Table of Contents

[CHAPTER 5: IMPLICIT DATA RETRIEVAL AND MANIPULATION 2](#_Toc412052601)

[Theory 2](#_Toc412052602)

[AIM 4](#_Toc412052603)

[Lab Exercise 5: IMPLICIT DATA RETRIEVAL & MANIPULATION 5](#_Toc412052604)

[1. Invoke SELECT Statement 6](#_Toc412052605)

[2. Invoke DML Statements 9](#_Toc412052606)

[3. Implicit Cursor Attribute 13](#_Toc412052607)

[4. Implicit FOR LOOP 20](#_Toc412052608)

[SUMMARY 24](#_Toc412052609)

[REFERENCES 25](#_Toc412052610)

[INDEX 26](#_Toc412052611)

# CHAPTER 5: IMPLICIT DATA RETRIEVAL AND MANIPULATION

## Theory

One of the interesting features of PL/SQL block is its simplicity in using SQL commands. In comparison to other programming languages such as C# and JAVA, PL/SQL is much easier and fully integrated with SQL commands. For example, most of Oracle SQL syntaxes are available exactly in PL/SQL. Thus, PL/SQL can use SELECT, UPDATE, DELETE, and INSERT syntaxes in PL/SQL. This is one of the greatest features of PL/SQL.

To run queries or Data Manipulation Language (DML) statements, PL/SQL needs an area in memory to -temporarily- process and store the information retrieved from the database. This memory area is usually referenced by a pointer called "cursor". It is important to recognize two kinds of cursors: (1) implicit and (2) explicit. The implicit cursor is defined, parsed, opened, and closed automatically by PL/SQL. Oracle opens an implicit cursor every time the code runs SELECT or DML statements. The explicit cursor, on the other hand, is defined, parsed, opened, and closed by programmer.

For every session connected to the database, Oracle reserves a memory area called session cursor. The session cursor lives in a session memory until the session ends. You cannot control an implicit cursor, but you can get information from its attributes. These attributes always refer to the most recent SELECT or DML statement.

|  |  |
| --- | --- |
| Attribute Name | Attribute Explanation |
| SQL%ISOPEN | Is the Cursor Open? |
| SQL%FOUND | Were Any Rows Affected? |
| SQL%NOTFOUND | Were No Rows Affected? |
| SQL%ROWCOUNT | How Many Rows Were Affected? |
| SQL%BULK\_ROWCOUNT | Getting Number of Rows Affected by FORALL Statement |
| SQL%BULK\_EXCEPTIONS | Handling FORALL Exceptions After FORALL Statement Completes" |

The cursor FOR LOOP statement lets you run a SELECT statement and then immediately loop through the rows of the result set. This statement can use either an implicit or explicit cursor. In an implicit cursor FOR LOOP statement, you specify the SELECT statement inside the cursor FOR LOOP statement. The cursor FOR LOOP statement implicitly declares its loop index as a %ROWTYPE record variable of the type that its cursor returns. This record is local to the loop and exists only during loop execution. In other words, the scope of this record starts from FOR keyword and ends at END LOOP keywords. Statements inside the loop can reference the record and its fields; outside the loop, the record is not defined.

## AIM

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

The steps involved will include:

* Invoke SELECT Statement
* Invoke DML Statements
* Implicit Cursor Attribute
* Implicit 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.

Estimated Completion Time:

30 minutes

# Lab Exercise 5: IMPLICIT DATA RETRIEVAL & MANIPULATION

|  |
| --- |
|  |

## Invoke SELECT Statement

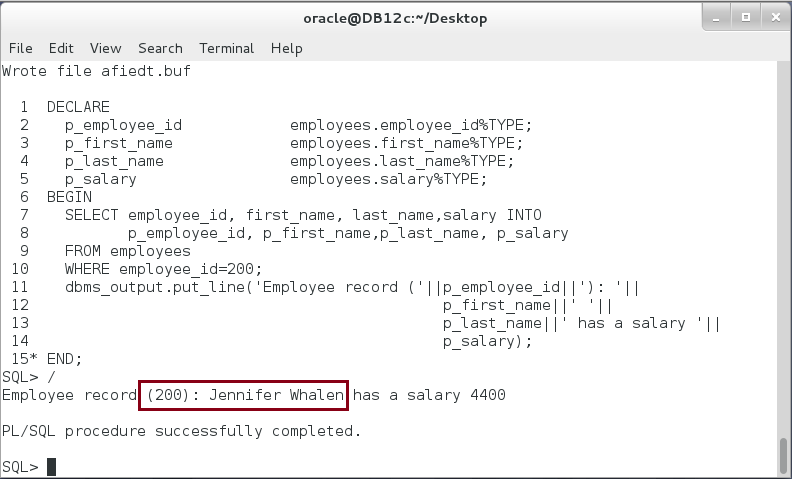
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

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

****

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

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_employee\_id employees.employee\_id%TYPE; | Define variables as table columns |
| p\_first\_name employees.first\_name%TYPE; |
| p\_last\_name employees.last\_name%TYPE; |
| p\_salary employees.salary%TYPE; |
| BEGIN |  |
| SELECT employee\_id, first\_name, last\_name,salary INTO | Retrieve data using implicit SELECT statement |
| p\_employee\_id, p\_first\_name,p\_last\_name, p\_salary |
| FROM employees |
| WHERE employee\_id=200; |
| dbms\_output.put\_line('Employee record ('||p\_employee\_id||'): '|| |  |
| p\_first\_name||' '|| |  |
| p\_last\_name||' has a salary '|| |  |
| p\_salary); |  |
| END; |  |
| / |  |

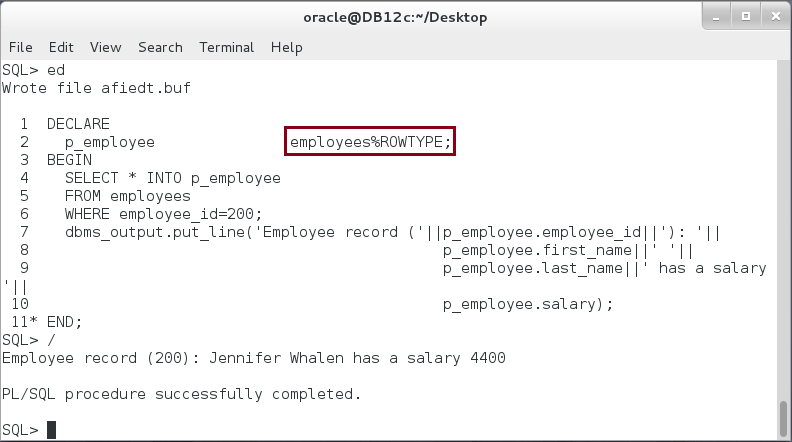
****

**Please note**: In the previous PL/SQL block, one variable was defined for each column: EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, and SALARY. In some cases, however, it is difficult to list all variables in a table.

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

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_employee employees%ROWTYPE; | **Table record data type.** |
| BEGIN |  |
| SELECT \* INTO p\_employee | Fetch the entire table columns. |
| FROM employees |
| WHERE employee\_id=200; |
| dbms\_output.put\_line('Employee record ('||p\_employee.employee\_id||'): '|| | Note who to use a Table Record variable. |
| p\_employee.first\_name||' '|| |
| p\_employee.last\_name||  ' has a salary '||p\_employee.salary); |
| END; |
| / |

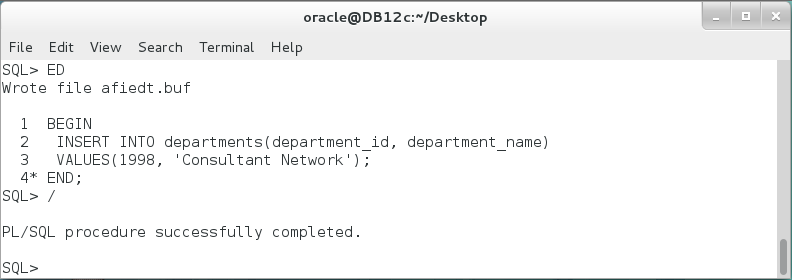
In this block, one variable was used to collect all columns of the table.



## Invoke DML Statements

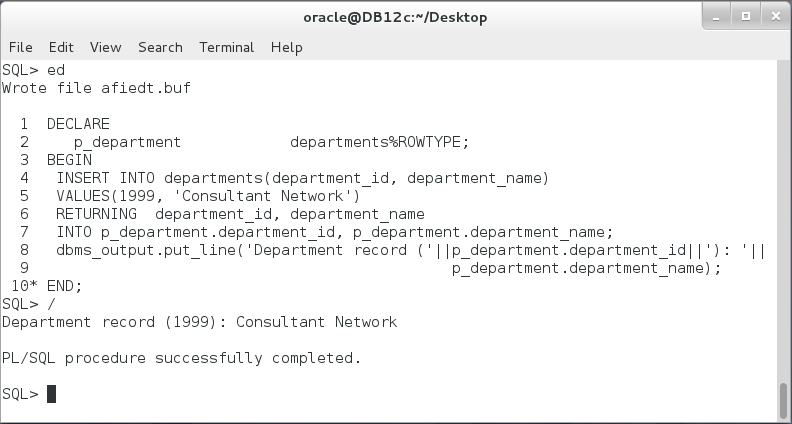
**Step 1:** Execute the following PL/SQL block:

|  |  |
| --- | --- |
| Line | Description |
| BEGIN |  |
| INSERT INTO departments(department\_id, department\_name)  VALUES(1998, 'Consultant Network'); | INSERT statement is exactly as SQL INSERT statement. |
| END; |  |
| / |  |



**Step 2:** Modify previous PL/SQL block as the following:

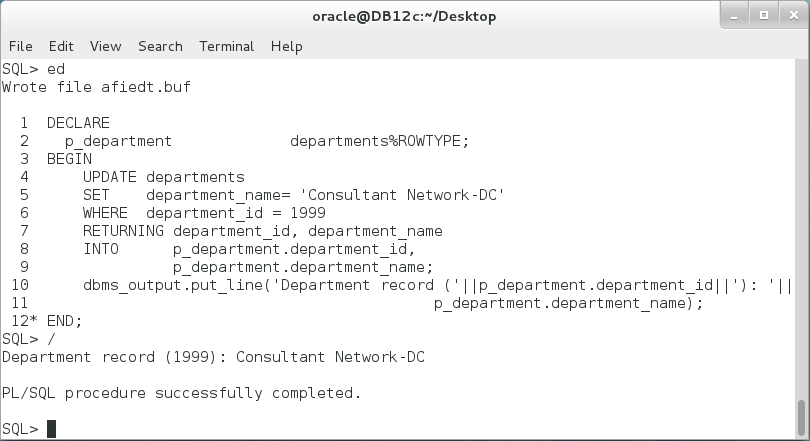
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| INSERT INTO departments(department\_id, department\_name)  VALUES(1999, 'Consultant Network')  **RETURNING department\_id, department\_name**  **INTO p\_department.department\_id, p\_department.department\_name;** | You can make use of RETURNING INTO clause in PL/SQL. |
| dbms\_output.put\_line('Department record ('||p\_department.department\_id||'): '|| p\_department.department\_name); |  |
| END; |  |
| / |  |



**Please note**: RETURNING INTO clause is optional.

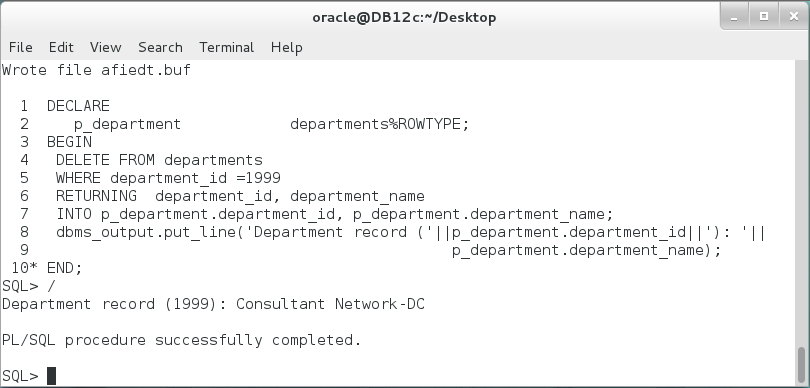
**Step 3:** UPDATE statement as INSERT statement, it can contain RETURNING INTO clause. Run the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| UPDATE departments | Think when RETURNING INTO clause would be useful. |
| SET department\_name= 'Consultant Network-DC' |
| WHERE department\_id = 1999 |
| RETURNING department\_id, department\_name |
| INTO p\_department.department\_id, |
| p\_department.department\_name; |
| dbms\_output.put\_line('Department record ('||p\_department.department\_id||'): '|| |  |
| p\_department.department\_name); |  |
| END; |  |
| / |  |

****

**Step 4:** Execute DELETE statement in the following block:

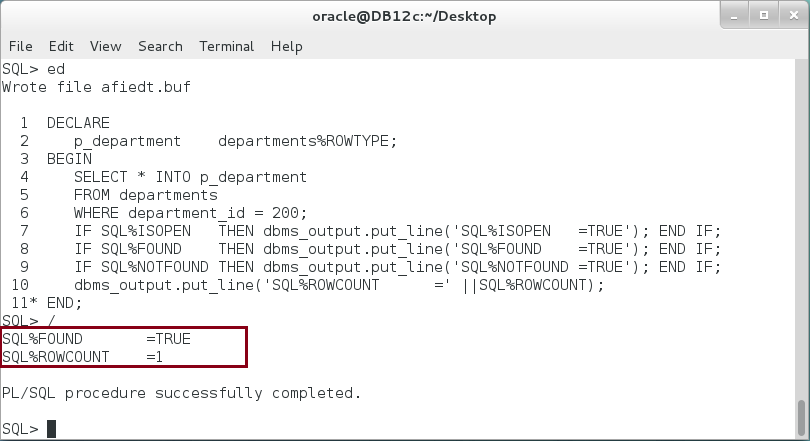
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| DELETE FROM departments  WHERE department\_id =1999  RETURNING department\_id, department\_name  INTO p\_department.department\_id, p\_department.department\_name; | DELETE statement with RETURNING INTO clause. Do you think this clause would be useful here? |
| dbms\_output.put\_line('Department record ('||p\_department.department\_id||'): '||  p\_department.department\_name); |  |
| END; |  |
| / |

****

## Implicit Cursor Attribute

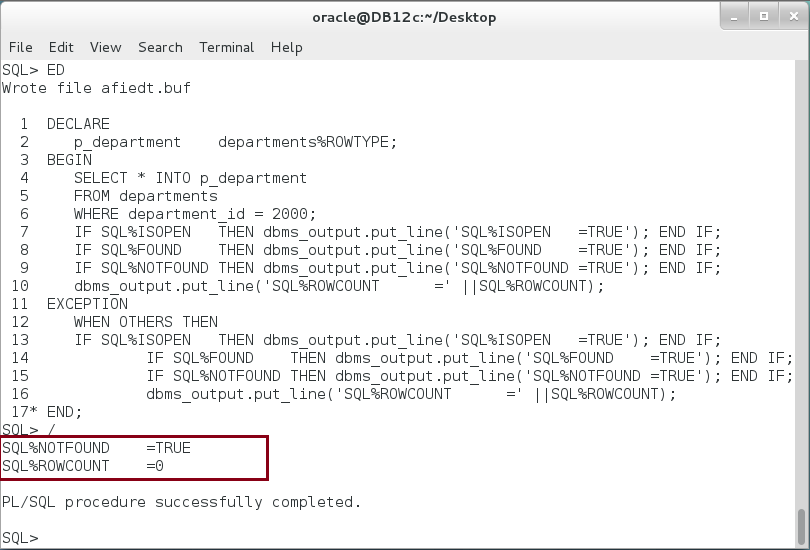
**Step 1:** In this section, you will examine the implicit cursor attributes. Most of these attributes are BOOLEAN variables. To show these variables, you may need IF statement. IF statement will be covered in the next chapters; please take it as it is now. Execute the following PL/SQL block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| SELECT \* INTO p\_department | Select **ONE** row from the table. |
| FROM departments |
| WHERE department\_id = 200; |
| IF SQL%ISOPEN THEN  dbms\_output.put\_line('SQL%ISOPEN =TRUE');  END IF; | **%ISOPEN = FALSE** because implicit cursor automatically closed after execution. |
| IF SQL%FOUND THEN  dbms\_output.put\_line('SQL%FOUND =TRUE');  END IF; | **%FOUND = TRUE** |
| IF SQL%NOTFOUND THEN  dbms\_output.put\_line('SQL%NOTFOUND =TRUE');  END IF; | **%NOTFOUND=FALSE** |
| dbms\_output.put\_line('SQL%ROWCOUNT =' ||SQL%ROWCOUNT); | **%ROWCOUNT= 1** |
| END; |  |
| / |  |

****

**Step 2:** Modify the previous block to retrieve no row as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| SELECT \* INTO p\_department | Select **ZERO** row from the table. |
| FROM departments |
| WHERE department\_id = 2000; |
| IF SQL%ISOPEN THEN dbms\_output.put\_line('SQL%ISOPEN=TRUE'); END IF; | **Never executed.** |
| IF SQL%FOUND THEN dbms\_output.put\_line('SQL%FOUND =TRUE'); END IF; |
| IF SQL%NOTFOUND THEN dbms\_output.put\_line('SQL%NOTFOUND=TRUE'); END IF; |
| dbms\_output.put\_line('SQL%ROWCOUNT =' ||SQL%ROWCOUNT); |
| **EXCEPTION** |  |
| **WHEN OTHERS THEN** |  |
| **IF SQL%ISOPEN THEN dbms\_output.put\_line('SQL%ISOPEN=TRUE'); END IF;** | **FALSE** |
| **IF SQL%FOUND THEN dbms\_output.put\_line('SQL%FOUND =TRUE'); END IF;** | **FALSE** |
| **IF SQL%NOTFOUND THEN dbms\_output.put\_line('SQL%NOTFOUND=TRUE'); END IF;** | **TRUE** |
| **dbms\_output.put\_line('SQL%ROWCOUNT =' ||SQL%ROWCOUNT);** | **0** |
| END; |  |

****

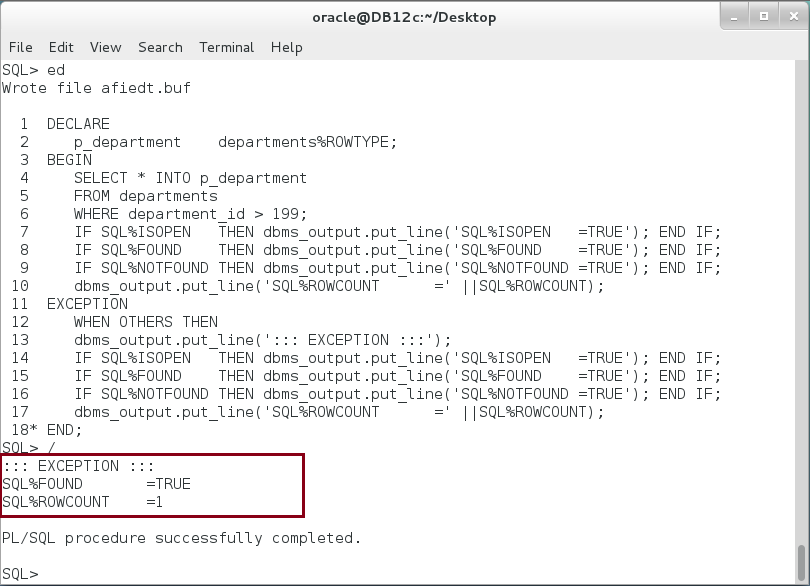
**Step 3:** Save the previous block; we will reuse it in the next steps. We will examine how many rows in DEPARTMENTS table where DEPARTMENT\_ID is greater than 199. Execute the following SQL statement right on SQL\*Plus console:

|  |  |
| --- | --- |
| Command | Description |
| SELECT count(\*) FROM departments WHERE department\_id > 199  / | There are **9** rows. |

****

**Step 4:** Return back to previous PL/SQL block, modify, and execute it**:**

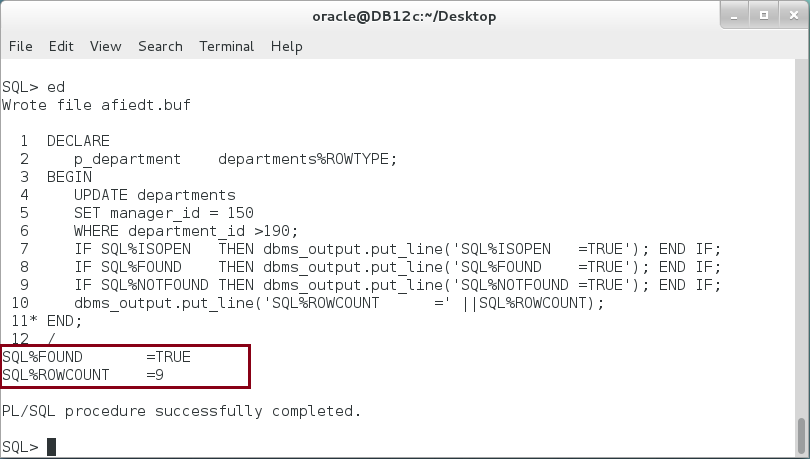
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| SELECT \* INTO p\_department | Select **9** rows from the table. |
| FROM departments |
| **WHERE department\_id > 199;** |
| IF SQL%ISOPEN THEN dbms\_output.put\_line('SQL%ISOPEN=TRUE'); END IF; | **Never executed.** |
| IF SQL%FOUND THEN dbms\_output.put\_line('SQL%FOUND =TRUE'); END IF; |
| IF SQL%NOTFOUND THEN dbms\_output.put\_line('SQL%NOTFOUND=TRUE'); END IF; |
| dbms\_output.put\_line('SQL%ROWCOUNT =' ||SQL%ROWCOUNT); |
| EXCEPTION |  |
| WHEN OTHERS THEN |  |
| IF SQL%ISOPEN THEN dbms\_output.put\_line('SQL%ISOPEN=TRUE'); END IF; | **FALSE** |
| IF SQL%FOUND THEN dbms\_output.put\_line('SQL%FOUND =TRUE'); END IF; | **TRUE** |
| IF SQL%NOTFOUND THEN dbms\_output.put\_line('SQL%NOTFOUND=TRUE'); END IF; | **FALSE** |
| dbms\_output.put\_line('SQL%ROWCOUNT =' ||SQL%ROWCOUNT); | **(1): WHY?** |
| END; |  |

****

**Please note:** If a SELECT INTO statement without a BULK COLLECT clause returns multiple rows, PL/SQL raises the predefined exception TOO\_MANY\_ROWS and **SQL%ROWCOUNT returns 1**, not the actual number of rows that satisfy the query.

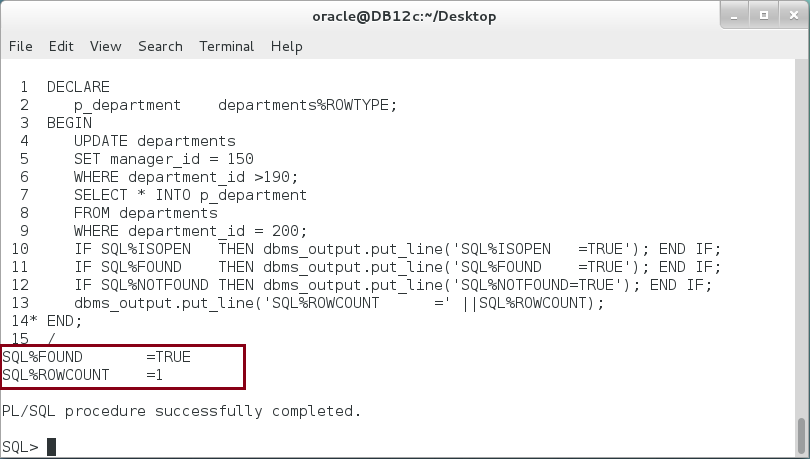
**Step 5:** INSERT, UPDATE, and DELETE statements with RETURNINING INTO clause follow the same rules as SELECT INTO statement. But DML statements without RETURNINGING INTO are quite different; they do not raise exceptions when no row or multiple rows affected by the query. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| UPDATE departments  SET manager\_id = 150  WHERE department\_id >190; | Update **9** rows. |
| IF SQL%ISOPEN THEN dbms\_output.put\_line('SQL%ISOPEN =TRUE'); END IF; | **FALSE** |
| IF SQL%FOUND THEN dbms\_output.put\_line('SQL%FOUND =TRUE'); END IF; | **TRUE** |
| IF SQL%NOTFOUND THEN dbms\_output.put\_line('SQL%NOTFOUND=TRUE'); END IF; | **FALSE** |
| dbms\_output.put\_line('SQL%ROWCOUNT =' ||SQL%ROWCOUNT); | **9** |
| END; |  |
| / |  |



**Step 6:** Modify the previous block to add a SELECT statement after the UPDATE statement as the following:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| p\_department departments%ROWTYPE; |  |
| BEGIN |  |
| UPDATE departments  SET manager\_id = 150  WHERE department\_id >190; | Update **9** rows. |
| SELECT \* INTO p\_department | Select **ONE** row. |
| FROM departments |
| WHERE department\_id = 200; |
| IF SQL%ISOPEN THEN dbms\_output.put\_line('SQL%ISOPEN =TRUE'); END IF; | **FALSE** |
| IF SQL%FOUND THEN dbms\_output.put\_line('SQL%FOUND =TRUE'); END IF; | **TRUE** |
| IF SQL%NOTFOUND THEN dbms\_output.put\_line('SQL%NOTFOUND=TRUE'); END IF; | **FALSE** |
| dbms\_output.put\_line('SQL%ROWCOUNT =' ||SQL%ROWCOUNT); | **(1): WHY?** |
| END; |  |
| / |  |



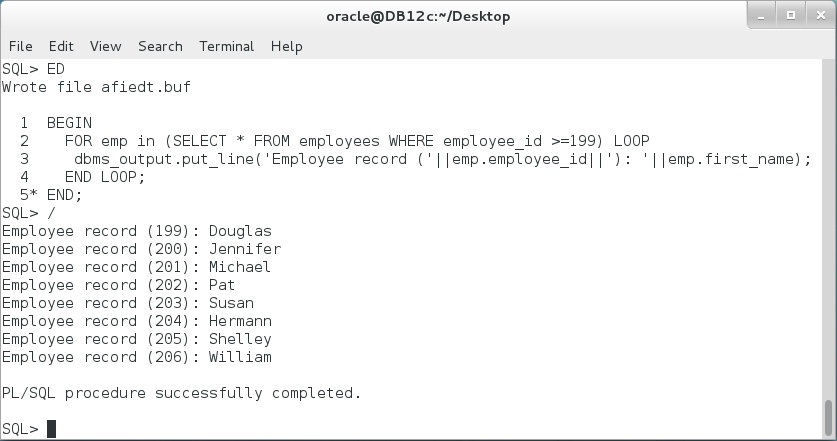
**Please note:**

* In the previous block, only the attributes of the last statement executed (SELECT statement) appears on session cursor's attributes.
* In all above situation (Step 1 - 6), SQL%ISOPEN is FALSE. Implicit cursors are automatically closed after execution.

## Implicit FOR LOOP

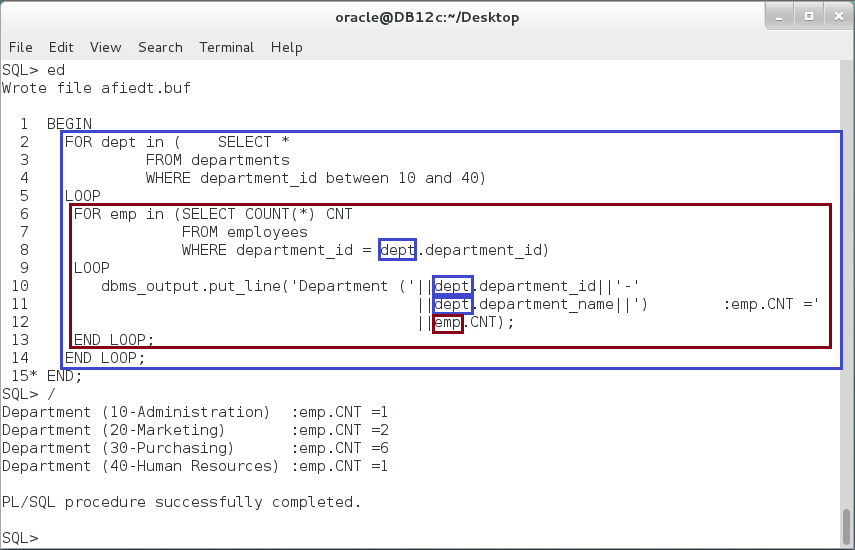
**Step 1:** Using SELECT INTO clause, you can fetch exactly one row but more than that. To fetch more one row in PL/SQL block, you can use implicit FOR LOOP statement as shown below:

|  |  |
| --- | --- |
| Command | Description |
| BEGIN |  |
| FOR **emp** in (SELECT \*  FROM employees  WHERE employee\_id >=199)  LOOP | **emp** is an **implicit record** of employees table |
| dbms\_output.put\_line('Employee record ('||  emp.employee\_id||'): '||emp.first\_name); | The **scope** of emp variable is just in this area. |
| END LOOP; |  |
| END; |  |
| / |  |



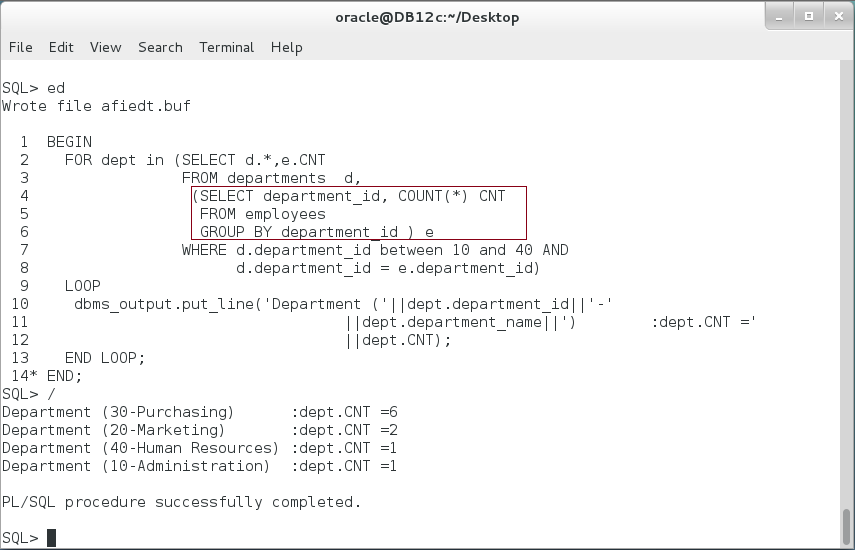
**Step 2:** You can use a nested FOR LOOP cursor as show below:

|  |  |
| --- | --- |
| Command | Description |
| BEGIN |  |
| FOR dept in (SELECT \* | Outer FOR LOOP. |
| FROM departments |
| WHERE department\_id between 10 and 40) |
| LOOP |
| FOR emp in (SELECT COUNT(\*) CNT | Inner FOR LOOP. |
| FROM employees |
| WHERE department\_id = dept.department\_id) |
| LOOP |
| dbms\_output.put\_line('Department ('||**dept**.department\_id||'-' | Note the **outer** and **inner** records. |
| ||**dept**.department\_name||') :emp.CNT =' |
| ||**emp**.CNT); |
| END LOOP; | End Inner LOOP |
| END LOOP; | End Outer LOOP |
| END; |  |
| / |  |



**Step 3:** Oracle highly recommend not using nested implicit FOR LOOP cursors when you can remove one. In the previous block, for example, the inner SELECT statement can be joined with the outer SELECT. The block should look like the following:

|  |  |
| --- | --- |
| Command | Description |
| BEGIN |  |
| FOR dept in (SELECT d.\*,e.CNT |  |
| FROM departments d, |  |
| **(SELECT department\_id, COUNT(\*) CNT** | Use Subquery instead of INNER LOOP. |
| **FROM employees** |
| **GROUP BY department\_id ) e** |
| WHERE d.department\_id between 10 and 40 AND |  |
| d.department\_id = e.department\_id) |  |
| LOOP |  |
| dbms\_output.put\_line('Department ('||dept.department\_id||'-' |  |
| ||dept.department\_name||') :dept.CNT =' |  |
| ||dept.CNT); |  |
| END LOOP; |  |
| END; |  |
| / |  |



**Please note:** Subquery is faster than using an inner LOOP.

# SUMMARY

PL/SQL is completely coherent with SQL syntax. Data retrieval is much easier than other programming languages. One way in which you can access/modify table data is using implicit cursor. A normal syntax of SELECT, INSERT,UPDATE, and DELETE statements can be used. However, a tiny addition has been added to specify how to handle data in PL/SQL. For example, INTO clause is used in SELECT statement; RETURNING INTO clause may be used in INSERT, UPDATE, and DELETE statements. SELECT INTO and DML with RETURNING INTO clauses are used when exactly one row affected by the operation. Oracle stores the attributes of the last SELECT or DML statement into a session cursor. You can access these attributes in a PL/SQL block. SQL%ISOPEN attribute is always FALSE for implicit cursors; SQL%FOUND and SQL%NOTFOUND attributes are used to determine if any rows or no rows were affected; SQL%ROWCOUNT attribute lists the number of rows affected. The implicit FOR LOOP cursor is used to retrieve multiple rows and loop through them. Implicit FOR LOOP opens and closes the cursor automatically. It also automatically defines a variable or a record which its scope starts and ends in the FOR LOOP.

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

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/static.htm#LNPLS99878
* http://docs.oracle.com/database/121/LNPLS/static.htm#LNPLS00603

# INDEX

cursor 2, 12, 17, 18, 21

DML 2, 4, 8, 16, 21

explicit cursor 2

implicit cursor 2

session cursor 2, 21