**Security Credentials**

### Security Credentials

#### username: ora41 password: ora41

.

# Practices for Lesson 1:

## Introduction

**Chapter 1**

### Practices for Lesson 1

#### Lesson Overview

In these practices, you perform the following:

* Start SQL Developer
* Create a new database connection
* Browse the schema tables
* Set a SQL Developer preference

**Note:** All written practices use SQL Developer as the development environment. Although it is recommended that you use SQL Developer, you can also use the SQL\*Plus environment that is available in this course.

### Practice 1-1: Getting Started

1. Start SQL Developer.

Double Click the SQL Developer icon on your desktop.

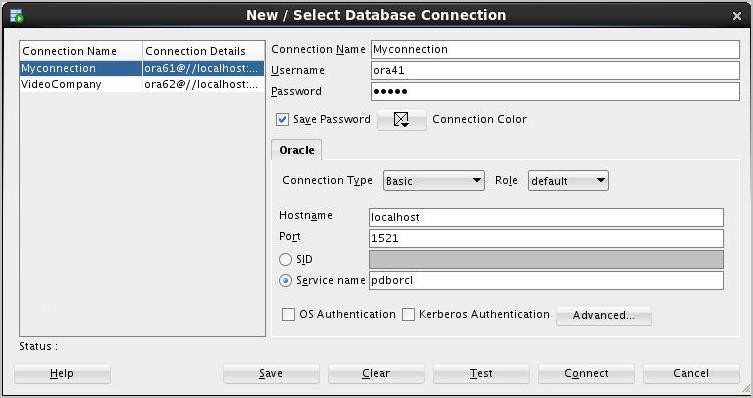


1. Create a database connection by using the following information (**Hint:** Select the Save Password check box):
2. Connection Name: MyConnection
3. Username: ora41
4. Password: ora41
5. Hostname: localhost
6. Port: 1521
7. Service name : pdborcl

Right-click the Connections node on the Connections tabbed page and select **New Connection.**

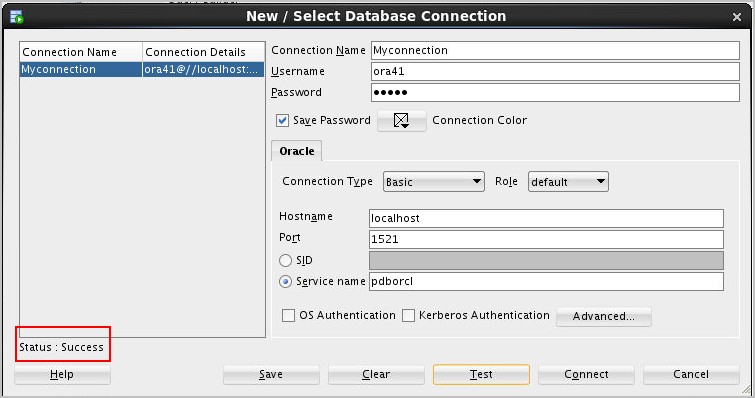
Result: The New/Select Database Connection window appears.

Use the preceding information to create the new database connection. In addition, select the Save Password check box. For example:

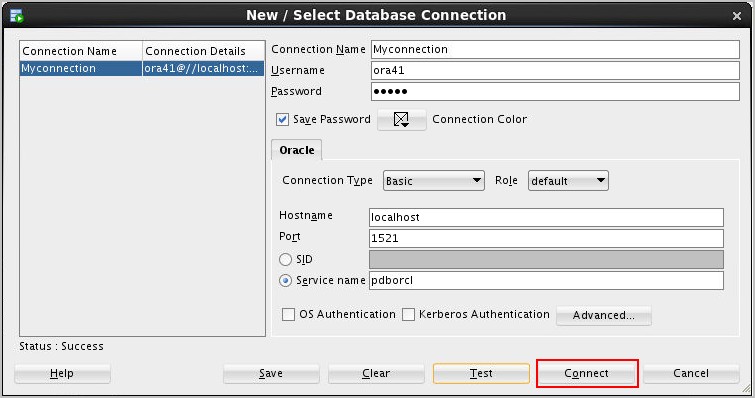


1. Test the new connection. If the Status is Success, connect to the database by using this new connection.
2. In the Database Connection window, click the Test button.

**Note:** The connection status appears in the lower-left corner of the window.



1. If the Status is Success, click the Connect button.



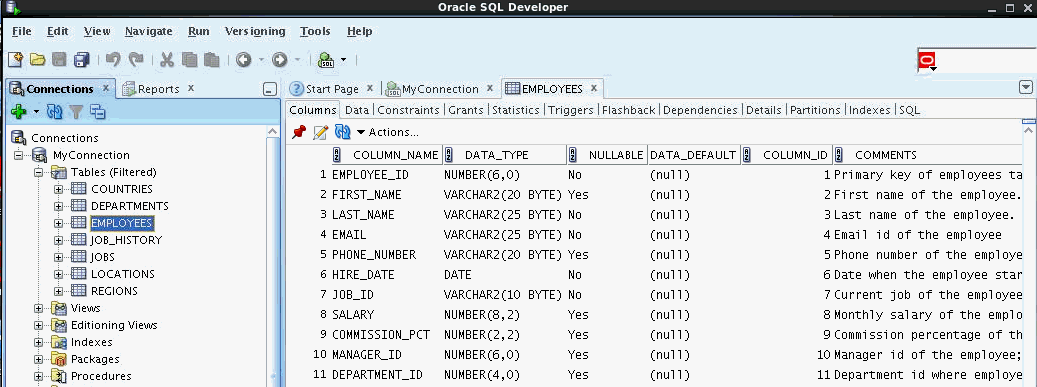
**Note:** To display the properties of an existing connection, right-click the connection name on the Connections tab and select Properties from the shortcut menu.

1. Browse the structure of the EMPLOYEES table and display its data.

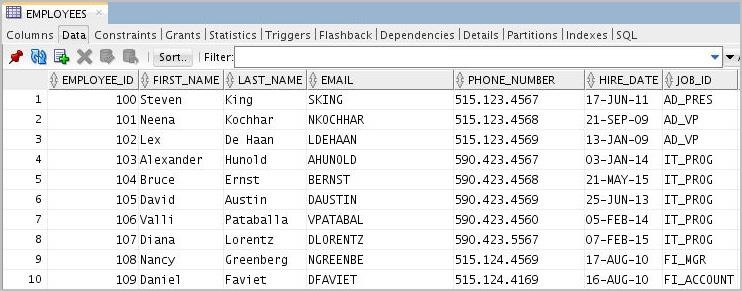
.

* 1. Expand the MyConnection connection by clicking the plus symbol next to it.
  2. Expand Tables by clicking the plus symbol next to it.
  3. Display the structure of the EMPLOYEES table.
     + Drill down on the EMPLOYEES table by clicking the plus symbol next to it.
     + Click the EMPLOYEES table.

Result: The Columns tab displays the columns in the EMPLOYEES table as follows:



1. Use the Data tab to view the data in the EMPLOYEES table. Result: The EMPLOYEES table data is displayed as follows:



1. Use the SQL Worksheet to select the last names and salaries of all employees whose annual salary is greater than $10,000. Use both the Execute Statement (F9) and Run Script (F5) icons to execute the SELECT statement. Review the results of both methods of executing the SELECT statements on the appropriate tabs.

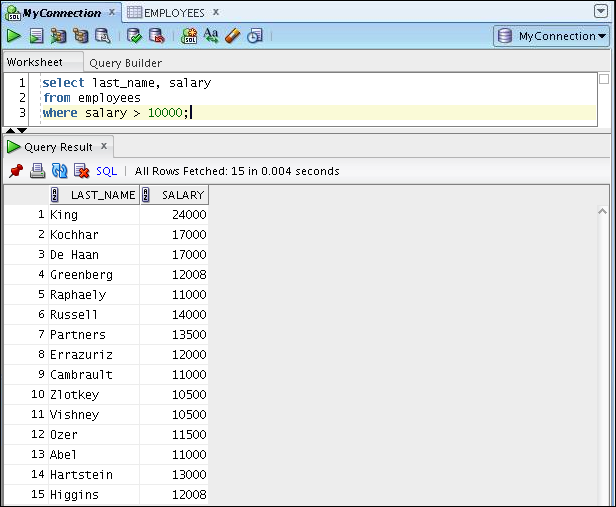
**Note:** Take a few minutes to familiarize yourself with the data, or consult Appendix A, which provides the description and data for all the tables in the HR schema that you will use in this course.

To display the SQL Worksheet, click the MyConnection tab.

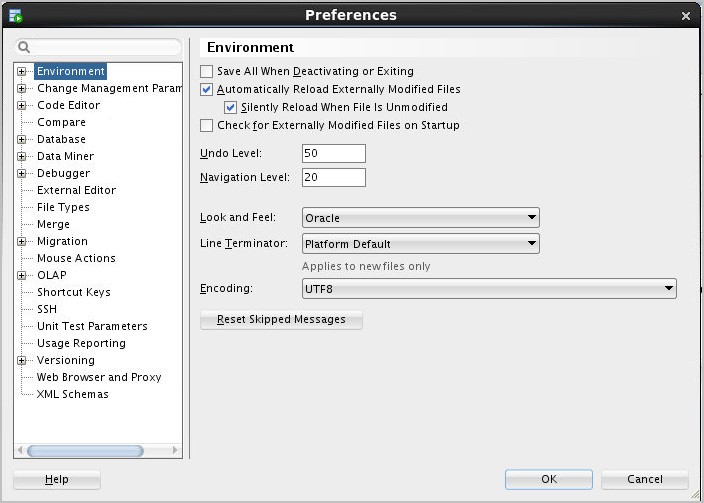
**Note:** This tab was opened previously when you drilled down on your database connection. Enter the appropriate SELECT statement. Press F9 to execute the query and F5 to execute the

query by using the Run Script method.

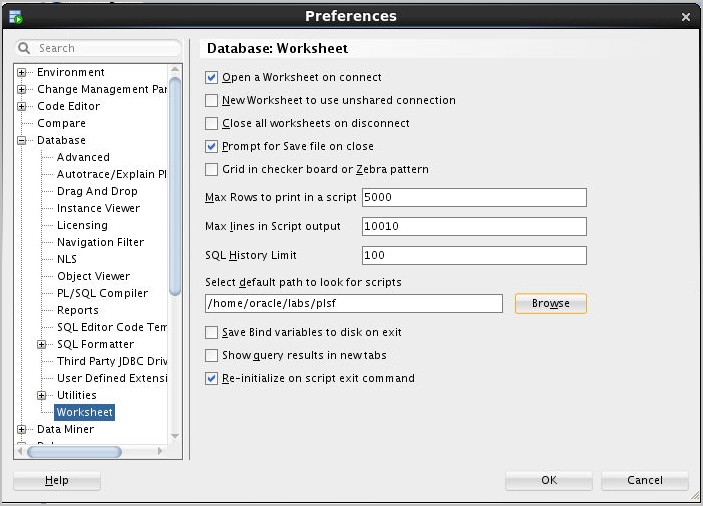
For example, when you press F9, the results appear similar to the following:



1. From the SQL Developer menu, select Tools > Preferences. The Preferences window appears.



Then, in the Preferences window, click OK to save the Worksheet Parameter setting.



# Practices for Lesson 2: Introduction to PL/SQL Chapter 2

**Practice 2: Introduction to PL/SQL**

1. Which of the following PL/SQL blocks execute successfully?
2. BEGIN commit; END;
3. DECLARE

v\_amount INTEGER(10); END;

1. DECLARE BEGIN

END;

1. SET SERVEROUTPUT ON; DECLARE

v\_amount INTEGER(10); BEGIN

DBMS\_OUTPUT.PUT\_LINE(v\_amount); END;

1. Create and execute a simple anonymous block that outputs “Hello World.” Execute and save this script as lab\_02\_02\_soln.sql.

### Solution 2: Introduction to PL/SQL

1. Which of the following PL/SQL blocks execute successfully?
2. BEGIN commit; END;
3. DECLARE

v\_amount INTEGER(10); END;

1. DECLARE BEGIN

END;

1. SET SERVEROUTPUT ON; DECLARE

v\_amount INTEGER(10); BEGIN

DBMS\_OUTPUT.PUT\_LINE(v\_amount); END;

##### The block in a executes successfully.

The block in b does not have the mandatory executable section that starts with the BEGIN

keyword.

The block in c has all the necessary parts, but no executable statements.

##### The block in d executes successfully.

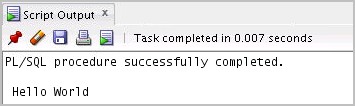
1. Create and execute a simple anonymous block that outputs “Hello World.” Execute and save this script as lab\_02\_02\_soln.sql.

Enter the following code in the workspace, and then press F5.

SET SERVEROUTPUT ON BEGIN

DBMS\_OUTPUT.PUT\_LINE(' Hello World '); END;

You should see the following output on the Script Output tab:



Click the Save button. Select the folder in which you want to save the file. Enter

lab\_02\_02\_soln.sql as the file name and click Save.

# Practices for Lesson 3: Declaring PL/SQL Variables Chapter 3

.

### Practice 3: Declaring PL/SQL Variables

In this practice, you declare PL/SQL variables.

1. Identify valid and invalid identifiers:
2. today
3. last\_name
4. today’s\_date
5. Number\_of\_days\_in\_February\_this\_year
6. Isleap$year
7. #number
8. NUMBER#
9. number1to7
10. Identify valid and invalid variable declaration and initialization:
11. number\_of\_copies PLS\_INTEGER;
12. PRINTER\_NAME constant VARCHAR2(10);
13. deliver\_to VARCHAR2(10):=Johnson;
14. by\_when DATE:= CURRENT\_DATE+1;
15. Examine the following anonymous block, and then select a statement from the following that is true.

DECLARE

v\_fname VARCHAR2(20);

v\_lname VARCHAR2(15) DEFAULT 'fernandez'; BEGIN

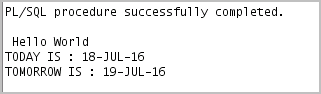
DBMS\_OUTPUT.PUT\_LINE(v\_fname ||' ' ||v\_lname); END;

1. The block executes successfully and prints “fernandez.”
2. The block produces an error because the fname variable is used without initializing.
3. The block executes successfully and prints “null fernandez.”
4. The block produces an error because you cannot use the DEFAULT keyword to initialize a variable of type VARCHAR2.
5. The block produces an error because the v\_fname variable is not declared.

.

1. Modify an existing anonymous block and save it as a new script.
2. Open the lab\_02\_02\_soln.sql script, which you created in Practice 2 titled “Introduction to PL/SQL.”
3. In this PL/SQL block, declare the following variables:
4. v\_today of type DATE. Initialize today with SYSDATE.
5. v\_tomorrow of type today. Use the %TYPE attribute to declare this variable.
6. In the executable section:
7. Initialize the v\_tomorrow variable with an expression, which calculates tomorrow’s date (add one to the value in today)
8. Print the value of v\_today and v\_tomorrow after printing “Hello World”
9. Save your script as lab\_03\_04\_soln.sql, and then execute.

The sample output is as follows (the values of v\_today and v\_tomorrow will be different to reflect your current today’s and tomorrow’s date):



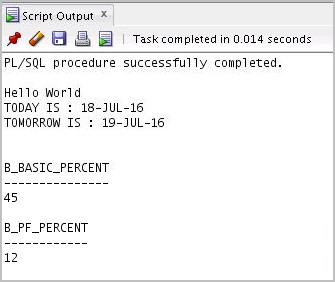
1. Edit the lab\_03\_04\_soln.sql script.
   1. Add code to create two bind variables named b\_basic\_percent and

b\_pf\_percent. Both bind variables are of type NUMBER.

* 1. In the executable section of the PL/SQL block, assign the values 45 and 12 to

b\_basic\_percent and b\_pf\_percent, respectively.

* 1. Terminate the PL/SQL block with “/” and display the value of the bind variables by using the PRINT command.
  2. Execute and save your script as lab\_03\_05\_soln.sql. The sample output is as follows:



### Solution 3: Declaring PL/SQL Variables

1. Identify valid and invalid identifiers:
   1. today **Valid**
   2. last\_name **Valid**
   3. today’s\_date **Invalid** – character “’” not allowed
   4. Number\_of\_days\_in\_February\_this\_year **Invalid** – Too long
   5. Isleap$year **Valid**
   6. #number **Invalid –** Cannot start with “#”
   7. NUMBER# **Valid**
   8. number1to7 **Valid**
2. Identify valid and invalid variable declaration and initialization:
3. number\_of\_copies PLS\_INTEGER; **Valid**
4. PRINTER\_NAME constant VARCHAR2(10); **Invalid**
5. deliver\_to VARCHAR2(10):=Johnson; **Invalid**
6. by\_when DATE:= CURRENT\_DATE+1; **Valid**

*The declaration in* ***b*** *is invalid because constant variables must be initialized during declaration.*

*The declaration in* ***c*** *is invalid because string literals should be enclosed within single quotation*

*marks.*

1. Examine the following anonymous block, and then select a statement from the following that is true.

DECLARE

v\_fname VARCHAR2(20);

v\_lname VARCHAR2(15) DEFAULT 'fernandez'; BEGIN

DBMS\_OUTPUT.PUT\_LINE(v\_fname ||' ' ||v\_lname); END;

1. The block executes successfully and prints “fernandez.”
2. The block produces an error because the fname variable is used without initializing.
3. The block executes successfully and prints “null fernandez.”
4. The block produces an error because you cannot use the DEFAULT keyword to initialize a variable of type VARCHAR2.
5. The block produces an error because the v\_fname variable is not declared.

##### The block will execute successfully and print “fernandez.”

1. Modify an existing anonymous block and save it as a new script.
2. Open the lab\_02\_02\_soln.sql script, which you created in Practice 2 titled “Introduction to PL/SQL.”
3. In the PL/SQL block, declare the following variables:
4. Variable v\_today of type DATE. Initialize today with SYSDATE. DECLARE

v\_today DATE:=SYSDATE;

1. Variable v\_tomorrow of type today. Use the %TYPE attribute to declare this variable.

v\_tomorrow v\_today%TYPE;

In the executable section:

1. Initialize the v\_tomorrow variable with an expression, which calculates tomorrow’s date (add one to the value in v\_today)
2. Print the value of v\_today and v\_tomorrow after printing “Hello World”

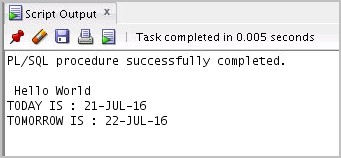
BEGIN

v\_tomorrow:=v\_today +1; DBMS\_OUTPUT.PUT\_LINE(' Hello World ');

DBMS\_OUTPUT.PUT\_LINE('TODAY IS : '|| v\_today); DBMS\_OUTPUT.PUT\_LINE('TOMORROW IS : ' || v\_tomorrow); END;

1. Save your script as lab\_03\_04\_soln.sql, and then execute.

The sample output is as follows (the values of v\_today and v\_tomorrow will be different to reflect your current today’s and tomorrow’s date):



.

1. Edit the lab\_03\_04\_soln.sql script.
   1. Add code to create two bind variables named b\_basic\_percent and

b\_pf\_percent. Both bind variables are of type NUMBER. VARIABLE b\_basic\_percent NUMBER

VARIABLE b\_pf\_percent NUMBER

* 1. In the executable section of the PL/SQL block, assign the values 45 and 12 to

b\_basic\_percent and b\_pf\_percent, respectively.

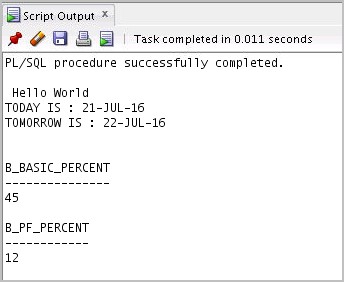
:b\_basic\_percent:=45;

:b\_pf\_percent:=12;

* 1. Terminate the PL/SQL block with “/” and display the value of the bind variables by using the PRINT command.

/

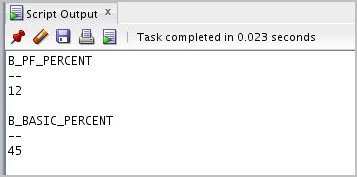
PRINT b\_basic\_percent PRINT b\_pf\_percent



OR

PRINT

* 1. Execute and save your script as lab\_03\_05\_soln.sql. The sample output is as follows:



# Practices for Lesson 4: Writing Executable Statements

**Chapter 4**

### Practice 4: Writing Executable Statements

**Note:** If you have executed the code examples for this lesson, make sure that you execute the

following code before starting this practice:

DROP sequence my\_seq;

In this practice, you examine and write executable statements.

DECLARE

v\_weight NUMBER(3) := 600;

v\_message VARCHAR2(255) := 'Product 10012'; BEGIN

DECLARE

v\_weight NUMBER(3) := 1;

v\_message VARCHAR2(255) := 'Product 11001'; v\_new\_locn VARCHAR2(50) := 'Europe';

BEGIN

v\_weight := v\_weight + 1;

v\_new\_locn := 'Western ' || v\_new\_locn; 1

END;

v\_weight := v\_weight + 1;

v\_message := v\_message || ' is in stock'; v\_new\_locn := 'Western ' || v\_new\_locn;

2

END;

/

* + 1. Evaluate the preceding PL/SQL block and determine the data type and value of each of the following variables, according to the rules of scoping.

1. The value of v\_weight at position 1 is:
2. The value of v\_new\_locn at position 1 is:
3. The value of v\_weight at position 2 is:
4. The value of v\_message at position 2 is:
5. The value of v\_new\_locn at position 2 is:

DECLARE

v\_customer VARCHAR2(50) := 'Womansport'; v\_credit\_rating VARCHAR2(50) := 'EXCELLENT'; BEGIN

DECLARE

v\_customer NUMBER(7) := 201;

v\_name VARCHAR2(25) := 'Unisports'; BEGIN

v\_credit\_rating :='GOOD';

… END;

… END;

* + 1. In the preceding PL/SQL block, determine the value and data type of each of the following cases:

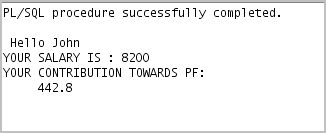
1. The value of v\_customer in the nested block is:
2. The value of v\_name in the nested block is:
3. The value of v\_credit\_rating in the nested block is:
4. The value of v\_customer in the main block is:
5. The value of v\_name in the main block is:
6. The value of v\_credit\_rating in the main block is:
   * 1. Use the same session that you used to execute the practices in the lesson titled “Declaring PL/SQL Variables.” If you have opened a new session, execute lab\_03\_05\_soln.sql. Then, edit lab\_03\_05\_soln.sql as follows:
        1. Use single-line comment syntax to comment the lines that create the bind variables, and turn on SERVEROUTPUT.
        2. Use multiple-line comments in the executable section to comment the lines that assign values to the bind variables.
        3. In the declaration section:
7. Declare and initialize two temporary variables to replace the commented out bind variables
8. Declare two additional variables: v\_fname of type VARCHAR2 and size 15, and

v\_emp\_sal of type NUMBER and size 10

* + - 1. Include the following SQL statement in the executable section: SELECT first\_name, salary INTO v\_fname, v\_emp\_sal FROM employees WHERE employee\_id=110;
      2. Change the line that prints “Hello World” to print “Hello” and the first name. Then, comment the lines that display the dates and print the bind variables.
      3. Calculate the contribution of an employee toward the provident fund (PF).

PF is 12% of the basic salary, and the basic salary is 45% of the salary. Use local variables for the calculation. Try to use only one expression to calculate the PF. Print the employee’s salary and his or her contribution toward PF.

* + - 1. Execute and save your script as lab\_04\_03\_soln.sql. The sample output is as follows:



### Solution 4: Writing Executable Statements

In this practice, you examine and write executable statements.

DECLARE

v\_weight NUMBER(3) := 600;

v\_message VARCHAR2(255) := 'Product 10012'; BEGIN

DECLARE

v\_weight NUMBER(3) := 1;

v\_message VARCHAR2(255) := 'Product 11001'; v\_new\_locn VARCHAR2(50) := 'Europe';

BEGIN

v\_weight := v\_weight + 1;

v\_new\_locn := 'Western ' || v\_new\_locn; 1

END;

v\_weight := v\_weight + 1;

v\_message := v\_message || ' is in stock'; v\_new\_locn := 'Western ' || v\_new\_locn; 2

END;

/

* + - * 1. Evaluate the preceding PL/SQL block and determine the data type and value of each of the

following variables, according to the rules of scoping:

1. The value of v\_weight at position 1 is:

##### 2

**The data type is NUMBER**.

1. The value of v\_new\_locn at position 1 is:

##### Western Europe

**The data type is VARCHAR2**.

1. The value of v\_weight at position 2 is:

##### 601

**The data type is NUMBER**.

1. The value of v\_message at position 2 is:

##### Product 10012 is in stock The data type is VARCHAR2.

1. The value of v\_new\_locn at position 2 is:

##### Illegal because v\_new\_locn is not visible outside the subblock

DECLARE

v\_customer VARCHAR2(50) := 'Womansport'; v\_credit\_rating VARCHAR2(50) := 'EXCELLENT'; BEGIN

DECLARE

v\_customer NUMBER(7) := 201;

v\_name VARCHAR2(25) := 'Unisports'; BEGIN

v\_credit\_rating :='GOOD';

… END;

… END;

* + - * 1. In the preceding PL/SQL block, determine the value and data type for each of the following cases:

1. The value of v\_customer in the nested block is:

##### 201

**The data type is NUMBER**.

1. The value of v\_name in the nested block is:

##### Unisports

**The data type is VARCHAR2**.

1. The value of v\_credit\_rating in the nested block is:

##### GOOD

**The data type is VARCHAR2**.

1. The value of v\_customer in the main block is:

##### Womansport

**The data type is VARCHAR2**.

1. The value of v\_name in the main block is:

##### Null. name is not visible in the main block and you would see an error.

1. The value of v\_credit\_rating in the main block is:

##### EXCELLENT

**The data type is VARCHAR2**.

* + - * 1. Use the same session that you used to execute the practices in the lesson titled “Declaring PL/SQL Variables.” If you have opened a new session, execute lab\_03\_05\_soln.sql.

Then, edit lab\_03\_05\_soln.sql as follows:

Use single-line comment syntax to comment the lines that create the bind variables, and turn on SERVEROUTPUT.

-- VARIABLE b\_basic\_percent NUMBER

-- VARIABLE b\_pf\_percent NUMBER SET SERVEROUTPUT ON

Use multiple-line comments in the executable section to comment the lines that assign values to the bind variables.

/\*:b\_basic\_percent:=45;

:b\_pf\_percent:=12;\*/

In the declaration section:

1. Declare and initialize two temporary variables to replace the commented out bind variables
2. Declare two additional variables: v\_fname of type VARCHAR2 and size 15, and

v\_emp\_sal of type NUMBER and size 10

DECLARE

v\_basic\_percent NUMBER:=45; v\_pf\_percent NUMBER:=12; v\_fname VARCHAR2(15); v\_emp\_sal NUMBER(10);

Include the following SQL statement in the executable section: SELECT first\_name, salary INTO v\_fname, v\_emp\_sal FROM employees WHERE employee\_id=110;

Change the line that prints “Hello World” to print “Hello” and the first name. Then, comment the lines that display the dates and print the bind variables. DBMS\_OUTPUT.PUT\_LINE(' Hello '|| v\_fname);

/\* DBMS\_OUTPUT.PUT\_LINE('TODAY IS : '|| v\_today);

DBMS\_OUTPUT.PUT\_LINE('TOMORROW IS : ' || v\_tomorrow);\*/

...

...

/

--PRINT b\_basic\_percent

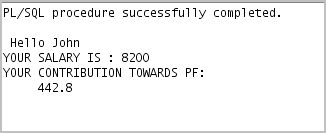
--PRINT b\_basic\_percent

Calculate the contribution of an employee toward the provident fund (PF).

PF is 12% of the basic salary, and the basic salary is 45% of the salary. Use local variables for the calculation. Try to use only one expression to calculate the PF. Print the employee’s salary and his or her contribution toward PF. DBMS\_OUTPUT.PUT\_LINE('YOUR SALARY IS : '||v\_emp\_sal); DBMS\_OUTPUT.PUT\_LINE('YOUR CONTRIBUTION TOWARDS PF:

'||v\_emp\_sal\*v\_basic\_percent/100\*v\_pf\_percent/100); END;

Execute and save your script as lab\_04\_03\_soln.sql. The sample output is as follows:



# Practices for Lesson 5: Using SQL Statements within a PL/SQL Block

**Chapter 5**

### Practice 5: Using SQL Statements Within a PL/SQL

**Note:** If you have executed the code examples for this lesson, make sure that you execute the following code before starting this practice:

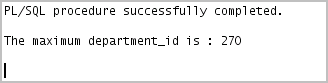
DROP table employees2; DROP table copy\_emp;

In this practice, you use PL/SQL code to interact with the Oracle Server.

1. Create a PL/SQL block that selects the maximum department ID in the departments

table and stores it in the v\_max\_deptno variable. Display the maximum department ID.

* 1. Declare a variable v\_max\_deptno of type NUMBER in the declarative section.
  2. Start the executable section with the BEGIN keyword and include a SELECT statement to retrieve the maximum department\_id from the departments table.
  3. Display v\_max\_deptno and end the executable block.
  4. Execute and save your script as lab\_05\_01\_soln.sql. The sample output is as follows:



1. Modify the PL/SQL block that you created in step 1 to insert a new department into the

departments table.

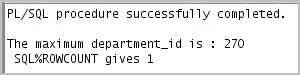
1. Load the lab\_05\_01\_soln.sql script. Declare two variables: v\_dept\_name of type departments.department\_name and v\_dept\_id of type NUMBER.

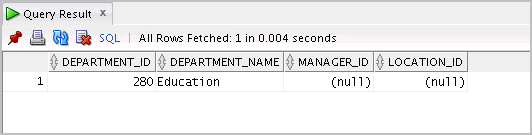
Assign 'Education' to v\_dept\_name in the declarative section.

1. You have already retrieved the current maximum department number from the

departments table. Add 10 to it and assign the result to v\_dept\_id.

1. Include an INSERT statement to insert data into the department\_name, department\_id, and location\_id columns of the departments table. Use the values in v\_dept\_name and v\_dept\_id for department\_name and department\_id, respectively, and use NULL for location\_id.
2. Use the SQL attribute SQL%ROWCOUNT to display the number of rows that are affected.
3. Execute a SELECT statement to check whether the new department is inserted. You can terminate the PL/SQL block with “/” and include the SELECT statement in your script.
4. Execute and save your script as lab\_05\_02\_soln.sql. The sample output is as follows:



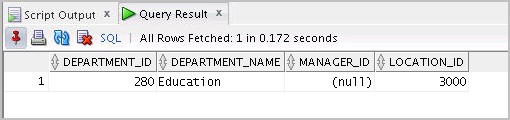
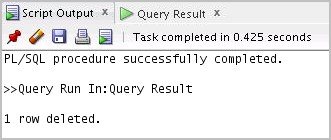


1. In step 2, you set location\_id to NULL. Create a PL/SQL block that updates

location\_id to 3000 for the new department.

**Note:** If you successfully completed step 2, continue with step 3a. If not, first execute the solution script /soln/sol\_05.sql. (Task 2 in sol\_05.sql)

1. Start the executable block with the BEGIN keyword. Include the UPDATE statement to set location\_id to 3000 for the new department (v\_dept\_id =280).
2. End the executable block with the END keyword. Terminate the PL/SQL block with “/” and include a SELECT statement to display the department that you updated.
3. Include a DELETE statement to delete the department that you added.
4. Execute and save your script as lab\_05\_03\_soln.sql. The sample output is as follows:



### Solution 5: Using SQL Statements Within a PL/SQL

In this practice, you use PL/SQL code to interact with the Oracle Server.

1. Create a PL/SQL block that selects the maximum department ID in the departments

table and stores it in the v\_max\_deptno variable. Display the maximum department ID.

* 1. Declare a variable v\_max\_deptno of type NUMBER in the declarative section.

DECLARE

v\_max\_deptno NUMBER;

* 1. Start the executable section with the BEGIN keyword and include a SELECT statement to retrieve the maximum department\_id from the departments table.

BEGIN

SELECT MAX(department\_id) INTO v\_max\_deptno FROM departments;

* 1. Display v\_max\_deptno and end the executable block. DBMS\_OUTPUT.PUT\_LINE('The maximum department\_id is : ' || v\_max\_deptno);

END;

* 1. Execute and save your script as lab\_05\_01\_soln.sql. The sample output is as follows:

1. Modify the PL/SQL block that you created in step 1 to insert a new department into the

departments table.

1. Load the lab\_05\_01\_soln.sql script. Declare two variables: v\_dept\_name of type departments.department\_name and v\_dept\_id of type NUMBER.

Assign 'Education' to v\_dept\_name in the declarative section.

v\_dept\_name departments.department\_name%TYPE:= 'Education'; v\_dept\_id NUMBER;

1. You have already retrieved the current maximum department number from the departments table. Add 10 to it and assign the result to v\_dept\_id. v\_dept\_id := 10 + v\_max\_deptno;
2. Include an INSERT statement to insert data into the department\_name, department\_id, and location\_id columns of the departments table.

Use the values in v\_dept\_name and v\_dept\_id for department\_name and

department\_id, respectively, and use NULL for location\_id.

…

INSERT INTO departments (department\_id, department\_name, location\_id)

VALUES (v\_dept\_id, v\_dept\_name, NULL);

1. Use the SQL attribute SQL%ROWCOUNT to display the number of rows that are affected.

DBMS\_OUTPUT.PUT\_LINE (' SQL%ROWCOUNT gives ' || SQL%ROWCOUNT);

…

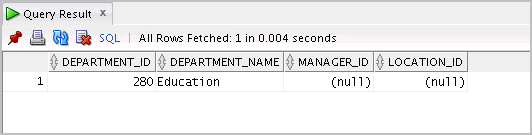
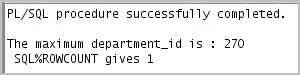
1. Execute a SELECT statement to check whether the new department is inserted. You can terminate the PL/SQL block with “/” and include the SELECT statement in your script.

…

/

SELECT \* FROM departments WHERE department\_id= 280;

1. Execute and save your script as lab\_05\_02\_soln.sql. The sample output is as follows:



1. In step 2, you set location\_id to NULL. Create a PL/SQL block that updates the

location\_id to 3000 for the new department.

**Note:** If you successfully completed step 2, continue with step 3a. If not, first execute the solution script /soln/sol\_05.sql. (Task 2 in sol\_05.sql)

1. Start the executable block with the BEGIN keyword. Include the UPDATE statement to set location\_id to 3000 for the new department (v\_dept\_id =280).

BEGIN

UPDATE departments SET location\_id=3000 WHERE department\_id=280;

1. End the executable block with the END keyword. Terminate the PL/SQL block with “/” and include a SELECT statement to display the department that you updated.

END;

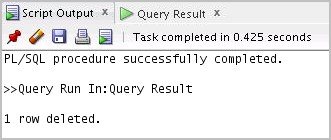
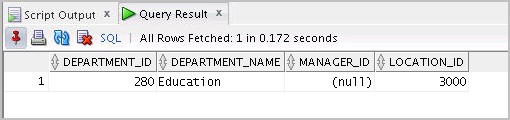
/

SELECT \* FROM departments WHERE department\_id=280;

1. Include a DELETE statement to delete the department that you added.

DELETE FROM departments WHERE department\_id=280;

1. Execute and save your script as lab\_05\_03\_soln.sql. The sample output is as follows:



**Practices for Lesson 6: Writing Control Structures**

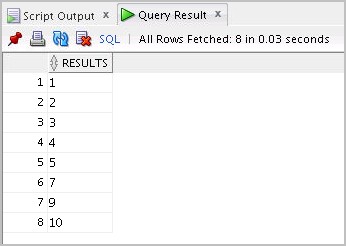
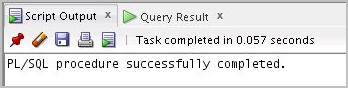
**Chapter 6**

### Practice 6: Writing Control Structures

In this practice, you create PL/SQL blocks that incorporate loops and conditional control structures. This practice tests your understanding of various IF statements and LOOP constructs.

* 1. Execute the command in the lab\_06\_01.sql file to create the messages table. Write a PL/SQL block to insert numbers into the messages table.

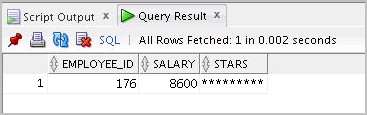
1. Insert the numbers 1 through 10, excluding 6 and 8.
2. Commit before the end of the block.
3. Execute a SELECT statement to verify that your PL/SQL block worked. Result: You should see the following output:



* 1. Execute the lab\_06\_02.sql script. This script creates an emp table that is a replica of the employees table. It alters the emp table to add a new column, stars, of VARCHAR2 data type and size 50. Create a PL/SQL block that inserts an asterisk in the stars column for every $1000 of an employee’s salary. Save your script as lab\_06\_02\_soln.sql.
     1. In the declarative section of the block, declare a variable v\_empno of type emp.employee\_id and initialize it to 176. Declare a variable v\_asterisk of type emp.stars and initialize it to NULL. Create a variable v\_sal of type emp.salary.
     2. In the executable section, write logic to append an asterisk (\*) to the string for every

$1,000 of the salary. For example, if the employee earns $8,000, the string of asterisks should contain eight asterisks. If the employee earns $12,500, the string of asterisks should contain 13 asterisks (rounded to the nearest whole number).

* + 1. Update the stars column for the employee with the string of asterisks. Commit before the end of the block.
    2. Display the row from the emp table to verify whether your PL/SQL block has executed successfully.
    3. Execute and save your script as lab\_06\_02\_soln.sql. The output is as follows:

.

### Solution 6: Writing Control Structures

1. Execute the command in the lab\_06\_01.sql file to create the messages table. Write a PL/SQL block to insert numbers into the messages table.

1. Insert the numbers 1 through 10, excluding 6 and 8.
2. Commit before the end of the block.

BEGIN

FOR i in 1..10 LOOP

IF i = 6 or i = 8 THEN

null;

ELSE

INSERT INTO messages(results) VALUES (i);

END IF;

END LOOP;

COMMIT;

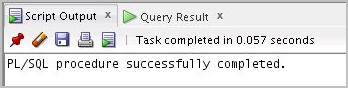
END;

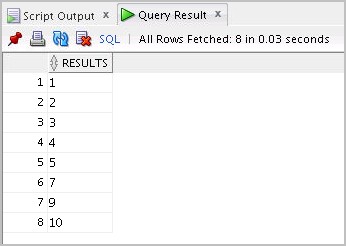
/

1. Execute a SELECT statement to verify that your PL/SQL block worked.

SELECT \* FROM messages;

**Result:** You should see the following output:





1. Execute the lab\_06\_02.sql script. This script creates an emp table that is a replica of the employees table. It alters the emp table to add a new column, stars, of VARCHAR2 data type and size 50. Create a PL/SQL block that inserts an asterisk in the stars column for every $1000 of the employee’s salary. Save your script as lab\_06\_02\_soln.sql.
   1. In the declarative section of the block, declare a variable v\_empno of type emp.employee\_id and initialize it to 176. Declare a variable v\_asterisk of type emp.stars and initialize it to NULL. Create a variable v\_sal of type emp.salary. DECLARE

v\_empno emp.employee\_id%TYPE := 176; v\_asterisk emp.stars%TYPE := NULL; v\_sal emp.salary%TYPE;

* 1. In the executable section, write logic to append an asterisk (\*) to the string for every

$1,000 of the salary. For example, if the employee earns $8,000, the string of asterisks should contain eight asterisks. If the employee earns $12,500, the string of asterisks should contain 13 asterisks.

BEGIN

SELECT NVL(ROUND(salary/1000), 0) INTO v\_sal FROM emp WHERE employee\_id = v\_empno;

FOR i IN 1..v\_sal LOOP

v\_asterisk := v\_asterisk ||'\*'; END LOOP;

* 1. Update the stars column for the employee with the string of asterisks. Commit before the end of the block.

UPDATE emp SET stars = v\_asterisk WHERE employee\_id = v\_empno; COMMIT;

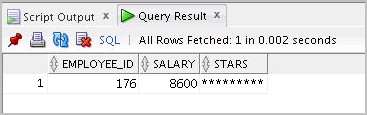
END;

/

* 1. Display the row from the emp table to verify whether your PL/SQL block has executed successfully.

SELECT employee\_id,salary, stars FROM emp WHERE employee\_id =176;

* 1. Execute and save your script as lab\_06\_02\_soln.sql. The output is as follows:



## Practices for Lesson 7:

**Working with Composite Data Types**

**Chapter 7**

### Practice 7: Working with Composite Data Types

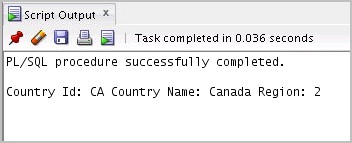
**Note:** If you have executed the code examples for this lesson, make sure that you execute the following code before starting this practice:

DROP table retired\_emps; DROP table empl;

* + 1. Write a PL/SQL block to print information about a given country.
       1. Declare a PL/SQL record based on the structure of the COUNTRIES table.
       2. Declare a variable v\_countryid. Assign CA to v\_countryid.
       3. In the declarative section, use the %ROWTYPE attribute and declare the

v\_country\_record variable of type countries.

* + - 1. In the executable section, get all the information from the COUNTRIES table by using v\_countryid. Display selected information about the country. The sample output is as follows:



* + - 1. You may want to execute and test the PL/SQL block for countries with the IDs DE, UK, and US.
    1. Create a PL/SQL block to retrieve the names of some departments from the

DEPARTMENTS

table and print each department name on the screen, incorporating an associative array. Save the script as lab\_07\_02\_soln.sql.

1. Declare an INDEX BY table dept\_table\_type of type departments.department\_name. Declare a variable my\_dept\_table of type dept\_table\_type to temporarily store the names of the departments.
2. Declare two variables: f\_loop\_count and v\_deptno of type NUMBER. Assign 10 to

f\_loop\_count and 0 to v\_deptno.

1. Using a loop, retrieve the names of 10 departments and store the names in the associative array. Start with department\_id 10. Increase v\_deptno by 10 for every loop iteration. The following table shows the department\_id for which you should retrieve the department\_name.

DEPARTMENT\_ID DEPARTMENT\_NAME

##### 10 Administration

**20 Marketing**

**30 Purchasing**

**40 Human Resources**

**50 Shipping**

**60 IT**

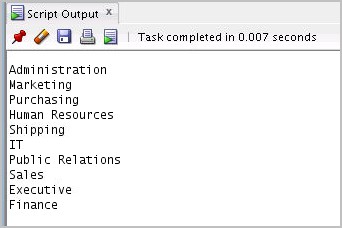
**70 Public Relations**

**80 Sales**

**90 Executive**

**100 Finance**

1. Using another loop, retrieve the department names from the associative array and display them.
2. Execute and save your script as lab\_07\_02\_soln.sql. The output is as follows:



* + 1. Modify the block that you created in Task 2 to retrieve all information about each department from the DEPARTMENTS table and display the information. Use an associative array with the INDEX BY table of records method.

1. Load the lab\_07\_02\_soln.sql script.
2. You have declared the associative array to be of type departments.department\_name. Modify the declaration of the associative array to temporarily store the number, name, and location of all the departments. Use the

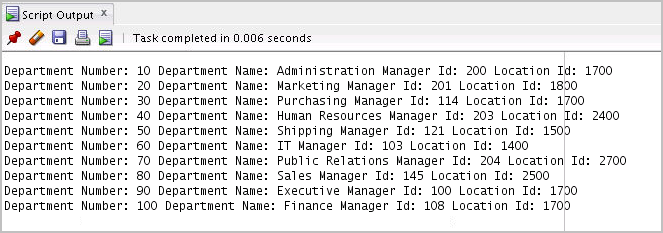
%ROWTYPE attribute.

1. Modify the SELECT statement to retrieve all department information currently in the

DEPARTMENTS table and store it in the associative array.

1. Using another loop, retrieve the department information from the associative array and display the information.

The sample output is as follows:



### Solution 7: Working with Composite Data Types

1. Write a PL/SQL block to print information about a given country.
   1. Declare a PL/SQL record based on the structure of the COUNTRIES table.
   2. Declare a variable v\_countryid. Assign CA to v\_countryid. SET SERVEROUTPUT ON

SET VERIFY OFF DECLARE

v\_countryid varchar2(20):= 'CA';

* 1. In the declarative section, use the %ROWTYPE attribute and declare the v\_country\_record variable of type countries. v\_country\_record countries%ROWTYPE;
  2. In the executable section, get all the information from the COUNTRIES table by using

v\_countryid. Display selected information about the country.

BEGIN SELECT \*

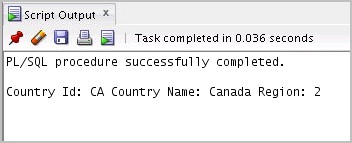
INTO v\_country\_record FROM countries

WHERE country\_id = UPPER(v\_countryid); DBMS\_OUTPUT.PUT\_LINE ('Country Id: ' || v\_country\_record.country\_id ||

' Country Name: ' || v\_country\_record.country\_name

|| ' Region: ' || v\_country\_record.region\_id); END;

The sample output after performing all the above steps is as follows:



* 1. You may want to execute and test the PL/SQL block for countries with the IDs DE, UK, and US.

1. Create a PL/SQL block to retrieve the names of some departments from the DEPARTMENTS table and print each department name on the screen, incorporating an associative array.

Save the script as lab\_07\_02\_soln.sql.

* 1. Declare an INDEX BY table dept\_table\_type of type departments.department\_name. Declare a variable my\_dept\_table of type dept\_table\_type to temporarily store the names of the departments.

SET SERVEROUTPUT ON DECLARE

TYPE dept\_table\_type is table of departments.department\_name%TYPE INDEX BY PLS\_INTEGER;

my\_dept\_table dept\_table\_type;

* 1. Declare two variables: f\_loop\_count and v\_deptno of type NUMBER. Assign 10 to

f\_loop\_count and 0 to v\_deptno. f\_loop\_count NUMBER (2):=10;

v\_deptno NUMBER (4):=0;

* 1. Using a loop, retrieve the names of 10 departments and store the names in the associative array. Start with department\_id 10. Increase v\_deptno by 10 for every iteration of the loop. The following table shows the department\_id for which you should retrieve the department\_name and store in the associative array.

DEPARTMENT\_ID DEPARTMENT\_NAME

##### 10 Administration

**20 Marketing**

**30 Purchasing**

**40 Human Resources**

**50 Shipping**

**60 IT**

**70 Public Relations**

**80 Sales**

**90 Executive**

**100 Finance**

BEGIN

FOR i IN 1..f\_loop\_count LOOP

v\_deptno:=v\_deptno+10; SELECT department\_name INTO my\_dept\_table(i) FROM departments

WHERE department\_id = v\_deptno; END LOOP;

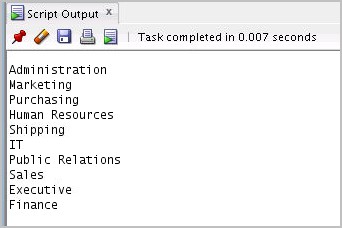
* 1. Using another loop, retrieve the department names from the associative array and display them.

FOR i IN 1..f\_loop\_count LOOP

DBMS\_OUTPUT.PUT\_LINE (my\_dept\_table(i)); END LOOP;

END;

* 1. Execute and save your script as lab\_07\_02\_soln.sql. The output is as follows:



1. Modify the block that you created in Task 2 to retrieve all information about each department from the DEPARTMENTS table and display the information. Use an associative array with the INDEX BY table of records method.
2. Load the lab\_07\_02\_soln.sql script.
3. You have declared the associative array to be of the departments.department\_name type. Modify the declaration of the associative array to temporarily store the number, name, and location of all the departments. Use the %ROWTYPE attribute.

SET SERVEROUTPUT ON DECLARE

TYPE dept\_table\_type is table of departments%ROWTYPE INDEX BY PLS\_INTEGER;

my\_dept\_table dept\_table\_type; f\_loop\_count NUMBER (2):=10;

v\_deptno NUMBER (4):=0;

1. Modify the SELECT statement to retrieve all department information currently in the

DEPARTMENTS table and store it in the associative array.

BEGIN

FOR i IN 1..f\_loop\_count LOOP

v\_deptno := v\_deptno + 10; SELECT \*

INTO my\_dept\_table(i) FROM departments

WHERE department\_id = v\_deptno; END LOOP;

1. Using another loop, retrieve the department information from the associative array and display the information.

FOR i IN 1..f\_loop\_count LOOP

DBMS\_OUTPUT.PUT\_LINE ('Department Number: ' || my\_dept\_table(i).department\_id

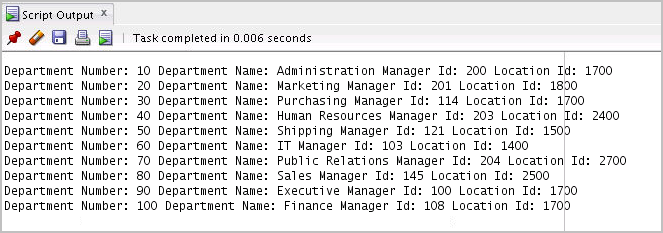
|| ' Department Name: ' || my\_dept\_table(i).department\_name

|| ' Manager Id: '|| my\_dept\_table(i).manager\_id

|| ' Location Id: ' || my\_dept\_table(i).location\_id); END LOOP;

END;

The sample output is as follows:



## Practices for Lesson 8: Using Explicit Cursors

**Chapter 8**

### Practice 8-1: Using Explicit Cursors

In this practice, you perform two exercises:

* + First, you use an explicit cursor to process a number of rows from a table and populate

another table with the results by using a cursor FOR loop.

* + Second, you write a PL/SQL block that processes information with two cursors, including one that uses a parameter.

1. Create a PL/SQL block to perform the following:
   1. In the declarative section, declare and initialize a variable named v\_deptno of type

NUMBER. Assign a valid department ID value (see table in step d for values).

* 1. Declare a cursor named c\_emp\_cursor, which retrieves the last\_name, salary, and manager\_id of employees working in the department specified in v\_deptno.
  2. In the executable section, use the cursor FOR loop to operate on the data retrieved. If the salary of the employee is less than 5,000 and if the manager ID is either 101 or

124, display the message “<<*last\_name*>> Due for a raise.” Otherwise, display the message “<<*last\_name*>> Not Due for a raise.”

* 1. Test the PL/SQL block for the following cases:

##### Department ID Message

10 Whalen Due for a raise

20 Hartstein Not Due for a raise Fay Not Due for a raise

50 Weiss Not Due for a raise Fripp Not Due for a raise Kaufling Not Due for a raise

Vollman Not Due for a raise. . .

. . .

OConnell Due for a raise Grant Due for a raise

80 Russell Not Due for a raise Partners Not Due for a raise Errazuriz Not Due for a raise Cambrault Not Due for a raise

. . .

Livingston Not Due for a raise Johnson Not Due for a raise

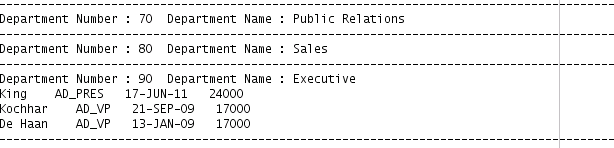
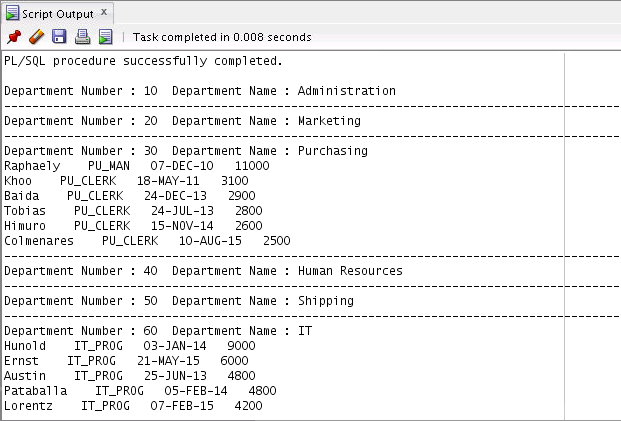
1. Next, write a PL/SQL block that declares and uses two cursors—one without a parameter and one with a parameter. The first cursor retrieves the department number and department name from the DEPARTMENTS table for all departments whose ID number is less than 100. The second cursor receives the department number as a parameter, and retrieves employee details for those who work in that department and whose employee\_id is less than 120.
   1. Declare a cursor c\_dept\_cursor to retrieve department\_id and department\_name for those departments with department\_id less than 100. Order by department\_id.
   2. Declare another cursor c\_emp\_cursor that takes the department number as parameter and retrieves the following data from the EMPLOYEES table: last\_name, job\_id, hire\_date, and salary of those employees who work in that department, with employee\_id less than 120.
   3. Declare variables to hold the values retrieved from each cursor. Use the %TYPE

attribute while declaring variables.

* 1. Open c\_dept\_cursor and use a simple loop to fetch values into the variables that are declared. Display the department number and department name. Use the appropriate cursor attribute to exit the loop.
  2. Open c\_emp\_cursor by passing the current department number as a parameter. Start another loop and fetch the values of emp\_cursor into variables, and print all the details retrieved from the EMPLOYEES table.

##### Notes

* + - Check whether c\_emp\_cursor is already open before opening the cursor.
    - Use the appropriate cursor attribute for the exit condition.
    - When the loop completes, print a line after you have displayed the details of each department, and close c\_emp\_cursor.
  1. End the first loop and close c\_dept\_cursor. Then end the executable section.
  2. Execute the script. The sample output is as follows:



### Solution 8-1: Using Explicit Cursors

In this practice, you perform two exercises:

* + - First, you use an explicit cursor to process a number of rows from a table and populate

another table with the results by using a cursor FOR loop.

* + - Second, you write a PL/SQL block that processes information with two cursors, including one that uses a parameter.

1. Create a PL/SQL block to perform the following:
2. In the declarative section, declare and initialize a variable named v\_deptno of the NUMBER type. Assign a valid department ID value (see table in step d for values). DECLARE

v\_deptno NUMBER := 10;

1. Declare a cursor named c\_emp\_cursor, which retrieves last\_name, salary, and manager\_id of employees working in the department specified in v\_deptno. CURSOR c\_emp\_cursor IS

SELECT last\_name, salary,manager\_id FROM employees

WHERE department\_id = v\_deptno;

1. In the executable section, use the cursor FOR loop to operate on the data retrieved. If the salary of the employee is less than 5,000 and if the manager ID is either 101 or

124, display the message “<<*last\_name*>> Due for a raise.” Otherwise, display the message “<<*last\_name*>> Not Due for a raise.”

BEGIN

FOR emp\_record IN c\_emp\_cursor LOOP

IF emp\_record.salary < 5000 AND (emp\_record.manager\_id=101 OR emp\_record.manager\_id=124) THEN

DBMS\_OUTPUT.PUT\_LINE (emp\_record.last\_name || ' Due for a raise');

ELSE

DBMS\_OUTPUT.PUT\_LINE (emp\_record.last\_name || ' Not Due for a raise');

END IF;

END LOOP;

END;.

b. Test the PL/SQL block for the following cases:

##### Department ID Message

10 Whalen Due for a raise

20 Hartstein Not Due for a raise Fay Not Due for a raise

50 Weiss Not Due for a raise Fripp Not Due for a raise Kaufling Not Due for a raise

Vollman Not Due for a raise. . .

. . .

OConnell Due for a raise Grant Due for a raise

80 Russell Not Due for a raise Partners Not Due for a raise Errazuriz Not Due for a raise Cambrault Not Due for a raise

. . .

Livingston Not Due for a raise Johnson Not Due for a raise

1. Next, write a PL/SQL block that declares and uses two cursors—one without a parameter and one with a parameter. The first cursor retrieves the department number and department name from the DEPARTMENTS table for all departments whose ID number is less than 100. The second cursor receives the department number as a parameter, and retrieves employee details for those who work in that department and whose employee\_id is less than 120.
   1. Declare a cursor c\_dept\_cursor to retrieve department\_id and department\_name for those departments with department\_id less than 100. Order by department\_id.

DECLARE

CURSOR c\_dept\_cursor IS

SELECT department\_id,department\_name FROM departments

WHERE department\_id < 100 ORDER BY department\_id;

* 1. Declare another cursor c\_emp\_cursor that takes the department number as parameter and retrieves the following data from the EMPLOYEES table: last\_name, job\_id, hire\_date, and salary of those employees who work in that department, with employee\_id less than 120.

CURSOR c\_emp\_cursor(v\_deptno NUMBER) IS SELECT last\_name,job\_id,hire\_date,salary FROM employees

WHERE department\_id = v\_deptno AND employee\_id < 120;

* 1. Declare variables to hold the values retrieved from each cursor. Use the %TYPE

attribute while declaring variables.

v\_current\_deptno departments.department\_id%TYPE; v\_current\_dname departments.department\_name%TYPE; v\_ename employees.last\_name%TYPE;

v\_job employees.job\_id%TYPE; v\_hiredate employees.hire\_date%TYPE; v\_sal employees.salary%TYPE;

* 1. Open c\_dept\_cursor and use a simple loop to fetch values into the variables that are declared. Display the department number and department name. Use the appropriate cursor attribute to exit the loop.

BEGIN

OPEN c\_dept\_cursor; LOOP

FETCH c\_dept\_cursor INTO v\_current\_deptno, v\_current\_dname;

EXIT WHEN c\_dept\_cursor%NOTFOUND; DBMS\_OUTPUT.PUT\_LINE ('Department Number : ' || v\_current\_deptno || ' Department Name : ' || v\_current\_dname);

* 1. Open c\_emp\_cursor by passing the current department number as a parameter. Start another loop and fetch the values of emp\_cursor into variables, and print all the details retrieved from the EMPLOYEES table.

##### Notes

* + - Check whether c\_emp\_cursor is already open before opening the cursor.
    - Use the appropriate cursor attribute for the exit condition.
    - When the loop completes, print a line after you have displayed the details of each department, and close c\_emp\_cursor.

IF c\_emp\_cursor%ISOPEN THEN CLOSE c\_emp\_cursor;

END IF;

OPEN c\_emp\_cursor (v\_current\_deptno); LOOP

FETCH c\_emp\_cursor INTO v\_ename,v\_job,v\_hiredate,v\_sal; EXIT WHEN c\_emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE (v\_ename || ' ' || v\_job

|| ' ' || v\_hiredate || ' ' || v\_sal); END LOOP;

DBMS\_OUTPUT.PUT\_LINE('

');

CLOSE c\_emp\_cursor;

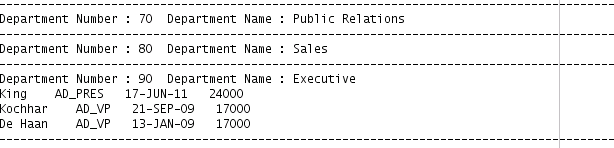
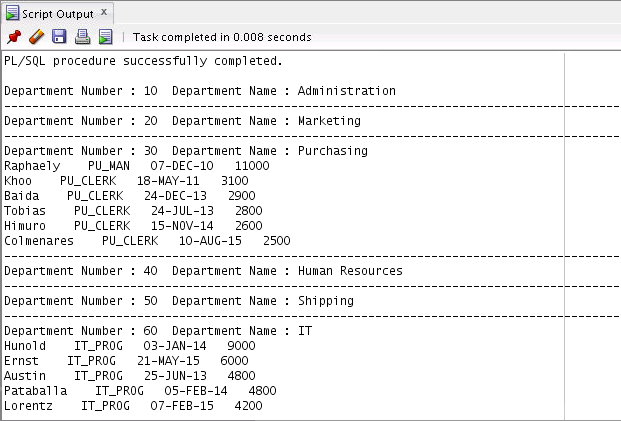
* 1. End the first loop and close c\_dept\_cursor. Then end the executable section.

END LOOP;

CLOSE c\_dept\_cursor; END;

.

* 1. Execute the script. The sample output is as follows:



### Practice 8-2: Using Explicit Cursors: Optional

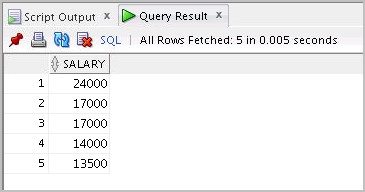
If you have time, complete the following optional practice. Here, create a PL/SQL block that uses an explicit cursor to determine the top *n* salaries of employees.

1. Run the lab\_08-02.sql script to create the TOP\_SALARIES table for storing the salaries of the employees.
2. In the declarative section, declare the v\_num variable of the NUMBER type that holds a number *n*, representing the number of top *n* earners from the employees table. For example, to view the top five salaries, enter 5. Declare another variable v\_sal of type employees.salary. Declare a cursor, c\_emp\_cursor, which retrieves the salaries of employees in descending order. Remember that the salaries should not be duplicated.
3. In the executable section, open the loop, fetch the top *n* salaries, and then insert them into the TOP\_SALARIES table. You can use a simple loop to operate on the data. Also, try and use the %ROWCOUNT and %FOUND attributes for the exit condition.

**Note:** Make sure that you add an exit condition to avoid having an infinite loop.

1. After inserting data into the TOP\_SALARIES table, display the rows with a SELECT

statement. The output shown represents the five highest salaries in the EMPLOYEES table.



1. Test a variety of special cases such as v\_num = 0 or where v\_num is greater than the number of employees in the EMPLOYEES table. Empty the TOP\_SALARIES table after each test.

### Solution 8-2: Using Explicit Cursors: Optional

If you have time, complete the following optional exercise. Here, create a PL/SQL block that uses an explicit cursor to determine the top *n* salaries of employees.

1. Execute the lab\_08\_02.sql script to create a new table, TOP\_SALARIES, for storing the salaries of the employees.
2. In the declarative section, declare a variable v\_num of type NUMBER that holds a number *n*,representing the number of top *n* earners from the EMPLOYEES table. For example, to view the top five salaries, enter 5. Declare another variable v\_sal of type employees.salary.

Declare a cursor, c\_emp\_cursor, which retrieves the salaries of employees in descending order. Remember that the salaries should not be duplicated.

DECLARE

v\_num NUMBER(3) := 5;

v\_sal employees.salary%TYPE; CURSOR c\_emp\_cursor IS SELECT salary

FROM employees

ORDER BY salary DESC;

1. In the executable section, open the loop, fetch the top *n* salaries, and then insert them into the TOP\_SALARIES table. You can use a simple loop to operate on the data. Also, try and use the %ROWCOUNT and %FOUND attributes for the exit condition.

**Note:** Make sure that you add an exit condition to avoid having an infinite loop.

BEGIN

OPEN c\_emp\_cursor;

FETCH c\_emp\_cursor INTO v\_sal;

WHILE c\_emp\_cursor%ROWCOUNT <= v\_num AND c\_emp\_cursor%FOUND LOOP INSERT INTO top\_salaries (salary)

VALUES (v\_sal);

FETCH c\_emp\_cursor INTO v\_sal; END LOOP;

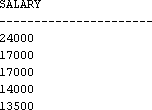
CLOSE c\_emp\_cursor; END;

1. After inserting data into the TOP\_SALARIES table, display the rows with a SELECT

statement. The output shown represents the five highest salaries in the EMPLOYEES table.

/

SELECT \* FROM top\_salaries;

The sample output is as follows:

1. Test a variety of special cases such as v\_num = 0 or where v\_num is greater than the number of employees in the EMPLOYEES table. Empty the TOP\_SALARIES table after each test.

.

## Practices for Lesson 9: Handling Exceptions

**Chapter 9**

### Practice 9-1: Handling Predefined Exceptions

In this practice, you write a PL/SQL block that applies a predefined exception to process only one record at a time. The PL/SQL block selects the name of the employee with a given salary value.

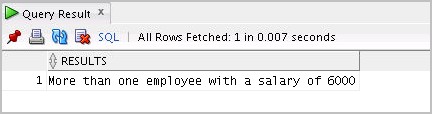
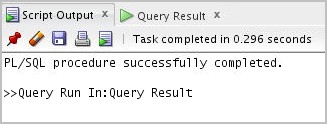
1. Execute the command in the lab\_06\_01.sql file to re-create the messages table.
2. In the declarative section, declare two variables: v\_ename of type employees.last\_name and v\_emp\_sal of type employees.salary. Initialize the latter to 6000.
3. In the executable section, retrieve the last names of employees whose salaries are equal to the value in v\_emp\_sal. If the salary entered returns only one row, insert the employee’s name and salary amount into the MESSAGES table.

**Note:** Do not use explicit cursors.

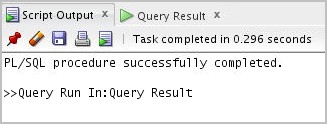
1. If the salary entered does not return any rows, handle the exception with an appropriate exception handler and insert the message “No employee with a salary of <*salary*>” into the MESSAGES table.
2. If the salary entered returns multiple rows, handle the exception with an appropriate exception handler and insert the message “More than one employee with a salary of

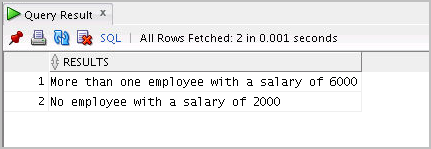
<*salary*>” into the MESSAGES table.

1. Handle any other exception with an appropriate exception handler and insert the message “Some other error occurred” into the MESSAGES table.
2. Display the rows from the MESSAGES table to check whether the PL/SQL block has executed successfully. The output is as follows:



1. Change the initialized value of v\_emp\_sal to 2000 and re-execute. The output is as follows:





### Solution 9-1: Handling Predefined Exceptions

In this practice, you write a PL/SQL block that applies a predefined exception to process only one record at a time. The PL/SQL block selects the name of the employee with a given salary value.

1. Execute the command in the lab\_06\_01.sql file to re-create the MESSAGES table.
2. In the declarative section, declare two variables: v\_ename of type employees.last\_name and v\_emp\_sal of type employees.salary. Initialize the latter

to 6000.

DECLARE

v\_ename employees.last\_name%TYPE; v\_emp\_sal employees.salary%TYPE := 6000;

1. In the executable section, retrieve the last names of employees whose salaries are equal to

the value in v\_emp\_sal. If the salary entered returns only one row, insert the employee’s name and the salary amount into the MESSAGES table.

**Note:** Do not use explicit cursors.

BEGIN

SELECT last\_name INTO v\_ename FROM employees

WHERE salary = v\_emp\_sal; INSERT INTO messages (results)

VALUES (v\_ename || ' - ' || v\_emp\_sal);

1. If the salary entered does not return any rows, handle the exception with an appropriate exception handler and insert the message “No employee with a salary of <*salary*>” into the MESSAGES table.

EXCEPTION

WHEN no\_data\_found THEN

INSERT INTO messages (results)

VALUES ('No employee with a salary of '|| TO\_CHAR(v\_emp\_sal));

1. If the salary entered returns multiple rows, handle the exception with an appropriate exception handler and insert the message “More than one employee with a salary of

<*salary*>” into the MESSAGES table.

WHEN too\_many\_rows THEN

INSERT INTO messages (results)

VALUES ('More than one employee with a salary of '|| TO\_CHAR(v\_emp\_sal));

1. Handle any other exception with an appropriate exception handler and insert the message “Some other error occurred” into the MESSAGES table.

WHEN others THEN

INSERT INTO messages (results)

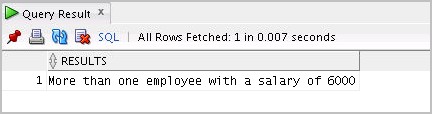
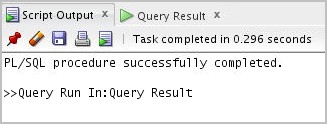
VALUES ('Some other error occurred.'); END;

1. Display the rows from the MESSAGES table to check whether the PL/SQL block has executed successfully.

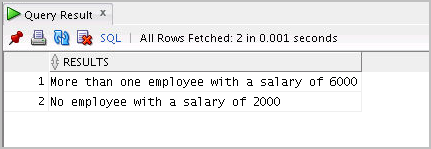
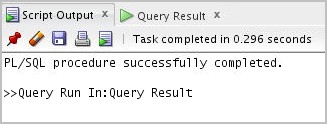
/

SELECT \* FROM messages;

The output is as follows:



1. Change the initialized value of v\_emp\_sal to 2000 and re-execute. The output is as follows:



### Practice 9-2: Handling Standard Oracle Server Exceptions

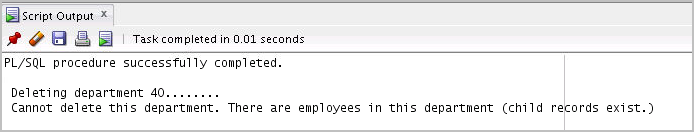
In this practice, you write a PL/SQL block that declares an exception for the Oracle Server error ORA-02292 (integrity constraint violated – child record found). The block tests for the exception and outputs the error message.

1. In the declarative section, declare an exception e\_childrecord\_exists. Associate the declared exception with the standard Oracle Server error –02292.
2. In the executable section, display “Deleting department 40 ” Include a DELETE

statement to delete the department with the department\_id 40.

1. Include an exception section to handle the e\_childrecord\_exists exception and display the appropriate message.

The sample output is as follows:



### Solution 9-2: Handling Standard Oracle Server Exceptions

In this practice, you write a PL/SQL block that declares an exception for the Oracle Server error ORA-02292 (integrity constraint violated – child record found). The block tests for the exception and outputs the error message.

1. In the declarative section, declare an exception e\_childrecord\_exists. Associate the declared exception with the standard Oracle Server error –02292.

SET SERVEROUTPUT ON DECLARE

e\_childrecord\_exists EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_childrecord\_exists, -02292);

1. In the executable section, display “Deleting department 40 ” Include a DELETE

statement to delete the department with department\_id 40.

BEGIN

DBMS\_OUTPUT.PUT\_LINE(' Deleting department 40. ');

delete from departments where department\_id=40;

1. Include an exception section to handle the e\_childrecord\_exists exception and display the appropriate message.

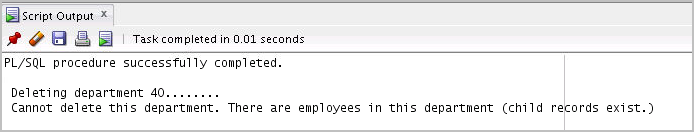
EXCEPTION

WHEN e\_childrecord\_exists THEN

DBMS\_OUTPUT.PUT\_LINE(' Cannot delete this department. There are employees in this department (child records exist.) ');

END;

The sample output is as follows:



## Practices for Lesson 10: Introducing Stored Procedures and Functions

**Chapter 10**

### Practice 10: Creating and Using Stored Procedures

**Note:** If you have executed the code examples for this lesson, make sure that you execute the following code before starting this practice:

DROP table dept;

DROP procedure add\_dept; DROP function check\_sal;

In this practice, you modify existing scripts to create and use stored procedures.

1. Open the sol\_03.sql script from the /home/oracle/labs/plsf/soln/ folder. Copy the code under task 4 into a new worksheet.

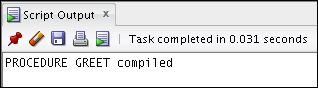
SET SERVEROUTPUT ON DECLARE

v\_today DATE:=SYSDATE; v\_tomorrow v\_today%TYPE; BEGIN

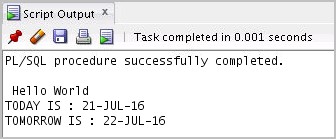
v\_tomorrow:=v\_today +1; DBMS\_OUTPUT.PUT\_LINE(' Hello World ');

DBMS\_OUTPUT.PUT\_LINE('TODAY IS : '|| v\_today); DBMS\_OUTPUT.PUT\_LINE('TOMORROW IS : ' || v\_tomorrow); END;

* 1. Modify the script to convert the anonymous block to a procedure called greet. (**Hint:** Also remove the SET SERVEROUTPUT ON command.)
  2. Execute the script to create the procedure. The output results should be as follows:



* 1. Save this script as lab\_10\_01\_soln.sql.
  2. Click the Clear button to clear the workspace.
  3. Create and execute an anonymous block to invoke the greet procedure. (**Hint:** Ensure that you enable SERVEROUTPUT at the beginning of the block.) The output should be similar to the following:



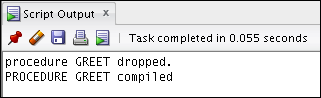
1. Modify the lab\_10\_01\_soln.sql script as follows:
   1. Drop the greet procedure by issuing the following command:

DROP PROCEDURE greet;

* 1. Modify the procedure to accept an argument of type VARCHAR2. Call the argument

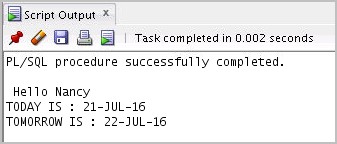
p\_name.

* 1. Print Hello <*name*> (that is, the contents of the argument) instead of printing Hello World.
  2. Save your script as lab\_10\_02\_soln.sql.
  3. Execute the script to create the procedure. The output results should be as follows:



* 1. Create and execute an anonymous block to invoke the greet procedure with a parameter value. The block should also produce the output.

The sample output should be similar to the following:



.

### Solution 10: Creating and Using Stored Procedures

In this practice, you modify existing scripts to create and use stored procedures.

* + 1. Open the sol\_03.sql script from the /home/oracle/labs/plsf/soln/ folder. Copy the code under task 4 into a new worksheet.

SET SERVEROUTPUT ON DECLARE

v\_today DATE:=SYSDATE; v\_tomorrow v\_today%TYPE; BEGIN

v\_tomorrow:=v\_today +1; DBMS\_OUTPUT.PUT\_LINE(' Hello World ');

DBMS\_OUTPUT.PUT\_LINE('TODAY IS : '|| v\_today); DBMS\_OUTPUT.PUT\_LINE('TOMORROW IS : ' || v\_tomorrow); END;

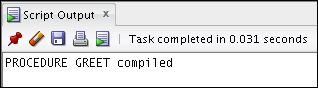
* + - 1. Modify the script to convert the anonymous block to a procedure called greet. (**Hint:** Also remove the SET SERVEROUTPUT ON command.)

CREATE PROCEDURE greet IS

v\_today DATE:=SYSDATE; v\_tomorrow v\_today%TYPE;

...

* + - 1. Execute the script to create the procedure. The output results should be as follows:



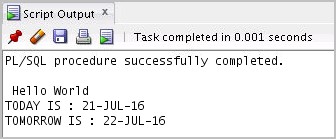
* + - 1. Save this script as lab\_10\_01\_soln.sql.
      2. Click the Clear button to clear the workspace.
      3. Create and execute an anonymous block to invoke the greet procedure. (**Hint:** Ensure that you enable SERVEROUTPUT at the beginning of the block.)

SET SERVEROUTPUT ON BEGIN

greet;

END;

The output should be similar to the following:



* + 1. Modify the lab\_10\_01\_soln.sql script as follows:
       1. Drop the greet procedure by issuing the following command:

DROP PROCEDURE greet;

* + - 1. Modify the procedure to accept an argument of type VARCHAR2. Call the argument

p\_name.

CREATE PROCEDURE greet(p\_name VARCHAR2) IS

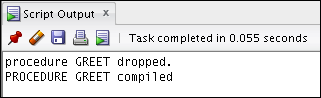
v\_today DATE:=SYSDATE; v\_tomorrow v\_today%TYPE;

* + - 1. Print Hello <*name*> instead of printing Hello World. BEGIN

v\_tomorrow:=v\_today +1; DBMS\_OUTPUT.PUT\_LINE(' Hello '|| p\_name);

...

* + - 1. Save your script as lab\_10\_02\_soln.sql.
      2. Execute the script to create the procedure. The output results should be as follows:



* + - 1. Create and execute an anonymous block to invoke the greet procedure with a parameter value. The block should also produce the output.

SET SERVEROUTPUT ON; BEGIN

greet('Nancy');

END;

The sample output should be similar to the following:

