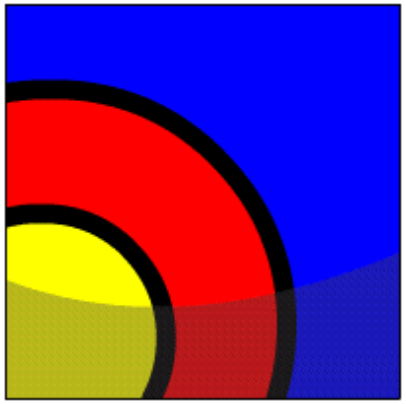
Introduction to Explicit Cursors

What Will I Learn?

In this lesson, you will learn to:

- Distinguish between an implicit and an explicit cursor
- Describe why and when to use an explicit cursor in PL/SQL code
- List two or more guidelines for declaring and controlling explicit cursors
- Create PL/SQL code that successfully opens a cursor and fetches a piece of data into a variable
- Use a simple loop to fetch multiple rows from a cursor
- Create PL/SQL code that successfully closes a cursor after fetching data into a variable



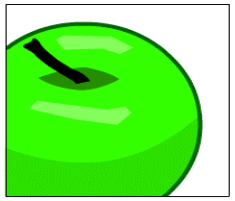
Why Learn It?

You have learned that an SQL SELECT statement in a PL/SQL block is successful only if it returns exactly one row.

What if you need to write a SELECT statement that returns more than one row? For example, you need to produce a report of all employees?

To return more than one row, you must declare and use an explicit cursor.

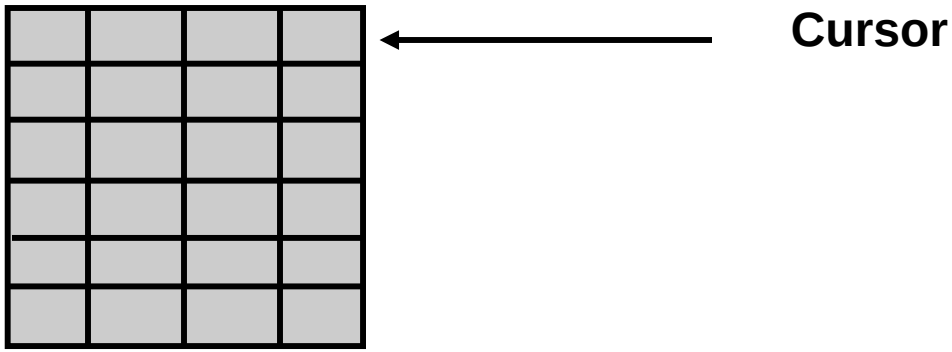
Tell Me/Show Me



Context Areas and Cursors

The Oracle server allocates a private memory area called a context area to store the data processed by an SQL statement.

Every context area (and therefore every SQL statement) has a cursor associated with it. You can think of a cursor either as a label for the context area, or as a pointer to the context area. In fact, a cursor is both of these items.



Tell Me/Show Me

Implicit and Explicit Cursors

There are two types of cursors:

- Implicit cursors: Defined automatically by Oracle for all SQL DML statements (`INSERT`, `UPDATE`, `DELETE`, and `MERGE`), and for `SELECT` statements that return only one row.

- Explicit cursors: Declared by the programmer for queries that return more than one row. You can use explicit cursors to name a context area and access its stored data.

Tell Me/Show Me

Limitations of Implicit Cursors

There is more than one row in the EMPLOYEES table:

```
DECLARE
  v_salary employees.salary%TYPE;
BEGIN
  SELECT salary INTO v_salary
    FROM employees;
  DBMS_OUTPUT.PUT_LINE(' Salary is : '||v_salary);
END;
```

```
ORA-01422: exact fetch returns more than requested number of rows
```

Tell Me/Show Me

Explicit Cursors

With an explicit cursor, you can retrieve multiple rows from a database table, have a pointer to each row that is retrieved, and work on the rows one at a time.

The following are some reasons to use an explicit cursor:

- It is the only way in PL/SQL to retrieve more than one row from a table.
- Each row is fetched by a separate program statement, giving the programmer more control over the processing of the rows.

Tell Me/Show Me

Example of an Explicit Cursor

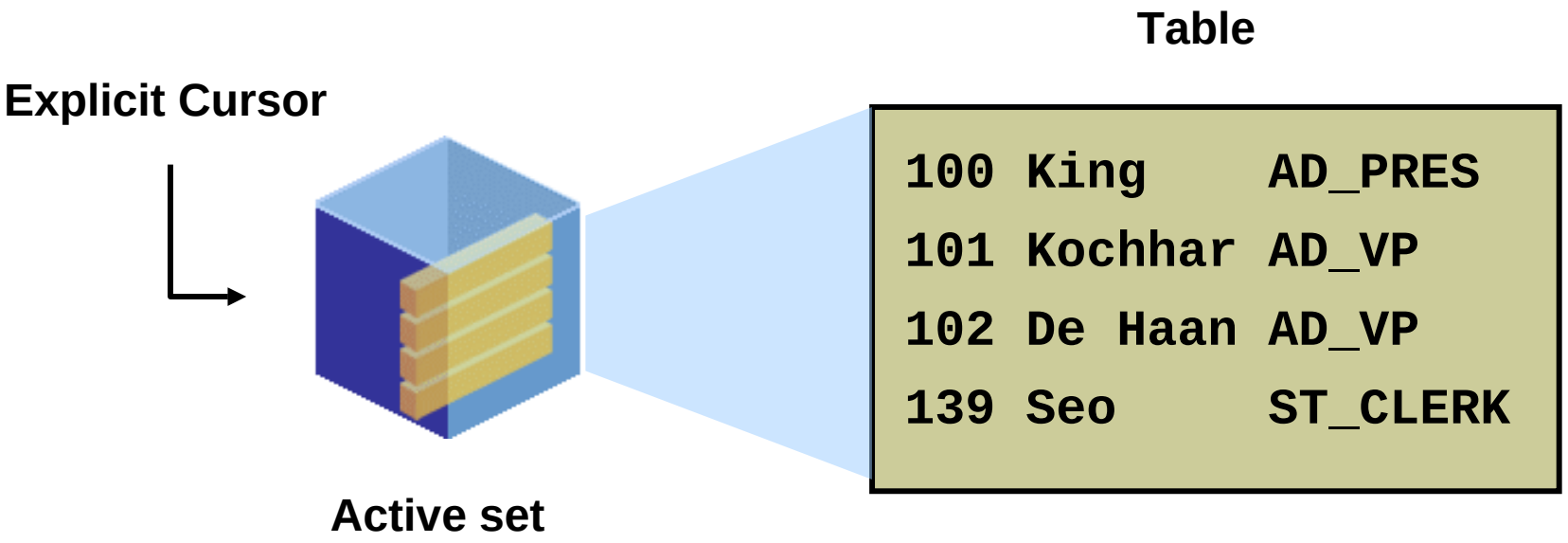
The following example uses an explicit cursor to obtain the country name and national holiday for countries in Asia.

```
DECLARE
  CURSOR wf_holiday_cursor IS
  SELECT country_name, national_holiday_date
  FROM wf_countries where region_id IN(30,34,35);
  v_country_name  wf_countries.country_name%TYPE;
  v_holiday       wf_countries.national_holiday_date%TYPE;
BEGIN
  OPEN wf_holiday_cursor;
  LOOP
    FETCH wf_holiday_cursor INTO v_country_name, v_holiday;
    EXIT WHEN wf_holiday_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_country_name||' '||v_holiday);
  END LOOP;
  CLOSE wf_holiday_cursor;
END;
```

Tell Me/Show Me

Explicit Cursor Operations

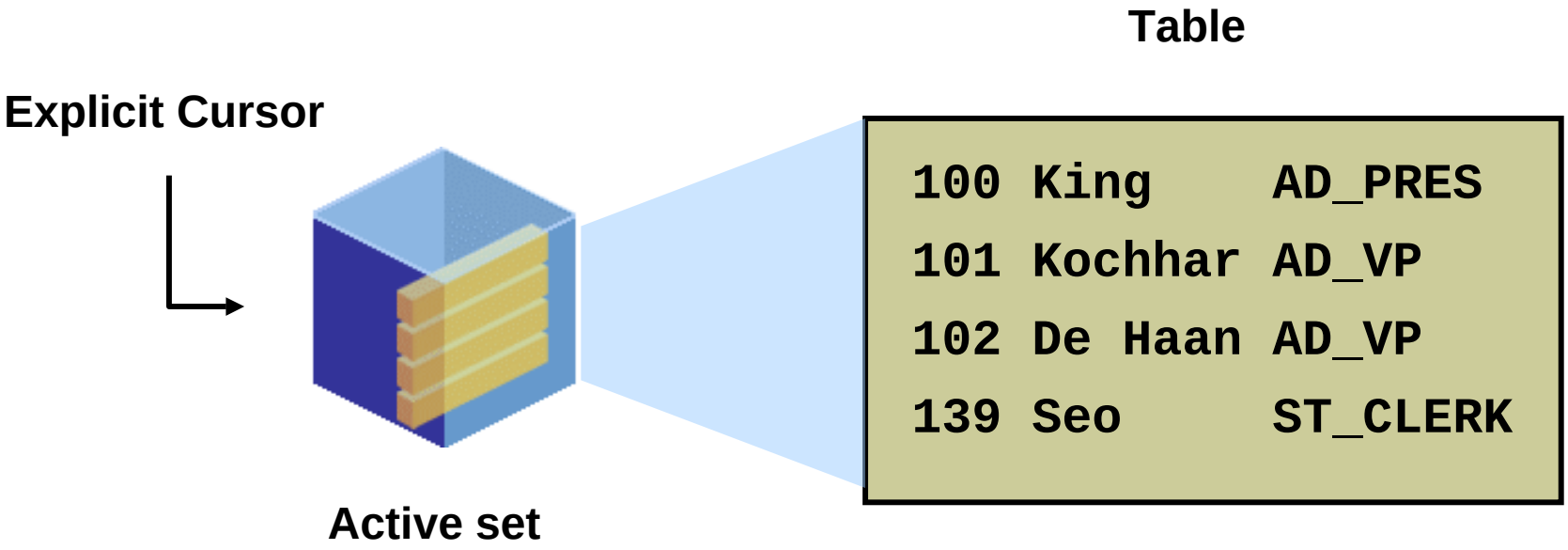
The set of rows returned by a multiple-row query is called the **active set**, and is stored in the context area. Its size is the number of rows that meet your search criteria.



Tell Me/Show Me

Explicit Cursor Operations

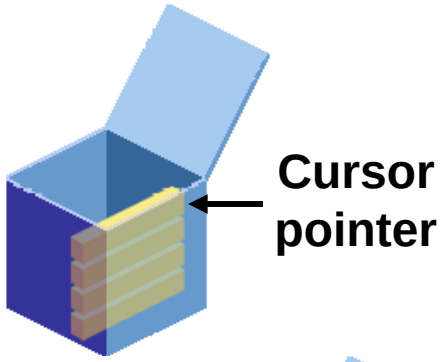
Think of the context area (named by the cursor) as a box, and the active set as the contents of the box. To get at the data, you must OPEN the box and FETCH each row from the box one at a time. When finished, you must CLOSE the box.



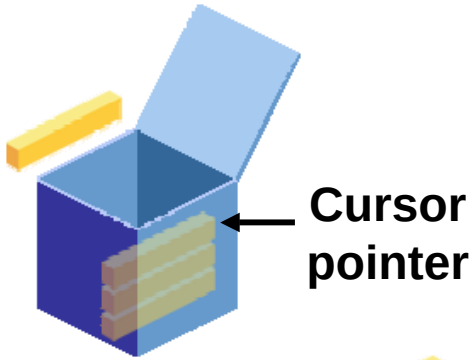
Tell Me/Show Me

Controlling Explicit Cursors

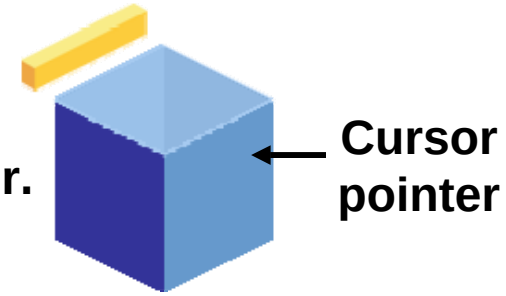
1 Open the cursor.



2 Fetch each row, one at a time.

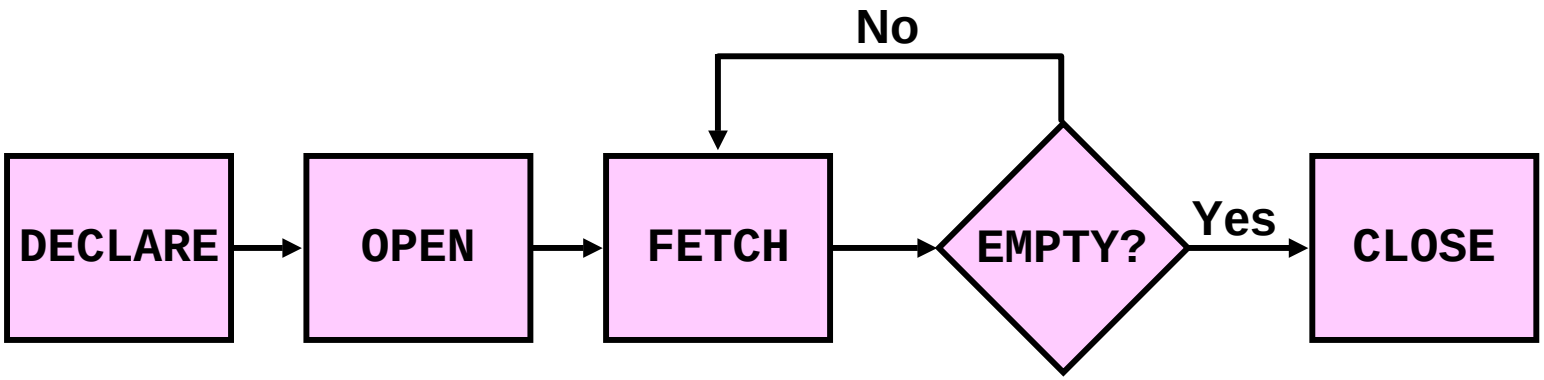


3 Close the cursor.



Tell Me/Show Me

Declaring and Controlling Explicit Cursors



- Name an active set.
- Fill the active set with data.
- Retrieve the current row into variables.
- Test for existing rows.
- Return to FETCH if rows are found.
- Release the active set.

Tell Me/Show Me

Declaring the Cursor

The active set of a cursor is determined by the SELECT statement in the cursor declaration.

Syntax:

```
CURSOR cursor_name IS
     select_statement;
```

In the syntax:

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

Tell Me/Show Me

Declaring the Cursor: Example 1

The `emp_cursor` cursor is declared to retrieve the `employee_id` and `last_name` columns of the employees working in the department with a `department_id` of 30.

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

Tell Me/Show Me

Declaring the Cursor: Example 3

A SELECT statement in a cursor declaration can include joins, group functions, and subqueries. This example retrieves each department that has at least two employees, giving the department name and number of employees.

```
DECLARE
  CURSOR dept_emp_cursor IS
    SELECT department_name, COUNT(*) AS how_many
      FROM departments d, employees e
         WHERE d.department_id = e.department_id
      GROUP BY d.department_name
      HAVING COUNT(*) > 1;
...
```

Tell Me/Show Me

Guidelines for Declaring the 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, then use the `ORDER BY` clause in the query.
- The cursor can be any valid `SELECT` statement, including joins, subqueries, and so on.
- If a cursor declaration references any PL/SQL variables, these variables must be declared before declaring the cursor.

Tell Me/Show Me

Opening the Cursor

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

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

Tell Me/Show Me

Opening the Cursor (continued)

- The `OPEN` statement performs the following operations:

1. Allocates memory for a context area (creates the box)
2. Executes the `SELECT` statement in the cursor declaration, returning the results into the active set (fills the box with data)
3. Positions the pointer to the first row in the active set

Tell Me/Show Me

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. Two variables, v_empno and v_lname, are declared to hold the fetched values from the cursor.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
      WHERE department_id =10;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO v_empno, v_lname;
  DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  ...
END;
```

```
200 Whalen

Statement processed.
```

Tell Me/Show Me

Fetching Data from the Cursor

You have successfully fetched the values from the cursor into the variables. However, there are six employees in department 30. Only one row has been fetched. To fetch all the rows, you have to make use of loops.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
       WHERE  department_id =50;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
   LOOP
    FETCH emp_cursor INTO v_empno, v_lname;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP; …
END;
```

```
124 Mourgos
141 Rajs
142 Davies
143 Matos
144 Vargas

Statement processed.
```

Tell Me/Show Me

Guidelines for Fetching Data From the Cursor

- Include the same number of variables in the `INTO` clause of the `FETCH` statement as columns in the `SELECT` statement, and be sure that the data types are compatible.
- Match each variable to correspond to the columns positionally.
- Test to see whether the cursor contains rows. If a fetch acquires no values, then there are no rows left to process in the active set and no error is recorded. The last row is re-processed.
- You can use the `%NOTFOUND` cursor attribute to test for the exit condition.

Tell Me/Show Me

Fetching Data From the Cursor

What is wrong with this example?

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

Tell Me/Show Me

Fetching Data From the Cursor (continued)

There is only one employee in department 10. What happens when this example is executed?

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
      WHERE  department_id =10;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO v_empno, v_lname;
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP; …
END;
```

Tell Me/Show Me

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 later if required.

Think of CLOSE as closing and emptying the box, so you can no longer FETCH its contents.

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

Tell Me/Show Me

Guidelines for Closing the Cursor

- 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 is raised.

- If you later reopen the cursor, the associated SELECT statement is re-executed to re-populate the context area with the most recent data from the database.

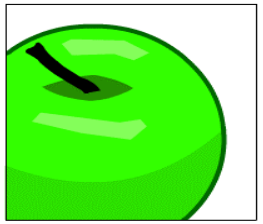
Tell Me/Show Me

Putting It All Together

The following example declares and processes a cursor to obtain the country name and national holiday for countries in Asia.

```
DECLARE
  CURSOR wf_holiday_cursor IS
    SELECT country_name, national_holiday_date
      FROM wf_countries where region_id IN(30,34,35);
  v_country_name wf_countries.country_name%TYPE;
  v_holiday      wf_countries.national_holiday_date%TYPE;
BEGIN
  OPEN wf_holiday_cursor;
  LOOP
    FETCH wf_holiday_cursor INTO v_country_name, v_holiday;
    EXIT WHEN wf_holiday_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_country_name||' '||v_holiday);
  END LOOP;
  CLOSE wf_holiday_cursor;
END;
```

Tell Me/Show Me



Terminology

Key terms used in this lesson include:

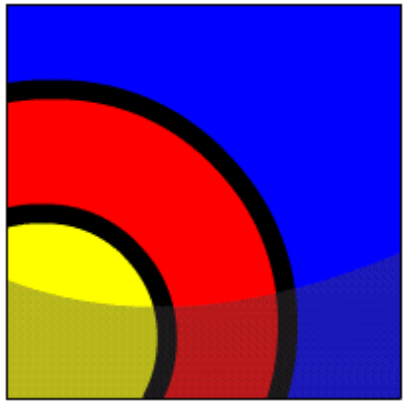
- Context area
- Cursor
- Implicit cursor
- Explicit cursor
- Active set
- FETCH
- OPEN
- CLOSE

Using Explicit Cursor Attributes

What Will I Learn?

In this lesson, you will learn to:

- Define a record structure using the %ROWTYPE attribute
- Create PL/SQL code to process the rows of an active set using record types in cursors
- Retrieve information about the state of an explicit cursor using cursor attributes



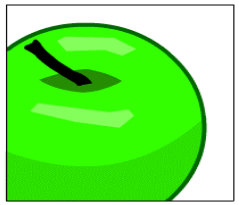
Why Learn It?

One of the reasons to use explicit cursors is that they give you greater programmatic control when handling your data. This lesson discusses techniques for using explicit cursors more effectively.

- Cursor records enable you to declare a single variable for all the selected columns in a cursor.

- Cursor attributes enable you to retrieve information about the state of your explicit cursor.

Tell Me/Show Me



Cursors and Records

The cursor in this example is based on a SELECT statement that retrieves only two columns of each table row:

```
DECLARE
  v_emp_id          employees.employee_id%TYPE;
  v_last_name       employees.last_name%TYPE;
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
      FROM employees
     WHERE  department_id =30;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor
    INTO v_emp_id, v_last_name;
  ...
```

What if it retrieved six columns .. or seven, or eight, or twenty?

Tell Me/Show Me

Cursors and Records

This cursor retrieves whole rows of EMPLOYEES:

```
DECLARE
  v_emp_id         employees.employee_id%TYPE;
  v_first_name     employees.first_name%TYPE;
  v_last_name      employees.last_name%TYPE;
  ...
  v_department_id employees.department_id%TYPE;
  CURSOR emp_cursor IS
    SELECT * FROM employees
      WHERE  department_id =30;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor
     INTO v_emp_id, v_first_name, v_last_name ...
          v_department_id;
  ...
```

Messy and long-winded, isn't it?

Tell Me/Show Me

Cursors and Records

Compare the following snippets of code. What differences do you see?

```
DECLARE
  v_emp_id         ...;
  v_first_name     ...;
  ...
  v_department_id ...:
  CURSOR emp_cursor IS
    SELECT * FROM employees
    WHERE  department_id =30;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor
    INTO v_emp_id, v_first_name,
    ... v_department_id;
  ...
```

```
DECLARE
  CURSOR emp_cursor IS
    SELECT * FROM employees
    WHERE  department_id =30;
  v_emp_record
    emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor
    INTO v_emp_record;
  ...
```

Tell Me/Show Me

Cursors and Records

The code on the right uses %ROWTYPE to declare a **record** structure based on the cursor. A record is a composite data type in PL/SQL.

Variables

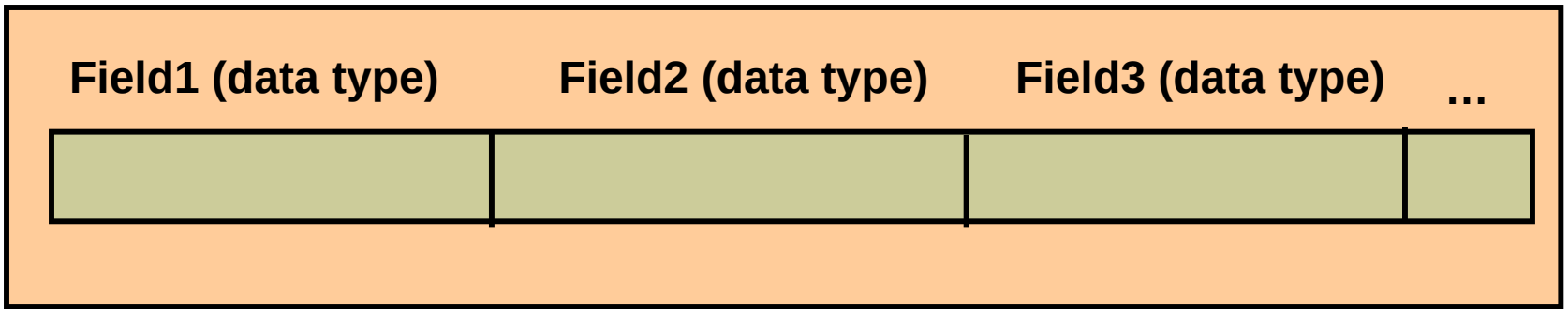
```
DECLARE
  v_emp_id          ...;
  v_first_name      ...;
  ...
  v_department_id ...:
  CURSOR emp_cursor IS
   SELECT * FROM employees
   WHERE  department_id =30;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor
    INTO v_emp_id, v_first_name,
    ... v_department_id;
  ...
```

Records

```
DECLARE
  CURSOR emp_cursor IS
   SELECT * FROM employees
   WHERE  department_id =30;
  v_emp_record
     emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor
    INTO v_emp_record;
  ...
```

Tell Me/Show Me

Structure of a PL/SQL Record:



A record is a composite data type, consisting of a number of fields each with their own name and data type.

You reference each field by dot-prefixing its field-name with the record-name.

%ROWTYPE declares a record with the same fields as the cursor on which it is based.

Tell Me/Show Me

Structure of *cursor_name*%ROWTYPE:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name, salary FROM employees
      WHERE  department_id =30;
  v_emp_record emp_cursor%ROWTYPE;
...
```

v_emp_record.employee_id	v_emp_record.last_name	v_emp_record.salary
100	King	24000

Tell Me/Show Me

Cursors and %ROWTYPE

%ROWTYPE is convenient for processing the rows of the active set because you can simply fetch into the record.

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

Tell Me/Show Me

Explicit Cursor Attributes

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

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch did not return a row
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returned a row; opposite of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the total number of rows FETCHed so far

Tell Me/Show Me

%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.

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

Example:

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

Tell Me/Show Me

%ROWCOUNT and %NOTFOUND Attributes

Usually the %ROWCOUNT and %NOTFOUND attributes are used in a loop to determine when to exit the loop.

Use the %ROWCOUNT cursor attribute for the following:

- To process an exact number of rows
- To count the number of rows fetched so far in a loop and/or determine when to exit the loop

Use the %NOTFOUND cursor attribute for the following:

- To determine whether the query found any rows matching your criteria
- To determine when to exit the loop

Tell Me/Show Me

Example of %ROWCOUNT and %NOTFOUND

This example shows how you can use %ROWCOUNT and %NOTFOUND attributes for exit conditions in a loop.

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

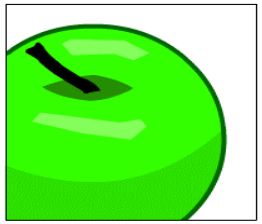
Tell Me/Show Me

Explicit Cursor Attributes in SQL Statements

You cannot use an explicit cursor attribute directly in an SQL statement. The following code returns an error:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, salary FROM employees
    ORDER BY SALARY DESC;
  v_emp_record  emp_cursor%ROWTYPE;
  v_count       NUMBER;
BEGIN
  OPEN emp_cursor;
  LOOP
   FETCH emp_cursor INTO v_emp_record;
   EXIT WHEN emp_cursor%NOTFOUND;
   INSERT INTO top_paid_emps
    (employee_id, rank, salary)
   VALUES
    (v_emp_record.employee_id, emp_cursor%ROWCOUNT,
     v_emp_record.salary);
...
```

Tell Me/Show Me



Terminology

Key terms used in this lesson include:

Record

%ROWTYPE

%ISOPEN

%ROWCOUNT

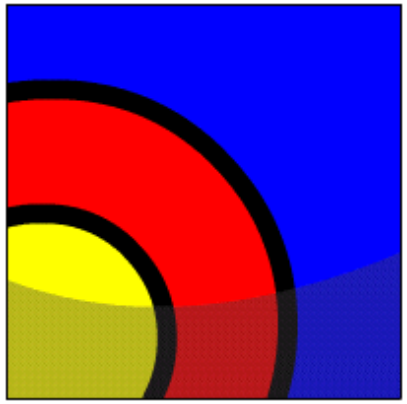
%NOTFOUND

Cursor F0R Loops

What Will I Learn?

In this lesson, you will learn to:

- List and explain the benefits of using cursor FOR loops
- Create PL/SQL code to declare a cursor and manipulate it in a FOR loop
- Create PL/SQL code containing a cursor FOR loop using a subquery



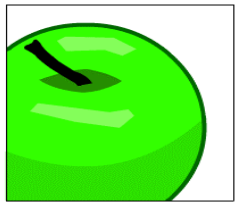
Why Learn It?

You have already learned how to declare and use a simple explicit cursor, using `DECLARE`, `OPEN`, and `FETCH` in a loop, testing for `%NOTFOUND`, and `CLOSE` statements.

Wouldn't it be easier if you could do all this with just one statement?

You can do all of this using a cursor FOR loop.

Tell Me/Show Me



Cursor FOR Loops

A cursor FOR loop 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 when the last row has been fetched.

Syntax:

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

Tell Me/Show Me

Cursor FOR Loops (continued)

In the syntax:

- *record_name* Is the name of the implicitly declared record (as *cursor_name*%ROWTYPE)
- *cursor_name* Is a PL/SQL identifier for the previously declared cursor

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

Tell Me/Show Me

Cursor FOR Loops

Note: `v_emp_record` is the record that is implicitly declared.

You can access the fetched data with this implicit record as shown in the slide. No variables are declared to hold the fetched data by using the `INTO` clause. The code does not have `OPEN` and `CLOSE` statements to open and close the cursor respectively.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =50;
BEGIN
  FOR v_emp_record IN emp_cursor
    LOOP
      DBMS_OUTPUT.PUT_LINE(v_emp_record.employee_id
                    || ' ' ||v_emp_record.last_name);
    END LOOP;
END;
```

Tell Me/Show Me

Cursor FOR Loops

Compare the cursor FOR loop code with the expanded code you learned in the previous lesson. The two forms of the code are logically identical to each other and produce exactly the same results.

```
DECLARE
  CURSOR emp_cursor IS
   SELECT employee_id, last_name
   FROM employees
   WHERE department_id =50;
BEGIN
  FOR v_emp_record IN emp_cursor
    LOOP
      DBMS_OUTPUT.PUT_LINE(…);
    END LOOP;
END;
```

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM employees
    WHERE department_id =50;
  v_emp_record  emp_cursor%ROWTYPE;
BEGIN
   OPEN emp_cursor;
   LOOP
     FETCH emp_cursor INTO
       v_emp_record;
     EXIT WHEN emp_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(…);
   END LOOP;
   CLOSE emp_cursor;
END;
```

Tell Me/Show Me

Guidelines for Cursor F0R Loops

- Do not declare the record that controls the loop because it is declared implicitly.
- The scope of the implicit record is restricted to the loop, so you cannot reference the record outside the loop.
- You can access fetched data by *record_name.column_name*.

Tell Me/Show Me

Testing Cursor Attributes

You can still test cursor attributes, such as %ROWCOUNT. This example exits from the loop after five rows have been fetched and processed. The cursor is still closed automatically.

```
DECLARE
  CURSOR emp_cursor IS
   SELECT employee_id, last_name FROM employees;

BEGIN
   FOR v_emp_record IN emp_cursor
    LOOP
     EXIT WHEN emp_cursor%ROWCOUNT > 5;
     DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                    || ' ' ||v_emp_record.last_name);
    END LOOP;
END;
```

Tell Me/Show Me

Cursor FOR Loops Using Subqueries

You can go one step further. You don't have to declare the cursor at all! Instead, you can specify the SELECT on which the cursor is based directly in the FOR loop.

The advantage of this is that all the cursor definition is contained in a single FOR … statement. This makes later changes to the code much easier and quicker.

The next slide shows an example.

Tell Me/Show Me

Cursor FOR Loops Using Subqueries: Example

```
BEGIN
  FOR v_emp_record IN (SELECT employee_id, last_name
    FROM employees WHERE department_id =50)
  LOOP
   DBMS_OUTPUT.PUT_LINE(v_emp_record.employee_id
                  ||' '||v_emp_record.last_name);
  END LOOP;
END;
```

The SELECT clause in the FOR statement is technically a subquery, so you must enclose it in parentheses.

Tell Me/Show Me

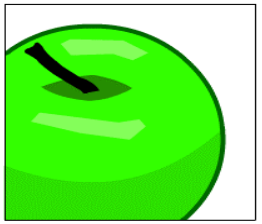
Cursor FOR Loops Using Subqueries (continued)

Again, compare these two forms of the code. They are logically identical. But which one would you rather write – especially if you hate typing!

```
BEGIN
  FOR v_dept_rec IN (SELECT *
     FROM departments)
  LOOP
    DBMS_OUTPUT.PUT_LINE(…);
  END LOOP;
END;
```

```
DECLARE
  CURSOR dept_cursor IS
    SELECT * FROM departments;
  v_dept_rec  dept_cursor%ROWTYPE;
BEGIN
   OPEN dept_cursor;
   LOOP
     FETCH dept_cursor INTO
       v_dept_rec;
     EXIT WHEN dept_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(…);
   END LOOP;
   CLOSE dept_cursor;
END;
```

Tell Me/Show Me



Terminology

Key terms used in this lesson include:

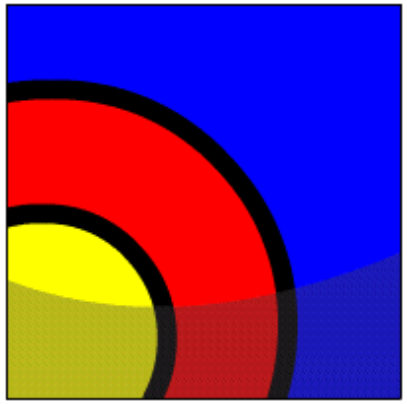
Cursor FOR loop

Cursors with Parameters

What Will I Learn?

In this lesson, you will learn to:

- List the benefits of using parameters with cursors
- Create PL/SQL code to declare and use manipulate a cursor with a parameter



Why Learn It?

Imagine a program which declares a cursor to fetch and process all the employees in a given department. The department is chosen by the user at runtime.

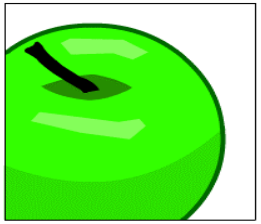
How would we declare our cursor? We could try:

```
DECLARE
  cursor country_cursor IS
    SELECT * FROM wf_countries
      WHERE region_id = ??? ;
```

Hmm… obviously not.

There are several regions. Do we need to declare several cursors, one for each region, each with a different value in the WHERE clause? No. We can declare just one cursor to handle all regions by using parameters.

Tell Me/Show Me



Cursors with Parameters

A parameter is a variable whose name is used in a cursor declaration. When the cursor is opened, the parameter value is passed to the Oracle server, which uses it to decide which rows to retrieve into the active set of the cursor.

This means that you can open and close an explicit cursor several times in a block, or in different executions of the same block, returning a different active set on each occasion.

Consider the example where you pass any region_id to a cursor and it returns the names of countries in that region. The next slide shows how.

Tell Me/Show Me

Cursors with Parameters: Example

```
DECLARE
  CURSOR c_country (p_region_id NUMBER) IS
    SELECT country_id, country_name
      FROM wf_countries
      WHERE region_id = p_region_id;
  v_country_record    c_country%ROWTYPE;
BEGIN
  OPEN c_country (5);
  LOOP
    FETCH c_country INTO v_country_record;
    EXIT WHEN c_country%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_country_record.country_id
                || ' ' || v_country_record.country_name);
  END LOOP;
  CLOSE c_country;
END;
```

Change to whichever region is required.

Tell Me/Show Me

Defining Cursors with Parameters

Each parameter named in the cursor declaration must have a corresponding value 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 used in the WHERE clause of the cursor SELECT statement.

Syntax:

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

Tell Me/Show Me

Defining Cursors with Parameters (continued)

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

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

Tell Me/Show Me

Opening Cursors with Parameters

The following is the syntax for opening a cursor with parameters:

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

Tell Me/Show Me

Cursors with Parameters

You pass parameter values to a cursor when the cursor is opened. Therefore you can open a single explicit cursor several times and fetch a different active set each time. In the following example, a cursor is opened several times.

```
DECLARE
  CURSOR c_country (p_region_id NUMBER) IS
    SELECT country_id, country_name
              FROM wf_countries
      WHERE region_id = p_region_id;
  v_country_record    c_country%ROWTYPE;
BEGIN
      OPEN c_country (5);

  ...
  CLOSE c_country;
  OPEN c_country (145);
  ...
```

Open the cursor and return different active sets.

Tell Me/Show Me

Cursor FOR Loops with a Parameter

We can use a cursor FOR loop if needed:

```
DECLARE
  CURSOR   emp_cursor (p_deptno NUMBER) IS
    SELECT  employee_id, last_name
      FROM    employees
      WHERE   department_id = p_deptno;
BEGIN
  FOR v_emp_record IN emp_cursor(10) LOOP
    ….
  END LOOP;
END;
```

Tell Me/Show Me

Cursors with Multiple Parameters

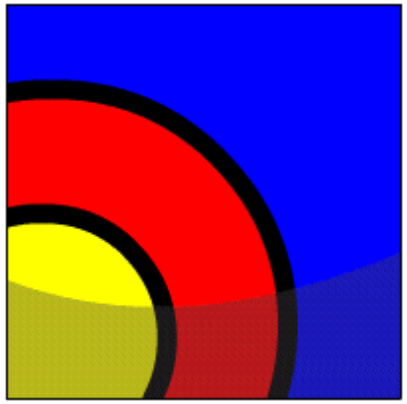
In the following example, a cursor is declared and is called with two parameters:

```
DECLARE
  CURSOR   countrycursor2 (p_region_id NUMBER,
                           p_population NUMBER) IS
    SELECT  country_id, country_name, population
      FROM    wf_countries
      WHERE   region_id = p_region_id
      OR     population > p_population;
BEGIN
  FOR v_country_record IN countrycursor2(145,10000000)
LOOP
DBMS_OUTPUT.PUT_LINE(v_country_record.country_id ||' '
                           || v_country_record.
country_name||' '|| v_country_record.population);
END LOOP;
END;
```

Summary

In this lesson, you learned to:

- List the benefits of using parameters with cursors
- Create PL/SQL code to declare and use manipulate a cursor with a parameter

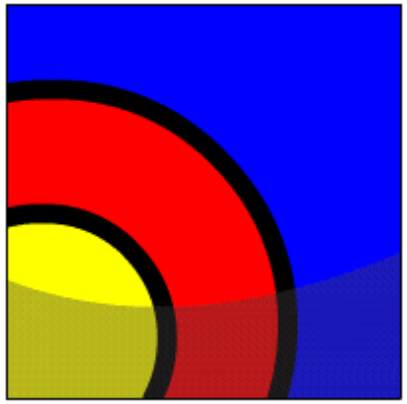


Using Cursors for Update

What Will I Learn?

In this lesson, you will learn to:

- Create PL/SQL code to lock rows before an update using the appropriate clause
- Explain the effect of using `NOWAIT` in an update cursor declaration
- Create PL/SQL code to use the current row of the cursor in an `UPDATE` or `DELETE` statement



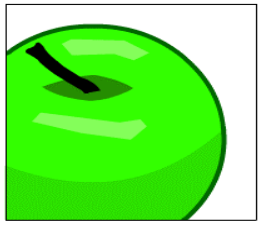
Why Learn It?

If there are multiple users connected to the database at the same time, there is the possibility that another user updated the rows of a particular table after you opened your cursor and fetched the rows.

We can lock rows as we open the cursor, to prevent other users from updating them.

It is important to do this if we want to update the same rows ourselves.

Tell Me/Show Me



Declaring a Cursor with the FOR UPDATE Clause

When we declare a cursor FOR UPDATE, each row is locked as we open the cursor. This prevents other users from modifying the rows while our cursor is open. It also allows us to modify the rows ourselves using a … WHERE CURRENT OF … clause.

Syntax:

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

This does not prevent other users from reading the rows.

Tell Me/Show Me

Declaring a Cursor with the FOR UPDATE Clause

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

column_reference is a column in the table whose rows we need to lock.

If the rows have already been locked by another session:

- NOWAIT returns an Oracle server error immediately
- WAIT n waits for n seconds, and returns an Oracle server error if the other session is still locking the rows at the end of that time.

Tell Me/Show Me

NOWAIT Keyword in the FOR UPDATE Clause

The optional NOWAIT keyword tells the Oracle server not to wait if any of the requested rows have already 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, then the Oracle server waits indefinitely until the rows are available.

Example:

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

Tell Me/Show Me

NOWAIT Keyword in the FOR UPDATE Clause

If the rows are already locked by another session and you have specified NOWAIT, then opening the cursor will result in an error. You can try to open the cursor later.

You can use WAIT *n* instead of NOWAIT and specify the number of seconds to wait and check whether the rows are unlocked. If the rows are still locked after *n* seconds, then an error is returned.

Tell Me/Show Me

FOR UPDATE OF column-name

If the cursor is based on a join of two tables, we may want to lock the rows of one table but not the other. To do this, we specify any column of the table we want to lock.

Example:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT e.employee_id, d.department_name
      FROM  employees e, departments d
      WHERE e.department_id = d.department_id
      AND department_id = 80 FOR UPDATE OF salary;
  ...
```

Tell Me/Show Me

WHERE CURRENT OF Clause

The WHERE CURRENT OF clause is used in conjunction with the FOR UPDATE clause to refer to the current row (the most recently FETCHed 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.

```
WHERE CURRENT OF cursor-name ;
```

Syntax:

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

Tell Me/Show Me

WHERE CURRENT OF Clause (continued)

You can use WHERE CURRENT OF for updating or 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 use a WHERE clause. You must include the FOR UPDATE clause in the cursor query so that the rows are locked on OPEN.

```
WHERE CURRENT OF cursor_name ;
```

Tell Me/Show Me

WHERE CURRENT OF Clause

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.

Example:

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

Tell Me/Show Me

NOWAIT, FOR UPDATE, and WHERE CURRENT OF Clauses

```
DECLARE
  CURSOR empcur IS
    SELECT employee_id, salary FROM my_employees
      WHERE salary <= 20000 FOR UPDATE NOWAIT;
  v_emp_rec empcur%ROWTYPE;
BEGIN
 OPEN empcur;
 LOOP
   FETCH empcur INTO v_emp_rec;
   EXIT WHEN empcur%NOTFOUND;
   UPDATE my_employees
     SET salary = v_emp_rec.salary*1.1
     WHERE CURRENT OF empcur;
 END LOOP;
 CLOSE empcur;
 COMMIT;
END;
```

In this example, we don't need a column-reference in the FOR UPDATE clause because the cursor is not based on a join.

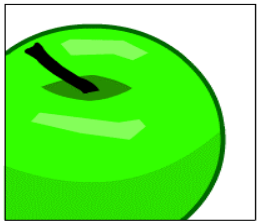
Tell Me/Show Me

A Second Example:

```
DECLARE
  CURSOR ed_cur IS
    SELECT employee_id, salary, department_name
      FROM my_employees e, my_departments d
      WHERE e.department_id = d.department_id
      FOR UPDATE OF salary NOWAIT;
BEGIN
  FOR v_ed_rec IN ed_cur LOOP
    UPDATE my_employees
      SET salary = v_ed_rec.salary*1.1
      WHERE CURRENT OF ed_cur;
  END LOOP;
  COMMIT;
END;
```

`FOR UPDATE OF salary` locks only the `MY_EMPLOYEES` rows, not the `MY_DEPARTMENTS` rows. Note that we update the table-name, not the cursor-name!

Tell Me/Show Me



Terminology

Key terms used in this lesson include:

FOR UPDATE

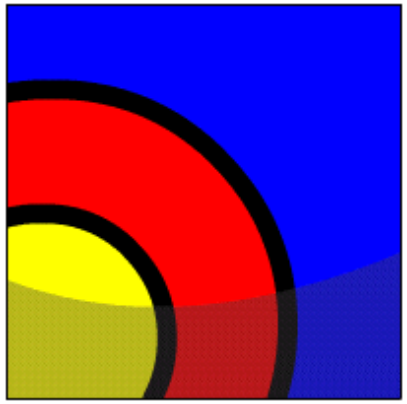
NOWAIT

Using Multiple Cursors

What Will I Learn?

In this lesson, you will learn to:

- Explain the need for using multiple cursors to produce multi-level reports
- Create PL/SQL code to declare and manipulate multiple cursors within nested loops
- Create PL/SQL code to declare and manipulate multiple cursors using parameters



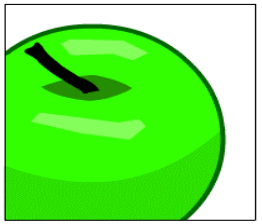
Why Learn It?

In real-life programs you often need to declare and use two or more cursors in the same PL/SQL block. Often these cursors are related to each other by parameters.

One common example is the need for multi-level reports in which each level of the report uses rows from a different cursor.

This lesson does not introduce new concepts or syntax. It shows more powerful uses for the concepts and syntax that you already know.

Tell Me/Show Me



A Sample Problem Statement

You need to produce a report that lists each department as a sub-heading, immediately followed by a listing of the employees in that department, followed by the next department, and so on.

You need two cursors, one for each of the two tables. The cursor based on EMPLOYEES is opened several times, once for each department.

Tell Me/Show Me

Problem Solution: Step 1

Declare two cursors, one for each table, plus associated record structures.

```
DECLARE
  CURSOR c_dept IS
    SELECT department_id, department_name
      FROM departments
      ORDER BY department_name;
  CURSOR c_emp (p_deptid NUMBER) IS
    SELECT first_name, last_name
      FROM employees
      WHERE department_id = p_deptid
      ORDER BY last_name;
  v_deptrec   c_dept%ROWTYPE;
  v_emprec    c_emp%ROWTYPE;
```

Why is cursor c_emp declared with a parameter?

Tell Me/Show Me

Problem Solution: Step 2

Open the c_dept cursor and fetch and display the DEPARTMENTS rows in the usual way.

```
DECLARE
  CURSOR c_dept IS .....;
  CURSOR c_emp (p_deptid NUMBER) IS .....;
  v_deptrec   c_dept%ROWTYPE;
  v_emprec    c_emp%ROWTYPE;
BEGIN
  OPEN c_dept;
  LOOP
    FETCH c_dept INTO v_deptrec;
    EXIT WHEN c_dept%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_deptrec.department_name);
  END LOOP;
  CLOSE c_dept;
END;
```

Tell Me/Show Me

Problem Solution: Step 3

After each DEPARTMENTS row has been fetched and displayed, you need to fetch and display the EMPLOYEES in that department.

To do this, you open the EMPLOYEES cursor, fetch and display its rows in a nested loop, and close the cursor.

Then, you do the same for the next DEPARTMENTS row. And so on.

The next slide shows the code for this.

Tell Me/Show Me

```
DECLARE
  CURSOR c_dept IS .....;
  CURSOR c_emp (p_deptid NUMBER) IS .....;
  v_deptrec   c_dept%ROWTYPE;
  v_emprec    c_emp%ROWTYPE;
BEGIN
  OPEN c_dept;
  LOOP
    FETCH c_dept INTO v_deptrec;
    EXIT WHEN c_dept%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_deptrec.department_name);
    OPEN c_emp (v_deptrec.department_id);
    LOOP
      FETCH c_emp INTO v_emprec;
      EXIT WHEN c_emp%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_emprec.last_name || ' ' ||
           v_emprec.first_name);
    END LOOP;
    CLOSE c_emp;
  END LOOP;
  CLOSE c_dept;
END;
```

Tell Me/Show Me

Using FOR Loops with Multiple Cursors

You can use FOR loops (and other cursor techniques, such as FOR UPDATE) with multiple cursors, just as you can with single cursors.

```
DECLARE
  CURSOR c_loc IS SELECT * FROM locations;
  CURSOR c_dept (p_locid NUMBER) IS
    SELECT * FROM departments WHERE location_id = p_locid;
BEGIN
  FOR v_locrec IN c_loc
  LOOP
    DBMS_OUTPUT.PUT_LINE(v_locrec.city);
    FOR v_deptrec IN c_dept (v_locrec.location_id)
    LOOP
      DBMS_OUTPUT.PUT_LINE(v_deptrec.department_name);
    END LOOP;
  END LOOP;
END;
```

Tell Me/Show Me

A Final Example

List all employees in all departments, and give a salary increase to some of them:

```
DECLARE
  CURSOR c_dept IS SELECT * FROM my_departments;
  CURSOR c_emp (p_dept_id NUMBER) IS
    SELECT * FROM my_employees WHERE department_id = p_dept_id
      FOR UPDATE NOWAIT;
BEGIN
  FOR v_deptrec IN c_dept LOOP
    DBMS_OUTPUT.PUT_LINE(v_deptrec.department_name);
    FOR v_emprec IN c_emp (v_deptrec.department_id) LOOP
      DBMS_OUTPUT.PUT_LINE(v_emprec.last_name);
      IF v_deptrec.location_id = 1700 AND v_emprec.salary < 10000
        THEN UPDATE my_employees SET salary = salary * 1.1
          WHERE CURRENT OF c_emp;
      END IF;
    END LOOP;
  END LOOP;
END;
```