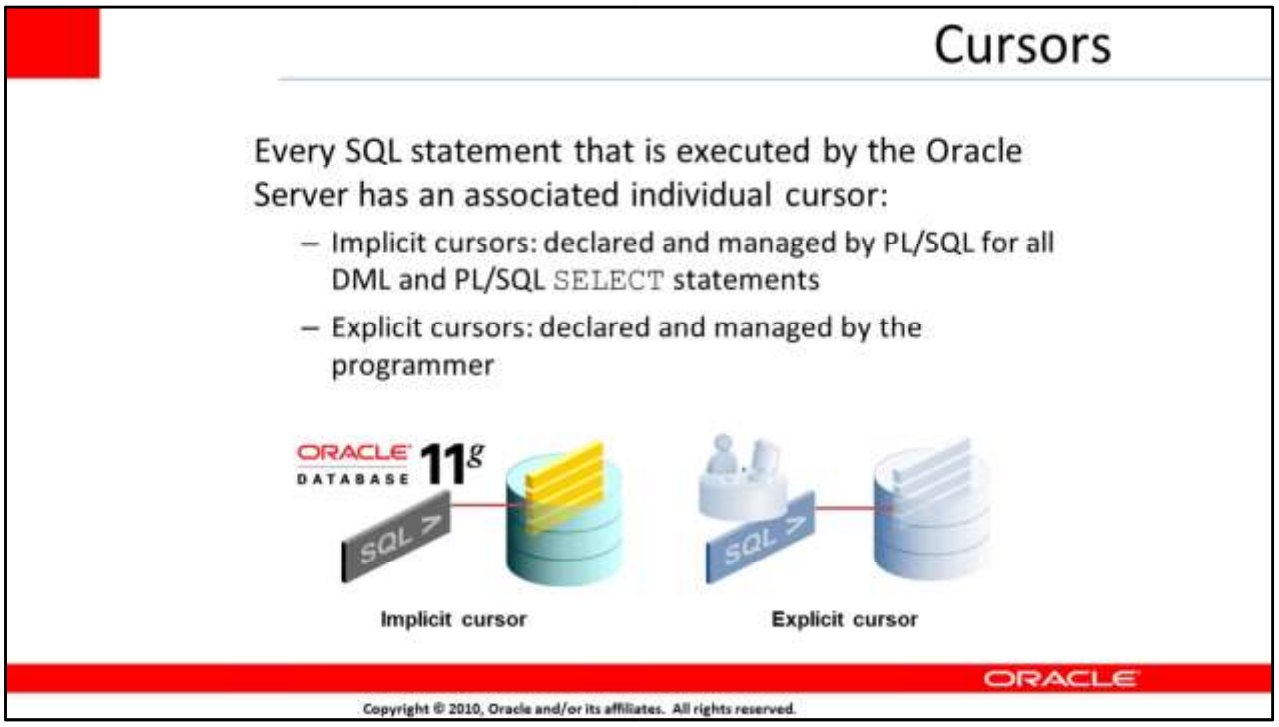
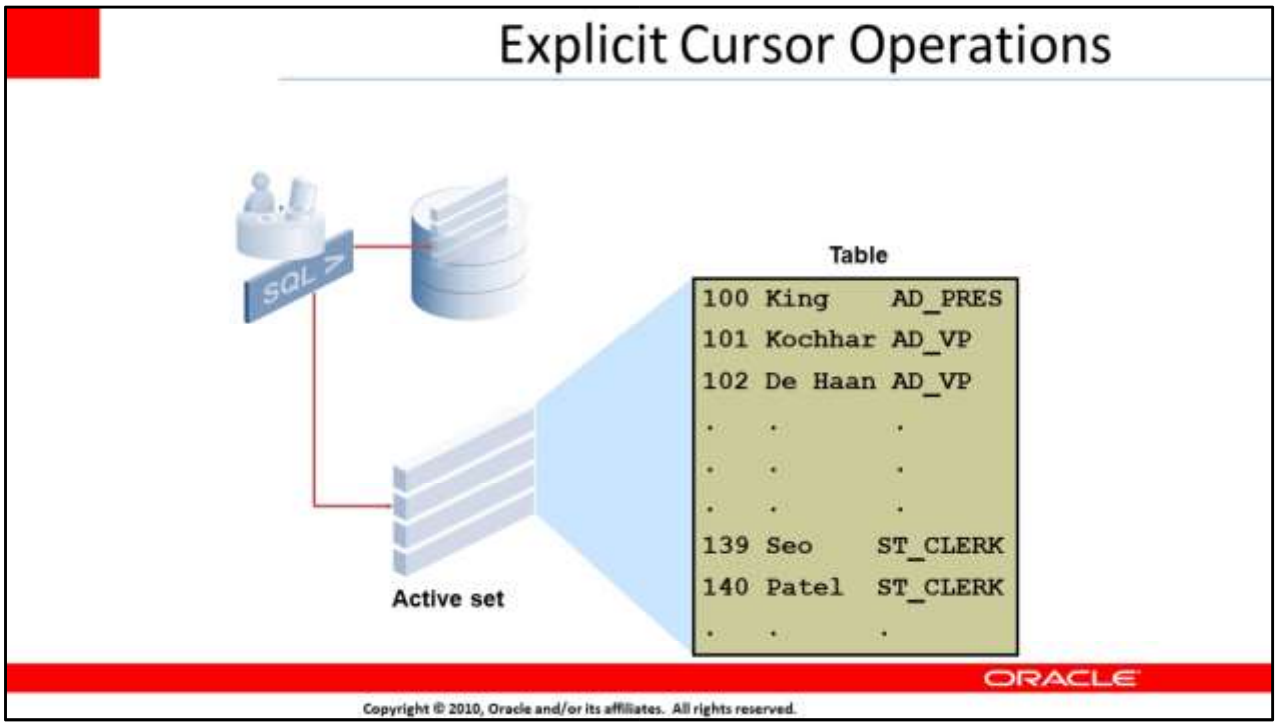


Cursors

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.



Explicit Cursor Operations

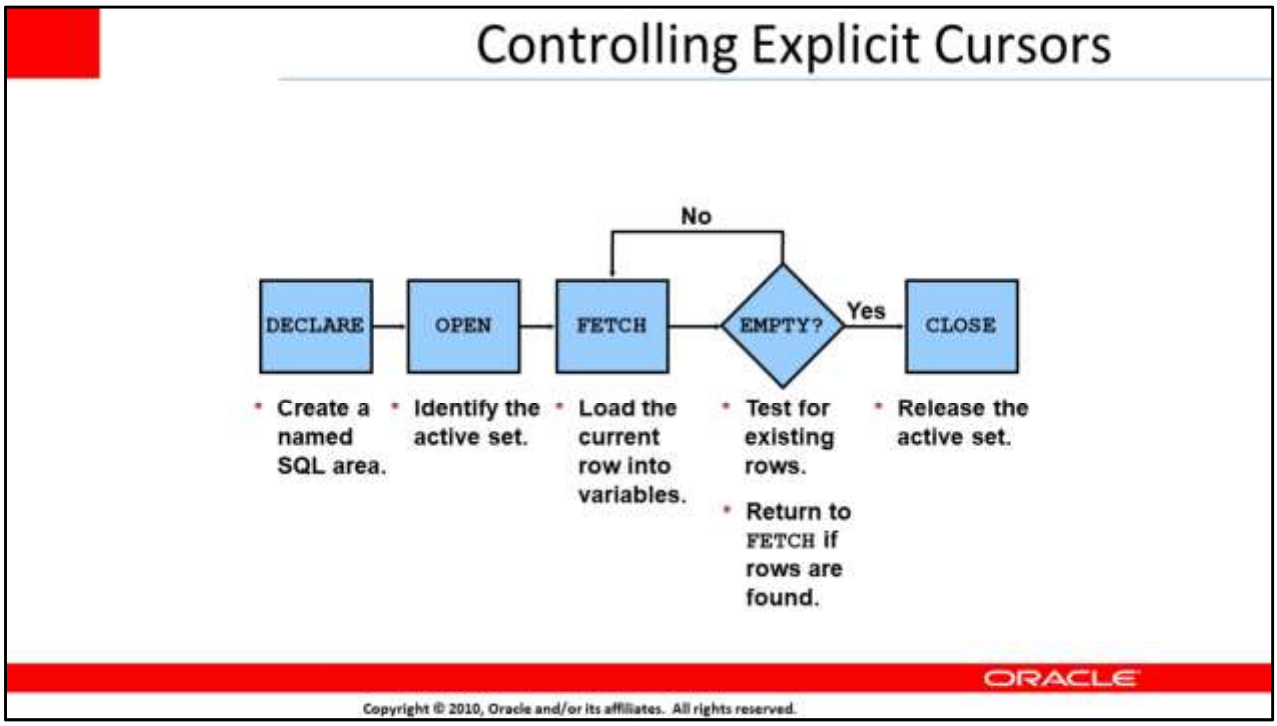
You declare explicit cursors in PL/SQL when you have a SELECT statement that returns multiple rows. You can process each row returned by the SELECT statement.

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. The diagram in the slide shows how an explicit cursor "points" to the current row in the active set. This enables your program to process the rows one at a time.

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

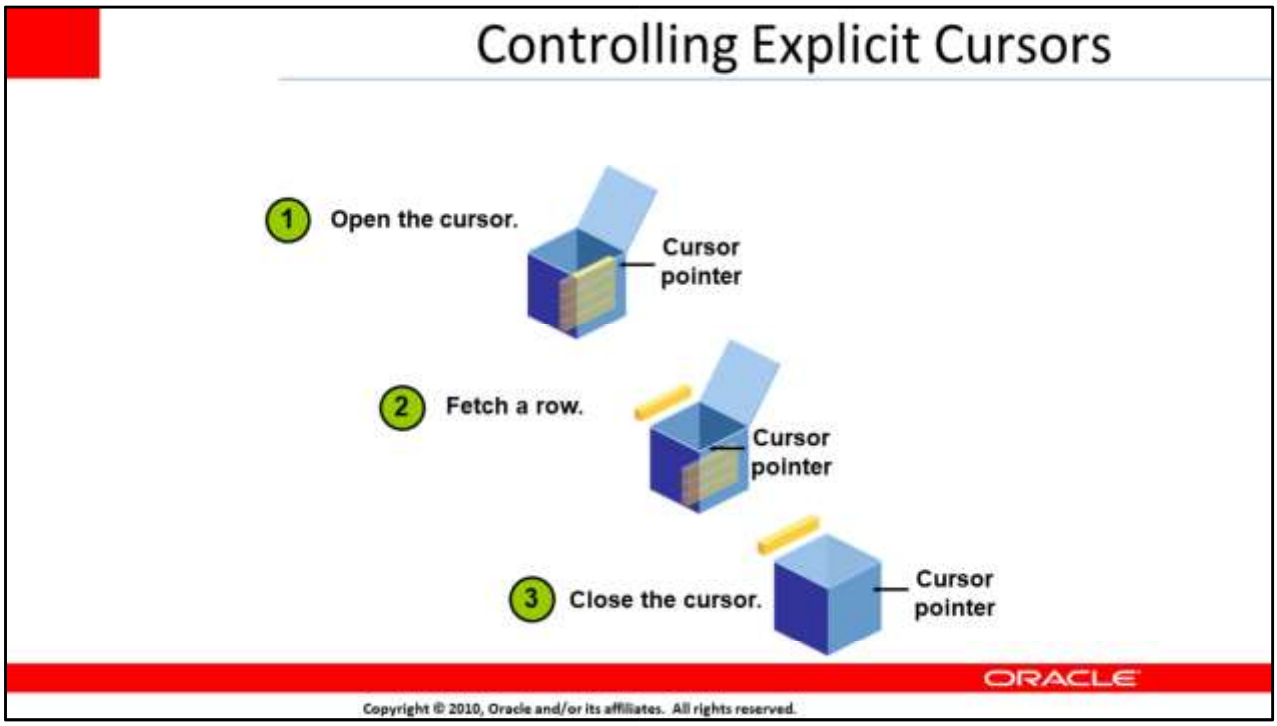
Now that you have a conceptual understanding of cursors, review the steps to use them.

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 and binds any variables that are referenced. 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.



4. Close the cursor.

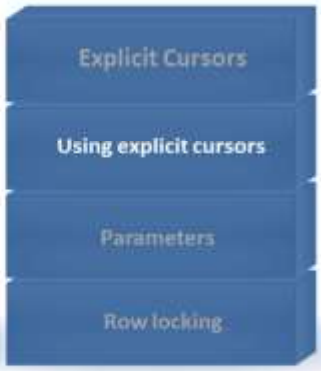
The CLOSE statement releases the active set of rows. It is now possible to reopen the cursor to establish a fresh active set.

Controlling Explicit Cursors (continued)

A PL/SQL program opens a cursor, processes rows returned by a query, and then closes the cursor. The cursor marks the current position in the active set.

1. The OPEN statement executes the query associated with the cursor, identifies the active set, and positions the cursor at the first row.
2. 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.
3. The CLOSE statement releases the cursor.

Session Plan



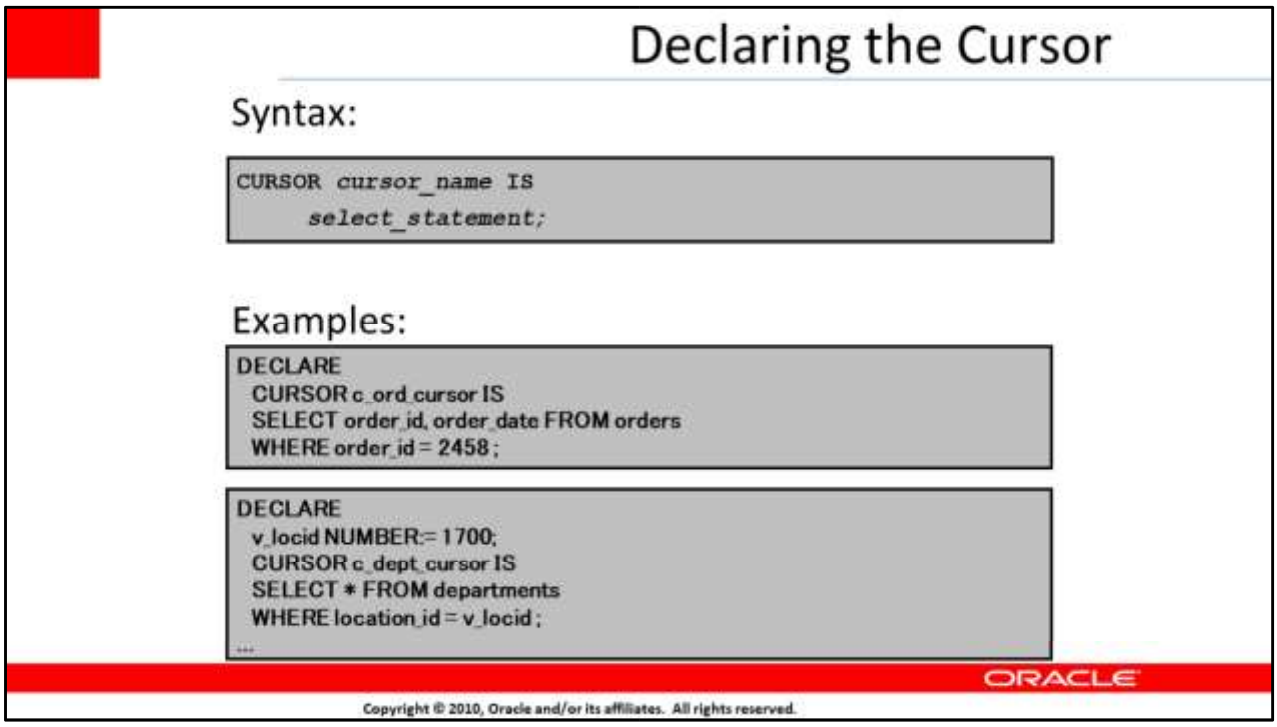
What are explicit cursors?

Performing operations using explicit cursors

Using cursors with parameters

Locking rows and referencing the current row

ORACLE



Declaring the Cursor

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.

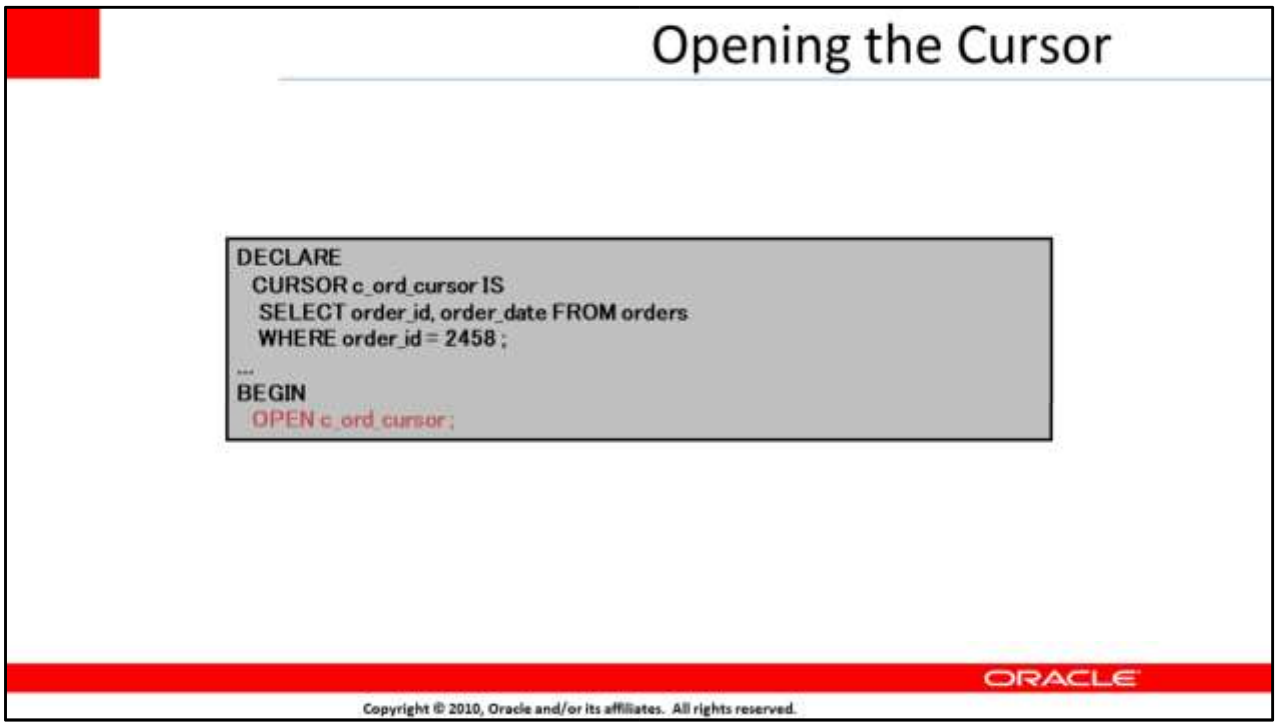
If you want the rows to be processed in a specific sequence, use the ORDERBY clause in the query.

The cursor can be any valid SELECT statement, including joins, subqueries, and so on.

Declaring the Cursor (continued)

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.

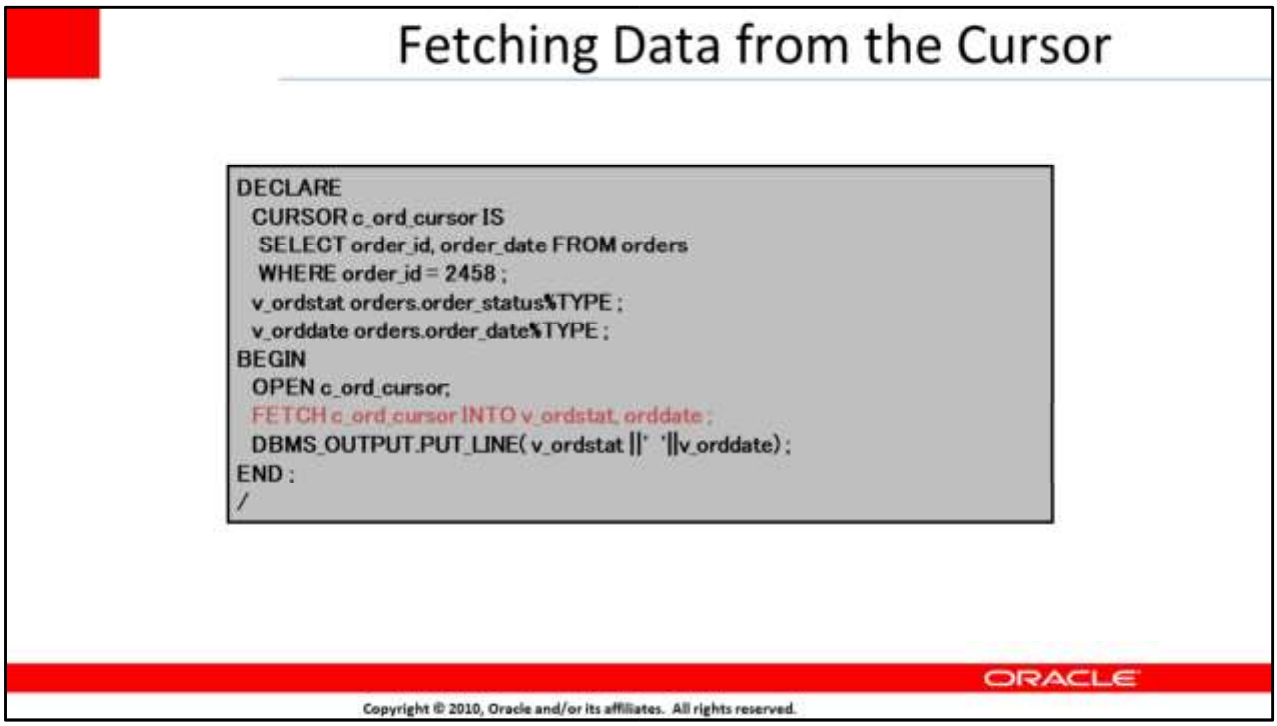
The c_dept_cursor cursor is declared to retrieve all the details for the department with the location_id 1700. Note that a variable is used while declaring the cursor. These variables are considered bind variables, which must be visible when you are declaring the cursor. These variables are examined 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

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.



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.

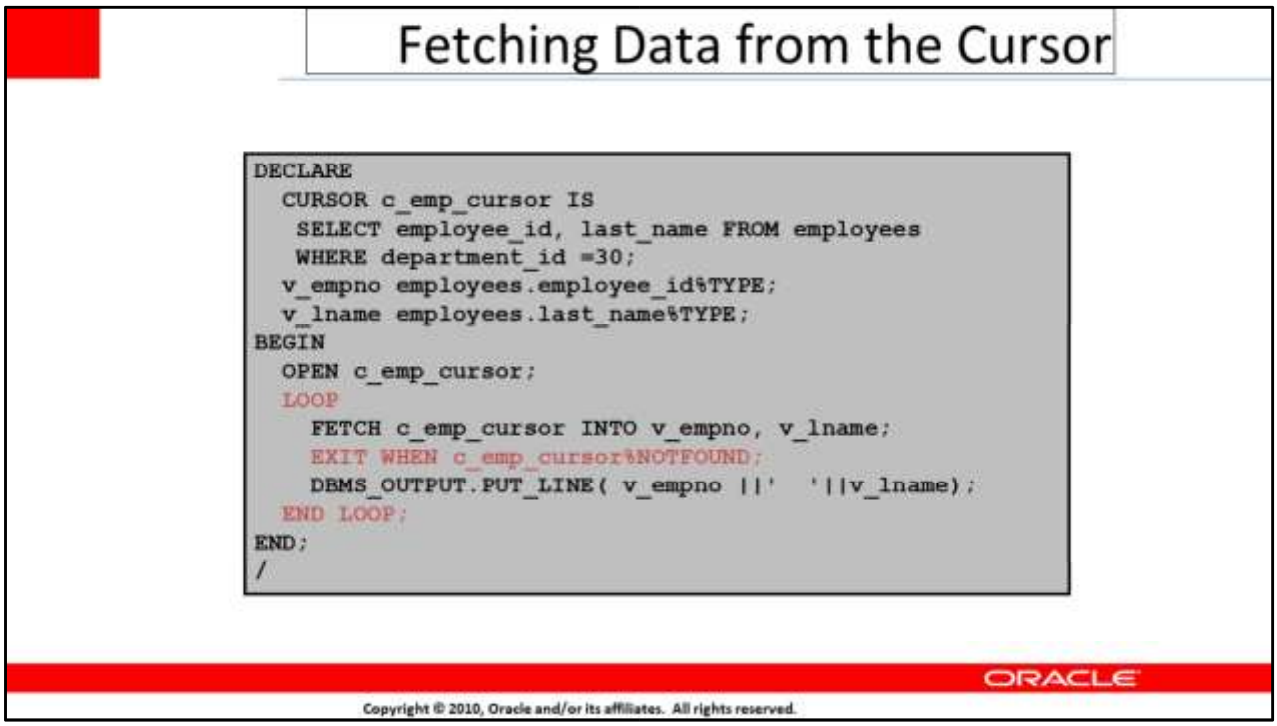
Consider the example shown in the slide. Two variables, ordstatand orddate, are declared to hold the fetched values from the cursor. Examine the FETCH statement.

You have successfully fetched the values from the cursor to the variables. 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

Fetching Data from the Cursor (continued)

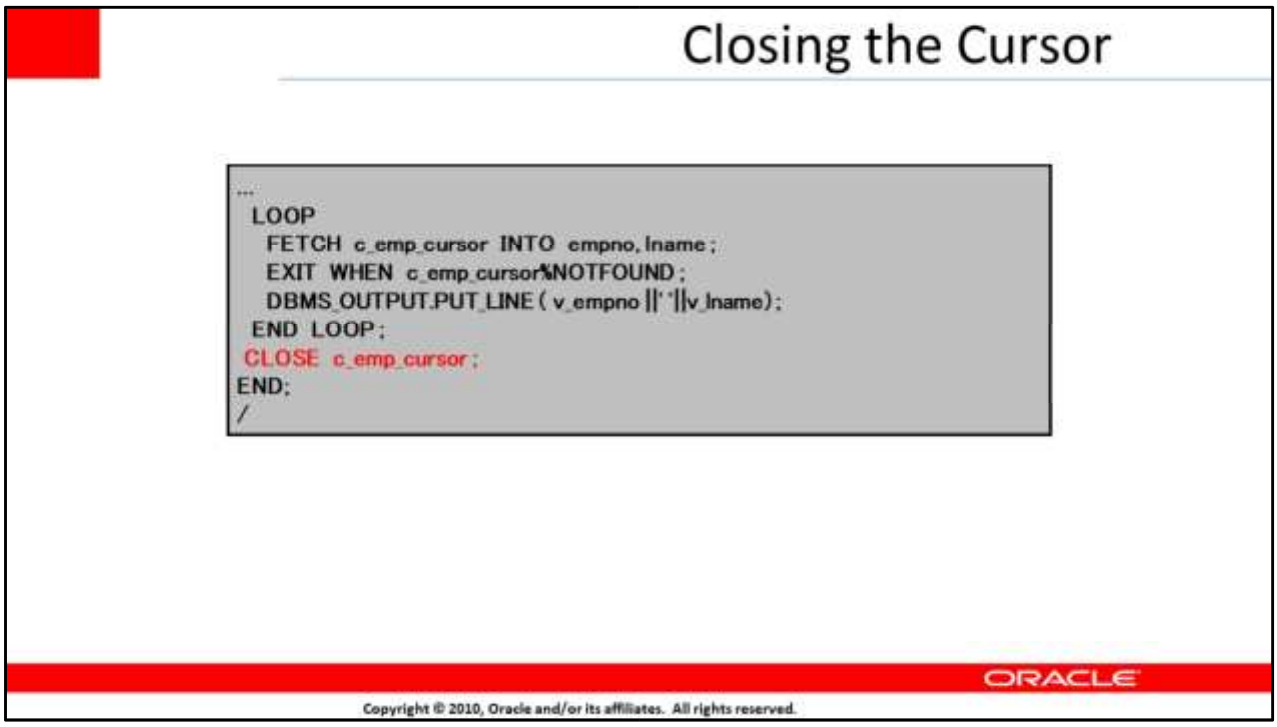
You can include the same number of variables in the INTO clause of the FETCH statement as there are columns in the SELECT statement; be sure that the data types are compatible. Match each variable to correspond to the columns positionally. Alternatively, you can also define a record for the cursor and reference the record in the FETCHINTO 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 (continued)

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
```

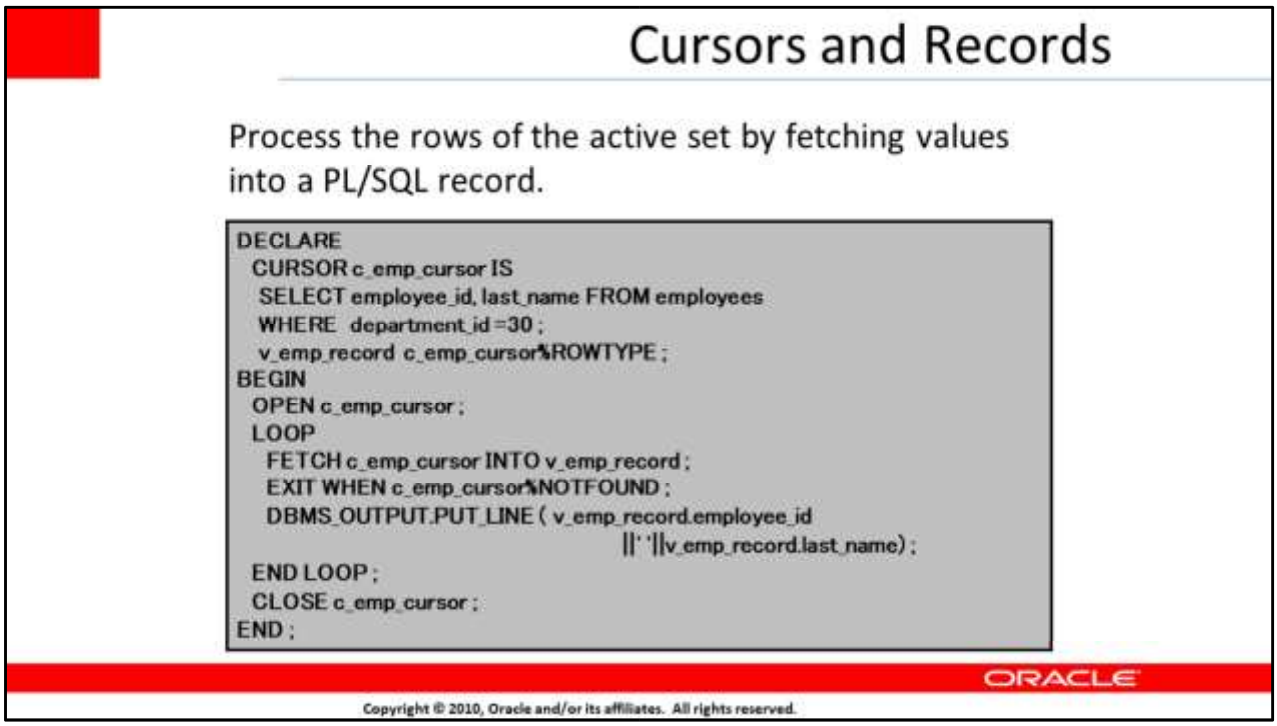


Closing the Cursor

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.

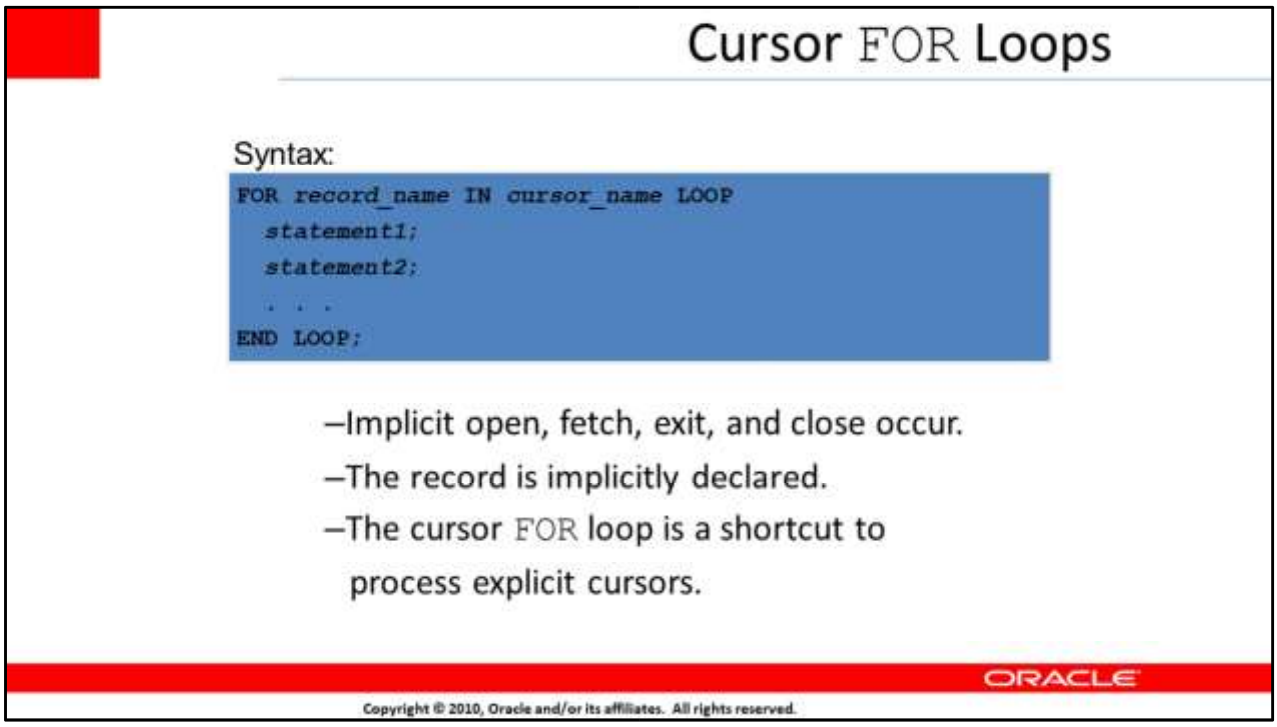
There is a maximum limit on the number of open cursors per session, which is determined by the OPEN_CURSORS parameter in the database parameter file. (OPEN_CURSORS = 50 by default.)



Cursors and Records

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

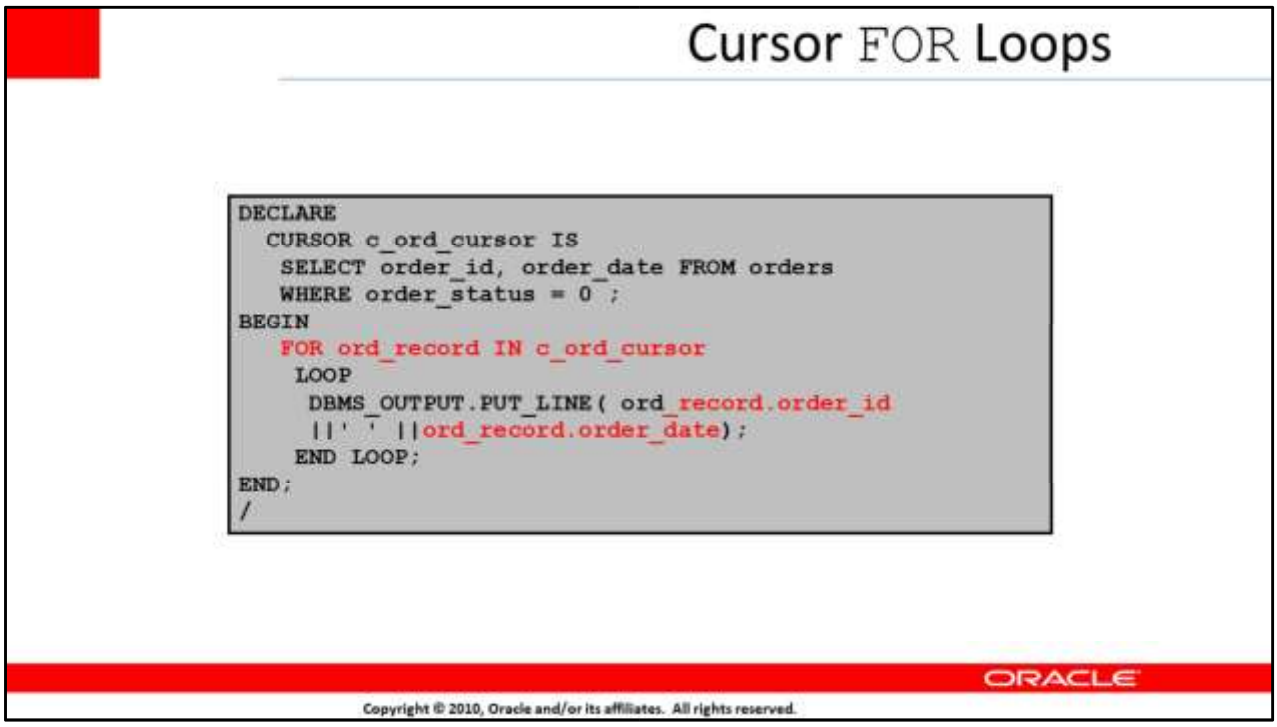
You learned to fetch data from cursors by using simple loops. You now learn to use a cursor FOR loop, which processes rows in an explicit cursor. It is a shortcut because the cursor is opened, a row is fetched once for each iteration in the loop, the loop exits when the last row is processed, and the 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 record — Is the name of the implicitly declared

cursor_name — Is a PL/SQL identifier for the previously declared cursor

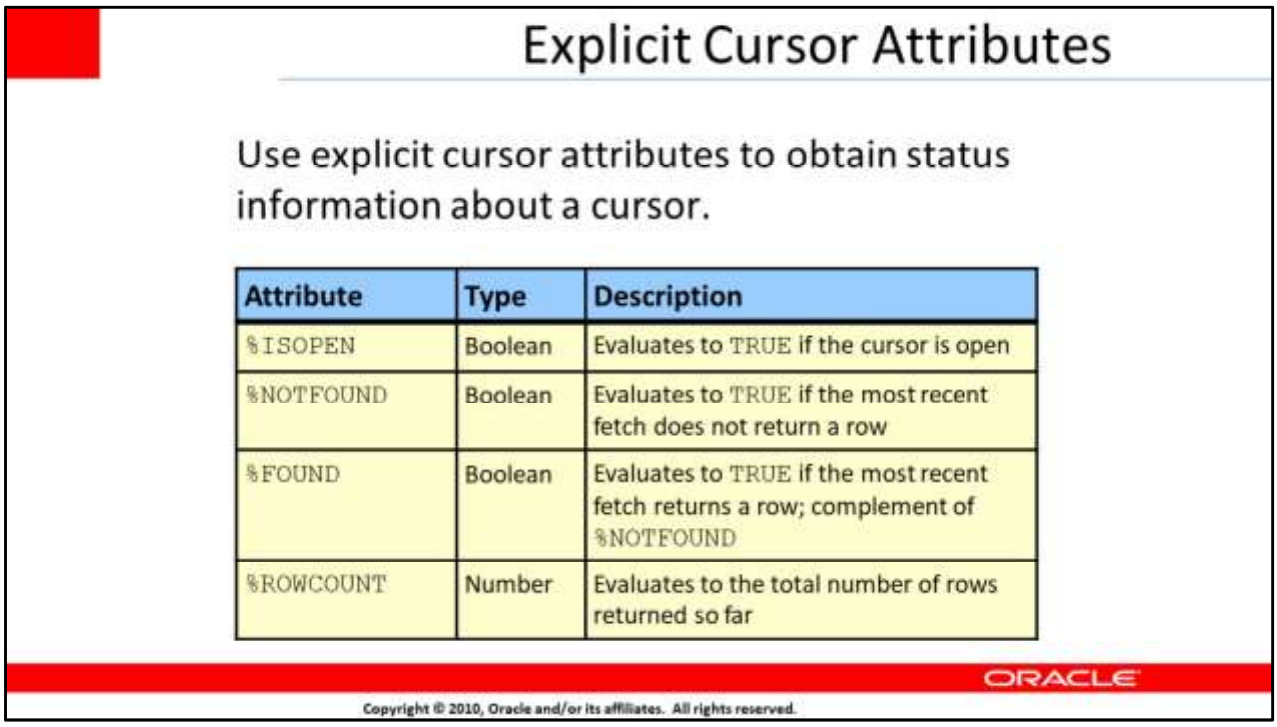
Guidelines

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 (continued)

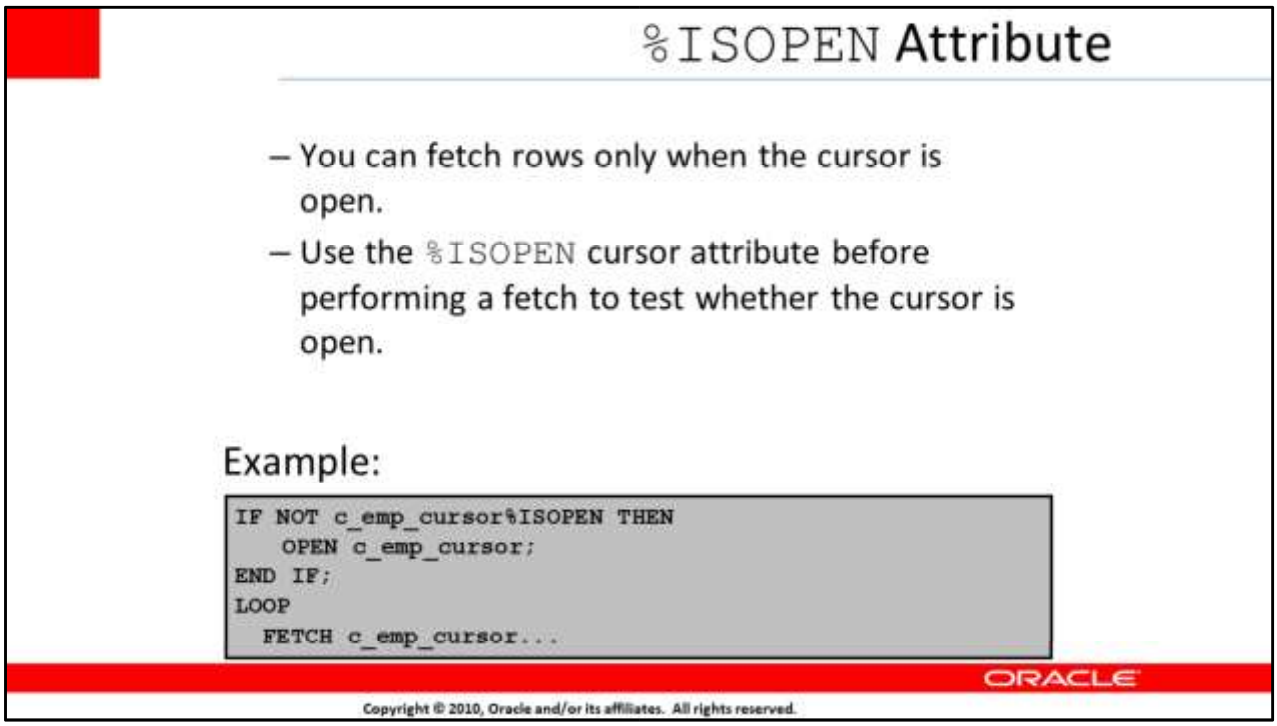
The example that was used to demonstrate the usage of a simple loop to fetch data from cursors is rewritten to use the cursor FOR loop. ord_record is the record that is implicitly declared. You can access the fetched data with this implicit record (as shown in the slide). Observe that no variables are 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

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 to determine whether the cursor is open.

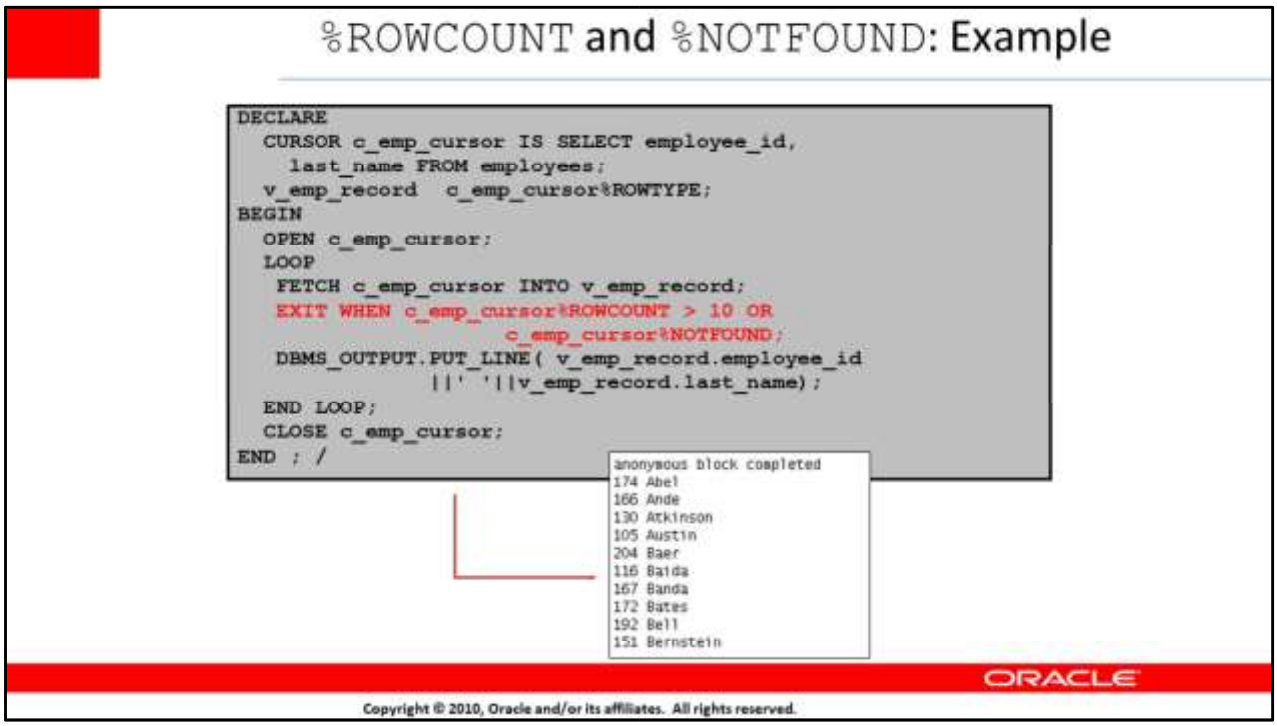
Fetch rows in a loop. Use cursor attributes to determine when to exit the loop.

Use the %ROWCOUNT cursor attribute to do the following:

Process an exact number of rows.

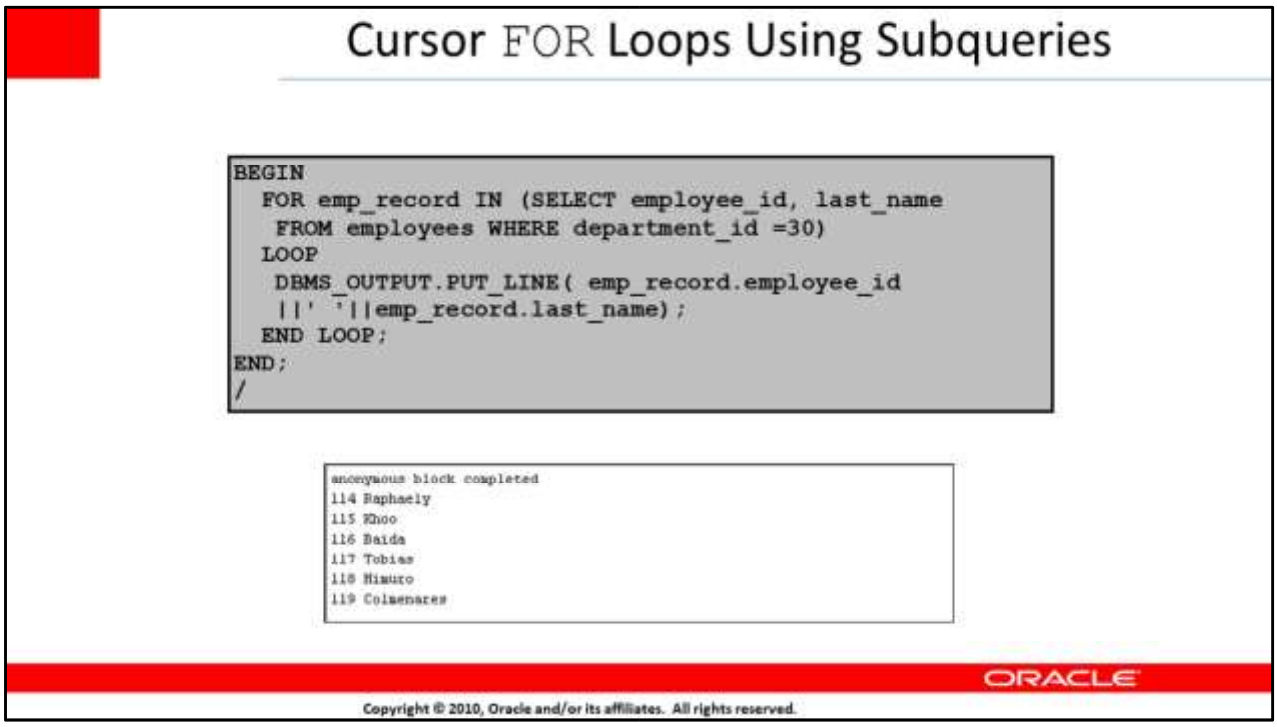
Fetch the rows in a loop and determine when to exit the loop.

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



%ROWCOUNT and %NOTFOUND: Example

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

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.