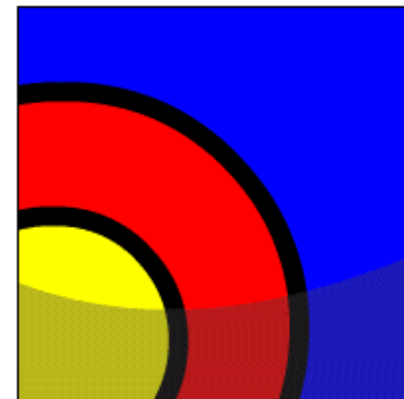


# Introduction to Explicit Cursors

## 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.

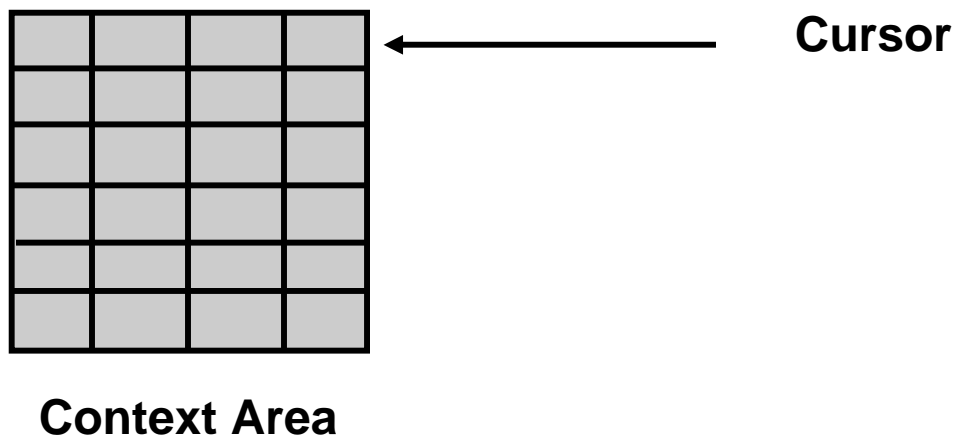
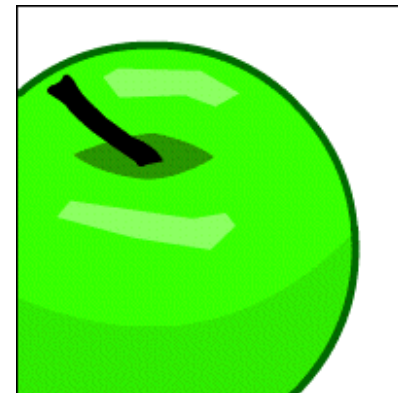


## Tell Me/Show Me

### 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.

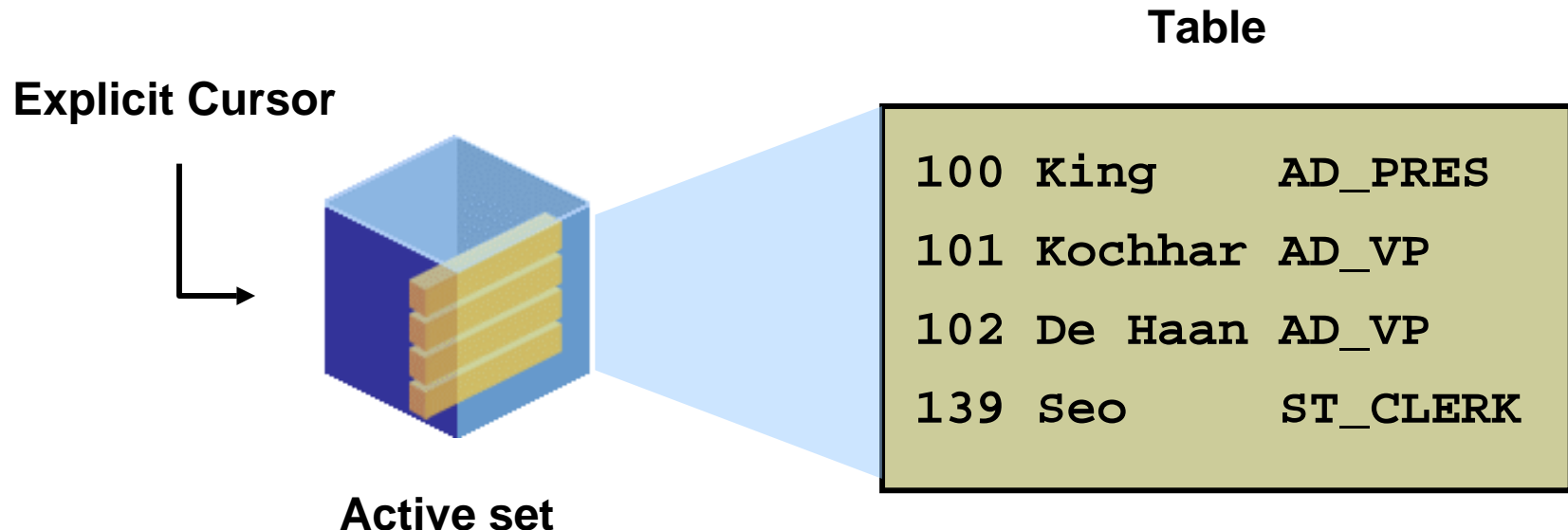
```
DECLARE
  CURSOR wf_holiday_cursor IS
    SELECT country_name, national_holiday_date
    FROM wf_countries where region_id IN(30,34,35);
  v_country_name  wf_countries.country_name%TYPE;
  v_holiday       wf_countries.national_holiday_date%TYPE;
BEGIN
  OPEN wf_holiday_cursor;
  LOOP
    FETCH wf_holiday_cursor 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

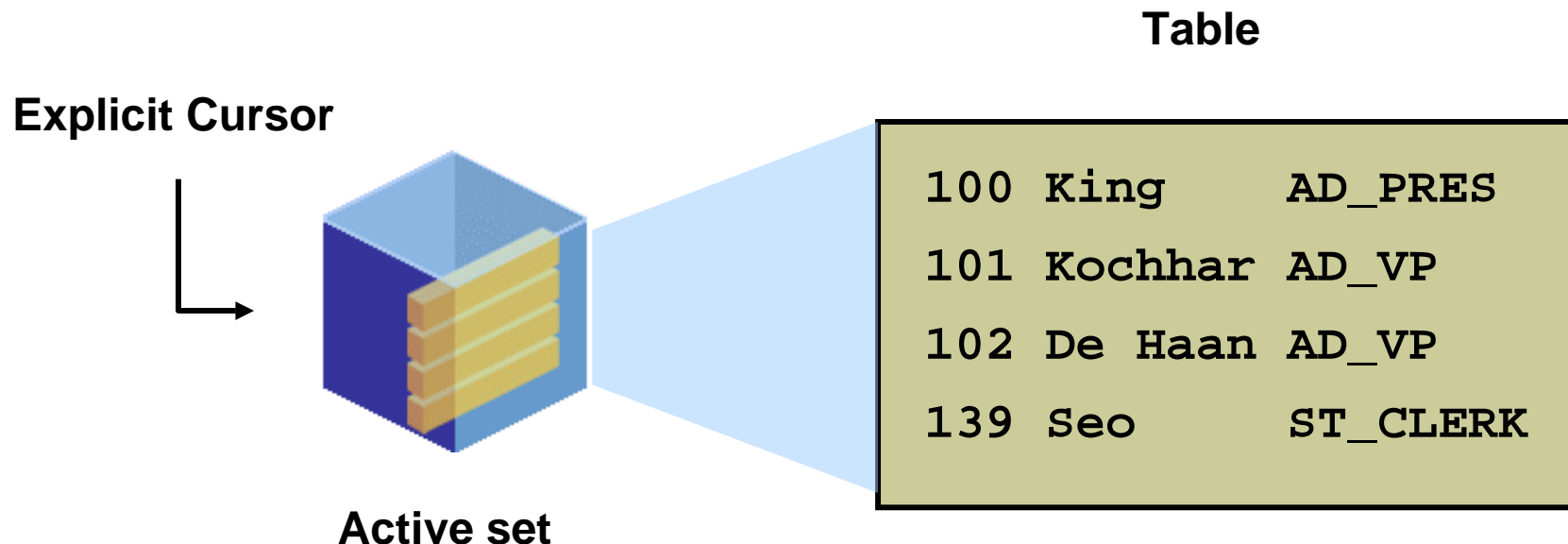
The set of rows returned by a multiple-row query is called the **active set**, and is stored in the context area. Its size is the number of rows that meet your search criteria.



# Tell Me/Show Me

## Explicit Cursor Operations

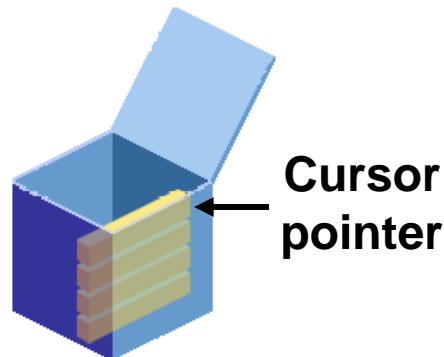
Think of the context area (named by the cursor) as a box, and the active set as the contents of the box. To get at the data, you must `OPEN` the box and `FETCH` each row from the box one at a time. When finished, you must `CLOSE` the box.



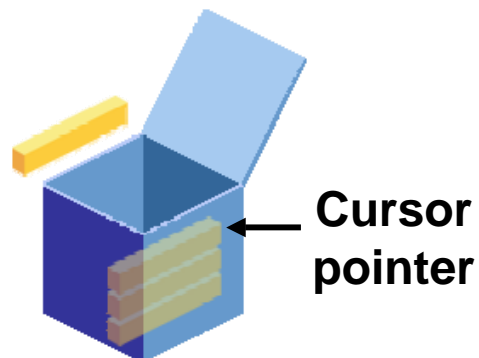
# Tell Me/Show Me

## Controlling Explicit Cursors

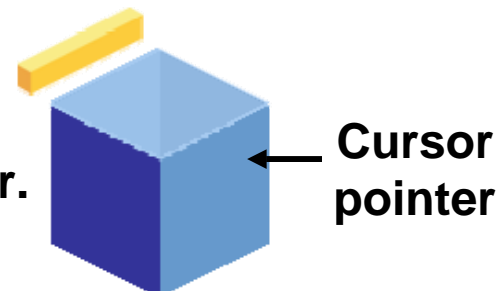
**1** Open the cursor.



**2** Fetch each row, one at a time.

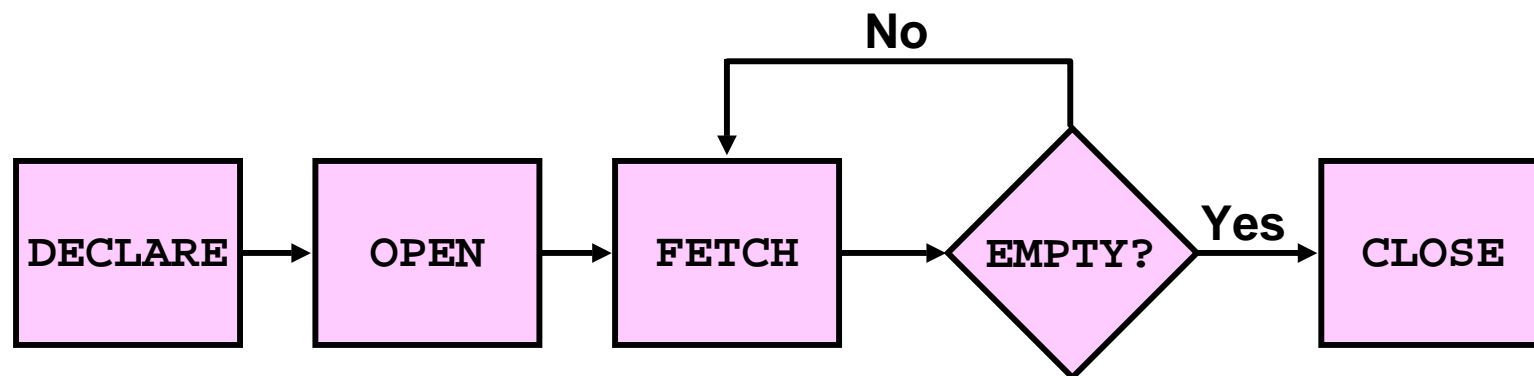


**3** Close the cursor.



# 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.
- Release the active set.
- Return to **FETCH** if rows are found.



## Tell Me/Show Me

### Steps for Using Explicit Cursors

You first DECLARE a cursor, and then you use the OPEN, FETCH, and CLOSE statements to control a cursor.

1. DECLARE the cursor in the declarative section.
2. OPEN the cursor.
3. FETCH each row from the active set until the box is empty.
4. CLOSE the cursor.

The OPEN statement executes the query associated with the cursor, identifies the result set, and positions the cursor before the first row. The FETCH statement retrieves the current row and advances the cursor to the next row. When the last row has been processed, the CLOSE statement disables the cursor.



## 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:

<i>cursor_name</i>	Is a PL/SQL identifier
<i>select_statement</i>	Is a <code>SELECT</code> statement without an <code>INTO</code> 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 2

The `dept_cursor` cursor is declared to retrieve all the details for the departments with the `location_id` 1700. You want to fetch and process these rows in ascending sequence by `department_name`.

```
DECLARE
  CURSOR dept_cursor IS
    SELECT * FROM departments
      WHERE location_id = 1700
      ORDER BY department_name;
...
```





# Tell Me/Show Me

## Declaring the Cursor: Example 3

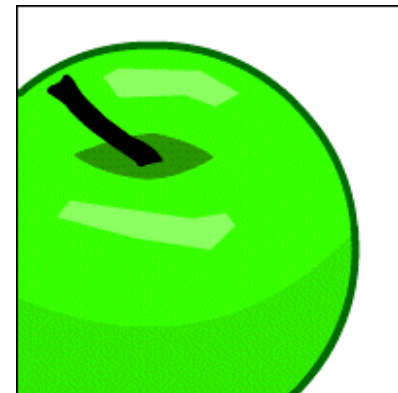
A SELECT statement in a cursor declaration can include joins, group functions, and subqueries. This example retrieves each department that has at least two employees, giving the department name and number of employees.

```
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.

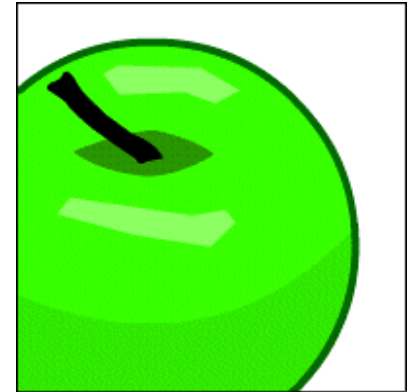
```
DECLARE
    CURSOR emp_cursor IS
        SELECT employee_id, last_name FROM employees
           WHERE department_id =30;
    ...
BEGIN
    OPEN emp_cursor;
    ...
```

## 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.

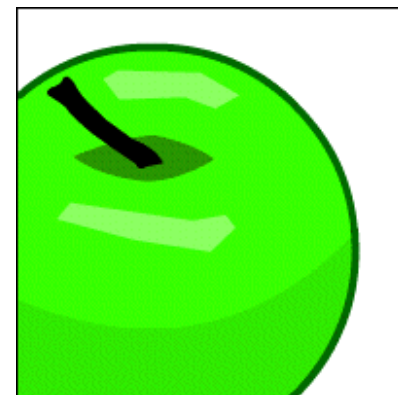
Think of `CLOSE` as closing and emptying the box, so you can no longer `FETCH` its contents.

```
...  
  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

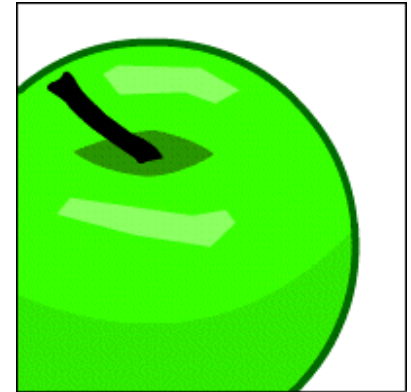
Explicit cursor

Active set

FETCH

OPEN

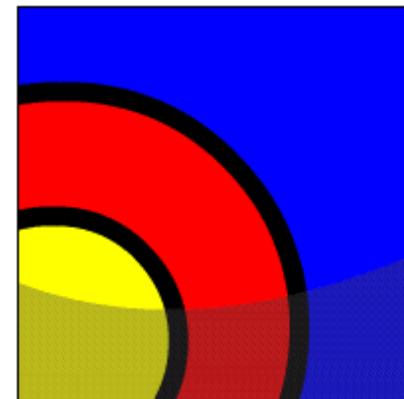
CLOSE



## Summary

In this lesson, you learned 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





## Try It/Solve It

The exercises in this lesson cover the following topics:

- Distinguishing between an implicit and an explicit cursor
- Discussing when and why to use an explicit cursor
- Declaring and controlling explicit cursors
- Using a simple loop to fetch multiple rows from a cursor

