Using Explicit Cursor

Cursors

Every SQL statement 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

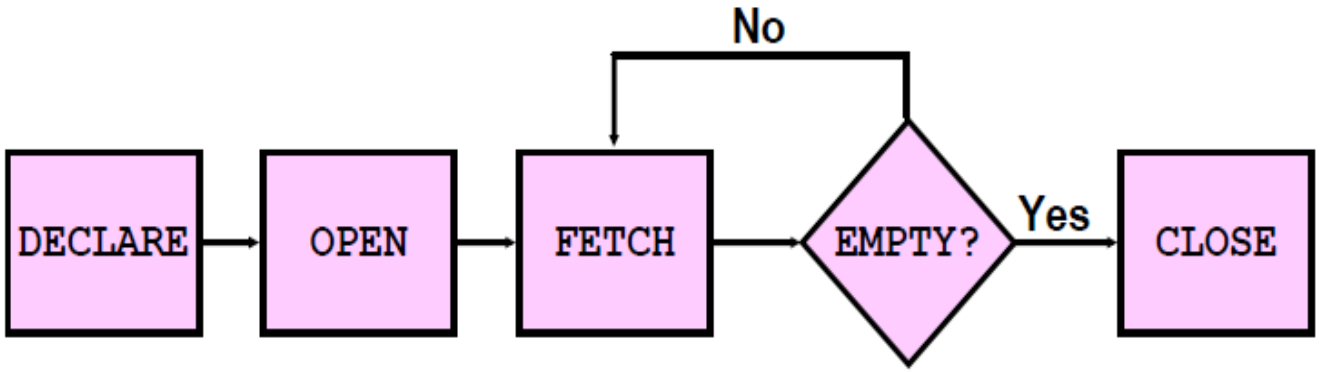
Explicit Cursor Operations

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.

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

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.

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 total number of rows returned so far

Hints when declaring a cursor

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

```
DECLARE
  CURSOR c_emp_dept20 is
  SELECT employee_id, first_name FROM employees
  where department_id=30
  Order by first_name;
```

Hints when Opening a 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. You can find out the number of rows returned with an explicit cursor by using the <cursor_name>%ROWCOUNT attribute.

Hints when fetching data from a cursor

Fetching Data from the Cursor

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.

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

Hints when closing the Cursor

Closing the Cursor

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 has been closed, then an `INVALID_CURSOR` exception will be 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 up 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.)

```
DECLARE
  CURSOR c_emp_dept20 is
  SELECT employee_id, first_name FROM employees
  where department_id=30;

  v_empno employees.employee_id%type;
  v_first_name employees.first_name%type;

BEGIN
OPEN c_emp_dept20;

  loop
      fetch c_emp_dept20 into v_empno, v_first_name;

      exit when c_emp_dept20%notfound;

      dbms_output.put_line(v_empno||' '||v_first_name);
  end loop;

  close c_emp_dept20;

END;
```

Declaring the cursor

Opening the Cursor

fetching Data from cursor

exit the loop using cursor attributes

close the cursor

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.

```
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;
/
```

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);
  ...
```

Parameter data types are the same as those for scalar variables, but you do not give them sizes.

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 ...
```

Using FOR UPDATE in Cursor

FOR UPDATE Clause

always last Statement in select

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.

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 lock the rows first.
- 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;
```

Thank You