Lab 7. Cursors

Databases II

Cursors are memory areas where Oracle executes SQL statements. In database programming cursors are internal data structures that allow processing of SQL query results.

7.1. Cursor Manipulation

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. Cursors have two important features:

- Cursors allow you to fetch and process rows returned by a SELECT statement one row at a time.
- A cursor is named so that it can be referenced.

TYPES OF CURSORS

There are two types of cursors:

- Oracle automatically declares an *implicit* cursor every time a SQL statement is executed. The user is unaware of this and cannot control or process the information in an implicit cursor.
- The program defines an *explicit* cursor for any query that returns more than one row of data. This means that 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 the cursor returns it.

IMPLICIT CURSOR

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

- Any given PL/SQL block issues an implicit cursor whenever a 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) 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 into the data- base; the implicit cursor fulfills this need.
- The most recently opened cursor is called the SQL cursor.

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

Example:

```
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;
```

EXPLICIT CURSOR

The only means of generating an explicit cursor is to name the cursor in the DECLARE section of the PL/SQL block.

The advantage of declaring an explicit cursor over an indirect implicit cursor is that the explicit cursor gives the programmer more programmatic control. Also, implicit cursors are less efficient than explicit cursors, so it is 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 declared cursor is opened, and memory is allotted.
3. *Fetching* the cursor. The declared and opened cursor can now retrieve data.
4. *Closing* the cursor. The declared, opened, and fetched cursor must be closed to release the memory allocation.

DECLARING A CURSOR

Declaring a cursor defines the cursor's name and associates it with a SELECT statement. You declare a cursor using the following syntax:

CURSOR c_cursor_name IS select statement

Example:

```
DECLARE
 CURSOR c_MyCursor IS
   SELECT *
     FROM zipcode
    WHERE state = 'NY';
```

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: table-based, cursor-based, and 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:

```
record_name table_name or cursor_name%ROWTYPE
```

Example:

```
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;
```

The variable vr_student is a record type of the existing database table student. In other words, it has the same components as a row in the student table. A cursor-based record is much the same, except that it is drawn from the select list of an explicitly declared cursor. When referencing elements of the record, you use the same syntax that you use with tables:

```
record_name.item_name
```

To define a variable that is based on a cursor record, first you must declare the cursor. In the following lab, you will start by declaring a cursor and then open the cursor, fetch from the cursor, and close the cursor.

A table-based record is drawn from a particular table structure. Consider the following code fragment:

Example:

```
DECLARE
  vr_zip ZIPCODE%ROWTYPE;
  vr_instructor INSTRUCTOR%ROWTYPE;
```

OPENING A CURSOR

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

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

c. The PL/SQL engine identifies the active set of data—the rows from all the involved tables that meet the WHERE clause criteria.
d. The active set pointer is set to the first row.

The syntax for opening a cursor is

```
OPEN cursor_name;
```

FETCHING ROWS IN A CURSOR

After the cursor has been declared and opened, you can retrieve data from the cursor. The process of getting data from the cursor is called fetching the cursor. There are two ways to fetch a cursor:

```
FETCH cursor_name INTO PL/SQL variables;
```
or
```
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 in the correspon- ding variables or PL/SQL record one at a time, performing operations on each one successively.
2. After each FETCH, the active set pointer is moved forward to the next row. Thus, each FETCH returns successive rows of the active set, until the entire set is returned. The last FETCH does not assign values to the output variables; they still contain their prior values.

CLOSING A CURSOR

As soon as all 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;
```

Example:

```
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;
```

CURSORS ATTRIBUTES

CURSOR ATTRIBUTE	SYNTAX	DESCRIPTION
%NOTFOUND	*cursor_name*%NOTFOUND	A Boolean attribute that returns TRUE if the previous FETCH did not return a row and FALSE if it did.
%FOUND	*cursor_name*%FOUND	A Boolean attribute that returns TRUE if the previous FETCH returned a row and FALSE if it did not.
%ROWCOUNT	*cursor_name*%ROWCOUNT	The number of records fetched from a cursor at that point in time.
%ISOPEN	*cursor_name*%ISOPEN	A Boolean attribute that returns TRUE if the cursor is open and FALSE if it is not.

Example:

```
DECLARE
  v_sid      student.student_id%TYPE;
  CURSOR c_student IS
    SELECT student_id
     FROM student
    WHERE student_id < 110;

 BEGIN
 OPEN c_student;
 LOOP
   FETCH c_student INTO v_sid;
   EXIT WHEN c_student%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('STUDENT ID : '||v_sid);
 END LOOP;
 CLOSE c_student;
 EXCEPTION
   WHEN OTHERS THEN
     IF c_student%ISOPEN THEN
        CLOSE c_student;
     END IF;
 END;
```

ASSORTED TIPS ON CURSORS

- **Cursor SELECT LIST** - Match the SELECT list with PL/SQL variables or PL/SQL record components. The number of variables must be equal to the number of columns or expressions in the SELECT list. The number of components in a record must match the columns or expressions in the SELECT list.
- **Cursor Scope** - The scope of a cursor declared in the main block (or an enclosing block) extends to the subblocks.
- **Expressions in a Cursor SELECT List** - PL/SQL variables, expressions, and even functions can be included in the cursor SELECT list.
- **Column Aliases in Cursors** - An alias is an alternative name you provide to a column or expression in the SELECT list. In an explicit cursor column, aliases are required for calculated columns when
  - You FETCH into a record declared with a %ROWTYPE declaration against that cursor.
  - You want to reference the calculated column in the program.

7.2. Using Cursor FOR Loops and Nested Cursors

There is an alternative way to handle cursors. It is called the cursor FOR loop because of the simplified syntax that is used. With a cursor FOR loop, the process of opening, fetching, and closing is handled implicitly. This makes the blocks much easier to code and maintain.

The cursor FOR loop specifies a sequence of statements to be repeated once for each row returned by the cursor. Use the cursor FOR loop if you need to FETCH and PROCESS every record from a cursor until you want to stop processing and exit the loop.

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

Example:

```
CREATE TABLE table_log
   (description VARCHAR2(250));
```

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;
```

Process Nested Cursors

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 were covered in previous chapters. If you have one parent cursor and two child cursors, each time the parent cursor makes a single loop, it loops through each child cursor once and then begins a second round. The following two examples show a nested cursor with a single child cursor.

Example:

```
SET SERVEROUTPUT ON
DECLARE
  v_zip zipcode.zip%TYPE;
  v_student_flag CHAR;
  CURSOR c_zip IS
    SELECT zip, city, state
     FROM zipcode
    WHERE state = 'CT';
  CURSOR c_student IS
    SELECT first_name, last_name
     FROM student
    WHERE zip = v_zip;
  BEGIN
    FOR r_zip IN c_zip LOOP
      v_student_flag := 'N';
      v_zip := r_zip.zip;
      DBMS_OUTPUT.PUT_LINE(CHR(10));
      DBMS_OUTPUT.PUT_LINE('Students living in '|| r_zip.city);
      FOR r_student in c_student LOOP
        DBMS_OUTPUT.PUT_LINE(r_student.first_name||' '||r_student.last_name);
        v_student_flag := 'Y';
      END LOOP;
      IF v_student_flag = 'N' THEN
        DBMS_OUTPUT.PUT_LINE('No Students for this zipcode');
      END IF;
    END LOOP;
  END;
```

7.3. Using Parameters with Cursors and Complex Nested Cursors

CURSORS WITH PARAMETERS

A cursor can be declared with parameters. This enables a cursor to generate a specific result set that is narrow but also reusable. A cursor of all the data from the zipcode table may be very useful, but it would be more useful for certain data processing if it held information for only one state. At this point, you know how to create such a cursor. But wouldn't it be more useful if you could create a cursor that could accept a parameter of a state and then run through only the city and zip for that state?

Example:

```
CURSOR c_zip (p_state IN zipcode.state%TYPE) IS
  SELECT zip, city, state
    FROM zipcode
   WHERE state = p_state;
```

Here are the main points to keep in mind for parameters in cursors:

- Cursor parameters make the cursor more reusable.
- Cursor parameters can be assigned default values.
- The scope of the cursor parameters is local to the cursor.
- The mode of the parameters can only be IN. When a cursor has been declared as taking a parameter, it must be called with a value for that parameter. The c_zip cursor declared in the preceding example is called as follows: OPEN c_zip (parameter_value)

The same cursor could be opened with a CURSOR FOR loop as follows:

```
  FOR r_zip IN c_zip('NY') LOOP  ...
```

Nesting cursors allows for looping through data at various stages. For example, one cursor could loop through zip codes. When it hits a zip code, a second, nested cursor would loop through students who live in that zip code. Working through a specific example will help you understand this in more detail.

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR c_student IS
   SELECT first_name, last_name, student_id
     FROM student
   WHERE last_name LIKE 'J%';
  CURSOR c_course (i_student_id IN student.student_id%TYPE) IS
   SELECT c.description, s.section_id sec_id
     FROM course c, section s, enrollment e
```

```
    WHERE e.student_id = i_student_id
      AND c.course_no = s.course_no
      AND s.section_id = e.section_id;
   CURSOR c_grade(i_section_id IN section.section_id%TYPE, i_student_id IN
                 student.student_id%TYPE) IS
     SELECT gt.description grd_desc, TO_CHAR (AVG(g.numeric_grade), '999.99') num_grd
       FROM enrollment e, grade g, grade_type gt
      WHERE e.section_id = i_section_id
        AND e.student_id = g.student_id
        AND e.student_id = i_student_id
        AND e.section_id = g.section_id
        AND g.grade_type_code = gt.grade_type_code
      GROUP BY gt.description ;
  BEGIN
    FOR r_student IN c_student LOOP
      DBMS_OUTPUT.PUT_LINE(CHR(10));
      DBMS_OUTPUT.PUT_LINE(r_student.first_name|| '  '||r_student.last_name);
      FOR r_course IN c_course(r_student.student_id) LOOP
        DBMS_OUTPUT.PUT_LINE ('Grades for course :'|| r_course.description);
          FOR r_grade IN c_grade(r_course.sec_id, r_student.student_id) LOOP
            DBMS_OUTPUT.PUT_LINE(r_grade.num_grd||'  '||r_grade.grd_desc);
          END LOOP;
      END LOOP;
    END LOOP;
  END;
```

7.4 FOR UPDATE and WHERE CURRENT Cursors

The cursor FOR UPDATE clause is used only with a cursor when you want to update tables in the database. Generally, when you execute a SELECT statement, you are not locking any rows. The purpose of using the FOR UPDATE clause is to lock the rows of the tables you want to update so that another user cannot perform an update until you perform your update and release the lock. The next COMMIT or ROLLBACK statement releases the lock. The FOR UPDATE clause changes the manner in which the cursor operates in only a few respects. When you open a cursor, all rows that meet the restriction criteria are identified as part of the active set. Using the FOR UPDATE clause locks these rows that have been identified in the active set. If the FOR UPDATE clause is used, rows may not be fetched from the cursor until a COMMIT has been issued. It is important to think about where to place the COMMIT.

The syntax is simply to add FOR UPDATE to the end of the cursor definition. If several items are being selected, but you want to lock only one of them, end the cursor definition with the following syntax:

```
   FOR UPDATE OF <item_name>
```

Example:

```
   DECLARE
     CURSOR c_grade(
        i_student_id IN enrollment.student_id%TYPE,
        i_section_id IN enrollment.section_id%TYPE)
      IS
        SELECT final_grade
          FROM enrollment
           WHERE student_id = i_student_id AND section_id = i_section_id FOR UPDATE;
     CURSOR c_enrollment IS
        SELECT e.student_id, e.section_id
         FROM enrollment e, section s
        WHERE s.course_no = 135 AND e.section_id = s.section_id;
     BEGIN
       FOR r_enroll IN c_enrollment LOOP
```

```
        FOR r_grade IN c_grade(r_enroll.student_id, r_enroll.section_id) LOOP
          UPDATE enrollment
            SET final_grade  = 90
           WHERE student_id = r_enroll.student_id AND section_id = r_enroll.section_id;
        END LOOP;
      END LOOP;
    END;
```

Exercices:

1. Write a nested cursor in which the parent cursor gathers information about each section of a course. The child cursor counts the enrollment. The only output is one line for each course, with the course name, section number, and total enrollment.
2. Write an anonymous PL/SQL block that finds all the courses that have at least one section that is at its maximum enrollment. If no courses meet that criterion, pick two courses and create that situation for each.
   a) For each of those courses, add another section. The instructor for the new section should be taken from the existing records in the instructor table. Use the instructor who is signed up to teach the fewest courses. Handle the fact that, during the execution of your program, the instructor teaching the most courses may change.
   b) Use any exception-handling techniques you think are useful to capture error conditions.
3. Construct 3 cursors. The first one, cursor c_student takes no parameters and is a collection of students with a last name beginning with J. The second one c_course takes in the parameter of student_ID to generate a list of courses that student is taking. The third one, c_grade takes in two parameters, section_id and student_id. In this way it can generate an average of the different grade types (quizzes, homework, final, etc.) for that student for that course. Display the student name for the first coursor. The second cursor takes the parameter of student_id from the first cursor. Only the description of the course is displayed. The third cursor takes in the parameter of section_id from the second cursor and student_id from the first cursor. The grades are then displayed.