Database Programming

4.Introduction to Cursors

Sagara Samarawickrama

CSD 4203 – 2024W

4.Cursors

Cursors are memory areas where the Oracle platform executes SQL statements. In database programming, cursors are internal data structures that allow processing of SQL query results. For example, you use a cursor to operate on all the rows of the STUDENT table for those students taking a particular course (having associated entries in the ENROLLMENT table)

Types of Cursors

For the Oracle platform to process an SQL statement, it needs to create an area of memory known as the context area; this will contain the information necessary to process the statement. A cursor is a handle, or pointer, to the context area. Through the cursor, a PL/SQL program can control the context area and what happens to it as the statement is processed

4.Cursors

Two important features about the cursor are as follows:

1. Cursors allow you to fetch and process rows returned by a SELECT statement, one row at a time.

2. A cursor is named so that it can be referenced.

There are two types of cursors:

1. An **implicit cursor** is automatically declared by Oracle every time an SQL statement is executed. The user will not be aware of this happening and will not be able to control or process the information in an implicit cursor.

2. An **explicit cursor** is defined by the program for any query that returns more than one row of data. That means the programmer has declared the cursor within the PL/SQL code block. This declaration allows the application to sequentially process each row of data as it is returned by the cursor.

4.Cursors

Making Use of an Implicit Cursor

To better understand the capabilities of an explicit cursor, you first need to understand the process followed for an implicit cursor.

Process of an Implicit Cursor

- Any given PL/SQL block issues an implicit cursor whenever an SQL statement is executed, as long as an explicit cursor does not exist for that SQL statement.
- A cursor is automatically associated with every DML (Data Manipulation Language) statement (UPDATE, DELETE, INSERT).
- All UPDATE and DELETE statements have cursors that identify the set of rows that will be affected by the operation.
- An INSERT statement needs a place to receive the data that is to be inserted in the database; the implicit cursor fulfills this need.
- The most recently opened cursor is called the “SQL” cursor.

4.Cursors

Making Use of an Implicit Cursor

The implicit cursor is used to process INSERT, UPDATE, DELETE, and SELECT INTO statements. During the processing of an implicit cursor, the Oracle platform automatically performs the OPEN, FETCH, and CLOSE operations

An implicit cursor can tell you how many rows were affected by an update.

Cursors have attributes such as ROWCOUNT. SQL%ROWCOUNT, for example, returns the numbers of rows updated

```
SET SERVEROUTPUT ON
BEGIN
  UPDATE student
     SET first_name = 'B'
   WHERE first_name LIKE 'B%';
  DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
END;
```

4.Cursors

Consider the following example of an implicit cursor:

```
SET SERVEROUTPUT ON;
DECLARE
  v_first_name VARCHAR2(35);
  v_last_name VARCHAR2(35);
BEGIN
  SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM student
   WHERE student_id = 123;
  DBMS_OUTPUT.PUT_LINE ('Student name: '||
    v_first_name||' '||v_last_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE
    ('There is no student with student ID 123');
END;
```

Explain in the class

4.Cursors

Making Use of an explicit Cursor

The only means of generating an explicit cursor is for the cursor to be named in the declaration section of the PL/SQL block.

The advantages of declaring an explicit cursor over using an indirect implicit cursor are that the explicit cursor gives more programmatic control to the programmer. Implicit cursors are less efficient than explicit cursors, which makes it harder to trap data errors

The process of working with an explicit cursor consists of the following steps:

1. Declaring the cursor. This initializes the cursor into memory.
2. Opening the cursor. The previously declared cursor can now be opened; memory is allocated.

4.Cursors

3. Fetching the cursor. The previously declared and opened cursor can now retrieve data; this is the process of fetching the cursor.

4. Closing the cursor. The previously declared, opened, and fetched cursor must now be closed to release memory allocation

Declaring a Cursor

Declaring a cursor defines the name of the cursor and associates it with a SELECT statement. The first step is to declare the cursor with the following syntax:

CURSOR c_cursor_name IS select statement

Example(in the next slide) is a PL/SQL fragment that demonstrates the first step of declaring a cursor. A cursor named C_MyCursor is declared as a SELECT statement of all the rows in the zipcode table that have the item state equal to “NY.”

4.Cursors

```
DECLARE
  CURSOR C_MyCursor IS
    SELECT *
      FROM zipcode
     WHERE state = 'NY';
…
   -<code would continue here with opening, fetching,
    and closing of the cursor>
```

Record Types

A record is a composite data structure, which means that it is composed of one or more elements. Records are very much like a row of a database table, but each element of the record does not stand on its own. PL/SQL supports three kinds of records: (1) table based,(2) cursor based, and (3) programmer defined.

A table-based record is one whose structure is drawn from the list of columns in the table. A cursor-based record is one whose structure matches the elements of a predefined cursor. To create a table-based or cursor-based record, use the %ROWTYPE attribute.

4.Cursors

```
SET SERVEROUTPUT ON
DECLARE
  vr_student student%ROWTYPE;
BEGIN
  SELECT *
    INTO vr_student
    FROM student
   WHERE student_id = 156;
  DBMS_OUTPUT.PUT_LINE (vr_student.first_name||' '
    ||vr_student.last_name||' has an ID of 156');
EXCEPTION
  WHEN no_data_found
    THEN
        RAISE_APPLICATION_ERROR(-2001,'The Student '||
         'is not in the database');
END;
```

Making Use of Record Types

Next is a example of a record type in an anonymous PL/SQL block.

4.Cursors

```
SET SERVEROUTPUT ON;
DECLARE
 vr_zip ZIPCODE%ROWTYPE;
BEGIN
  SELECT *
    INTO vr_zip
    FROM zipcode
   WHERE rownum < 2;
  DBMS_OUTPUT.PUT_LINE('City: '||vr_zip.city);
  DBMS_OUTPUT.PUT_LINE('State: '||vr_zip.state);
  DBMS_OUTPUT.PUT_LINE('Zip: '||vr_zip.zip);
END;
```

Cursor Loop

To process a cursor, you will have to loop through it. Next we explain the details of each step of the loop by going through a code example

4.Cursors

To process a cursor, you will have to loop through it. Next we explain the details of each step of the loop by going through a code example. The following example shows the declaration section of a PL/SQL block that defines a cursor named c_student

```
DECLARE
  CURSOR c_student is
    SELECT first_name||'  '||Last_name name
      FROM student;
  vr_student c_student%ROWTYPE;
```

Opening a Cursor

The next step in controlling an explicit cursor is to open it. When the open cursor statement is processed, four actions will take place automatically:

1. The variables (including bind variables) in the WHERE clause are examined.

2. Based on the values of the variables, the active set is determined and the PL/SQL engine executes the query for that cursor. Variables are examined at cursor open time only.

4.Cursors

Fetching Rows in a Cursor

After the cursor has been declared and opened, you can retrieve data from the cursor. The process of getting the data from the cursor is referred to as fetching the cursor. There are two methods of fetching a cursor, which use the following commands:

1.FETCH cursor_name INTO PL/SQL variables;

2.FETCH cursor_name INTO PL/SQL record;

When the cursor is fetched, the following occurs:

1. The fetch command is used to retrieve one row at a time from the active set. This is generally done inside a loop. The values of each row in the active set can then be stored into the corresponding variables or PL/SQL record one at a time, performing operations on each one successively.

4.Cursors

2. After each FETCH command, the active set pointer is moved forward to the next row. Thus, each fetch will return successive rows of the active set, until the entire set is returned. The last FETCH command will not assign values to the output variables;thus they will still contain their prior values.

Closing a Cursor

Once all of the rows in the cursor have been processed (retrieved), the cursor should be closed. This tells the PL/SQL engine that the program is finished with the cursor, and the resources associated with it can be freed. The syntax for closing the cursor is

```
CLOSE cursor_name;
```

Next, study the example given in the next slide

4.Cursors

```
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_student_name IS
    SELECT first_name, last_name
      FROM student
     WHERE rownum <= 5;
  vr_student_name c_student_name%ROWTYPE;
BEGIN
  OPEN c_student_name;
  LOOP
    FETCH c_student_name INTO vr_student_name;
    EXIT WHEN c_student_name%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Student name: '||
      vr_student_name.first_name
      ||'  '||vr_student_name.last_name);
  END LOOP;
  CLOSE c_student_name;
END;
```

What is the output of the above cursor ?

4.Cursors

Consider the same example with single modification. Notice that the DBMS_OUTPUT.PUT_LINE statement has been moved outside the loop.

```
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_student_name IS
    SELECT first_name, last_name
      FROM student
     WHERE rownum <= 5;
  vr_student_name c_student_name%ROWTYPE;
BEGIN
  OPEN c_student_name;
  LOOP
    FETCH c_student_name INTO vr_student_name;
    EXIT WHEN c_student_name%NOTFOUND;
  END LOOP;
  CLOSE c_student_name;
  DBMS_OUTPUT.PUT_LINE('Student name: '||
    vr_student_name.first_name||' '
    ||vr_student_name.last_name);
END;
```

4.Cursors

Putting It All Together

Here is an example of the complete cycle of declaring, opening, fetching, and closing a cursor, including use of cursor attributes.

```
1> DECLARE
2>   v_sid    student.student_id%TYPE;
3>   CURSOR c_student IS
4>     SELECT student_id
5>       FROM student
6>      WHERE student_id < 110;
7> BEGIN
8>   OPEN c_student;
9>   LOOP
10>    FETCH c_student INTO v_sid;
11>    EXIT WHEN c_student%NOTFOUND;
12>      DBMS_OUTPUT.PUT_LINE('STUDENT ID : '||v_sid);
13>  END LOOP;
14>  CLOSE c_student;
15> EXCEPTION
16>  WHEN OTHERS
17>  THEN
18>    IF c_student%ISOPEN
19>    THEN
20>      CLOSE c_student;
21>    END IF;
22> END;
```

4.Cursors

Cursor FOR Loops

An alternative method of handling cursors is called the cursor FOR LOOP because of the simplified syntax that is used. When using the cursor FOR LOOP, the process of opening, fetching, and closing is handled implicitly. This makes the blocks much simpler to code and easier to maintain.

The cursor FOR LOOP specifies a sequence of statements to be repeated once for each row returned by the cursor. You can use a cursor FOR LOOP when you need to fetch and process each and every record from a cursor until you want to stop processing and exit the loop.

To make use of this column, you need to create a new table called table_log with the following script:

```
create table table_log
       (description VARCHAR2(250));
```

4.Cursors

Then run this script:

```
DECLARE
  CURSOR c_student IS
    SELECT student_id, last_name, first_name
      FROM student
     WHERE student_id < 110;
BEGIN
  FOR r_student IN c_student
  LOOP
    INSERT INTO table_log
      VALUES(r_student.last_name);
  END LOOP;
END;
SELECT * from table_log;
```

Cursors can be nested inside each other. Although this may sound complex, it is really just a loop inside a loop, much like nested loops, which we studied earlier.

Example of nested loop will be available in the practical session