Using Explicit Cursors

Objectives

After completing this lesson, you should be able to do the following:

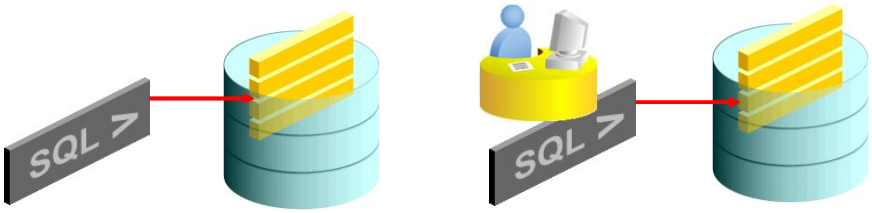
- Distinguish between implicit and explicit cursors
- Discuss the reasons for using explicit cursors
- Declare and control explicit cursors
- Use simple loops and cursor `FOR` loops to fetch data
- Declare and use cursors with parameters
- Lock rows with the `FOR UPDATE` clause
- Reference the current row with the `WHERE CURRENT OF` clause

You have learned about implicit cursors that are automatically created by PL/SQL when you execute a SQL `SELECT` or DML statement. In this lesson, you learn about explicit cursors. You learn to differentiate between implicit and explicit cursors. You also learn to declare and control simple cursors, as well as cursors with parameters.

Cursors

Every SQL statement that is executed by the Oracle Server has an associated individual cursor:

- Implicit cursors: Declared and managed by PL/SQL for all DML and PL/SQL `SELECT` statements
- Explicit cursors: Declared and managed by the programmer



Implicit cursor Explicit cursor

The Oracle Server uses work areas (called *private SQL areas*) to execute SQL statements and to store processing information. You can use explicit cursors to name a private SQL area and to access its stored information.

Cursor Type	Description
Implicit	Implicit cursors are declared by PL/SQL implicitly for all DML and PL/SQL `SELECT` statements.
Explicit	For queries that return multiple rows, explicit cursors are declared and managed by the programmer, and manipulated through specific statements in the block's executable actions.

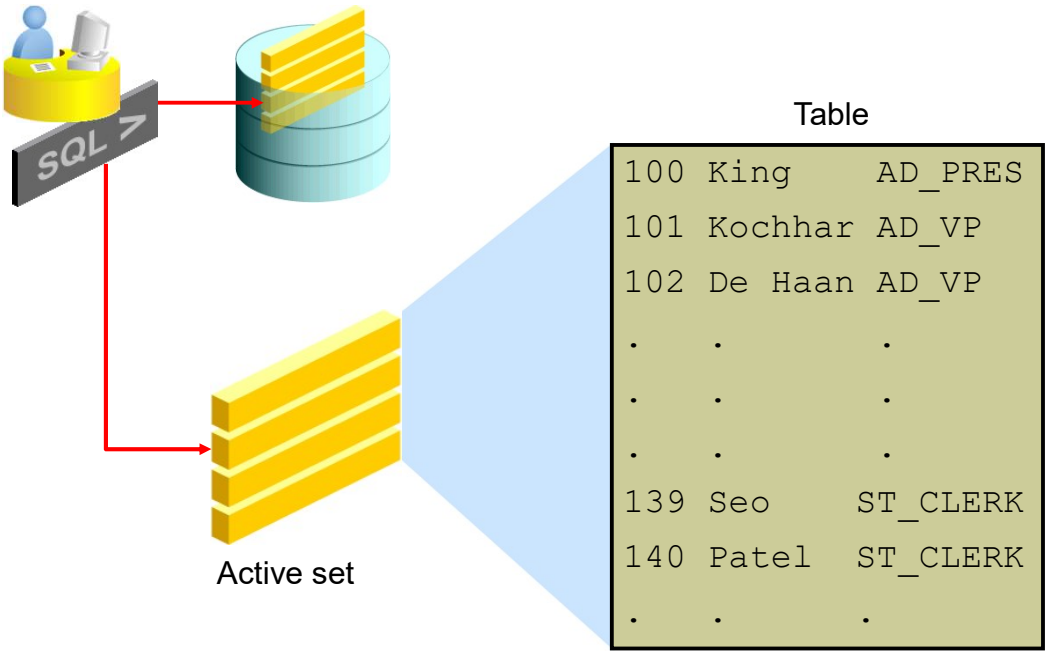
The Oracle Server implicitly opens a cursor to process each SQL statement that is not associated with an explicitly declared cursor. Using PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor.

Explicit Cursor Operations

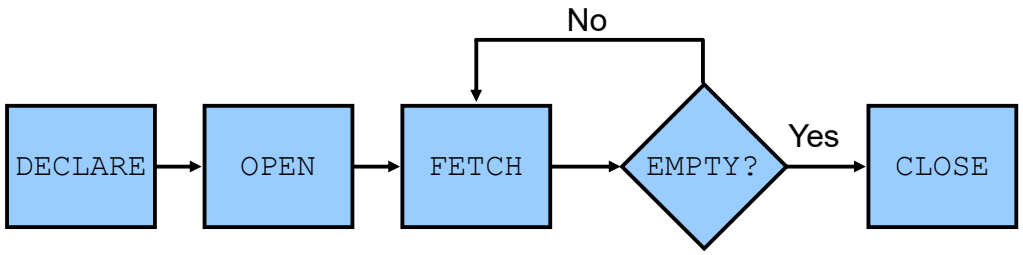
- You declare explicit cursors in PL/SQL when you have a `SELECT` statement that returns multiple rows. You can process each row returned by the `SELECT` statement.
- The set of rows returned by a multiple-row query is called the *active set*. Its size is the number of rows that meet your search criteria.

- Explicit cursor functions:
  - Can perform row-by-row processing beyond the first row returned by a query
  - Keep track of the row that is currently being processed
  - Enable the programmer to manually control explicit cursors in the PL/SQL block

Explicit Cursor Operations



Controlling Explicit Cursors



- Create a named SQL area.
- Identify the active set.
- Load the current row into variables.
- Test for existing rows.
- Return to `FETCH` if rows are found.
- Release the active set.

Now that you have a conceptual understanding of cursors, review the steps to use them.

1. In the declarative section of a PL/SQL block, declare the cursor by naming it and defining the structure of the query to be associated with it.
2. Open the cursor.
   The `OPEN` statement executes the query, binds any variables that are referenced and positions the cursor at the first row. Rows identified by the query are called the *active set* and are now available for fetching.
3. Fetch data from the cursor.
   In the flow diagram shown in the slide, after each fetch, you test the cursor for any existing row. If there are no more rows to process, you must close the cursor.
   The `FETCH` statement retrieves the current row and advances the cursor to the next row until there are no more rows or a specified condition is met.
4. Close the cursor.
   The `CLOSE` statement releases the active set of rows. It is now possible to reopen the cursor to establish a fresh active set.

Declaring the Cursor

Syntax:

```
CURSOR cursor_name IS
     select_statement;
```

Examples:

```
DECLARE
  CURSOR c_emp_cursor IS
  SELECT employee_id, last_name FROM employees
  WHERE department_id =30;
```

```
DECLARE
  v_locid NUMBER:= 1700;
  CURSOR c_dept_cursor IS
  SELECT * FROM departments
  WHERE location_id = v_locid;
...
```

ORACLE

The syntax to declare a cursor is shown in the slide. In the syntax:

cursor_name	Is a PL/SQL identifier
select_statement	Is a `SELECT` statement without an `INTO` clause

The active set of a cursor is determined by the `SELECT` statement in the cursor declaration. It is mandatory to have an `INTO` clause for a `SELECT` statement in PL/SQL. However, note that the `SELECT` statement in the cursor declaration cannot have an `INTO` clause. That is because you are only defining a cursor in the declarative section and not retrieving any rows into the cursor.

Note

- Do not include the `INTO` clause in the cursor declaration because it appears later in the `FETCH` statement.
- If you want the rows to be processed in a specific sequence, use the `ORDER BY` clause in the query.
- The cursor can be any valid `SELECT` statement, including joins, subqueries, and so on.

The `c_emp_cursor` cursor is declared to retrieve the `employee_id` and `last_name` columns for those employees working in the department with `department_id` 30.

The `c_dept_cursor` cursor is declared to retrieve all the details for the department with the `location_id` 1700. Note that a variable is used while declaring the cursor. These variables are considered bind variables, which must be visible when you are declaring the cursor. These variables are examined only once at the time the cursor opens. You have learned that explicit cursors are used when you have to retrieve and operate on multiple rows in PL/SQL. However, this example shows that you can use the explicit cursor even if your `SELECT` statement returns only one row.

Opening the Cursor

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
...
BEGIN
  OPEN c_emp_cursor;
```


The OPEN statement executes the query associated with the cursor, identifies the active set, and positions the cursor pointer at the first row. The OPEN statement is included in the executable section of the PL/SQL block.

OPEN is an executable statement that performs the following operations:

1. Dynamically allocates memory for a context area
2. Parses the SELECT statement
3. Binds the input variables (sets the values for the input variables by obtaining their memory addresses)
4. Identifies the active set (the set of rows that satisfy the search criteria). Rows in the active set are not retrieved into variables when the OPEN statement is executed. Rather, the FETCH statement retrieves the rows from the cursor to the variables.
5. Positions the pointer to the first row in the active set

Note: If a query returns no rows when the cursor is opened, PL/SQL does not raise an exception.

Fetching Data from the Cursor

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  FETCH c_emp_cursor INTO v_empno, v_lname;
  DBMS_OUTPUT.PUT_LINE( v_empno ||'  '||v_lname);
END;
/
```

```
anonymous block completed
114  Raphaely
```

ORACLE

The FETCH statement retrieves the rows from the cursor one at a time. After each fetch, the cursor advances to the next row in the active set. You can use the %NOTFOUND attribute to determine whether the entire active set has been retrieved.

Consider the example shown in the slide. Two variables, v_empno and v_lname, are declared to hold the fetched values from the cursor. Examine the FETCH statement.

You have successfully fetched the values from the cursor to the variables. However, there are six employees in department 30, but only one row was fetched. To fetch all rows, you must use loops. In the next slide, you see how a loop is used to fetch all the rows.

The FETCH statement performs the following operations:

1. Reads the data for the current row into the output PL/SQL variables
2. Advances the pointer to the next row in the active set

You can include the same number of variables in the `INTO` clause of the `FETCH` statement as there are columns in the `SELECT` statement; be sure that the data types are compatible. Match each variable to correspond to the columns positionally. Alternatively, you can also define a record for the cursor and reference the record in the `FETCH INTO` clause. Finally, test to see whether the cursor contains rows. If a fetch acquires no values, there are no rows left to process in the active set and no error is recorded.

Fetching Data from the Cursor

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_empno, v_lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||'  '||v_lname);
  END LOOP;
END;
/
```


Observe that a simple `LOOP` is used to fetch all the rows. Also, the cursor attribute `%NOTFOUND` is used to test for the exit condition. The output of the PL/SQL block is:

```
anonymous block completed
114  Raphaely
115  Khoo
116  Baida
117  Tobias
118  Himuro
119  Colmenares
```

Closing the Cursor

```
...
  LOOP
    FETCH c_emp_cursor INTO empno, lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP;
 CLOSE c_emp_cursor;
END;
/
```


The `CLOSE` statement disables the cursor, releases the context area, and "undefines" the active set. Close the cursor after completing the processing of the `FETCH` statement. You can reopen the cursor if required. A cursor can be reopened only if it is closed. If you attempt to fetch data from a cursor after it is closed, an `INVALID_CURSOR` exception is raised.

Note: Although it is possible to terminate the PL/SQL block without closing cursors, you should make it a habit to close any cursor that you declare explicitly to free resources. There is a maximum limit on the number of open cursors per session, which is determined by the `OPEN_CURSORS` parameter in the database parameter file. (`OPEN_CURSORS` = `50` by default.)

Cursors and Records

Process the rows of the active set by fetching values into a PL/SQL record.

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE  department_id =30;
   v_emp_record   c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                        ||' '||v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END;
```

ORACLE

You have already seen that you can define records that have the structure of columns in a table. You can also define a record based on the selected list of columns in an explicit cursor. This is convenient for processing the rows of the active set, because you can simply fetch into the record. Therefore, the values of the rows are loaded directly into the corresponding fields of the record.

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

Cursor FOR Loops

Syntax:

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- The cursor FOR loop is a shortcut to process explicit cursors.
- Implicit open, fetch, exit, and close occur.
- The record is implicitly declared.

You learned to fetch data from cursors by using simple loops. You now learn to use a cursor FOR loop, which processes rows in an explicit cursor. It is a shortcut because the cursor is opened, a row is fetched once for each iteration in the loop, the loop exits when the last row is processed, and the cursor is closed automatically. The loop itself is terminated automatically at the end of the iteration where the last row is fetched.

In the syntax:

record_name	Is the name of the implicitly declared record
cursor_name	Is a PL/SQL identifier for the previously declared cursor

Guidelines

- Do not declare the record that controls the loop; it is declared implicitly.
- Test the cursor attributes during the loop if required.
- Supply the parameters for a cursor, if required, in parentheses following the cursor name in the FOR statement.

Cursor FOR Loops

```
DECLARE
  CURSOR c_emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE department_id =30;
BEGIN
   FOR emp_record IN c_emp_cursor
    LOOP
     DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
     ||' ' ||emp_record.last_name);
    END LOOP;
END;
/
```

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```


The example that was used to demonstrate the usage of a simple loop to fetch data from cursors is rewritten to use the cursor `FOR` loop.

`emp_record` is the record that is implicitly declared. You can access the fetched data with this implicit record (as shown in the slide). Observe that no variables are declared to hold the fetched data using the `INTO` clause. The code does not have the `OPEN` and `CLOSE` statements to open and close the cursor, respectively.

Explicit Cursor Attributes

Use explicit cursor attributes to obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the number of rows that has been fetched

As with implicit cursors, there are four attributes for obtaining the status information of a cursor. When appended to the cursor variable name, these attributes return useful information about the execution of a cursor manipulation statement.

Note: You cannot reference cursor attributes directly in a SQL statement.

%ISOPEN Attribute

- You can fetch rows only when the cursor is open.
- Use the %ISOPEN cursor attribute before performing a fetch to test whether the cursor is open.

Example:

```
IF NOT c_emp_cursor%ISOPEN THEN
   OPEN c_emp_cursor;
END IF;
LOOP
  FETCH c_emp_cursor...
```


- You can fetch rows only when the cursor is open. Use the %ISOPEN cursor attribute to determine whether the cursor is open.
- Fetch rows in a loop. Use cursor attributes to determine when to exit the loop.
- Use the %ROWCOUNT cursor attribute to do the following:
  - Process an exact number of rows.
  - Fetch the rows in a loop and determine when to exit the loop.

Note: %ISOPEN returns the status of the cursor: TRUE if open and FALSE if not.

%ROWCOUNT and %NOTFOUND: Example

```
DECLARE
  CURSOR c_emp_cursor IS SELECT employee_id,
    last_name FROM employees;
  v_emp_record   c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
   FETCH c_emp_cursor INTO v_emp_record;
   EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR
                    c_emp_cursor%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
               ||' '||v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END ; /
```

```
anonymous block completed
174 Abel
166 Ande
130 Atkinson
105 Austin
204 Baer
116 Baida
167 Banda
172 Bates
192 Bell
151 Bernstein
```


The example in the slide retrieves the first 10 employees one by one. This example shows how the %ROWCOUNT and %NOTFOUND attributes can be used for exit conditions in a loop.

Cursor FOR Loops Using Subqueries

There is no need to declare the cursor.

```
BEGIN
  FOR emp_record IN (SELECT employee_id, last_name
   FROM employees WHERE department_id =30)
  LOOP
   DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
   ||' '||emp_record.last_name);
  END LOOP;
END;
/
```

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

ORACLE

Note that there is no declarative section in this PL/SQL block. The difference between the cursor FOR loops using subqueries and the cursor FOR loop lies in the cursor declaration. If you are writing cursor FOR loops using subqueries, you need not declare the cursor in the declarative section. You have to provide the SELECT statement that determines the active set in the loop itself.

The example that was used to illustrate a cursor FOR loop is rewritten to illustrate a cursor FOR loop using subqueries.

Note: You cannot reference explicit cursor attributes if you use a subquery in a cursor FOR loop because you cannot give the cursor an explicit name.

Cursors with Parameters

Syntax:

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

- Pass parameter values to a cursor when the cursor is opened and the query is executed.
- Open an explicit cursor several times with a different active set each time.

```
OPEN  cursor_name(parameter_value,.....) ;
```


You can pass parameters to a cursor. This means that you can open and close an explicit cursor several times in a block, returning a different active set on each occasion. For each execution, the previous cursor is closed and reopened with a new set of parameters.

Each formal parameter in the cursor declaration must have a corresponding actual parameter in the `OPEN` statement. Parameter data types are the same as those for scalar variables, but you do not give them sizes. The parameter names are for reference in the query expression of the cursor.

In the syntax:

cursor_name	Is a PL/SQL identifier for the declared cursor
parameter_name	Is the name of a parameter
datatype	Is the scalar data type of the parameter
select_statement	Is a `SELECT` statement without the `INTO` clause

The parameter notation does not offer greater functionality; it simply allows you to specify input values easily and clearly. This is particularly useful when the same cursor is referenced repeatedly.

Cursors with Parameters

```
DECLARE
  CURSOR   c_emp_cursor (deptno NUMBER) IS
   SELECT  employee_id, last_name
   FROM    employees
   WHERE   department_id = deptno;
   ...
BEGIN
  OPEN c_emp_cursor (10);
  ...
  CLOSE c_emp_cursor;
  OPEN c_emp_cursor (20);
  ...
```

```
anonymous block completed
200 Whalen
201 Hartstein
202 Fay
```


Parameter data types are the same as those for scalar variables, but you do not give them sizes. The parameter names are for reference in the cursor's query. In the following example, a cursor is declared and is defined with one parameter:

```
DECLARE
   CURSOR c_emp_cursor(deptno NUMBER) IS SELECT ...
```

The following statements open the cursor and return different active sets:

```
OPEN c_emp_cursor(10);
OPEN c_emp_cursor(20);
```

You can pass parameters to the cursor that is used in a cursor `FOR` loop:

```
DECLARE
CURSOR c_emp_cursor(p_deptno NUMBER, p_job VARCHAR2)IS
   SELECT ...
BEGIN
   FOR emp_record IN c_emp_cursor(10, 'Sales') LOOP ...
```

FOR UPDATE Clause

Syntax:

```
SELECT ...
FROM          ...
FOR UPDATE [OF column_reference][NOWAIT | WAIT n];
```

- Use explicit locking to deny access to other sessions for the duration of a transaction.
- Lock the rows *before* the update or delete.

If there are multiple sessions for a single database, there is the possibility that the rows of a particular table were updated after you opened your cursor. You see the updated data only when you reopen the cursor. Therefore, it is better to have locks on the rows before you update or delete rows. You can lock the rows with the `FOR UPDATE` clause in the cursor query.

In the syntax:

column_reference	Is a column in the table against which the query is performed (A list of columns may also be used.)
`NOWAIT`	Returns an Oracle Server error if the rows are locked by another session

The `FOR UPDATE` clause is the last clause in a `SELECT` statement, even after `ORDER BY` (if it exists). When you want to query multiple tables, you can use the `FOR UPDATE` clause to confine row locking to particular tables. `FOR UPDATE OF col_name(s)` locks rows only in tables that contain `col_name(s)`.

The SELECT ... FOR UPDATE statement identifies the rows that are to be updated or deleted, and then locks each row in the result set. This is useful when you want to base an update on the existing values in a row. In that case, you must make sure that the row is not changed by another session before the update.

The optional NOWAIT keyword tells the Oracle Server not to wait if the requested rows have been locked by another user. Control is immediately returned to your program so that it can do other work before trying again to acquire the lock. If you omit the NOWAIT keyword, the Oracle Server waits until the rows are available.

Example:

```
DECLARE
  CURSOR c_emp_cursor IS
  SELECT employee_id, last_name, FROM employees
  WHERE department_id = 80 FOR UPDATE OF salary NOWAIT;
  ...
```

If the Oracle Server cannot acquire the locks on the rows it needs in a SELECT FOR UPDATE operation, it waits indefinitely. Use NOWAIT to handle such situations. If the rows are locked by another session and you have specified NOWAIT, opening the cursor results in an error. You can try to open the cursor later. You can use WAIT instead of NOWAIT, specify the number of seconds to wait, and then determine whether the rows are unlocked. If the rows are still locked after *n* seconds, an error is returned.

It is not mandatory for the FOR UPDATE OF clause to refer to a column, but specify which tables to lock if the statement is a join of multiple tables.

WHERE CURRENT OF Clause

Syntax:

```
WHERE CURRENT OF cursor ;
```

- Use cursors to update or delete the current row.
- Include the FOR UPDATE clause in the cursor query to first lock the rows.
- Use the WHERE CURRENT OF clause to reference the current row from an explicit cursor.

```
UPDATE employees
   SET    salary = ...
   WHERE CURRENT OF c_emp_cursor;
```


In the syntax:

cursor Is the name of a declared cursor (The cursor must have been declared with the FOR UPDATE clause.)

WHERE CURRENT OF Clause

- The WHERE CURRENT OF clause is used in conjunction with the FOR UPDATE clause to refer to the current row in an explicit cursor.
- The WHERE CURRENT OF clause is used in the UPDATE or DELETE statement, whereas the FOR UPDATE clause is specified in the cursor declaration.
- You can use the combination for updating and deleting the current row from the corresponding database table.
- This enables you to apply updates and deletes to the row currently being addressed, without the need to explicitly reference the row ID.
- You must include the FOR UPDATE clause in the cursor query so that the rows are locked on OPEN.

Quiz

Explicit cursor functions enable the programmer to manually control explicit cursors in the PL/SQL block.

a. True
b. False

Answer: a

Summary

In this lesson, you should have learned to:

- Distinguish cursor types:
  - Implicit cursors are used for all DML statements and single-row queries.
  - Explicit cursors are used for queries of zero, one, or more rows.
- Create and handle explicit cursors
- Use simple loops and cursor `FOR` loops to handle multiple rows in the cursors
- Evaluate cursor status by using cursor attributes
- Use the `FOR UPDATE` and `WHERE CURRENT OF` clauses to update or delete the current fetched row

The Oracle Server uses work areas to execute SQL statements and store processing information. You can use a PL/SQL construct called a *cursor* to name a work area and access its stored information. There are two kinds of cursors: implicit and explicit. PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row. For queries that return multiple rows, you must explicitly declare a cursor to process the rows individually.

Every explicit cursor and cursor variable has four attributes: `%FOUND`, `%ISOPEN`, `%NOTFOUND`, and `%ROWCOUNT`. When appended to the cursor variable name, these attributes return useful information about the execution of a SQL statement. You can use cursor attributes in procedural statements but not in SQL statements.

Use simple loops or cursor `FOR` loops to operate on the multiple rows fetched by the cursor. If you are using simple loops, you have to open, fetch, and close the cursor; however, cursor `FOR` loops do this implicitly. If you are updating or deleting rows, lock the rows by using a `FOR UPDATE` clause. This ensures that the data you are using is not updated by another session after you open the cursor. Use a `WHERE CURRENT OF` clause in conjunction with the `FOR UPDATE` clause to reference the current row fetched by the cursor.