
cur_Name%NOTFOUND
```

# Cursor FETCH loops

- They are examples of simple loop statements.
- The FETCH statement should be followed by the EXIT condition to avoid infinite looping.
- Condition to be checked is  cursor%NOTFOUND.
- Examples: LOOP .. END LOOP, WHILE LOOP, etc

# Cursor using LOOP … END LOOP:

```
DECLARE
      cursor c1 is ……..
BEGIN
     open cursor c1;  /* open the cursor and identify the active result set.*/
LOOP
     fetch c1 into  variable_list ;
     -- exit out of the loop when there are no more rows.
     /* exit is done before processing to prevent handling of null rows.*/
     EXIT WHEN C1%NOTFOUND ;
     /* Process the fetched rows using variables and  PL/SQLstatements */
END LOOP;
     -- Free resources used by the cursor
     close  c1;
     -- commit
     commit;
END;
```

# FOR Cursor LOOP

- FOR Cursor Loop

```
FOR Variable in Cursor_Name
    LOOP
        Process the variables
    END LOOP;
```

- You can pass parameters to the cursor in a CURSOR FOR loop.

```
FOR Variable in Cursor_Name ( PARAM1 , PARAM 2 ….)
    LOOP
        Process the variables
    END LOOP;
```

# SELECT… FOR UPDATE

- SELECT … FOR UPDATE cursor:
  - The method of locking records which are selected for modification, consists of two parts:
    - The FOR UPDATE clause in CURSOR declaration.
    - The WHERE CURRENT OF clause in an UPDATE or DELETE statement.
    - Syntax: FOR UPDATE

```
CURSOR   Cursor_Name IS  SELECT  ….. FROM … WHERE   .. ORDER
BY     FOR UPDATE  [OF column names ] [ NOWAIT]
```

where column names are the names of the columns in the table against which the query is fired. The column names are optional.

# SELECT... FOR UPDATE

- **If the cursor is declared with a FOR UPDATE clause, the WHERE CURRENT OF clause can be used in an UPDATE or DELETE statement.**
  - Syntax: WHERE CURRENT OF

```
WHERE CURRENT OF Cursor_Name
```

  - The WHERE CURRENT OF clause evaluates up to the row that was just retrieved by the cursor.
  - When querying multiple tables Rows in a table are locked only if the FOR UPDATE OF clause refers to a column in that table.

contd.

# SELECT… FOR UPDATE

- **For example: Following query locks the staff_master table but not the department_master table.**

```
CURSOR C1 is SELECT staff_code, job, dname from emp,
dept WHERE emp.deptno=dept.deptno FOR UPDATE OF sal;
```

  - Using primary key simulates the WHERE CURRENT OF clause but does not create any locks.

# SELECT… FOR UPDATE - Examples

- **To promote professors who earn more than 20000**

```
DECLARE
CURSOR c_staff  is SELECT staff_code, staff_master.design_code
 FROM staff_master,designation_master
 WHERE design_name = 'Professor' and staff_sal > 20000
and staff_master.design_code =designation_master.design_code
FOR UPDATE OF design_code NOWAIT;
d_code  designation_master.design_code%type;
BEGIN
   SELECT design_code into d_code FROM designation_master
        WHERE design_name='Director';
      FOR v_rec in c_staff
      LOOP
          UPDATE staff_master SET design_code = d_code
           WHERE current of c_staff;
      END LOOP;
END;
```

# Parameterized Cursor

- You must use the OPEN statement to pass parameters to a cursor.
  - Unless you want to accept default values, each "formal parameter" in the Cursor declaration must have a corresponding "actual parameter" in the OPEN statement.
  - The scope of parameters is local to the cursor.
  - Syntax:

```
OPEN Cursor-name(param1, param2…..)
```

# Parameterized Cursor - Examples

- Parameters are passed to a parametric cursor using the syntax OPEN (param1, param2 …)as shown in the following example:

```
OPEN C_Select_staff( 800,5000);
        Query → SELECT  * from staff_master
      WHERE staff_sal BETWEEN 800 AND 5000;
```

# Usage

- Like a Cursor, a Cursor Variable points to the current row in the result set of a multi-row query.
  - A Cursor is static whereas a Cursor Variable is dynamic because it is not tied to a specific query.
  - You can open a Cursor Variable for any type-compatible query.
    - This offers more flexibility
  - You can assign new values to a Cursor Variable and pass it as a parameter to subprograms, including those in database.
    - This offers an easy way to centralize data retrieval.

# Usage

- Cursor variables are available to every PL/SQL client.
  - You can declare a cursor variable in a PL/SQL host environment, and then pass it as a bind variable to PL/SQL.
  - Oracle Forms and Oracle Reports, which have a PL/SQL engine, can use cursor variables entirely on the client side.

# Cursors and Cursor Variables - Comparison

- To access the processing information stored in an unnamed work area, you can use:
  - an Explicit Cursor, which names the work area or
  - a Cursor Variable, which points to the work area
- However, Cursors and Cursor Variables are not interoperable.
  - a Cursor always refers to the “same query work area”.
  - a Cursor Variable can refer to “different work areas”.

# Cursor Variables - Example

- Defining REF CURSOR types:
  - Syntax:

```
TYPE ref_type_name IS REF CURSOR RETURN return_type;
    DECLARE
        TYPE DeptCurTyp IS REF CURSOR RETURN
department_master%ROWTYPE;
```

  - where:
- ref_type_name is a type specifier used in subsequent declarations of cursor variables
- Return_type must represent a record or a row in a database table.
- REF CURSOR types are strong (restrictive), or weak (non-restrictive)

# Cursor Variables - Example

```
DECLARE
    TYPE staffCurTyp IS REF CURSOR
    RETURN staff_master%ROWTYPE;  -- Strong types


    TYPE GenericCurTyp IS REF CURSOR;  -- Weak types
```

# Cursor Variables - Example

- Declaring Cursor Variables:
  - Example 1:

```
DECLARE
TYPE DeptCurTyp IS REF CURSOR RETURN
department_master%ROWTYPE;
dept_cv                    DeptCurTyp;  -- Declare cursor variable
```

    - You cannot declare cursor variables in a package.
    -
  - Example 2:

```
TYPE TmpCurTyp IS REF CURSOR RETURN staff_master%ROWTYPE;
tmp_cv TmpCurTyp;  -- Declare cursor variable
```

# Cursor Variables - Example

```
DECLARE
      TYPE staffcurtyp is REF CURSOR RETURN
       staff_master%rowtype;
      staff_cv    staffcurtyp;  -- declare cursor variable
      staff_cur     staff_master%rowtype;
BEGIN
       open staff_cv for  select * from staff_master;
 LOOP
          EXIT WHEN staff_cv%notfound;
          FETCH staff_cv into staff_cur;
          INSERT into temp_table  VALUES (staff_cv.staff_code,
                staff_cv.staff_name,staff_cv.staff_sal);
        END LOOP;
        CLOSE staff_cv;
    END;
```

# Summary

- In this lesson, you have learnt:
  - Cursor is a “handle” or “name” for a private SQL area.
    - Implicit cursors are declared for queries that return only one row.
    - Explicit cursors are declared for queries that return more than one row.
  - Like a Cursor, a Cursor Variable points to the current row in the result set of a multi-row query.
    - However, Cursors and Cursor Variables are not interoperable.

# Review Question

- Question 1: A “Cursor” is static whereas a “Cursor Variable” is dynamic because it is not tied to a specific query.
  - True / False
- Question 2: %COUNT returns number of rows fetched from the cursor area by using FETCH command.
  - True / False

# Review Question

- Question 3: Implicit SQL cursor is opened or closed by the program.
  - True / False
- Question 4: A ____ specifies a Return Type.
- Question 5: PL/SQL provides a shortcut via a ____ Loop, which implicitly handles the cursor processing.