



Integrated Cloud Applications & Platform Services

# Oracle Database 12c R2: PL/SQL Fundamentals

Activity Guide

D80182GC20

Edition 2.0 | November 2016 | D98674

Learn more from Oracle University at [education.oracle.com](http://education.oracle.com)

The Oracle logo, consisting of the word "ORACLE" in a white, sans-serif font with a registered trademark symbol, set against a red background.

## **Author**

Jayashree Sharma

## **Technical Contributors and Reviewers**

Bryan Roberts

Miyuki Osato

Nancy Greenberg

Suresh Rajan

## **Editor**

Vijayalakshmi Narasimhan

## **Graphic Designers**

Prakash Dharmalingam

Seema Bopaiah

Maheshwari Krishnamurthy

## **Publishers**

Syed Imtiaz Ali

Sujatha Nagendra

**Copyright © 2016, Oracle and/or its affiliates. All rights reserved.**

## **Disclaimer**

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

## **Restricted Rights Notice**

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

### **U.S. GOVERNMENT RIGHTS**

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

## **Trademark Notice**

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

# Table of Contents

<b>Security Credentials .....</b>	<b>I-1</b>
Security Credentials.....	I-2
<b>Practices for Lesson 1: Introduction.....</b>	<b>1-1</b>
Practices for Lesson 1.....	1-2
Practice 1-1: Getting Started .....	1-3
Solution 1-1: Getting Started .....	1-5
<b>Practices for Lesson 2: Introduction to PL/SQL.....</b>	<b>2-1</b>
Practices for Lesson 2: Introduction to PL/SQL .....	2-2
Practice 2: Introduction to PL/SQL.....	2-3
Solution 2: Introduction to PL/SQL.....	2-4
<b>Practices for Lesson 3: Declaring PL/SQL Variables.....</b>	<b>3-1</b>
Practice 3: Declaring PL/SQL Variables .....	3-2
Solution 3: Declaring PL/SQL Variables.....	3-4
<b>Practices for Lesson 4: Writing Executable Statements.....</b>	<b>4-1</b>
Practice 4: Writing Executable Statements.....	4-2
Solution 4: Writing Executable Statements.....	4-4
<b>Practices for Lesson 5: Using SQL Statements within a PL/SQL Block.....</b>	<b>5-1</b>
Practice 5: Using SQL Statements Within a PL/SQL.....	5-2
Solution 5: Using SQL Statements Within a PL/SQL.....	5-4
<b>Practices for Lesson 6: Writing Control Structures.....</b>	<b>6-1</b>
Practice 6: Writing Control Structures .....	6-2
Solution 6: Writing Control Structures .....	6-4
<b>Practices for Lesson 7: Working with Composite Data Types.....</b>	<b>7-1</b>
Practice 7: Working with Composite Data Types .....	7-2
Solution 7: Working with Composite Data Types .....	7-5
<b>Practices for Lesson 8: Using Explicit Cursors .....</b>	<b>8-1</b>
Practice 8-1: Using Explicit Cursors .....	8-2
Solution 8-1: Using Explicit Cursors.....	8-5
Practice 8-2: Using Explicit Cursors: Optional .....	8-10
Solution 8-2: Using Explicit Cursors: Optional .....	8-11
<b>Practices for Lesson 9: Handling Exceptions.....</b>	<b>9-1</b>
Practice 9-1: Handling Predefined Exceptions.....	9-2
Solution 9-1: Handling Predefined Exceptions.....	9-4
Practice 9-2: Handling Standard Oracle Server Exceptions.....	9-6
Solution 9-2: Handling Standard Oracle Server Exceptions.....	9-7
<b>Practices for Lesson 10: Introducing Stored Procedures and Functions.....</b>	<b>10-1</b>
Practice 10: Creating and Using Stored Procedures.....	10-2
Solution 10: Creating and Using Stored Procedures.....	10-4

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## Security Credentials

## Security Credentials

---

**username:ora41**

**password:ora41**

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license to use this Student Guide.

# 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.
2. Create a database connection by using the following information (**Hint:** Select the Save Password check box):
  - a. Connection Name: MyConnection
  - b. Username: ora41
  - c. Password: (refer to security credential document)
  - d. Hostname: localhost
  - e. Port: 1521
  - f. Service name: pdborcl
3. Test the new connection. If the Status is Success, connect to the database by using this new connection.
  - a. In the Database Connection window, click the Test button.  
**Note:** The connection status appears in the lower-left corner of the window.
  - b. If the Status is Success, click the Connect button.
4. Browse the structure of the `EMPLOYEES` table and display its data.
  - a. Expand the MyConnection connection by clicking the plus symbol next to it.
  - b. Expand Tables by clicking the plus symbol next to it.
  - c. Display the structure of the `EMPLOYEES` table.
5. Use the Data tab to view the data in the `EMPLOYEES` table.
6. 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 the Run Script (F5) icons to execute the `SELECT` statement. Review the results of both methods of executing the `SELECT` statement 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.
7. From the SQL Developer menu, select Tools > Preferences. The Preferences window appears.
8. Select Database > Worksheet Parameters. In the “Select default path to look for scripts” text box, use the Browse icon to select the `/home/oracle/labs/plsf` directory. This directory contains the code example scripts, lab scripts, and practice solution scripts that are used in this course. Then, in the Preferences window, click OK to save the Worksheet Parameter setting.

9. Familiarize yourself with the structure of the `/home/oracle/labs/plsf` directory.
  - a. Select File > Open. The Open window automatically selects the `.../plsf` directory as your starting location. This directory contains three subdirectories:
    - The `/code_ex` directory contains the code examples that are found in the course materials. Each `.sql` script is associated with a particular page in the lesson.
    - The `/labs` directory contains the code that is used in certain lesson practices. You are instructed to run the required script in the appropriate practice.
    - The `/soln` directory contains the solutions for each practice. Each `.sql` script is numbered with the associated `practice_exercise` reference.
  - b. You can also use the Files tab to navigate through the directories to open the script files.
  - c. Using the Open window, and the Files tab, navigate through the directories and open a script file without executing the code.
  - d. Close the SQL Worksheet.

## Solution 1-1: Getting Started

1. Start SQL Developer.  
Double Click the SQL Developer icon on your desktop.

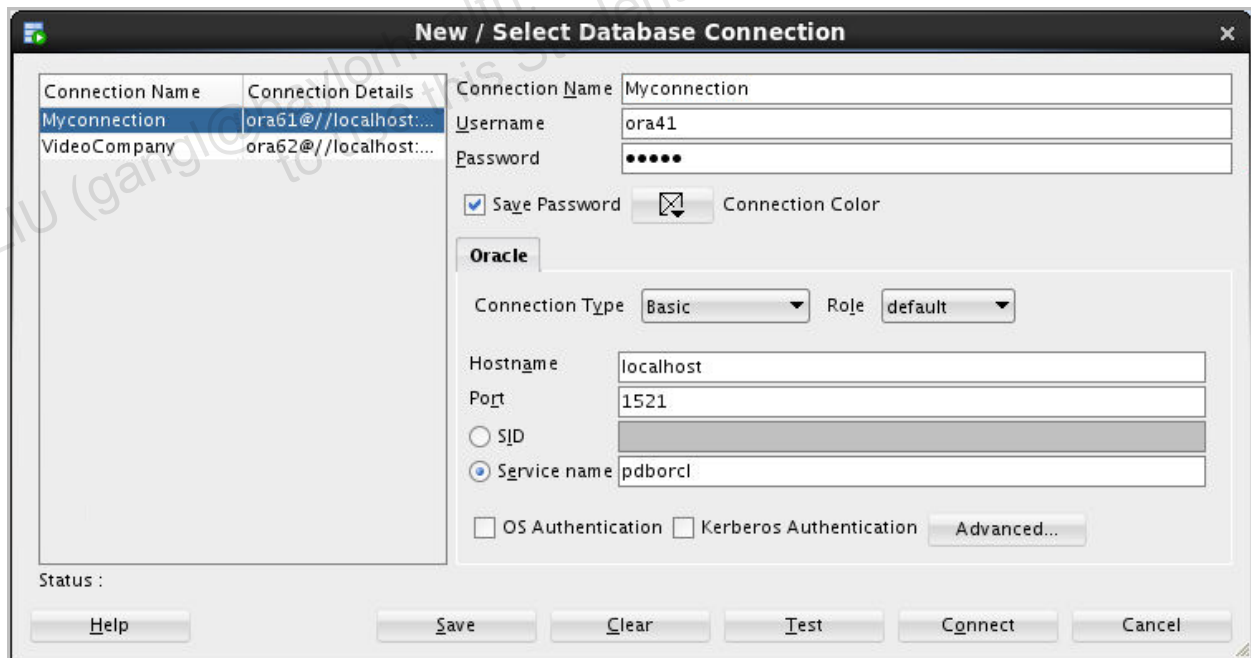


2. Create a database connection by using the following information (**Hint:** Select the Save Password check box):
  - a. Connection Name: MyConnection
  - b. Username: ora41
  - c. Password: refer to the security credential document
  - d. Hostname: localhost
  - e. Port: 1521
  - f. Service name : pdborcl

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

Result: The New/Select Database Connection window appears.

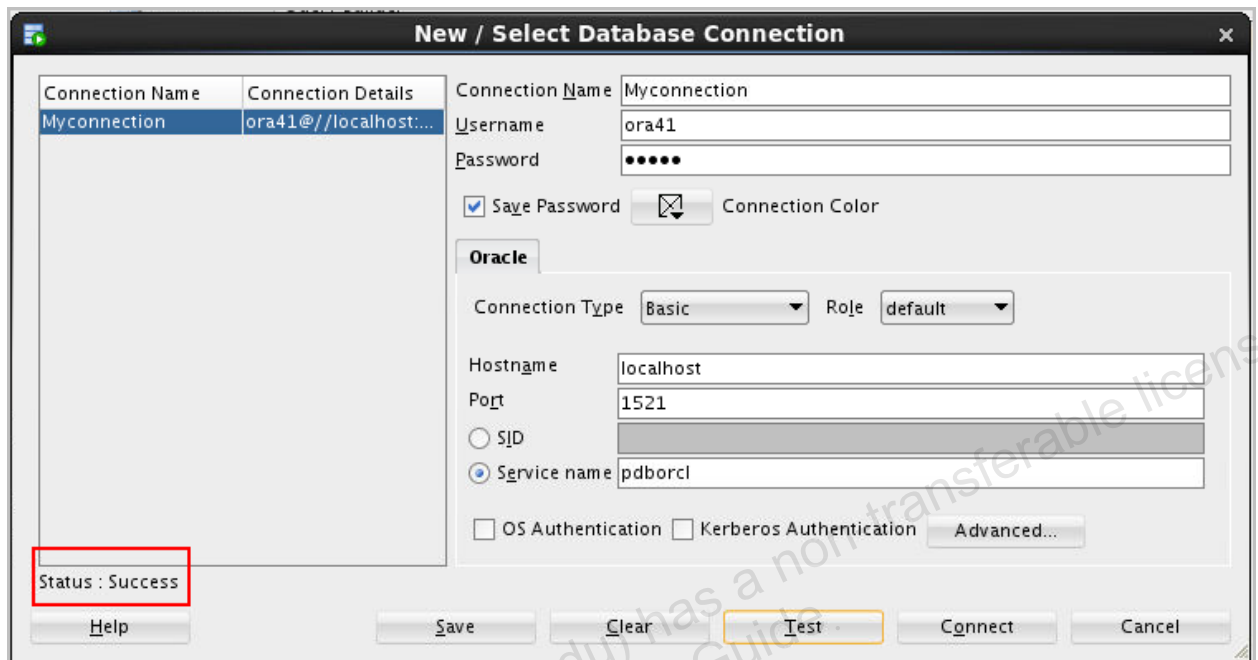
Use the preceding information to create the new database connection. In addition, select the Save Password check box. For example:



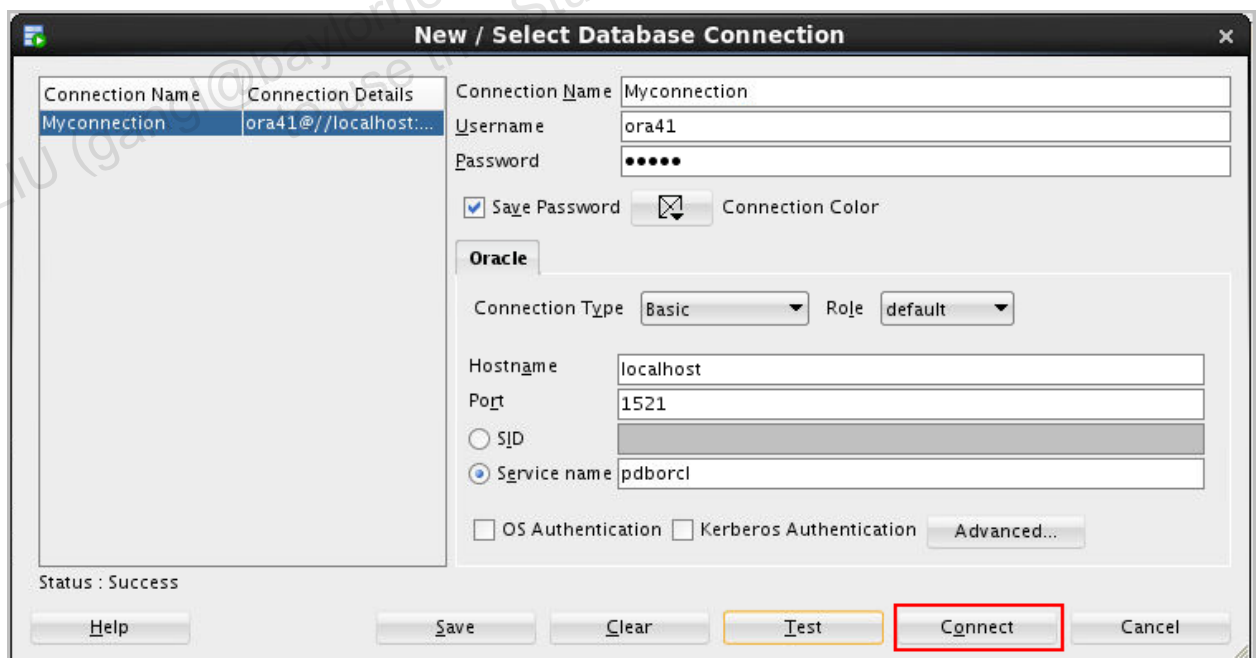
3. Test the new connection. If the Status is Success, connect to the database by using this new connection.

- a. In the Database Connection window, click the Test button.

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



- b. 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.

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

- Expand the MyConnection connection by clicking the plus symbol next to it.
- Expand Tables by clicking the plus symbol next to it.
- 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:

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 EMPLOYEE_ID	NUMBER(6,0)	No	(null)		1 Primary key of employees ta
2 FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)		2 First name of the employee.
3 LAST_NAME	VARCHAR2(25 BYTE)	No	(null)		3 Last name of the employee.
4 EMAIL	VARCHAR2(25 BYTE)	No	(null)		4 Email id of the employee
5 PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)		5 Phone number of the employe
6 HIRE_DATE	DATE	No	(null)		6 Date when the employee star
7 JOB_ID	VARCHAR2(10 BYTE)	No	(null)		7 Current job of the employee
8 SALARY	NUMBER(8,2)	Yes	(null)		8 Monthly salary of the emplo
9 COMMISSION_PCT	NUMBER(2,2)	Yes	(null)		9 Commission percentage of th
10 MANAGER_ID	NUMBER(6,0)	Yes	(null)		10 Manager id of the employee;
11 DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)		11 Department id where employe

- Use the Data tab to view the data in the EMPLOYEES table.

Result: The EMPLOYEES table data is displayed as follows:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID
100	Steven	King	SKING	515.123.4567	17-JUN-11	AD_PRES
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-09	AD_VP
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-09	AD_VP
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-14	IT_PROG
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-15	IT_PROG
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-13	IT_PROG
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-14	IT_PROG
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-15	IT_PROG
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-10	FI_MGR
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-10	FI_ACCOUNT

- 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:

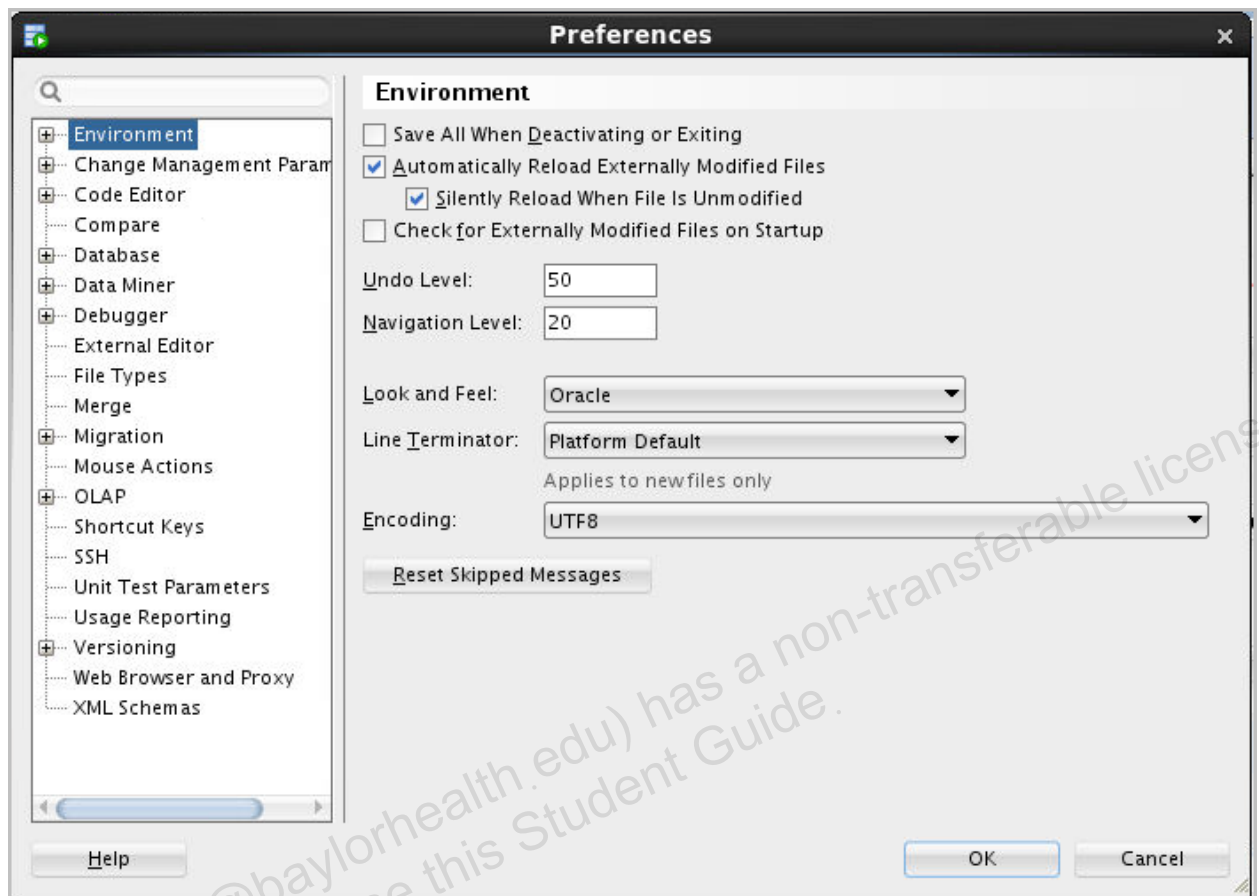
The screenshot shows the SQL Developer interface with a connection named 'MyConnection' and a table named 'EMPLOYEES'. The 'Query Builder' tab is active, displaying the following SQL query:

```
1 select last_name, salary
2 from employees
3 where salary > 10000;
```

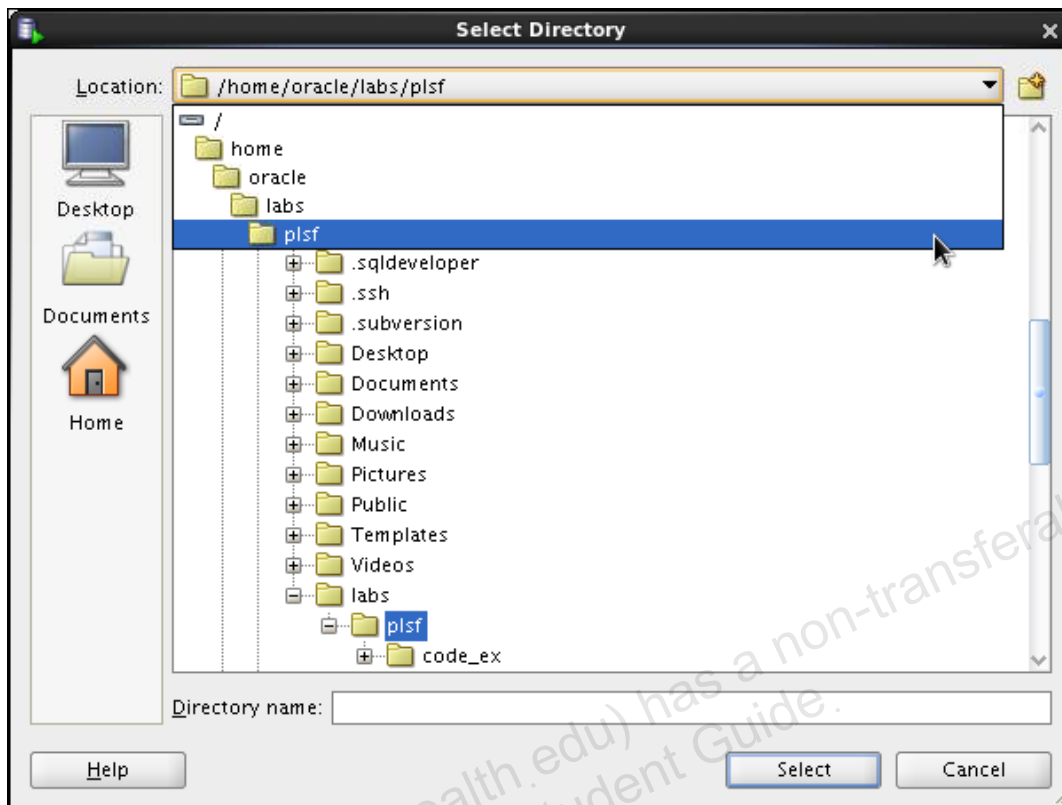
Below the query, the 'Query Result' tab shows the results of the query. The status bar indicates 'All Rows Fetched: 15 in 0.004 seconds'. The results are displayed in a table with two columns: 'LAST\_NAME' and 'SALARY'.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Greenberg	12008
5	Raphaely	11000
6	Russell	14000
7	Partners	13500
8	Errazuriz	12000
9	Cambrault	11000
10	Zlotkey	10500
11	Vishney	10500
12	Ozer	11500
13	Abel	11000
14	Hartstein	13000
15	Higgins	12008

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



8. Select Database > Worksheet Parameters. In the “Select default path to look for scripts” text box, use the Browse icon to select the /home/oracle/labs/plsf directory.

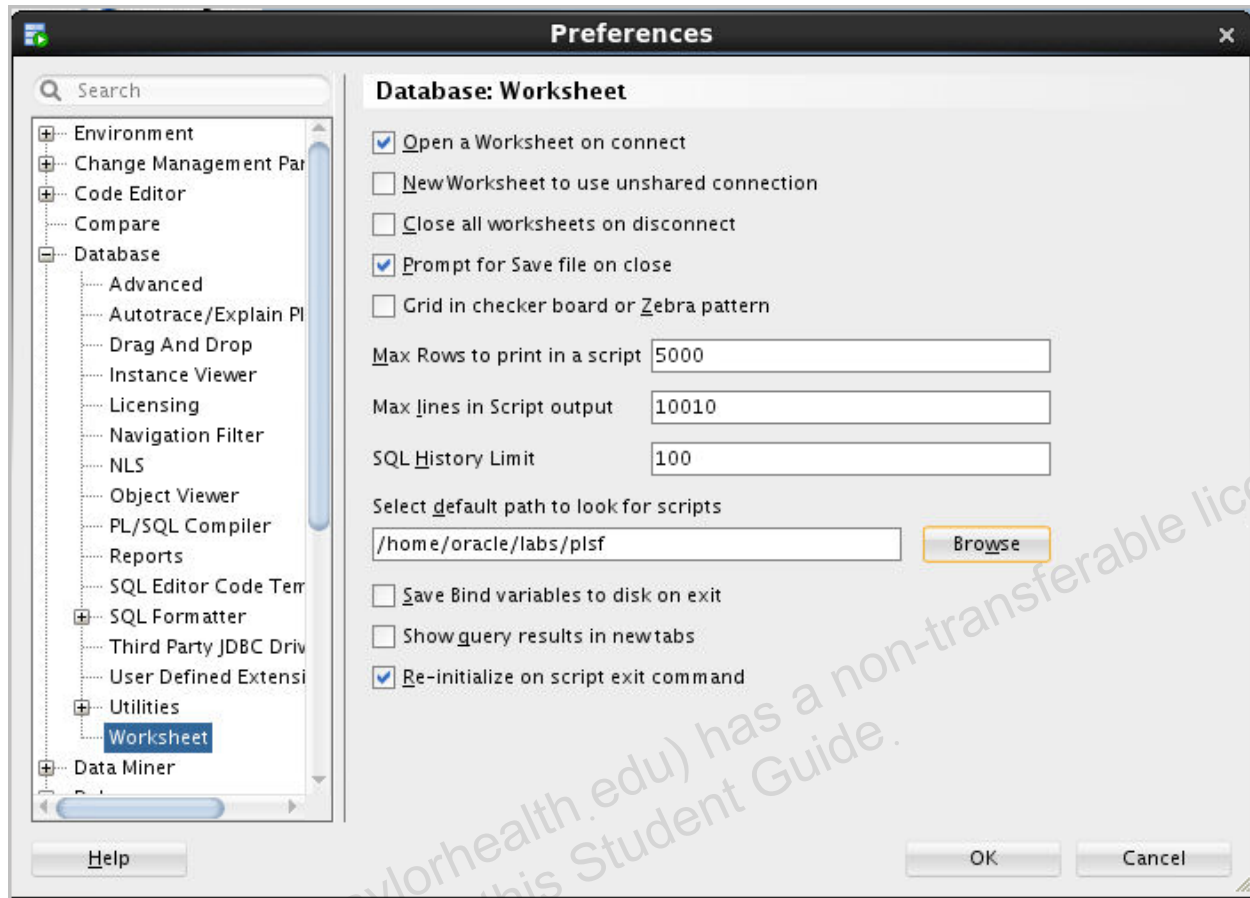


This directory contains the code example scripts, lab scripts, and practice solution scripts that are used in this course.

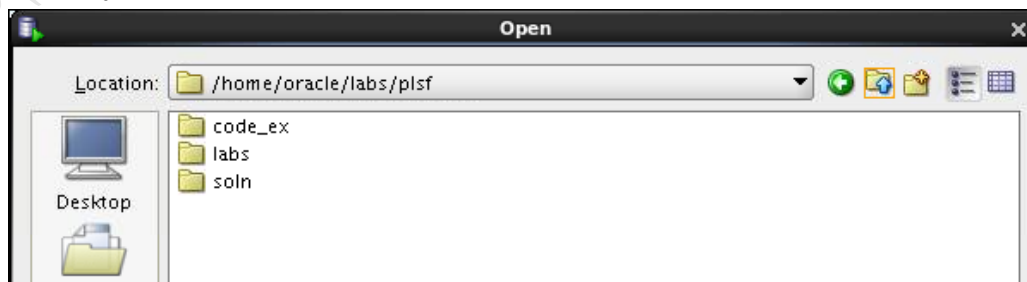
Click Select to choose the directory.



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

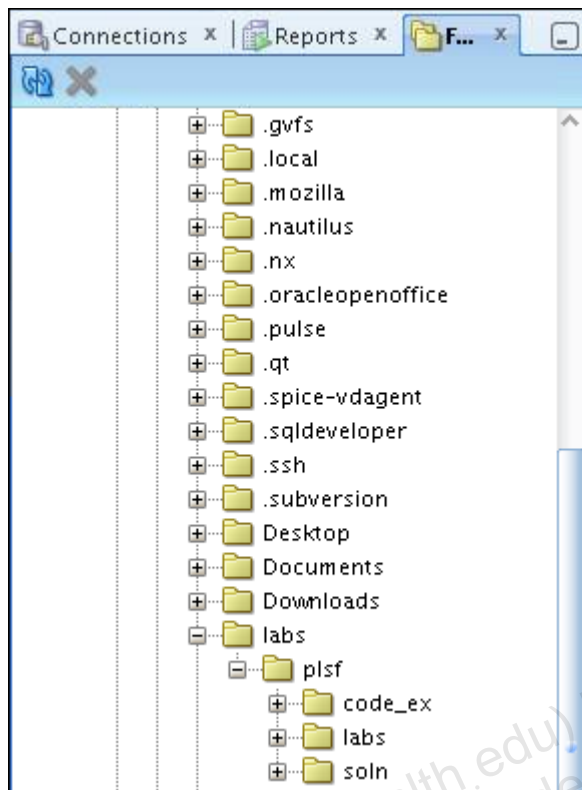


9. Familiarize yourself with the structure of the `/home/oracle/labs/plsf` directory.
  - a. Select File > Open. Navigate to the `/home/oracle/labs/plsf` directory. This directory contains three subdirectories:



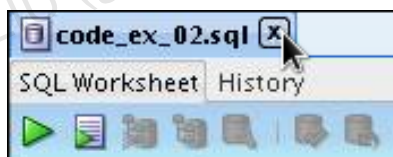
- The `/code_ex` directory contains the code examples found in the course materials. Each `.sql` script is associated with a particular page in the lesson.
- The `/labs` directory contains the code that is used in certain lesson practices. You are instructed to run the required script in the appropriate practice.
- The `/soln` directory contains the solutions for each practice. Each `.sql` script is numbered with the associated practice\_exercise reference.

- b. You can also use the Files tab to navigate through the directories to open the script files.



- c. Using the Open window, and the Files tab, navigate through the directories and open a script file without executing the code.
- d. Close the SQL Worksheet.

To close any SQL Worksheet tab, click X on the tab, as shown here:



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

### **Chapter 2**

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

---

### Lesson Overview

The `/home/oracle/labs/plsf/labs` folder is the working directory where you save the scripts that you create.

The solutions for all the practices are in the `/home/oracle/labs/plsf/soln` folder.

## Practice 2: Introduction to PL/SQL

---

1. Which of the following PL/SQL blocks execute successfully?
  - a. 

```
BEGIN
  commit;
END;
```
  - b. 

```
DECLARE
  v_amount INTEGER(10);
END;
```
  - c. 

```
DECLARE
BEGIN
END;
```
  - d. 

```
SET SERVEROUTPUT ON;
DECLARE
  v_amount INTEGER(10);
BEGIN
  DBMS_OUTPUT.PUT_LINE(v_amount);
END;
```
2. 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?

- a. `BEGIN`  
`commit;`  
`END;`
- b. `DECLARE`  
`v_amount INTEGER(10);`  
`END;`
- c. `DECLARE`  
`BEGIN`  
`END;`
- d. `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.

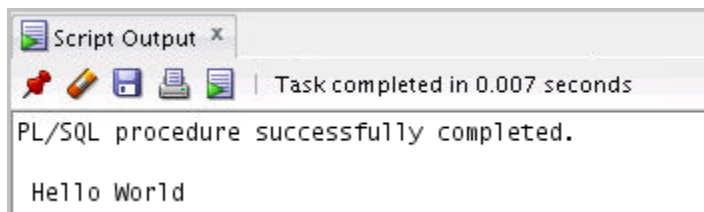
**The block in d executes successfully.**

2. 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:
  - a. today
  - b. last\_name
  - c. today's\_date
  - d. Number\_of\_days\_in\_February\_this\_year
  - e. Isleap\$year
  - f. #number
  - g. NUMBER#
  - h. number1to7
2. Identify valid and invalid variable declaration and initialization:
  - a. number\_of\_copies            PLS\_INTEGER;
  - b. PRINTER\_NAME                constant VARCHAR2(10);
  - c. deliver\_to                    VARCHAR2(10):=Johnson;
  - d. by\_when                        DATE:= CURRENT\_DATE+1;
3. 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;
```

- a. The block executes successfully and prints "fernandez."
- b. The block produces an error because the `fname` variable is used without initializing.
- c. The block executes successfully and prints "null fernandez."
- d. The block produces an error because you cannot use the `DEFAULT` keyword to initialize a variable of type `VARCHAR2`.
- e. The block produces an error because the `v_fname` variable is not declared.

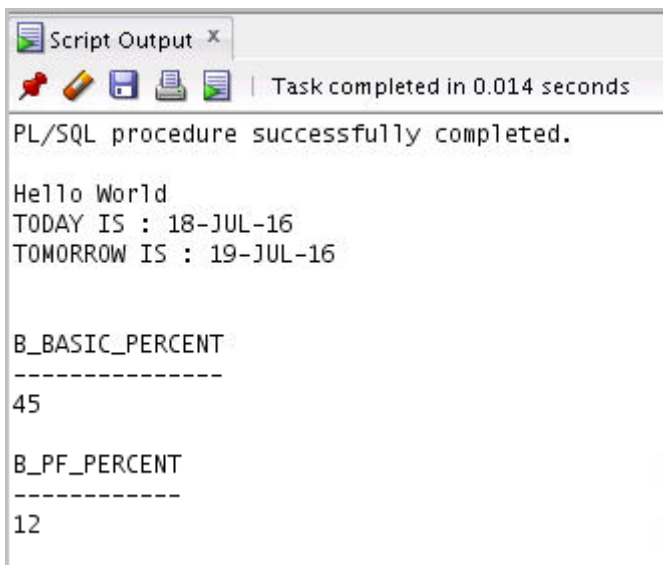


4. Modify an existing anonymous block and save it as a new script.
  - a. Open the `lab_02_02_soln.sql` script, which you created in Practice 2 titled "Introduction to PL/SQL."
  - b. In this PL/SQL block, declare the following variables:
    - 1) `v_today` of type `DATE`. Initialize `today` with `SYSDATE`.
    - 2) `v_tomorrow` of type `today`. Use the `%TYPE` attribute to declare this variable.
  - c. In the executable section:
    - 1) Initialize the `v_tomorrow` variable with an expression, which calculates tomorrow's date (add one to the value in `today`)
    - 2) Print the value of `v_today` and `v_tomorrow` after printing "Hello World"
  - d. 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):

```
PL/SQL procedure successfully completed.

Hello World
TODAY IS : 18-JUL-16
TOMORROW IS : 19-JUL-16
```

5. Edit the `lab_03_04_soln.sql` script.
  - a. Add code to create two bind variables named `b_basic_percent` and `b_pf_percent`. Both bind variables are of type `NUMBER`.
  - b. In the executable section of the PL/SQL block, assign the values 45 and 12 to `b_basic_percent` and `b_pf_percent`, respectively.
  - c. Terminate the PL/SQL block with `/` and display the value of the bind variables by using the `PRINT` command.
  - d. Execute and save your script as `lab_03_05_soln.sql`. The sample output is as follows:



```
Script Output x
Task completed in 0.014 seconds

PL/SQL procedure successfully completed.

Hello World
TODAY IS : 18-JUL-16
TOMORROW IS : 19-JUL-16

B_BASIC_PERCENT
-----
45

B_PF_PERCENT
-----
12
```

## Solution 3: Declaring PL/SQL Variables

1. Identify valid and invalid identifiers:
  - a. today **Valid**
  - b. last\_name **Valid**
  - c. today's\_date **Invalid** – character “'” not allowed
  - d. Number\_of\_days\_in\_February\_this\_year **Invalid** – Too long
  - e. Isleap\$year **Valid**
  - f. #number **Invalid** – Cannot start with “#”
  - g. NUMBER# **Valid**
  - h. number1to7 **Valid**
2. Identify valid and invalid variable declaration and initialization:
  - a. number\_of\_copies      PLS\_INTEGER; **Valid**
  - b. PRINTER\_NAME      constant VARCHAR2(10); **Invalid**
  - c. deliver\_to      VARCHAR2(10) := Johnson; **Invalid**
  - d. 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.*

3. 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;
```

- a. The block executes successfully and prints “fernandez.”
  - b. The block produces an error because the `fname` variable is used without initializing.
  - c. The block executes successfully and prints “null fernandez.”
  - d. The block produces an error because you cannot use the `DEFAULT` keyword to initialize a variable of type `VARCHAR2`.
  - e. The block produces an error because the `v_fname` variable is not declared.
- a. The block will execute successfully and print “fernandez.”**

4. Modify an existing anonymous block and save it as a new script.

- a. Open the `lab_02_02_soln.sql` script, which you created in Practice 2 titled "Introduction to PL/SQL."
- b. In the PL/SQL block, declare the following variables:
  - 1) Variable `v_today` of type `DATE`. Initialize today with `SYSDATE`.

```
DECLARE
    v_today DATE:=SYSDATE;
```

- 2) Variable `v_tomorrow` of type today. Use the `%TYPE` attribute to declare this variable.

```
v_tomorrow v_today%TYPE;
```

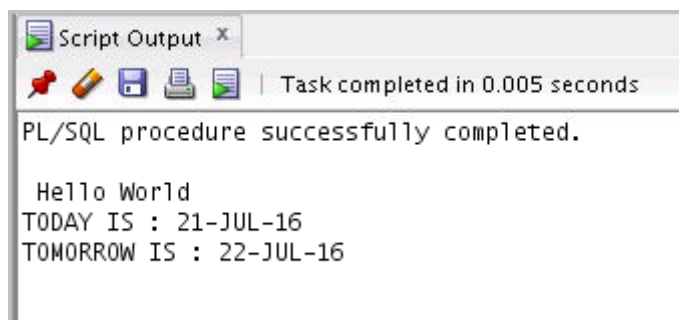
In the executable section:

- 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;
```

- c. 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):



5. Edit the lab\_03\_04\_soln.sql script.
- Add code to create two bind variables named b\_basic\_percent and b\_pf\_percent. Both bind variables are of type NUMBER.

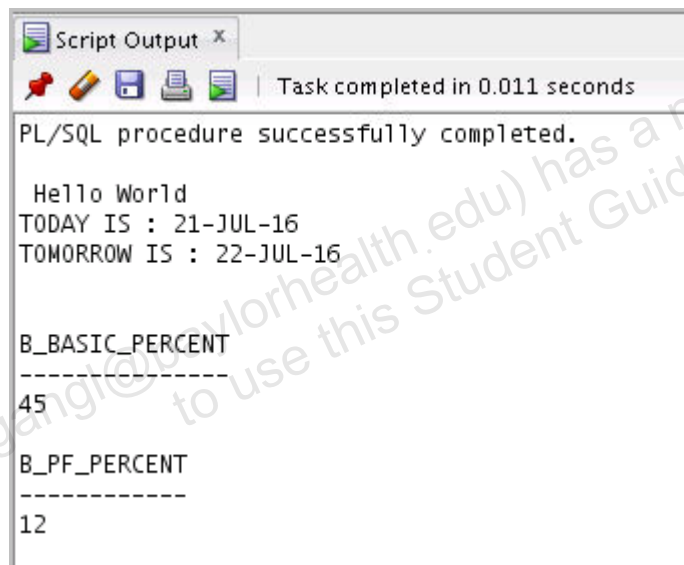
```
VARIABLE b_basic_percent NUMBER  
VARIABLE b_pf_percent NUMBER
```

- 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;
```

- 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
```



Script Output x

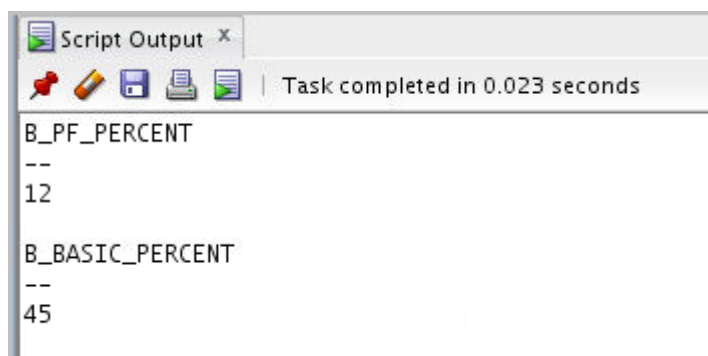
Task completed in 0.011 seconds

```
PL/SQL procedure successfully completed.  
  
Hello World  
TODAY IS : 21-JUL-16  
TOMORROW IS : 22-JUL-16  
  
B_BASIC_PERCENT  
-----  
45  
  
B_PF_PERCENT  
-----  
12
```

OR

```
PRINT
```

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



Script Output x

Task completed in 0.023 seconds

```
B_PF_PERCENT  
--  
12  
  
B_BASIC_PERCENT  
--  
45
```

## **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;
  END;
  v_weight := v_weight + 1;
  v_message := v_message || ' is in stock';
  v_new_locn := 'Western ' || v_new_locn;
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.
  - a. The value of `v_weight` at position 1 is:
  - b. The value of `v_new_locn` at position 1 is:
  - c. The value of `v_weight` at position 2 is:
  - d. The value of `v_message` at position 2 is:
  - e. 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;
```

2. In the preceding PL/SQL block, determine the value and data type of each of the following cases:
  - a. The value of `v_customer` in the nested block is:
  - b. The value of `v_name` in the nested block is:
  - c. The value of `v_credit_rating` in the nested block is:
  - d. The value of `v_customer` in the main block is:
  - e. The value of `v_name` in the main block is:
  - f. The value of `v_credit_rating` in the main block is:
3. 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:
  - a. Use single-line comment syntax to comment the lines that create the bind variables, and turn on `SERVEROUTPUT`.
  - b. Use multiple-line comments in the executable section to comment the lines that assign values to the bind variables.
  - c. 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
  - d. 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;
```

- e. 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.
- f. 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.
- g. Execute and save your script as `lab_04_03_soln.sql`. The sample output is as follows:

```
PL/SQL procedure successfully completed.

Hello John
YOUR SALARY IS : 8200
YOUR CONTRIBUTION TOWARDS PF:
      442.8
```

## 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:
  - a. The value of `v_weight` at position 1 is:  
**2**  
**The data type is NUMBER.**
  - b. The value of `v_new_locn` at position 1 is:  
**Western Europe**  
**The data type is VARCHAR2.**
  - c. The value of `v_weight` at position 2 is:  
**601**  
**The data type is NUMBER.**
  - d. The value of `v_message` at position 2 is:  
**Product 10012 is in stock**  
**The data type is VARCHAR2.**
  - e. 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;

```

2. In the preceding PL/SQL block, determine the value and data type for each of the following cases:
  - a. The value of v\_customer in the nested block is:  
**201**  
**The data type is NUMBER.**
  - b. The value of v\_name in the nested block is:  
**Unisports**  
**The data type is VARCHAR2.**
  - c. The value of v\_credit\_rating in the nested block is:  
**GOOD**  
**The data type is VARCHAR2.**
  - d. The value of v\_customer in the main block is:  
**Womansport**  
**The data type is VARCHAR2.**
  - e. 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.**
  - f. The value of v\_credit\_rating in the main block is:  
**EXCELLENT**  
**The data type is VARCHAR2.**
3. 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:
  - a. 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

```

- b. 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;*/
```

- c. 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);
```

- d. 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;
```

- e. 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
```

- f. 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;
```

- g. Execute and save your script as `lab_04_03_soln.sql`. The sample output is as follows:

```
PL/SQL procedure successfully completed.
```

```
Hello John  
YOUR SALARY IS : 8200  
YOUR CONTRIBUTION TOWARDS PF:  
442.8
```

GANG LIU (gangl@baylorhealth.edu) has a non-transferable license to use this Student Guide.

## **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.
  - a. Declare a variable `v_max_deptno` of type `NUMBER` in the declarative section.
  - b. Start the executable section with the `BEGIN` keyword and include a `SELECT` statement to retrieve the maximum `department_id` from the `departments` table.
  - c. Display `v_max_deptno` and end the executable block.
  - d. Execute and save your script as `lab_05_01_soln.sql`. The sample output is as follows:

```
PL/SQL procedure successfully completed.  
  
The maximum department_id is : 270  
|
```

2. Modify the PL/SQL block that you created in step 1 to insert a new department into the `departments` table.
  - a. 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.
  - b. 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`.
  - c. 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`.
  - d. Use the SQL attribute `SQL%ROWCOUNT` to display the number of rows that are affected.
  - e. 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.
  - f. Execute and save your script as `lab_05_02_soln.sql`. The sample output is as follows:

```
PL/SQL procedure successfully completed.  
  
The maximum department_id is : 270  
SQL%ROWCOUNT gives 1
```

Query Result x			
SQL   All Rows Fetched: 1 in 0.004 seconds			
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	280 Education	(null)	(null)

3. 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`)

- 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`).
- 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.
- Include a `DELETE` statement to delete the department that you added.
- Execute and save your script as `lab_05_03_soln.sql`. The sample output is as follows:

Script Output x Query Result x  
Task completed in 0.425 seconds  
PL/SQL procedure successfully completed.  
>>Query Run In:Query Result  
1 row deleted.

Script Output x Query Result x			
SQL   All Rows Fetched: 1 in 0.172 seconds			
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	280 Education	(null)	3000

## 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.
  - a. Declare a variable `v_max_deptno` of type `NUMBER` in the declarative section.

```
DECLARE
    v_max_deptno NUMBER;
```

- b. 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;
```

- c. Display `v_max_deptno` and end the executable block.

```
DBMS_OUTPUT.PUT_LINE('The maximum department_id is : ' ||
    v_max_deptno);
END;
```

- d. Execute and save your script as `lab_05_01_soln.sql`. The sample output is as follows:

```
PL/SQL procedure successfully completed.
The maximum department_id is : 270
|
```

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

- a. 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;
```

- b. 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;
```



- c. 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);
```

- d. Use the SQL attribute `SQL%ROWCOUNT` to display the number of rows that are affected.

```
DBMS_OUTPUT.PUT_LINE (' SQL%ROWCOUNT gives ' || SQL%ROWCOUNT);
...
```

- e. 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;
```

- f. Execute and save your script as `lab_05_02_soln.sql`. The sample output is as follows:

```
PL/SQL procedure successfully completed.
The maximum department_id is : 270
SQL%ROWCOUNT gives 1
```

Query Result x				
SQL   All Rows Fetched: 1 in 0.004 seconds				
	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	280	Education	(null)	(null)

3. 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`)
- a. 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;
```

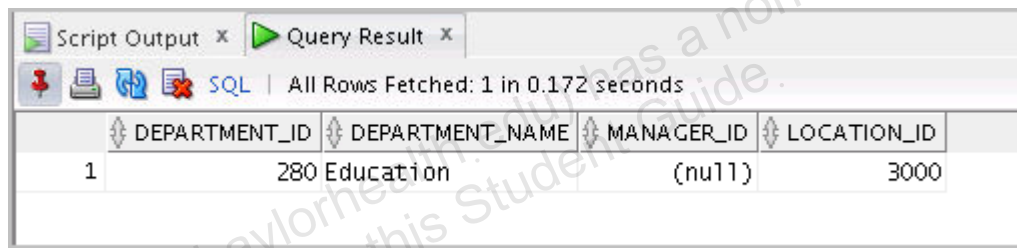
- b. End the executable block with the `END` keyword. Terminate the PL/SQL block with `/` and include a `SELECT` statement to display the department that you updated.

```
END;  
/  
SELECT * FROM departments WHERE department_id=280;
```

- c. Include a `DELETE` statement to delete the department that you added.

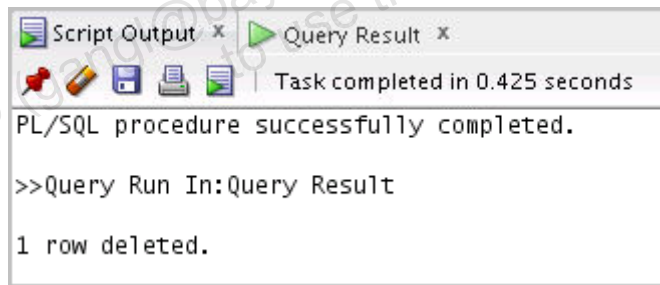
```
DELETE FROM departments WHERE department_id=280;
```

- d. Execute and save your script as `lab_05_03_soln.sql`. The sample output is as follows:



The screenshot shows the 'Query Result' window in SQL Developer. It displays a single row of data from the 'departments' table. The columns are DEPARTMENT\_ID, DEPARTMENT\_NAME, MANAGER\_ID, and LOCATION\_ID. The values in the row are 1, 280 Education, (null), and 3000 respectively. The status bar indicates 'All Rows Fetched: 1 in 0.172 seconds'.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	280 Education	(null)	3000



The screenshot shows the 'Script Output' window in SQL Developer. It displays the message 'PL/SQL procedure successfully completed.' followed by 'Task completed in 0.425 seconds'. Below this, it shows the prompt '>>Query Run In:Query Result' and the message '1 row deleted.'.

```
PL/SQL procedure successfully completed.  
Task completed in 0.425 seconds  
>>Query Run In:Query Result  
1 row deleted.
```

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

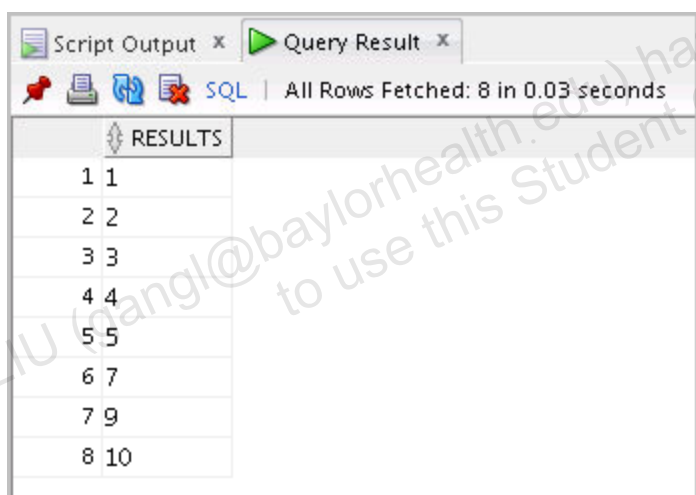
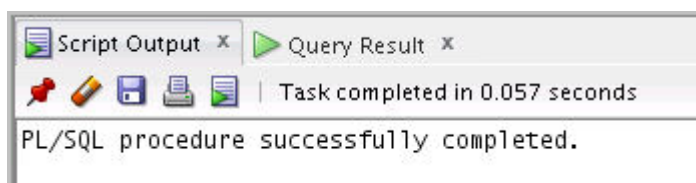
### **Chapter 6**

## 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.
  - a. Insert the numbers 1 through 10, excluding 6 and 8.
  - b. Commit before the end of the block.
  - c. Execute a `SELECT` statement to verify that your PL/SQL block worked.

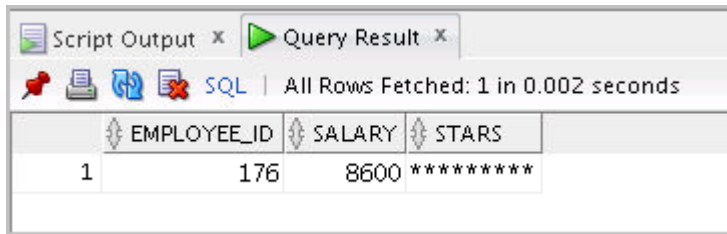
Result: You should see the following output:



	RESULTS
1	1
2	2
3	3
4	4
5	5
6	7
7	9
8	10

2. 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`.
  - a. 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`.
  - b. 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).
  - c. Update the `stars` column for the employee with the string of asterisks. Commit before the end of the block.

- d. Display the row from the `emp` table to verify whether your PL/SQL block has executed successfully.
- e. Execute and save your script as `lab_06_02_soln.sql`. The output is as follows:



The screenshot shows a SQL query result window with two tabs: 'Script Output' and 'Query Result'. The 'Query Result' tab is active, displaying a table with three columns: 'EMPLOYEE\_ID', 'SALARY', and 'STARS'. The table contains one row with the values 1, 176, and 8600, and a seventh column with the value '\*\*\*\*\*'.

	EMPLOYEE_ID	SALARY	STARS	
1	176	8600	*****	

## 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.
  - a. Insert the numbers 1 through 10, excluding 6 and 8.
  - b. 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;
/
```

- c. Execute a `SELECT` statement to verify that your PL/SQL block worked.

```
SELECT * FROM messages;
```

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

The screenshot shows two windows from SQL Developer. The top window, titled 'Script Output', shows the message 'PL/SQL procedure successfully completed.' with a task completion time of 0.057 seconds. The bottom window, titled 'Query Result', shows the results of a SELECT statement. It displays a table with 8 rows and 1 column, where the values are 1 through 10, excluding 6 and 8. The window also shows 'All Rows Fetched: 8 in 0.03 seconds'.

RESULTS
1 1
2 2
3 3
4 4
5 5
6 7
7 9
8 10

2. 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`.

- a. 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;
```

- b. 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;
```

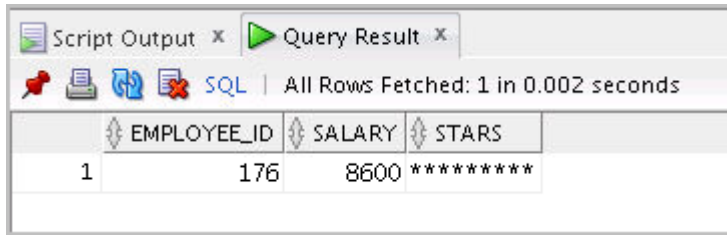
- c. Update the `stars` column for the employee with the string of asterisks. Commit before the end of the block.

```
UPDATE emp SET stars = v_asterisk
WHERE employee_id = v_empno;
COMMIT;
END;
/
```

- d. 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;
```

- e. Execute and save your script as `lab_06_02_soln.sql`. The output is as follows:



The screenshot shows a SQL query result window with two tabs: 'Script Output' and 'Query Result'. The 'Query Result' tab is active, displaying a table with three columns: 'EMPLOYEE\_ID', 'SALARY', and 'STARS'. The table contains one row of data with values 1, 176, and 8600, and a star rating of eight asterisks. The status bar indicates 'All Rows Fetched: 1 in 0.002 seconds'.

EMPLOYEE_ID	SALARY	STARS
1	176	8600 *****



## **Practices for Lesson 7: Working with Composite Data Types**

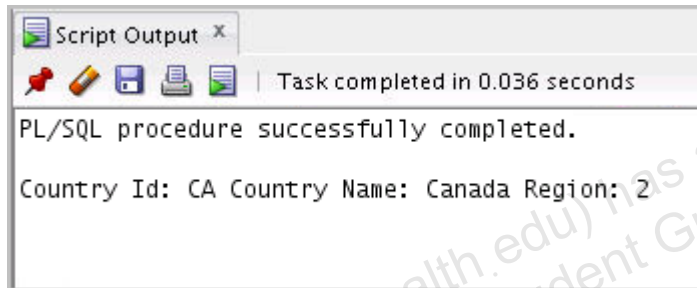
### **Chapter 7**

## 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.
  - a. Declare a PL/SQL record based on the structure of the `COUNTRIES` table.
  - b. Declare a variable `v_countryid`. Assign `CA` to `v_countryid`.
  - c. In the declarative section, use the `%ROWTYPE` attribute and declare the `v_country_record` variable of type `countries`.
  - d. 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:

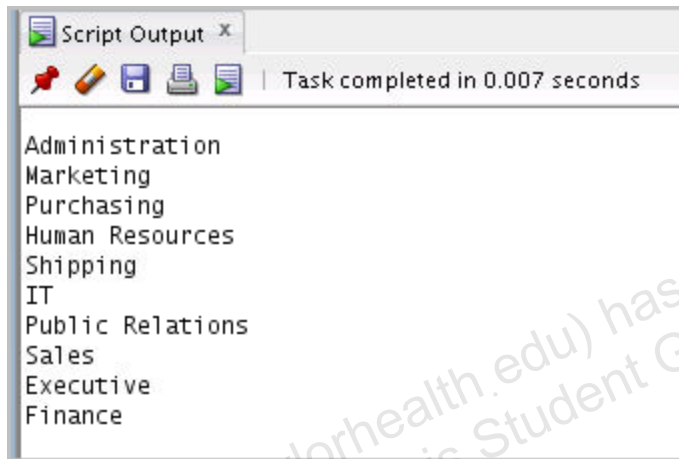


- e. You may want to execute and test the PL/SQL block for countries with the IDs `DE`, `UK`, and `US`.
2. Create a PL/SQL block to retrieve the names of some departments from the `DEPARTMENTS` table and print each department name on the screen, incorporating an associative array. Save the script as `lab_07_02_soln.sql`.
  - a. 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.
  - b. Declare two variables: `f_loop_count` and `v_deptno` of type `NUMBER`. Assign 10 to `f_loop_count` and 0 to `v_deptno`.
  - c. 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

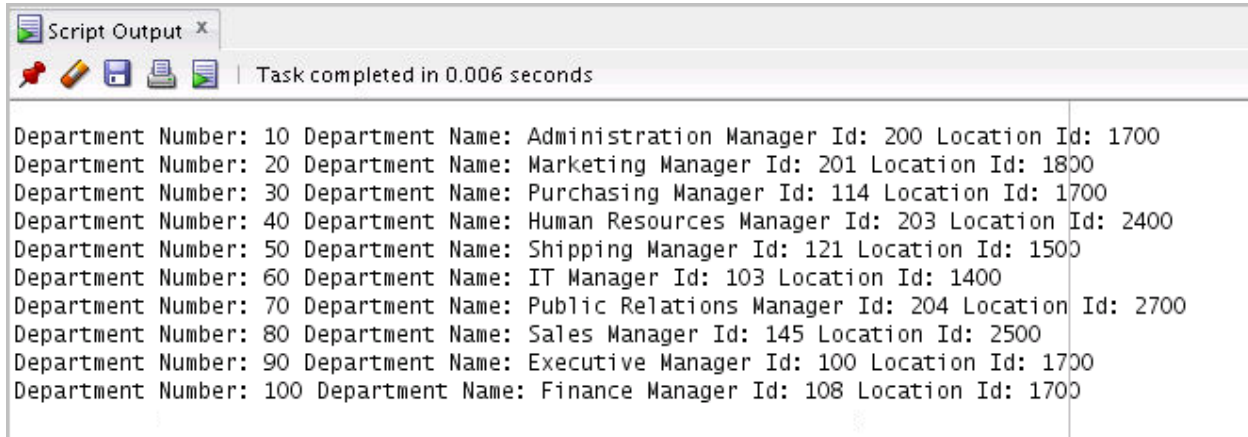
<b>50</b>	<b>Shipping</b>
<b>60</b>	<b>IT</b>
<b>70</b>	<b>Public Relations</b>
<b>80</b>	<b>Sales</b>
<b>90</b>	<b>Executive</b>
<b>100</b>	<b>Finance</b>

- d. Using another loop, retrieve the department names from the associative array and display them.
- e. Execute and save your script as `lab_07_02_soln.sql`. The output is as follows:



3. 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.
  - a. Load the `lab_07_02_soln.sql` script.
  - b. 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.
  - c. Modify the `SELECT` statement to retrieve all department information currently in the `DEPARTMENTS` table and store it in the associative array.
  - d. Using another loop, retrieve the department information from the associative array and display the information.

The sample output is as follows:



Department Number: 10	Department Name: Administration	Manager Id: 200	Location Id: 1700
Department Number: 20	Department Name: Marketing	Manager Id: 201	Location Id: 1800
Department Number: 30	Department Name: Purchasing	Manager Id: 114	Location Id: 1700
Department Number: 40	Department Name: Human Resources	Manager Id: 203	Location Id: 2400
Department Number: 50	Department Name: Shipping	Manager Id: 121	Location Id: 1500
Department Number: 60	Department Name: IT	Manager Id: 103	Location Id: 1400
Department Number: 70	Department Name: Public Relations	Manager Id: 204	Location Id: 2700
Department Number: 80	Department Name: Sales	Manager Id: 145	Location Id: 2500
Department Number: 90	Department Name: Executive	Manager Id: 100	Location Id: 1700
Department Number: 100	Department Name: Finance	Manager Id: 108	Location Id: 1700

## Solution 7: Working with Composite Data Types

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

```
SET SERVEROUTPUT ON

SET VERIFY OFF
DECLARE
    v_countryid varchar2(20) := 'CA';
```

- c. In the declarative section, use the %ROWTYPE attribute and declare the v\_country\_record variable of type countries.

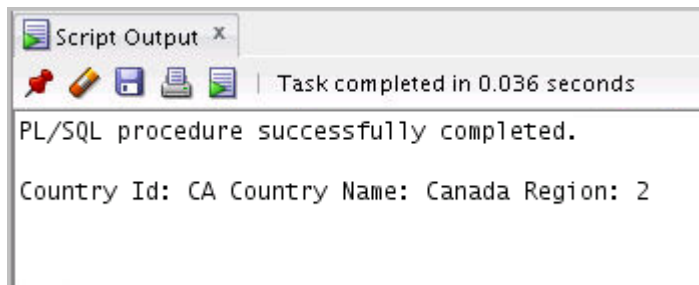
```
v_country_record countries%ROWTYPE;
```

- d. 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:



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

2. 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`.

- a. 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;
```

- b. 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;
```

- c. 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.

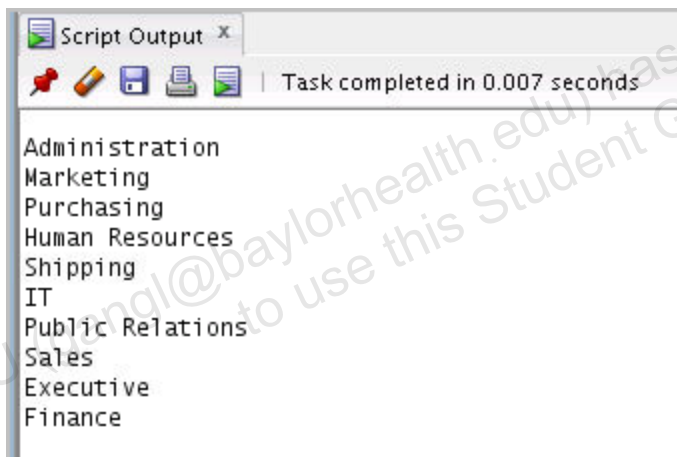
DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive
100	Finance

```
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;
```

- d. 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;
```

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



3. 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.
- Load the lab\_07\_02\_soln.sql script.
  - 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;

```

- c. 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;

```

- d. 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:

Script Output x	
Task completed in 0.006 seconds	
Department Number: 10	Department Name: Administration Manager Id: 200 Location Id: 1700
Department Number: 20	Department Name: Marketing Manager Id: 201 Location Id: 1800
Department Number: 30	Department Name: Purchasing Manager Id: 114 Location Id: 1700
Department Number: 40	Department Name: Human Resources Manager Id: 203 Location Id: 2400
Department Number: 50	Department Name: Shipping Manager Id: 121 Location Id: 1500
Department Number: 60	Department Name: IT Manager Id: 103 Location Id: 1400
Department Number: 70	Department Name: Public Relations Manager Id: 204 Location Id: 2700
Department Number: 80	Department Name: Sales Manager Id: 145 Location Id: 2500
Department Number: 90	Department Name: Executive Manager Id: 100 Location Id: 1700
Department Number: 100	Department Name: Finance Manager Id: 108 Location Id: 1700



## **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:
    - a. 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).
    - b. 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`.
    - c. 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.”
    - d. 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

2. 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.
  - a. 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`.
  - b. 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.
  - c. Declare variables to hold the values retrieved from each cursor. Use the `%TYPE` attribute while declaring variables.
  - d. 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.
  - e. 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`.
- f. End the first loop and close `c_dept_cursor`. Then end the executable section.

g. Execute the script. The sample output is as follows:

```
Script Output x
Task completed in 0.008 seconds

PL/SQL procedure successfully completed.

Department Number : 10  Department Name : Administration
-----
Department Number : 20  Department Name : Marketing
-----
Department Number : 30  Department Name : Purchasing
Raphaely      PU_MAN      07-DEC-10      11000
Khoo          PU_CLERK    18-MAY-11       3100
Baida         PU_CLERK    24-DEC-13       2900
Tobias        PU_CLERK    24-JUL-13       2800
Himuro        PU_CLERK    15-NOV-14       2600
Colmenares    PU_CLERK    10-AUG-15       2500
-----
Department Number : 40  Department Name : Human Resources
-----
Department Number : 50  Department Name : Shipping
-----
Department Number : 60  Department Name : IT
Hunold        IT_PROG     03-JAN-14       9000
Ernst         IT_PROG     21-MAY-15       6000
Austin        IT_PROG     25-JUN-13       4800
Pataballa     IT_PROG     05-FEB-14       4800
Lorentz       IT_PROG     07-FEB-15       4200
-----
Department Number : 70  Department Name : Public Relations
-----
Department Number : 80  Department Name : Sales
-----
Department Number : 90  Department Name : Executive
King          AD_PRE      17-JUN-11      24000
Kochhar       AD_VP       21-SEP-09      17000
De Haan       AD_VP       13-JAN-09      17000
-----
```

## 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:
    - a. 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;
```

- b. 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;
```

- c. 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;
```

- d. 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

2. 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.
- a. 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;
```

- b. 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;
```

- c. 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;
```

- d. 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);
```

- e. 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;
```

- f. End the first loop and close `c_dept_cursor`. Then end the executable section.

```
END LOOP;
CLOSE c_dept_cursor;
END;
```



g. Execute the script. The sample output is as follows:

```

Script Output x
Task completed in 0.008 seconds

PL/SQL procedure successfully completed.

Department Number : 10  Department Name : Administration
-----
Department Number : 20  Department Name : Marketing
-----
Department Number : 30  Department Name : Purchasing
Raphaely    PU_MAN    07-DEC-10    11000
Khoo        PU_CLERK   18-MAY-11     3100
Baida       PU_CLERK   24-DEC-13     2900
Tobias      PU_CLERK   24-JUL-13     2800
Himuro      PU_CLERK   15-NOV-14     2600
Colmenares  PU_CLERK   10-AUG-15     2500
-----
Department Number : 40  Department Name : Human Resources
-----
Department Number : 50  Department Name : Shipping
-----
Department Number : 60  Department Name : IT
Hunold      IT_PROG    03-JAN-14     9000
Ernst       IT_PROG    21-MAY-15     6000
Austin      IT_PROG    25-JUN-13     4800
Pataballa   IT_PROG    05-FEB-14     4800
Lorentz     IT_PROG    07-FEB-15     4200
-----
Department Number : 70  Department Name : Public Relations
-----
Department Number : 80  Department Name : Sales
-----
Department Number : 90  Department Name : Executive
King        AD_PRE     17-JUN-11    24000
Kochhar     AD_VP      21-SEP-09    17000
De Haan     AD_VP      13-JAN-09    17000
-----

```

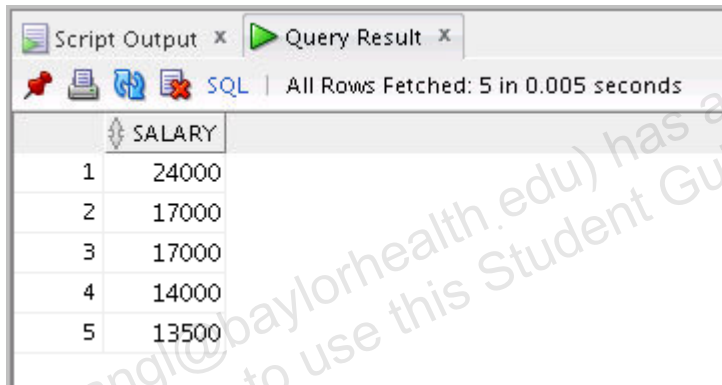
## Practice 8-2: Using Explicit Cursors: Optional

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.

4. 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.



The screenshot shows a 'Query Result' window with a table containing 5 rows of salary data. The status bar indicates 'All Rows Fetched: 5 in 0.005 seconds'.

	SALARY
1	24000
2	17000
3	17000
4	14000
5	13500

5. 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;
```

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.

```
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;
```

4. 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:

SALARY
-----
24000
17000
17000
14000
13500

5. 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.

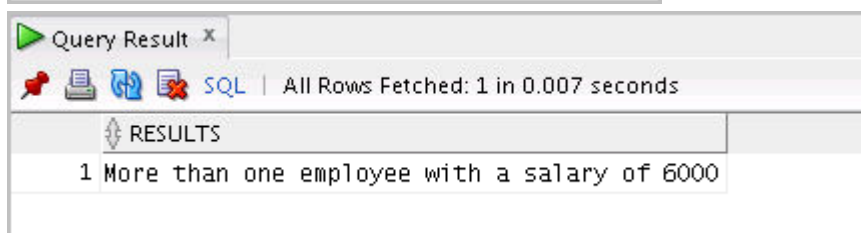
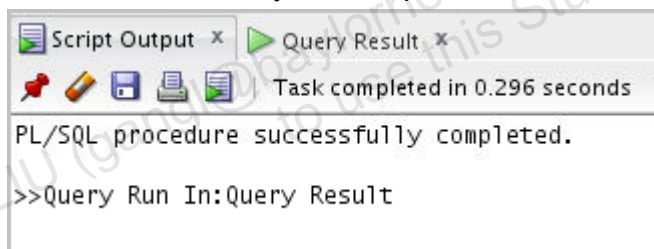
## **Practices for Lesson 9: Handling Exceptions**

### **Chapter 9**

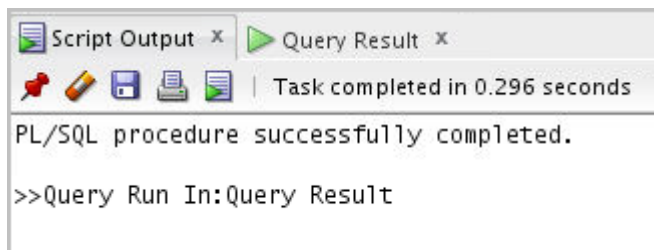
## Practice 9-1: Handling Predefined Exceptions

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.
4. 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.
5. 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.
6. Handle any other exception with an appropriate exception handler and insert the message "Some other error occurred" into the `MESSAGES` table.
7. Display the rows from the `MESSAGES` table to check whether the PL/SQL block has executed successfully. The output is as follows:



8. Change the initialized value of `v_emp_sal` to 2000 and re-execute. The output is as follows:



Query Result x	
All Rows Fetched: 2 in 0.001 seconds	
RESULTS	
1	More than one employee with a salary of 6000
2	No employee with a salary of 2000

## 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;
```

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 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);
```

4. 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));
```

5. 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));
```



6. 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.');
```

7. 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:

The screenshot shows two windows from SQL Developer. The top window, titled 'Script Output', displays the message 'PL/SQL procedure successfully completed.' and the prompt '>>Query Run In:Query Result'. The bottom window, titled 'Query Result', shows the results of a query: 'All Rows Fetched: 1 in 0.007 seconds'. The results table has one row with the value '1 More than one employee with a salary of 6000'.

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

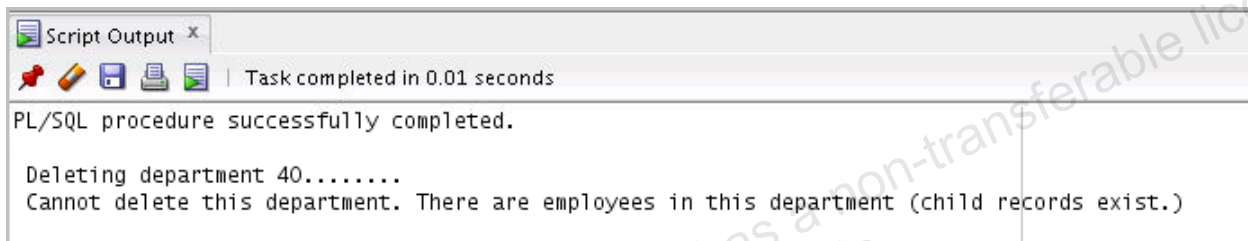
The screenshot shows two windows from SQL Developer. The top window, titled 'Script Output', displays the message 'PL/SQL procedure successfully completed.' and the prompt '>>Query Run In:Query Result'. The bottom window, titled 'Query Result', shows the results of a query: 'All Rows Fetched: 2 in 0.001 seconds'. The results table has two rows: '1 More than one employee with a salary of 6000' and '2 No employee with a salary of 2000'.

## 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.
3. Include an exception section to handle the `e_childrecord_exists` exception and display the appropriate message.

The sample output is as follows:



```
Script Output x
Task completed in 0.01 seconds
PL/SQL procedure successfully completed.
Deleting department 40.....
Cannot delete this department. There are employees in this department (child records exist.)
```

## 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);
```

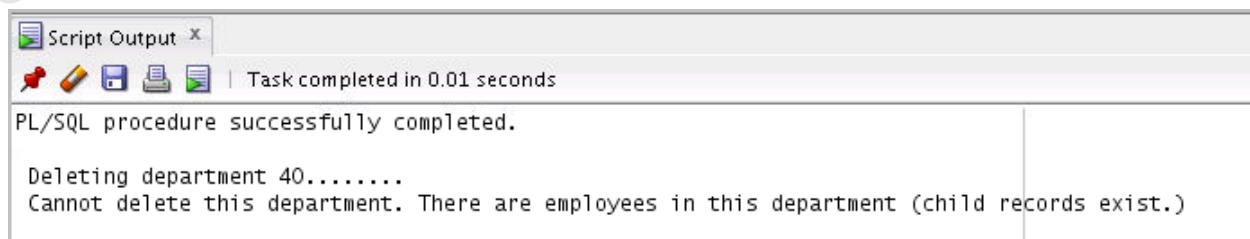
2. In the executable section, display "Deleting department 40...." Include a `DELETE` statement to delete the department with `department_id` 40.

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(' Deleting department 40.....');
    delete from departments where department_id=40;
```

3. 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:



GANG LIU (gangl@baylorhealth.edu) has a non-transferable license to use this Student Guide.

# **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;
```

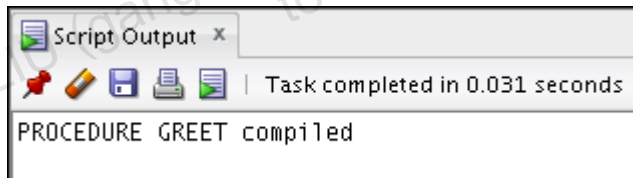
In this practice, you modify existing scripts to create and use stored procedures.

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

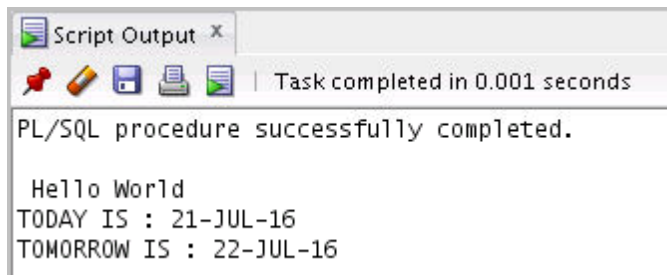
DECLARE
    v_today DATE:=SYSDATE;
    v_tomorrow v_today%TYPE;
BEGIN
    v_tomorrow:=v_today +1;
    DBMS_OUTPUT.PUT_LINE(' Hello World ');
    DBMS_OUTPUT.PUT_LINE('TODAY IS : '|| v_today);
    DBMS_OUTPUT.PUT_LINE('TOMORROW IS : '|| v_tomorrow);
END;
```

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



- c. Save this script as `lab_10_01_soln.sql`.
- d. Click the Clear button to clear the workspace.
- e. Create and execute an anonymous block to invoke the `greet` procedure.  
(Hint: Ensure that you enable `SERVEROUTPUT` at the beginning of the block.)

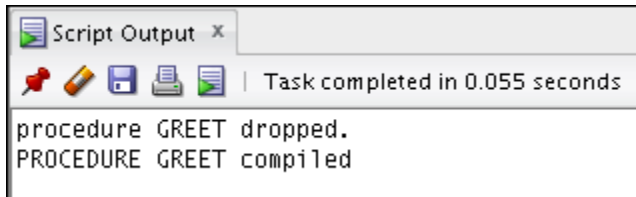
The output should be similar to the following:



2. Modify the `lab_10_01_soln.sql` script as follows:
- Drop the `greet` procedure by issuing the following command:

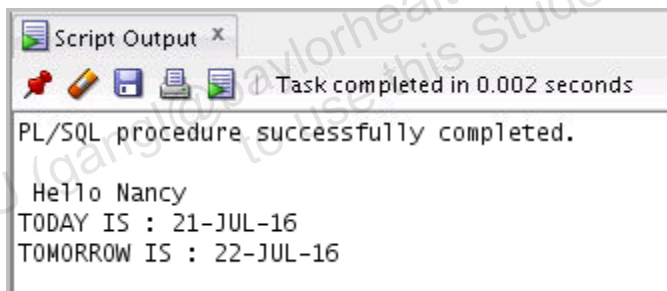
```
DROP PROCEDURE greet;
```

- Modify the procedure to accept an argument of type `VARCHAR2`. Call the argument `p_name`.
- Print `Hello <name>` (that is, the contents of the argument) instead of printing `Hello World`.
- Save your script as `lab_10_02_soln.sql`.
- Execute the script to create the procedure. The output results should be as follows:



- 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.

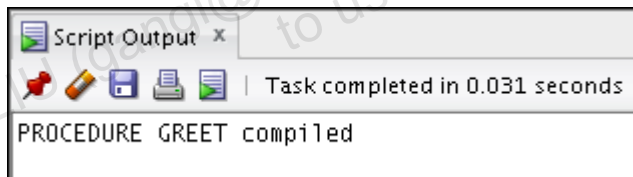
```
SET SERVEROUTPUT ON

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;
```

- a. 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;
    ...
```

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



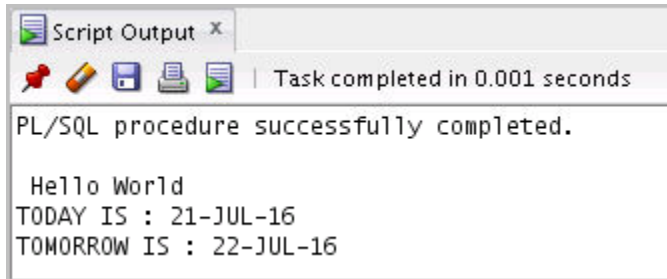
- c. Save this script as `lab_10_01_soln.sql`.
- d. Click the Clear button to clear the workspace.
- e. Create and execute an anonymous block to invoke the `greet` procedure. (Hint: Ensure that you enable `SERVEROUTPUT` at the beginning of the block.)

```
SET SERVEROUTPUT ON

BEGIN
    greet;
END;
```



The output should be similar to the following:



2. Modify the `lab_10_01_soln.sql` script as follows:

a. Drop the `greet` procedure by issuing the following command:

```
DROP PROCEDURE greet;
```

b. Modify the procedure to accept an argument of type `VARCHAR2`. Call the argument `p_name`.

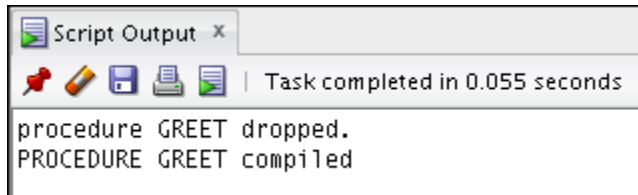
```
CREATE PROCEDURE greet(p_name VARCHAR2) IS
    v_today DATE:=SYSDATE;
    v_tomorrow v_today%TYPE;
```

c. Print `Hello <name>` instead of printing `Hello World`.

```
BEGIN
    v_tomorrow:=v_today +1;
    DBMS_OUTPUT.PUT_LINE(' Hello ' || p_name);
...
```

d. Save your script as `lab_10_02_soln.sql`.

e. Execute the script to create the procedure. The output results should be as follows:



f. 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:

