Table of Contents

[CHAPTER 6: EXPLICIT DATA RETRIEVAL AND MANIPULATION 2](#_Toc412144432)

[Theory 2](#_Toc412144433)

[AIM 4](#_Toc412144434)

[Lab Exercise 6: EXPLICIT DATA RETRIEVAL & MANIPULATION 5](#_Toc412144435)

[1. OPEN, FETCH, and CLOSE 6](#_Toc412144436)

[2. Cursor Variable (SYS\_REFCURSOR) 15](#_Toc412144437)

[3. Explicit FOR LOOP Cursor 22](#_Toc412144438)

[SUMMARY 25](#_Toc412144439)

[REFERENCES 26](#_Toc412144440)

[INDEX 27](#_Toc412144441)

# CHAPTER 6: EXPLICIT DATA RETRIEVAL AND MANIPULATION

## Theory

This chapter is actually an extension to the previous chapter. Explicit cursor is somehow similar to the techniques used in other programming languages. Unlike implicit cursor, explicit cursor is a cursor you define and manage by yourself. The life cycle of explicit cursor usually follows the steps below:

Release cursor resources. The result set is no longer exists.

Create cursor variable and define the query it should run. Declaration and definition can be separated.

Allocate database resources and execute cursor query. The cursor now refers to the position **before** the first row of the result set.

Now you can fetch rows from the result set one by one. Typically, you need scalar variables or a **record** variable to store the row. You also need a **loop** to repeat until no more rows remains in the result set.

Open

Declare & Define

Close

Fetch

It is clearly seen that explicit cursor is more complex than implicit cursor. However, it is more flexible. For example, you can:

* Process multiple result sets in parallel, using multiple cursors.
* Process multiple rows in a single loop iteration, skip rows, or split the processing into multiple loops.
* Specify the query in one PL/SQL unit but retrieve the rows in another.

However, it seems that implicit cursor is much faster than explicit cursor. Some informal tests done on the performance of both methods shows that implicit cursor (SELECT INTO, FOR LOOP cursor) requires less time than explicit cursor (OPEN FETCH CLOSE). Therefore, be careful not to use explicit cursor unless you have a reason for that.

**Explicit Cursor FOR LOOP** statement is a special case of explicit cursor. It does not follow all the steps shown above. It has a Declare & Define step but it reduces the OPEN, FETCH, and CLOSE steps by using FOR LOOP statement. This reduces the coding complexity as in implicit cursor but you still can refer to the cursor name as in the explicit cursor.

The third type of explicit cursors is **cursor variable**. A cursor variable is like an explicit cursor, except that:

* It is not limited to one query: You can open a cursor variable for a query, process the result set, and then use the cursor variable for another query.
* You can use it in an expression.
* **It can be a subprogram parameter**: You can use cursor variable to pass query result sets between subprograms.
* It cannot accept parameters: You cannot pass parameters to a cursor variable, but you can pass whole queries to it. The queries can include variables.

To create a cursor variable, either declare a variable of the predefined type SYS\_REFCURSOR or define a REF CURSOR type and then declare a variable of that type. In this tutorial, we will examine **SYS\_REFCURSOR** only.

## AIM

The AIM of the following exercise is to demonstrate how to use explicit data retrieval.

The steps involved will include:

* OPEN, FETCH, and CLOSE
* Cursor Variable (SYS\_REFCURSOR)
* Explicit FOR LOOP

In general, lab exercises are done in sequential order. Thus, it is assumed that you successfully completed the previous labs. However, not all previous labs are required. Please be sure to run the following lab before proceeding:

* Installing Oracle Database 12c.
* Implicit Data Retrieval and Manipulation

Estimated Completion Time:

25 minutes

# Lab Exercise 6: EXPLICIT DATA RETRIEVAL & MANIPULATION

|  |
| --- |
|  |

## OPEN, FETCH, and CLOSE

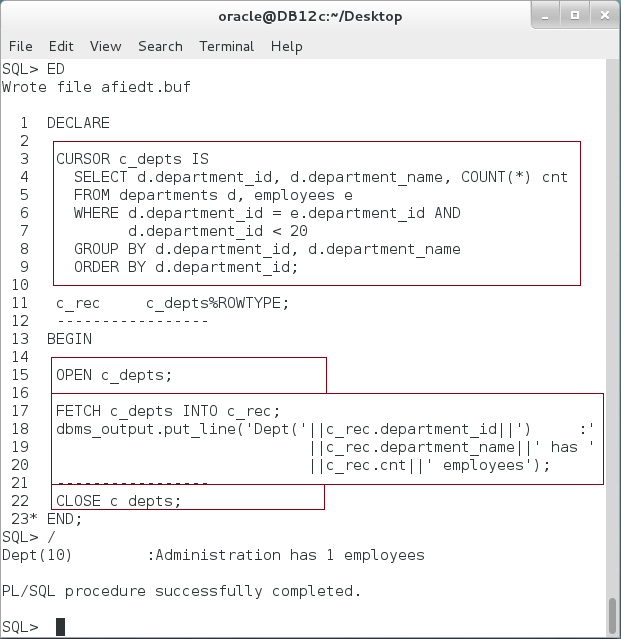
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

**Step 2:** Execute the following block:

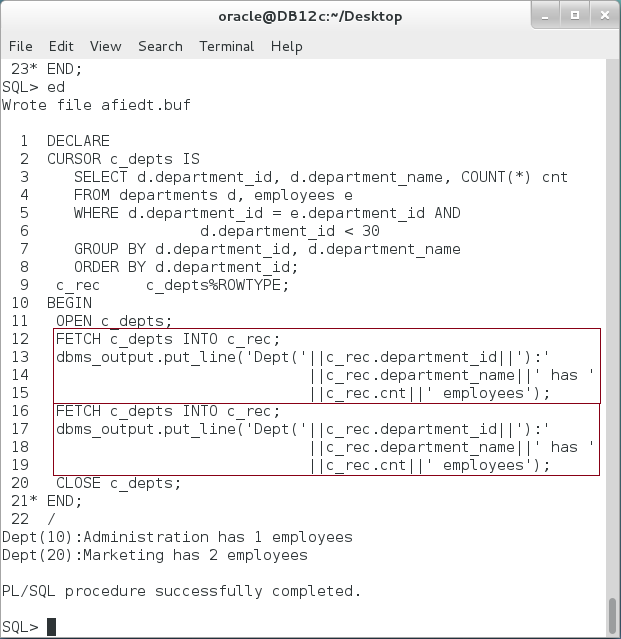
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| **CURSOR** c\_depts **IS** **SELECT** d.department\_id, d.department\_name, COUNT(\*) cnt FROM departments d, employees e  WHERE d.department\_id = e.department\_id AND  d.department\_id < 20  GROUP BY d.department\_id, d.department\_name  ORDER BY d.department\_id; | **Declare** and **Define** a cursor c\_depts |
| c\_rec c\_depts%ROWTYPE; | Define a record variable of c\_depts cursor. |
| BEGIN |  |
| OPEN c\_depts; | **Open cursor** |
| FETCH c\_depts INTO c\_rec; | **Fetch record** |
| dbms\_output.put\_line('Dept('||c\_rec.department\_id||'):' |
| ||c\_rec.department\_name||' has ' |
| ||c\_rec.cnt||' employees'); |
| CLOSE c\_depts; | **Close cursor.** |
| END; |  |

****

**Please note**: In the previous PL/SQL block, one record was fetched from the result set. To fetch more records, you may need to repeat FETCH step.

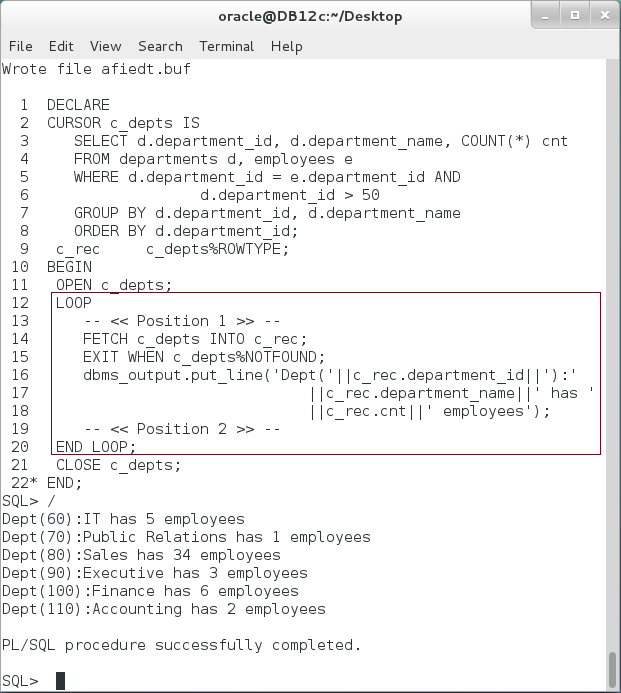
**Step 3:** Modify the previous PL/SQL block as show below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| CURSOR c\_depts IS SELECT d.department\_id, d.department\_name, COUNT(\*) cnt FROM departments d, employees e  WHERE d.department\_id = e.department\_id AND  **d.department\_id < 30**  GROUP BY d.department\_id, d.department\_name  ORDER BY d.department\_id; | Declare and Define a cursor c\_depts |
| c\_rec c\_depts%ROWTYPE; | Define a record variable of c\_depts cursor. |
| BEGIN |  |
| OPEN c\_depts; | Open cursor |
| FETCH c\_depts INTO c\_rec; | Fetch one record |
| dbms\_output.put\_line('Dept('||c\_rec.department\_id||'):' |
| ||c\_rec.department\_name||' has ' |
| ||c\_rec.cnt||' employees'); |
| **FETCH c\_depts INTO c\_rec;** | **Fetch second record** |
| **dbms\_output.put\_line('Dept('||c\_rec.department\_id||'):'** |
| **||c\_rec.department\_name||' has '** |
| **||c\_rec.cnt||' employees');** |
| CLOSE c\_depts; | Close cursor. |
| END; |  |

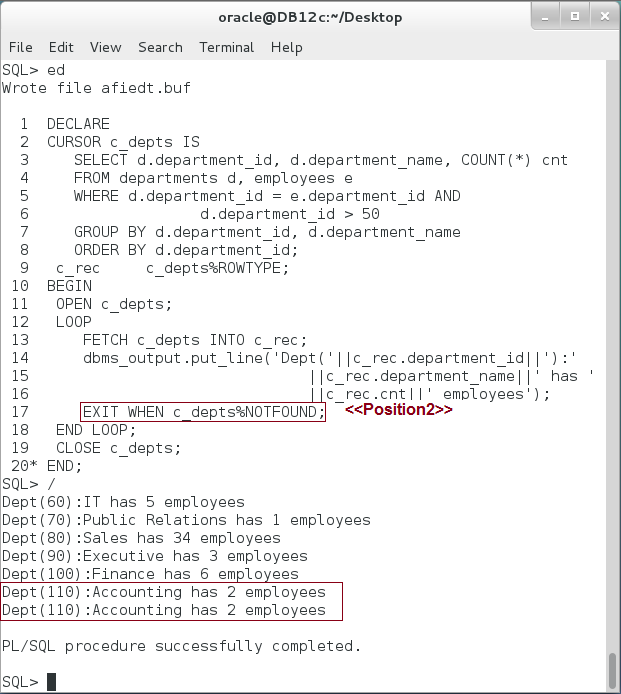
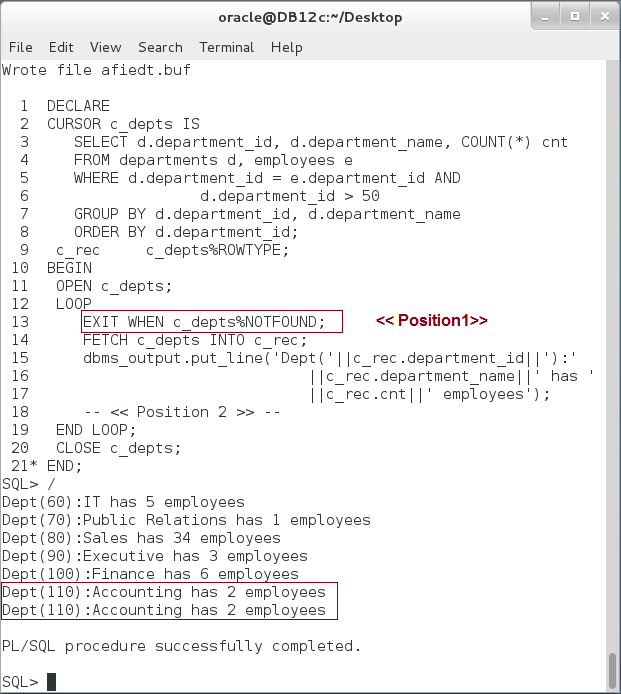


**Step 4:** Obviously, this is not a right way to fetch multiple records; there should be some kind of LOOP that repeats the fetch process until no more rows are left in the record set. Modify the previous PL/SQL block as show below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| CURSOR c\_depts IS SELECT d.department\_id, d.department\_name, COUNT(\*) cnt FROM departments d, employees e  WHERE d.department\_id = e.department\_id AND  **d.department\_id > 50**  GROUP BY d.department\_id, d.department\_name  ORDER BY d.department\_id; | Declare and Define a cursor c\_depts |
| c\_rec c\_depts%ROWTYPE; | Define a record. |
| BEGIN |  |
| OPEN c\_depts; | Open cursor |
| **LOOP** |  |
| **-- << Position 1 >> --** |  |
| FETCH c\_depts INTO c\_rec; | Fetch one record |
| **EXIT WHEN c\_depts%NOTFOUND;** | Stop when no more rows available. |
| dbms\_output.put\_line('Dept('||c\_rec.department\_id||'):' |  |
| ||c\_rec.department\_name||' has ' |  |
| ||c\_rec.cnt||' employees'); |  |
| **-- << Position 2 >> --** |  |
| **END LOOP;** |  |
| CLOSE c\_depts; | Close cursor. |
| END; |  |

****

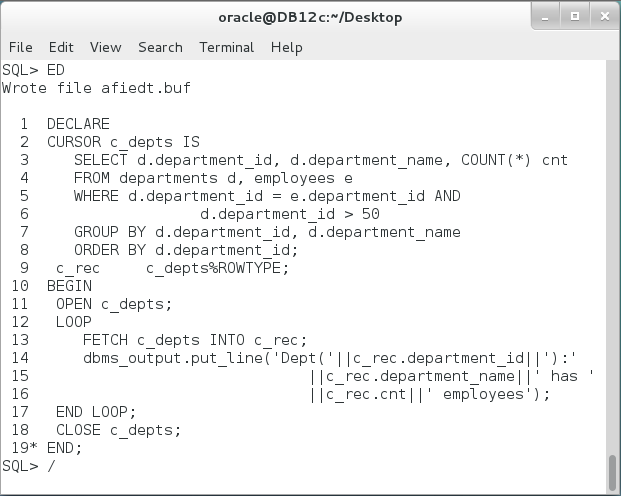
**Step 5:** Please note the position in where EXIT statement was placed in the previous block: right after a FETCH statement. What if we change the position of EXIT statement to the **<< Position 1>>** or **<< Position 2>>**? **Explain the output**.



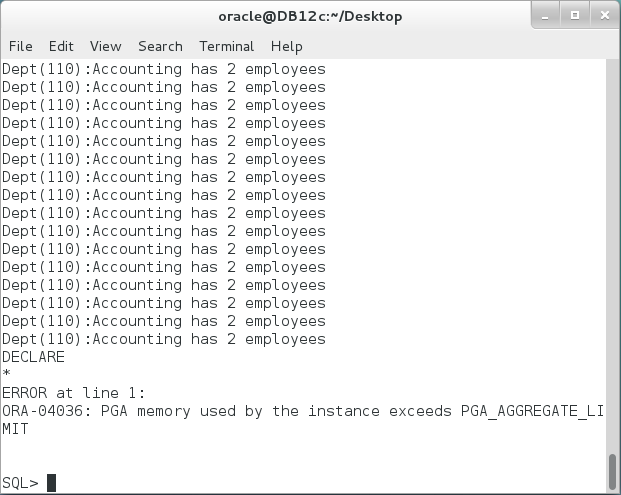
**Please note**: The value of %NOTFOUND attribute is NULL before the first FETCH statement. After that, its value is either TRUE or FALSE.

**Step 6:** In the previous block, remove EXIT statement at all as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| CURSOR c\_depts IS SELECT d.department\_id, d.department\_name, COUNT(\*) cnt FROM departments d, employees e  WHERE d.department\_id = e.department\_id AND  d.department\_id > 50  GROUP BY d.department\_id, d.department\_name  ORDER BY d.department\_id; | Declare and Define a cursor c\_depts |
| c\_rec c\_depts%ROWTYPE; | Define a record. |
| BEGIN |  |
| OPEN c\_depts; | Open cursor |
| **LOOP** |  |
| FETCH c\_depts INTO c\_rec; | Fetch one record |
| dbms\_output.put\_line('Dept('||c\_rec.department\_id||'):' |  |
| ||c\_rec.department\_name||' has ' |  |
| ||c\_rec.cnt||' employees'); |  |
| **END LOOP;** |  |
| CLOSE c\_depts; | Close cursor. |
| END; |  |

It seems the code runs in **infinite loop.** ****

If you wait too long, Oracle will run out of allowed memory and raise an exception as shown below:

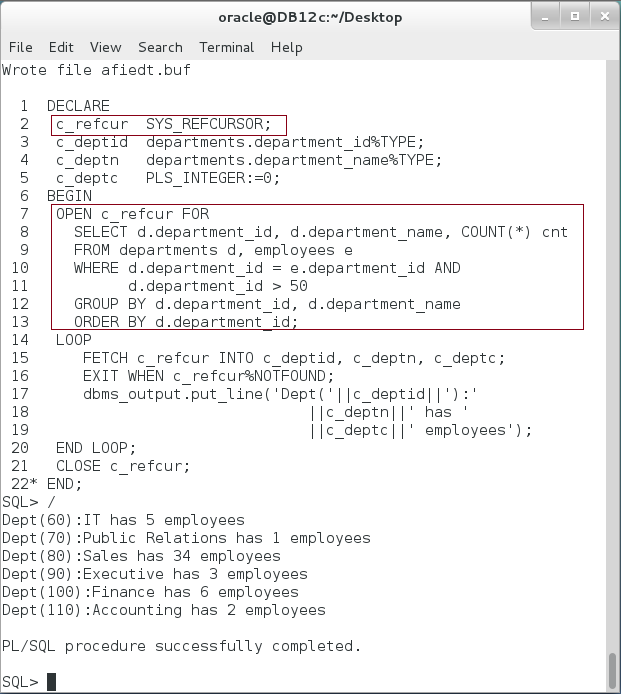


**Please note:** Oracle does not raise exception when a FETCH statement returns no row. Be careful when using LOOP with FETCH statement; the place of EXIT statement matters and forgetting it at all leads to infinite loop.

## Cursor Variable (SYS\_REFCURSOR)

**Step 1:** Execute the following PL/SQL block:

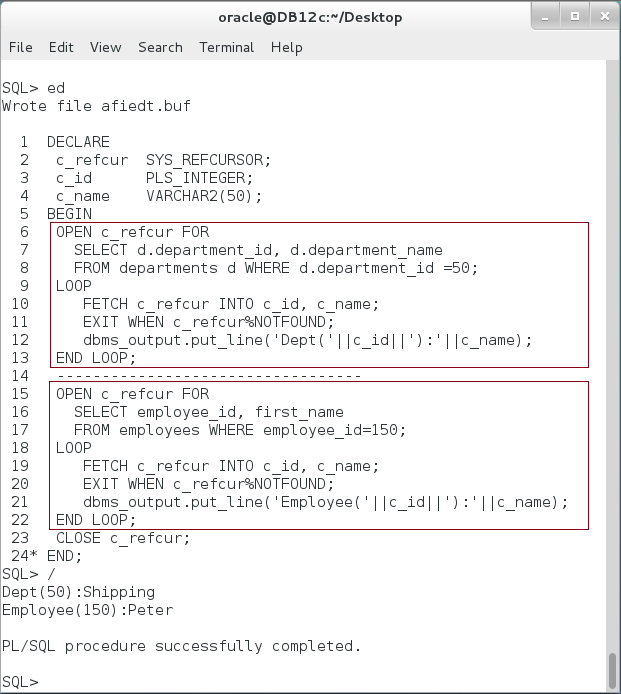
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **c\_refcur SYS\_REFCURSOR;** | Declare a variable "c\_refcur" from a pre-defined system cursor. |
| c\_deptid departments.department\_id%TYPE; |  |
| c\_deptn departments.department\_name%TYPE; |  |
| c\_deptc PLS\_INTEGER:=0; |  |
| BEGIN |  |
| **OPEN** c\_refcur **FOR** | Define the SELECT statement and open the cursor. |
| SELECT d.department\_id, d.department\_name,COUNT(\*) cnt |
| FROM departments d, employees e |
| WHERE d.department\_id = e.department\_id AND |
| d.department\_id > 50 |
| GROUP BY d.department\_id, d.department\_name |
| ORDER BY d.department\_id; |
| LOOP | The loop is exactly as a regular explicit cursor. |
| FETCH c\_refcur INTO c\_deptid, c\_deptn, c\_deptc; |
| EXIT WHEN c\_refcur%NOTFOUND; |
| dbms\_output.put\_line('Dept('||c\_deptid||'):' |
| ||c\_deptn||' has ' |
| ||c\_deptc||' employees'); |
| END LOOP; |
| CLOSE c\_refcur; |
| END; |  |
| / |  |



**Please note:** So far, it seems a variable cursor add nothing but extra OPEN FOR clause. The entire loop and the result is the same as regular explicit cursor.

**Step 2:** Execute the following PL/SQL block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| c\_refcur SYS\_REFCURSOR; | Define ref cursor |
| c\_id PLS\_INTEGER; |  |
| c\_name VARCHAR2(50); |  |
| BEGIN |  |
| **OPEN c\_refcur FOR** | Open the ref cursor for the first SQL statement. |
| SELECT d.department\_id, d.department\_name |
| FROM departments d WHERE d.department\_id =50; |
| LOOP |
| FETCH c\_refcur INTO c\_id, c\_name; |
| EXIT WHEN c\_refcur%NOTFOUND; |
| dbms\_output.put\_line('Dept('||c\_id||'):'||c\_name); |
| END LOOP; |
| ---------------------------------- |  |
| **OPEN c\_refcur FOR** | Open the same ref cursor for another SQL statement. |
| SELECT employee\_id, first\_name |
| FROM employees WHERE employee\_id=150; |
| LOOP |
| FETCH c\_refcur INTO c\_id, c\_name; |
| EXIT WHEN c\_refcur%NOTFOUND; |
| dbms\_output.put\_line('Employee('||c\_id||'):'||c\_name); |
| END LOOP; |
| CLOSE c\_refcur; | Close ref cursor at the end; |
| END; |  |
| / |  |

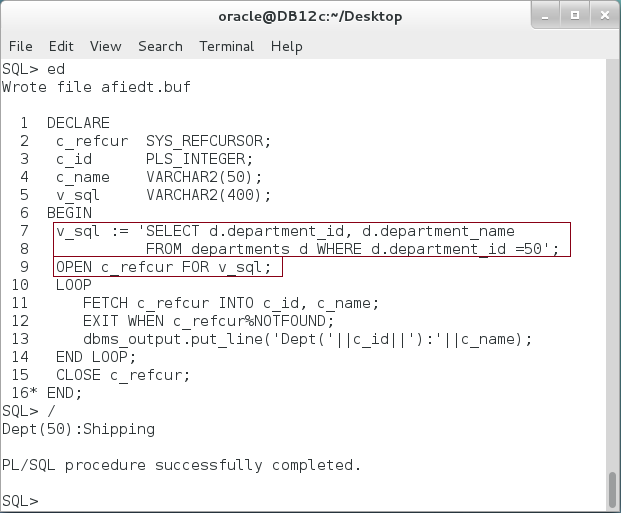


**Please note**: A ref cursor is declared once but defined and used several times. This cannot be done using a regular explicit cursor in which you declare and define a cursor in a declaration part.

**Step 3:** Another important feature of variable cursor (REF CURSOR, SYS\_REFCURSOR) is that you can pass it as function parameter. It is important to bear in mind that all other explicit or implicit cursors cannot achieve this purpose. This feature will be covered in a stored function and procedure.

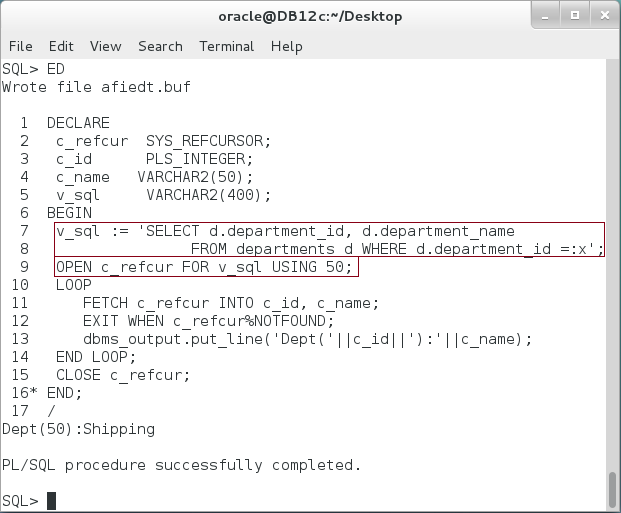
**Step 4:** Variable cursors can be defined with a text variable as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| c\_refcur SYS\_REFCURSOR; |  |
| c\_id PLS\_INTEGER; |  |
| c\_name VARCHAR2(50); |  |
| v\_sql VARCHAR2(400); |  |
| BEGIN |  |
| **v\_sql** := 'SELECT d.department\_id, d.department\_name | Text variable holding a query |
| FROM departments d WHERE d.department\_id =50'; |
| **OPEN** c\_refcur **FOR** v\_sql; | Open the ref cursor using a text variable. |
| LOOP |  |
| FETCH c\_refcur INTO c\_id, c\_name; |  |
| EXIT WHEN c\_refcur%NOTFOUND; |  |
| dbms\_output.put\_line('Dept('||c\_id||'):'||c\_name); |  |
| END LOOP; |  |
| CLOSE c\_refcur; |  |
| END; |  |
| / |  |

****

**Step 5:** Oracle highly recommends **not** embedding variable values inside the query as shown in the previous PL/SQL block. Instead, it is better performed and more secure using a USING clause as show in the following block:

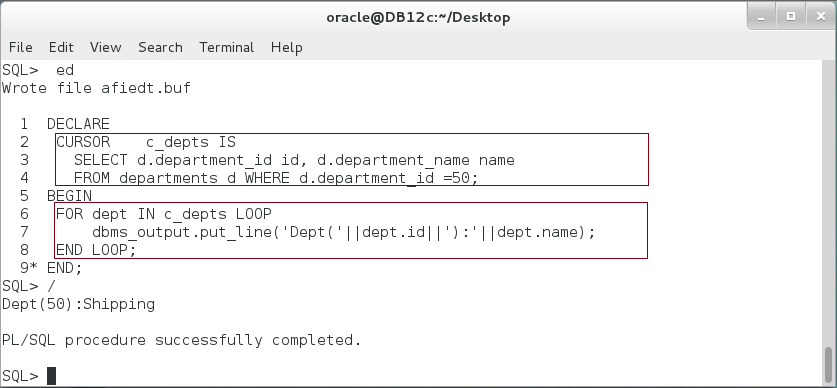
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| c\_refcur SYS\_REFCURSOR; |  |
| c\_id PLS\_INTEGER; |  |
| c\_name VARCHAR2(50); |  |
| v\_sql VARCHAR2(400); |  |
| BEGIN |  |
| v\_sql := 'SELECT d.department\_id, d.department\_name | ":x" is used as variable placeholder |
| FROM departments d WHERE d.department\_id **=:x'**; |
| OPEN c\_refcur FOR v\_sql **USING** **50**; | Binding 50 to variable placeholder **:x** |
| LOOP |  |
| FETCH c\_refcur INTO c\_id, c\_name; |  |
| EXIT WHEN c\_refcur%NOTFOUND; |  |
| dbms\_output.put\_line('Dept('||c\_id||'):'||c\_name); |  |
| END LOOP; |  |
| CLOSE c\_refcur; |  |
| END; |  |
| / |  |



## Explicit FOR LOOP Cursor

**Step 1:** The last method used is an Explicit FOR LOOP cursor. It is quite similar to an implicit FOR LOOP cursor but its SQL statement defined outside FOR LOOP as show below:

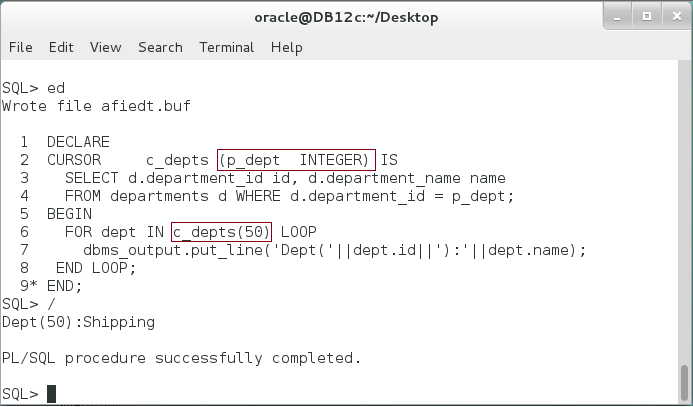
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| **CURSOR** c\_depts **IS** | Define a cursor first |
| SELECT d.department\_id id, d.department\_name name |
| FROM departments d WHERE d.department\_id =50; |
| BEGIN |  |
| **FOR** dept **IN** c\_depts **LOOP** | Use a pre-defined cursor in a **FOR LOOP** |
| dbms\_output.put\_line('Dept('||dept.id||'):'||dept.name); |
| **END LOOP;** |
| END; |  |

****

**Please note:** Explicit FOR LOOP cursor automatically open, fetch and close a cursor. However, it does not automatically declare and define a cursor as in the implicit FOR LOOP cursor.

**Step 2:** Again, Oracle recommends referring variables by parameters not by handwriting them inside the query. Modify the previous block as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| CURSOR **c\_depts (p\_dept INTEGER )** IS | Define a cursor with parameter |
| SELECT d.department\_id id, d.department\_name name |
| FROM departments d WHERE d.department\_id = p\_dept; |
| BEGIN |  |
| FOR dept IN **c\_depts(50)** LOOP | Use the cursor with parameter in a **FOR LOOP** |
| dbms\_output.put\_line('Dept('||dept.id||'):'||dept.name); |
| END LOOP; |
| END; |  |

****

# SUMMARY

Explicit cursor is more complex than its counterpart: implicit cursor. However, it is more flexible. A regular explicit cursor should follow 5 steps: Declaration, Definition, Opening, Fetching, and Closing. Declaration and Definition are usually done at the same time. Typically, fetching step is done repeatedly using LOOP statement. To avoid entering in an infinite LOOP, you should use one of a cursor attributes such as %NOTFOUND. Be careful when using these attributes, their positions matter. There are some special kinds of explicit cursors such as cursor variable and explicit FOR LOOP cursor. A cursor variable (REF CURSOR, SYS\_REFCURSOR) has an additional features such as the ability to use the same cursor for multiple queries and the ability to pass it as function/procedure parameter. Explicit FOR LOOP cursor is very similar implicit cursor except it declares and defines the cursor outside the FOR LOOP. In general, explicit cursors are slower than implicit cursors. It is logical not use explicit cursor unless you need more flexibility which cannot be done using implicit cursors.

After completing this lab exercise, you should be able to retrieve and manipulate data using explicit cursors.

# REFERENCES

* http://oracle-base.com/articles/misc/implicit-vs-explicit-cursors-in-oracle-plsql.php
* http://docs.oracle.com/database/121/LNPLS/static.htm#LNPLS00605

# INDEX

cursor variable 3, 25

explicit cursor 2, 3, 16, 17, 19, 25

Explicit Cursor FOR LOOP 3

implicit cursor 2, 3, 25