Oracle Database 12*c*: PL/SQL Fundamentals

Activity Guide

D80182GC10
Edition 1.0
August 2013
D83267

ORACLE

Author

Supriya Ananth

Technical Contributors and Reviewers

Nancy Greenberg, Swarnapriya Shridhar, KimSeong Loh, Miyuki Osato, Laszlo Czinkoczki, Diganta Choudhury, Jim Spiller, Christopher Wensley, Tom Best, Christoph Burandt, Yanti Chang, Chaitanya Kortamaddi, Bryan Roberts, Manish Pawar

This book was published using: Oracle Tutor

Table of Contents

Practices for Lesson 1: Introduction

Chapter 1

Practices for Lesson 1

Lesson Overview

In these practices, you identify information resources for SQL Developer, execute SQL statements using SQL Developer, and examine data in the class schema. Specifically, you:

- 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: Accessing SQL Developer Resources

In this practice, you view a demonstration on introduction to SQL Developer interface. Also, you navigate to the SQL Developer home page and browse helpful information on the tool.

1. Access the SQL Developer home page.
   a. Access the online SQL Developer Home Page, which is available at: http://www.oracle.com/technology/products/database/sql_developer/index.html
   b. Bookmark the page for easier access in future.
2. Access the SQL Developer tutorial, which is available online at http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm
   Review the following sections and associated demonstrations:
   a. What to Do First
   b. Working with Database Objects
   c. Accessing Data

Solution 1-1: Accessing SQL Developer Resources

1. Access the SQL Developer home page.
   a. Access the online SQL Developer Home Page, which is available at:
   http://www.oracle.com/technology/products/database/sql_developer/index.html

   Note: The screenshots in this course reflect the 3.2 version of SQL Developer. However, the online SQL Developer Home Page points to the latest version of SQL Developer available for download.

   b. Bookmark the page for easier access in future.
2. Access the SQL Developer tutorial, which is available online at
   http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm
   Then review the following sections and associated demos:
   a. What to Do First
   b. Working with Database Objects
   c. Accessing Data

Practice 1-2: 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: `ora41`
   d. Hostname: localhost
   e. Port: 1521
   f. SID: `orcl`
3. Test the new connection. If the Status is Success, connect to the database 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 the Tables icon by clicking the plus symbol next to it.
   c. Display the structure of the `EMPLOYEES` table.
5. Use the Data tab to view 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` 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.
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 button 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 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 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 1-2: Getting Started

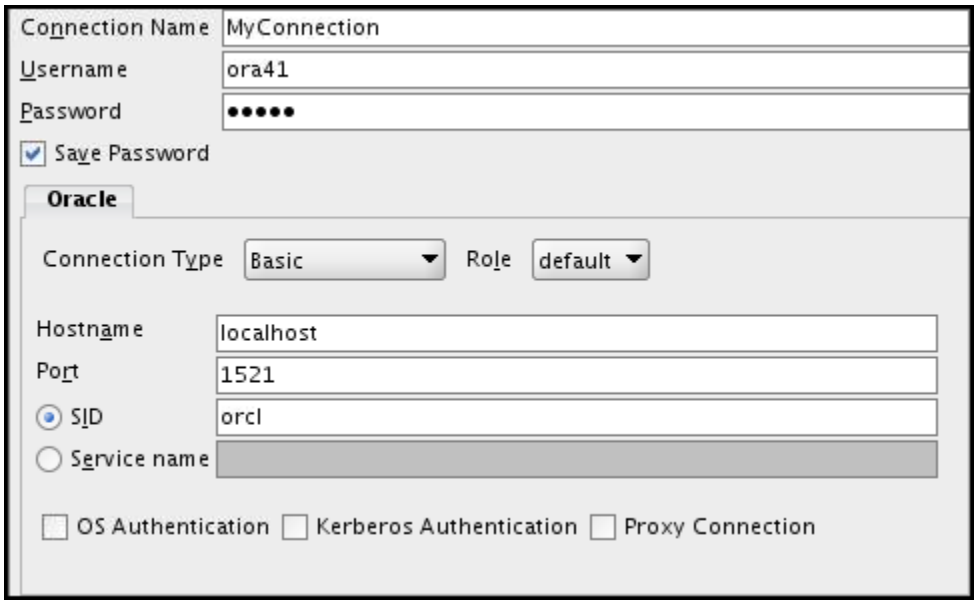
1. Start SQL Developer.
   Click the SQL Developer icon on your desktop.

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: `ora41`
   d. Hostname: localhost
   e. Port: 1521
   f. SID: `orcl`

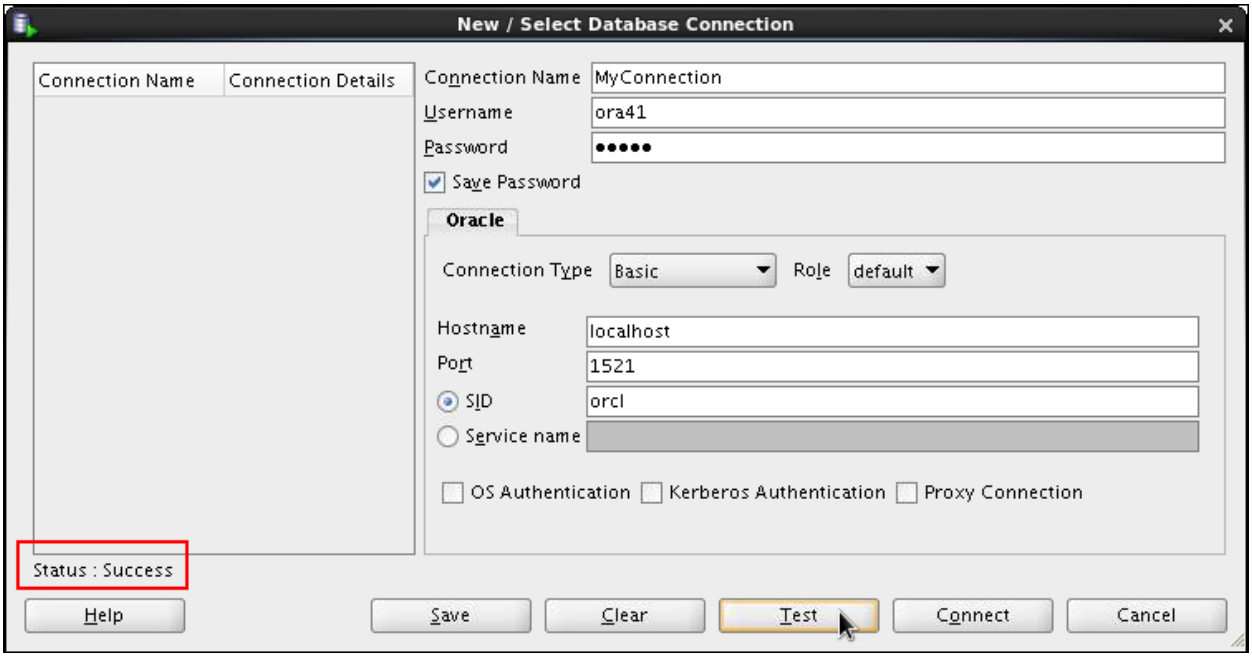
Right-click the Connections node on the Connections tabbed page and select **New Connection…**

Result: The New/Select Database Connection window appears.

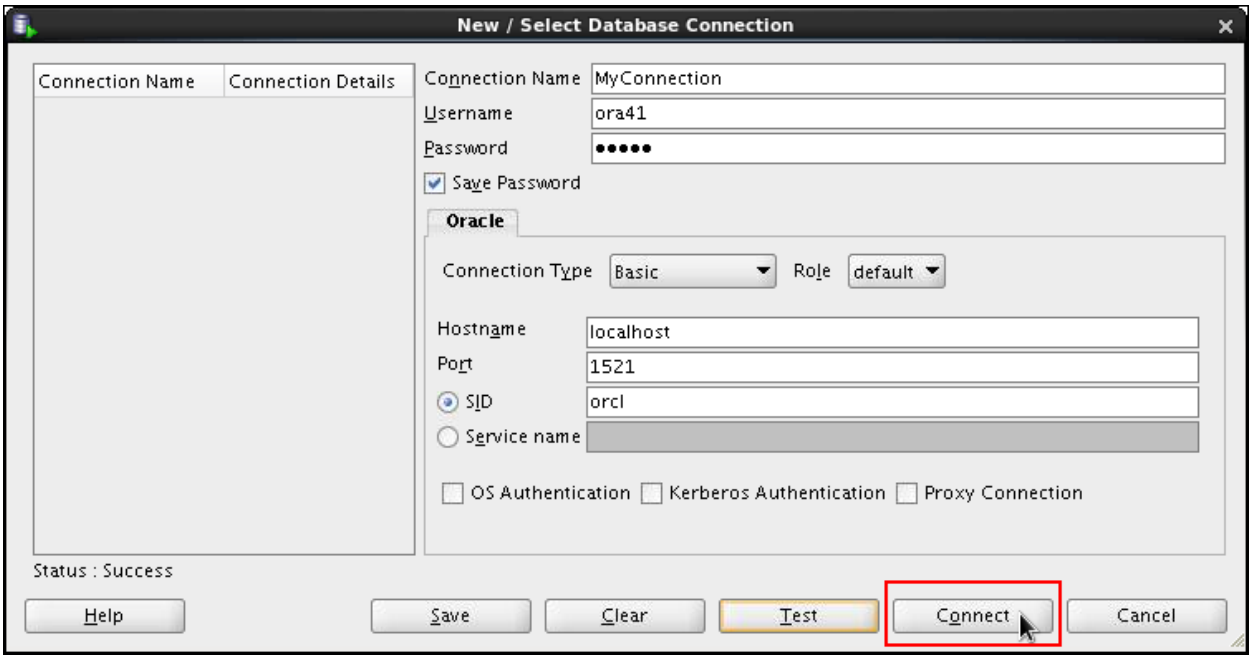
Use the preceding information to create the new database connection. In addition, select the Save Password check box. For example:



3. Test the new connection. If the Status is Success, connect to the database 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.
   a. Expand the MyConnection connection by clicking the plus symbol next to it.

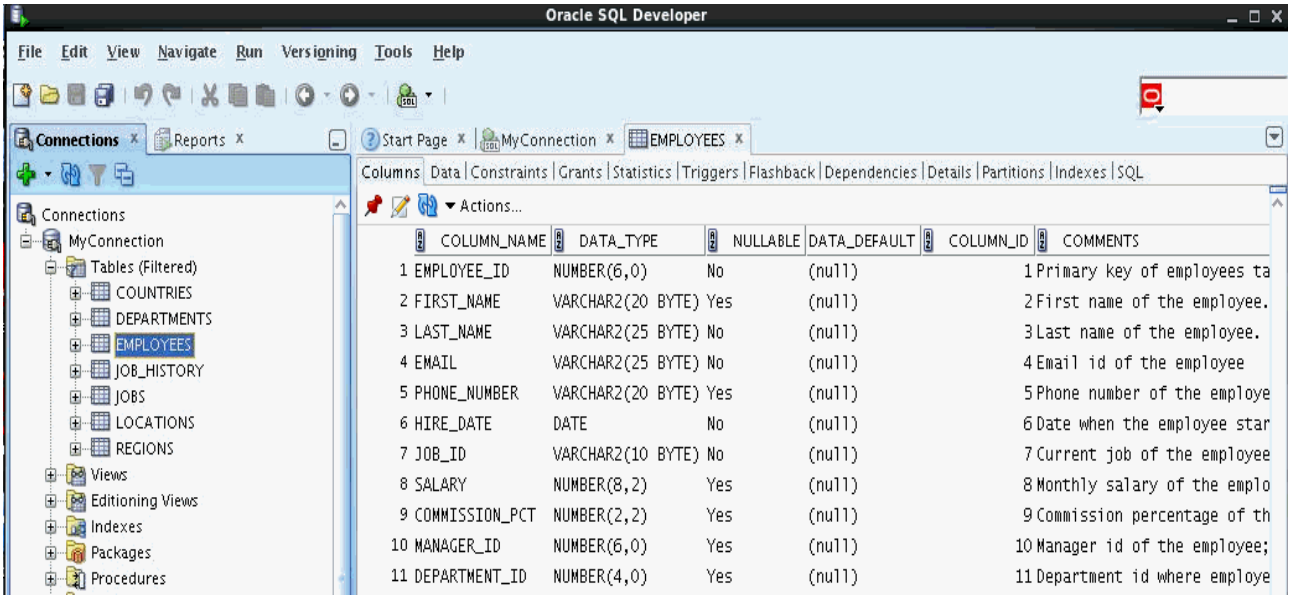
b. Expand Tables by clicking the plus symbol next to it.

c. Display the structure of the `EMPLOYEES` table.

Drill down on the `EMPLOYEES` table by clicking the plus symbol next to it.

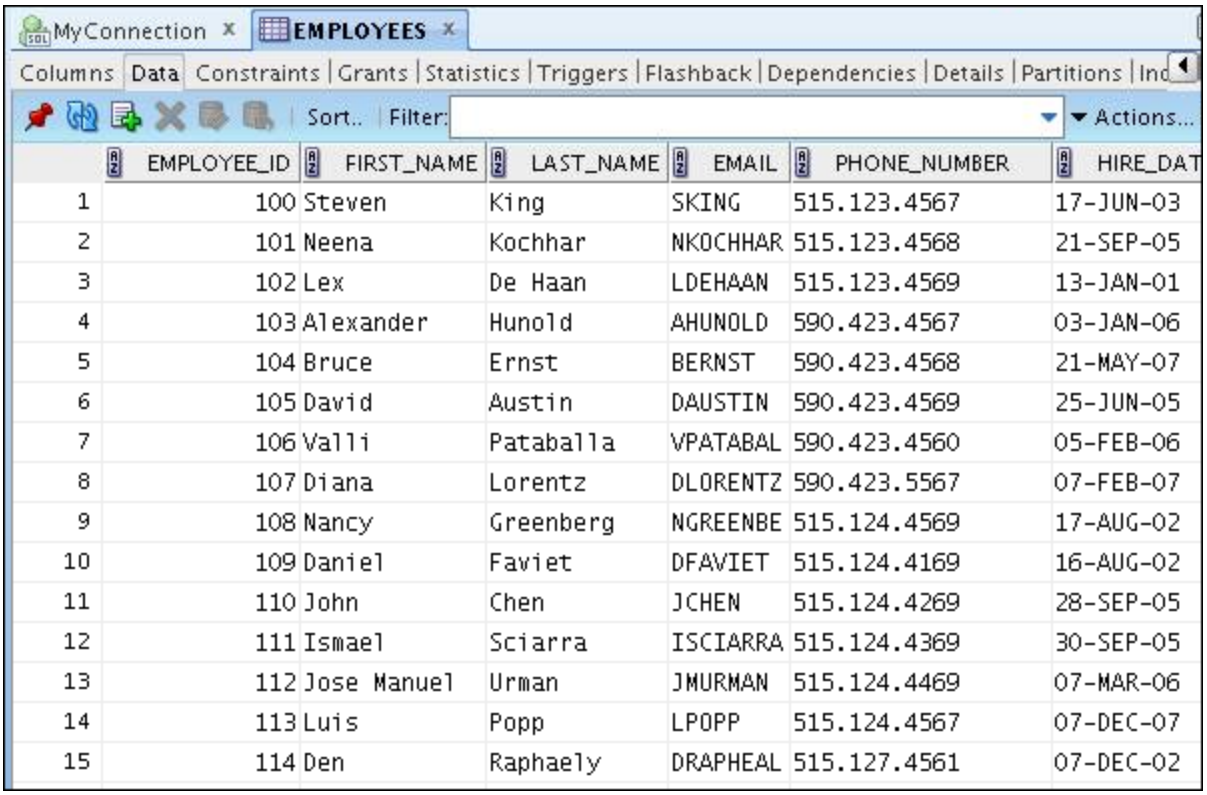
Click the `EMPLOYEES` table.

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



5. Use the Data tab to view the data in the `EMPLOYEES` table.

Result: The `EMPLOYEES` table data is displayed as follows:



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 Run Script (F5) icons to execute the `SELECT` statement. Review the results of both methods of executing the `SELECT` statements on the appropriate tabs.

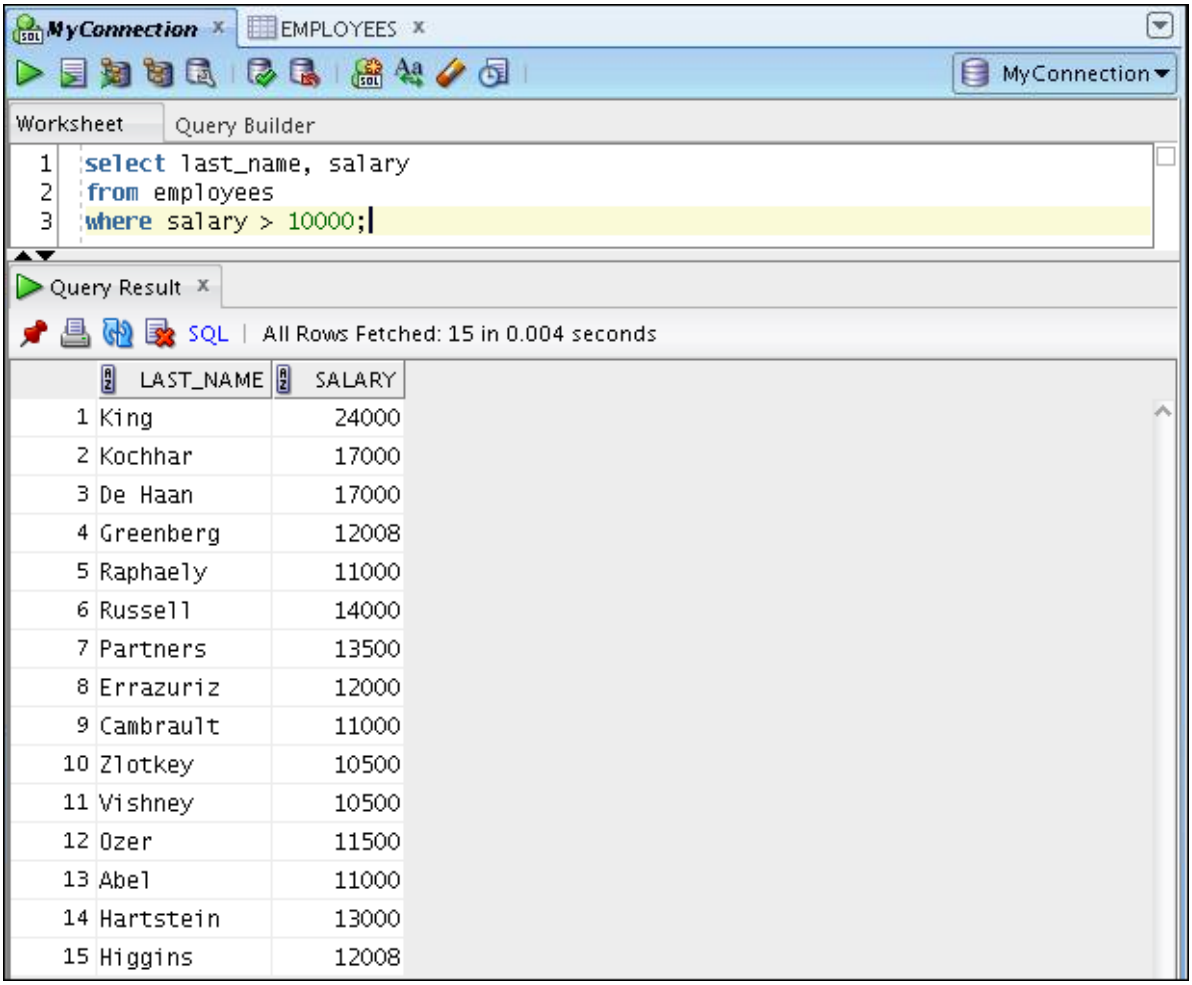
   Note: Take a few minutes to familiarize yourself with the data, or consult Appendix A, which provides the description and data for all the tables in the HR schema that you will use in this course.

To display the SQL Worksheet, click the MyConnection tab.

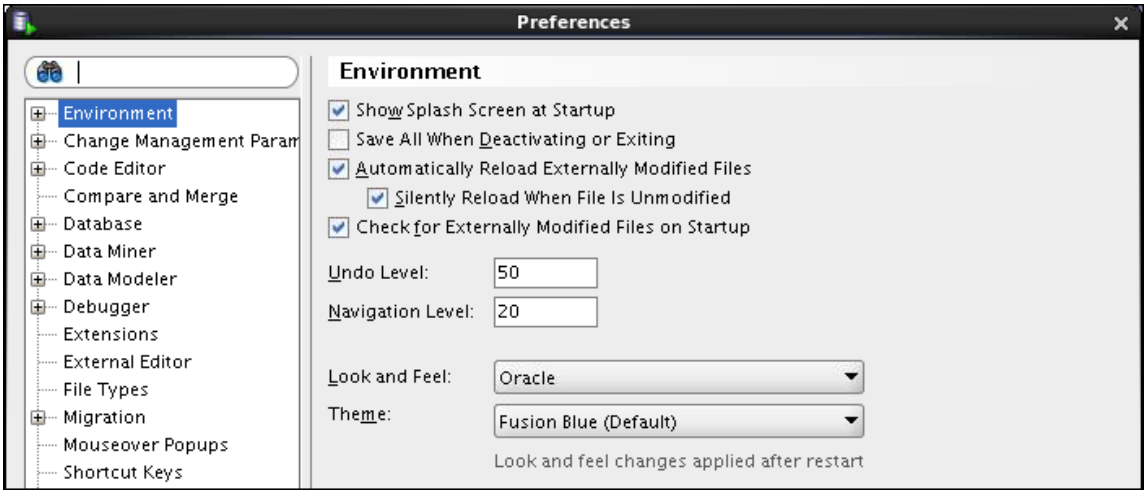
Note: This tab was opened previously when you drilled down on your database connection.

Enter the appropriate `SELECT` statement. Press F9 to execute the query and F5 to execute the query using the Run Script method.

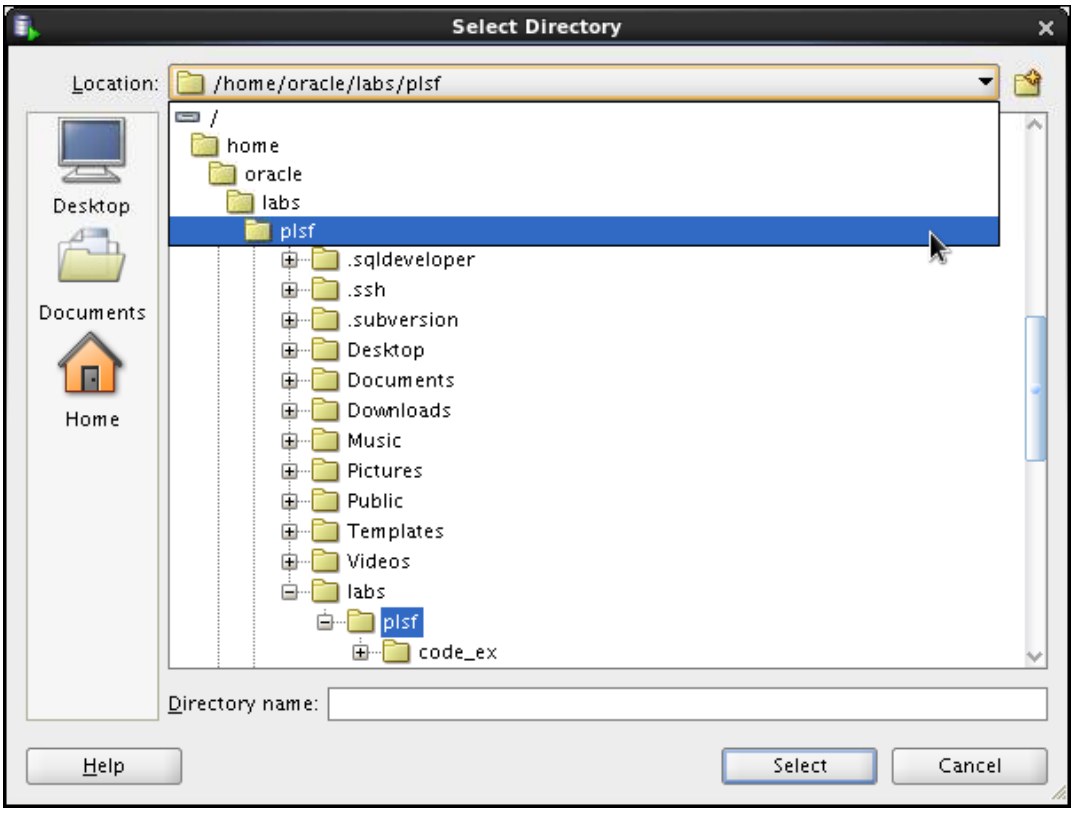
For example, when you press F9, the results appear similar to the following:



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



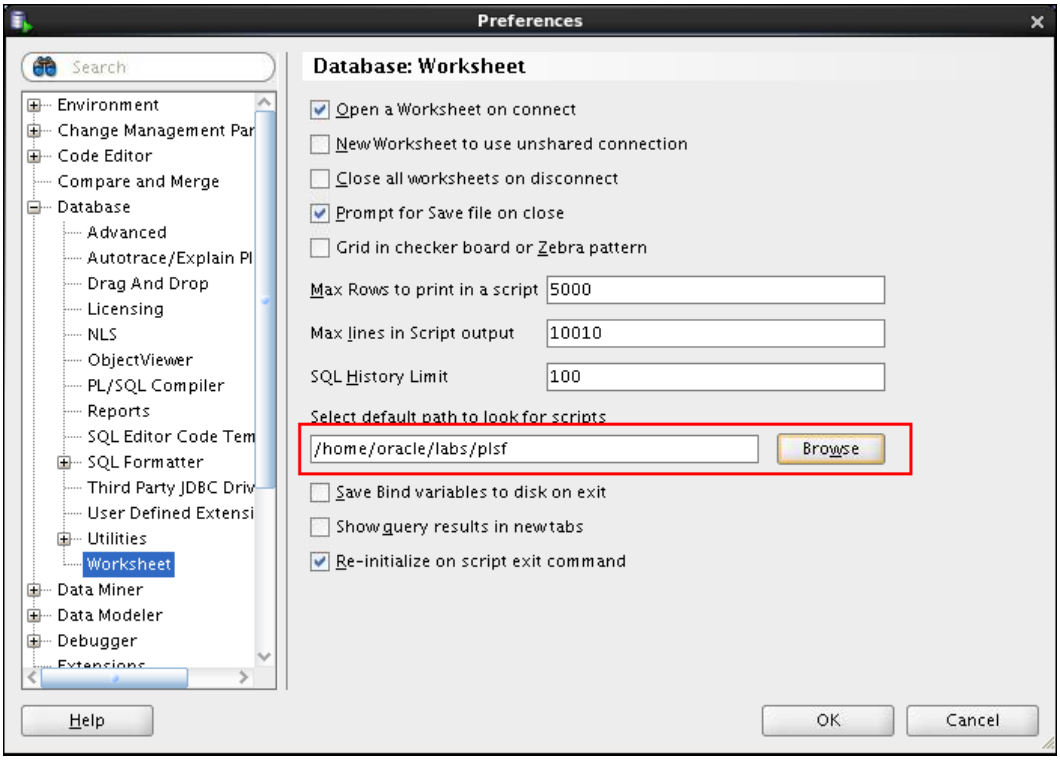
8. Select Database > Worksheet Parameters. In the "Select default path to look for scripts" text box, use the Browse button to select the `/home/oracle/labs/plsf` directory.



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.

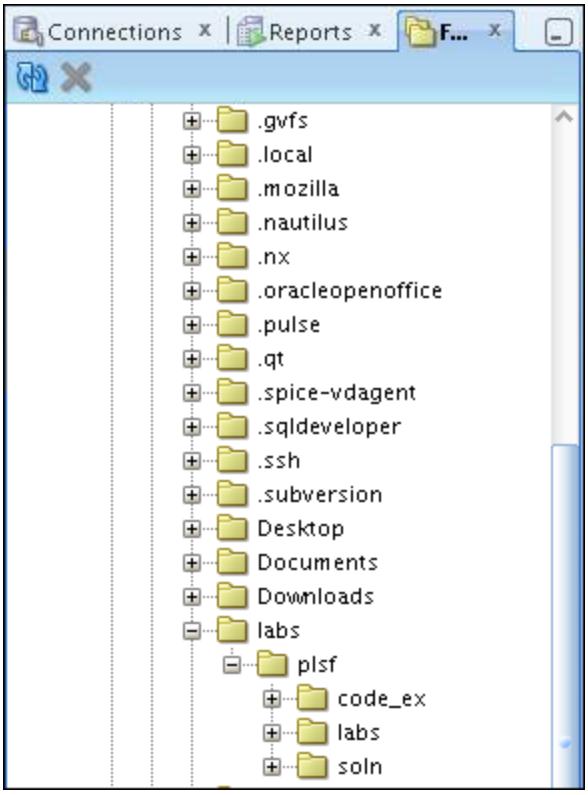
Then, in the Preferences window, click OK to save the Worksheet Parameter setting.



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 directories to open 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
   END;
   ```

   b.
   ```
   DECLARE
   v_amount INTEGER(10);
   END;
   ```

   c.
   ```
   DECLARE
   BEGIN
   END;
   ```

   d.
   ```
   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
   END;
b. DECLARE
   v_amount INTEGER(10);
   END;
c. DECLARE
   BEGIN
   END;
d. DECLARE
   v_amount INTEGER(10);
   BEGIN
   DBMS_OUTPUT.PUT_LINE(v_amount);
   END;
```

The block in a does not execute. It has no executable statements.
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.

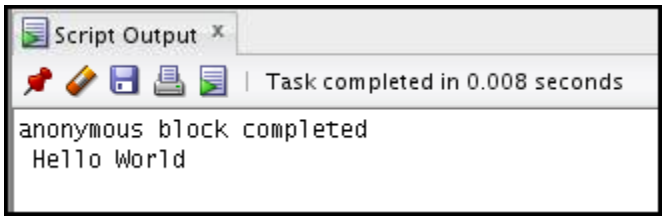
The block in d executes successfully.

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.

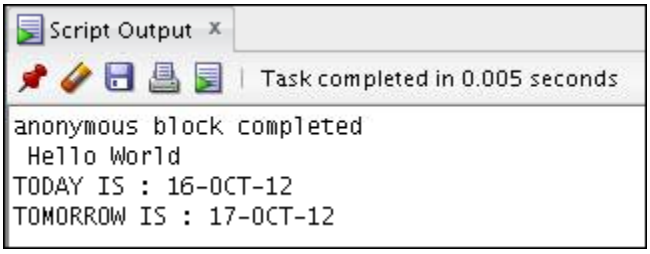
1. Identify valid and invalid identifiers:
   a. `today`
   b. `last_name`
   c. `today's_date`
   d. `Number_of_days_in_February_this_year`
   e. `Isleap$year`
   f. `#number`
   g. `NUMBER#`
   h. `number1to7`
2. Identify valid and invalid variable declaration and initialization:
   a. `number_of_copies 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.
   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 `tomorrow` after printing "Hello World"
   d. Save your script as `lab_03_04_soln.sql`, and then execute.

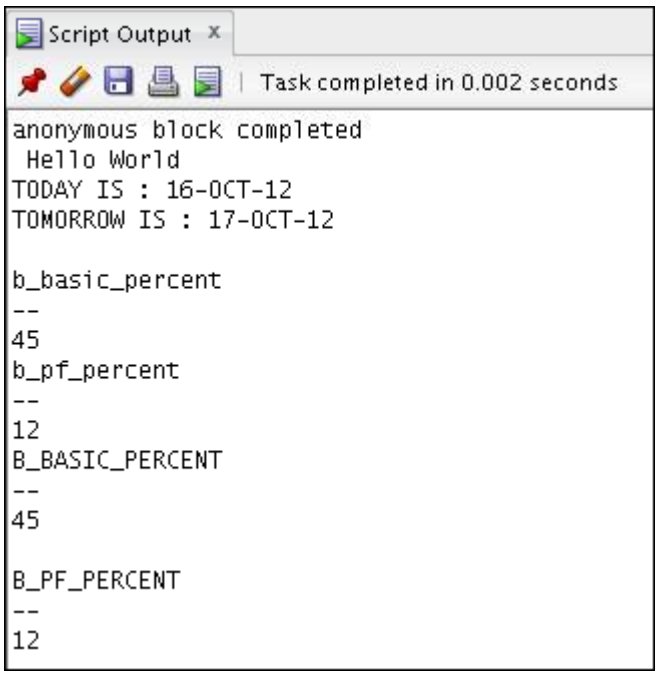
      The sample output is as follows (the values of `v_today` and `v_tomorrow` will be different to reflect your current today's and tomorrow's date):



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:



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.
   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;
```

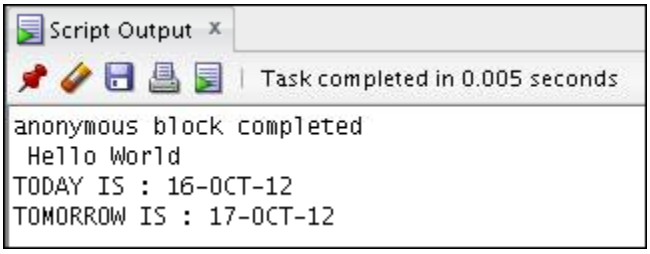
In the executable section:

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.
   a. Add the 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
```

   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.

```
   :b_basic_percent:=45;
   :b_pf_percent:=12;
```

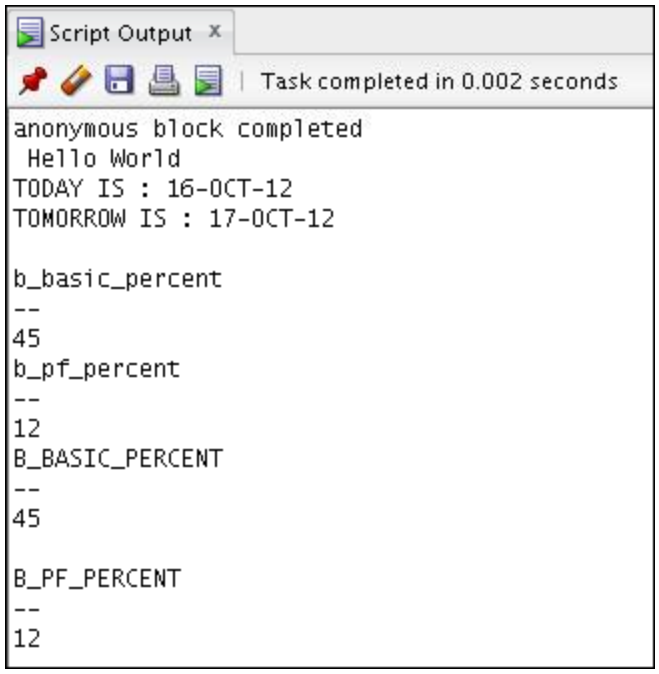
   c. Terminate the PL/SQL block with "/" and display the value of the bind variables by using the `PRINT` command.

```
/
PRINT b_basic_percent
PRINT b_pf_percent
```

OR

```
PRINT
```

   d. 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 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;
/
```

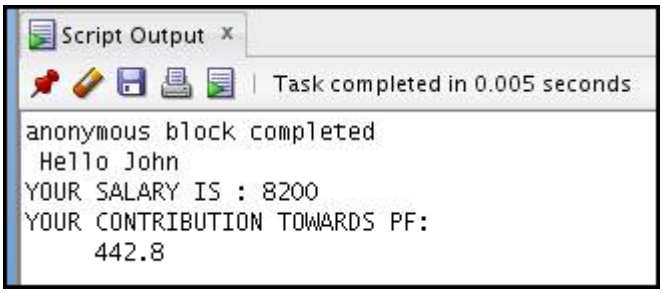
1. Evaluate the preceding PL/SQL block and determine the data type and value of each of the following variables, according to the rules of scoping.
   a. The value of `v_weight` at position 1 is:
   b. The value of `v_new_locn` at position 1 is:
   c. The value of `v_weight` at position 2 is:
   d. The 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 values and data types for 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 the employee towards 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:



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 values and data types 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;
```

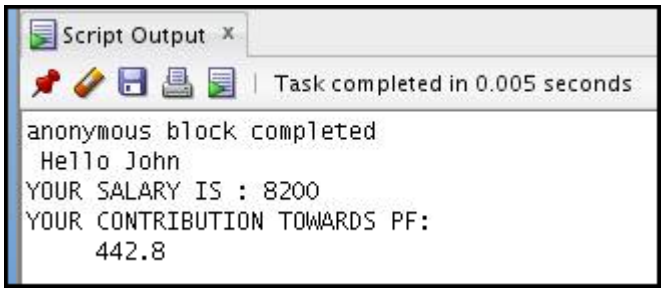
e. Change the line that prints “Hello World” to print “Hello” and the first name. Then, comment the lines that display the dates and print the bind variables.

```
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 the employee towards 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:



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 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:

```
anonymous block 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 values in `dept_name` and `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:

```
anonymous block completed
The maximum department_id is : 270
 SQL%ROWCOUNT gives 1

DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          280 Education
```

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_02.sql`.

   a. Start the executable block with the `BEGIN` keyword. Include the `UPDATE` statement to set the `location_id` to `3000` for the new department (`dept_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.

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

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

```
anonymous block completed
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          280 Education                                        3000

1 rows deleted.
```

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:

```
anonymous block 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 values in dept_name and 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:

```
anonymous block completed
The maximum department_id is : 270
 SQL%ROWCOUNT gives 1

DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          280 Education
```

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_02.sql.

a. Start the executable block with the BEGIN keyword. Include the UPDATE statement to set location_id to 3000 for the new department (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:

```
anonymous block completed
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          280 Education                                        3000

1 rows deleted.
```

Practices for Lesson 6: Writing Control Structures

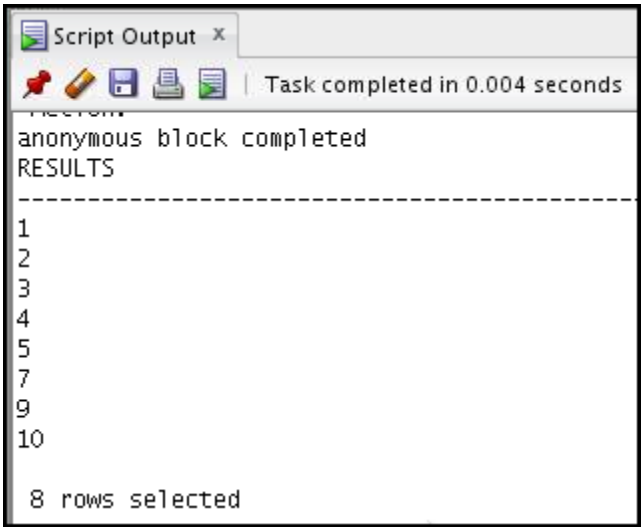
Chapter 6

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:



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.
   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:

Solution 6: Writing Control Structures

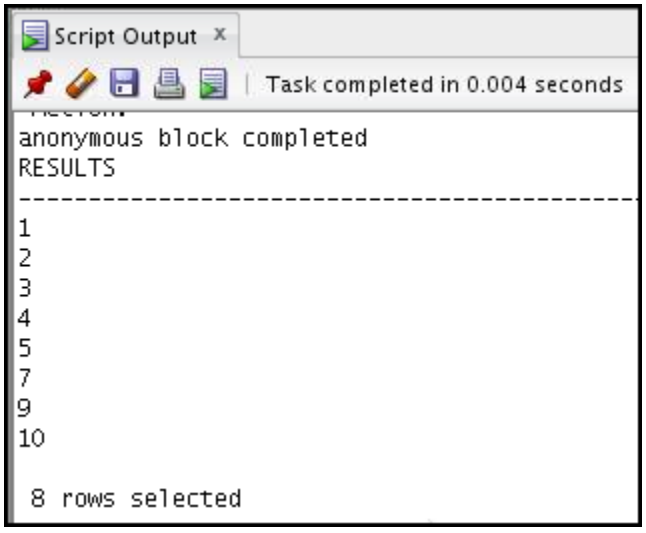
1. Execute the command in the `lab_06_01.sql` file to create the `messages` table. Write a PL/SQL block to insert numbers into the `messages` table.
   a. Insert the numbers 1 through 10, excluding 6 and 8.
   b. Commit before the end of the block.

```
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:



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;
```

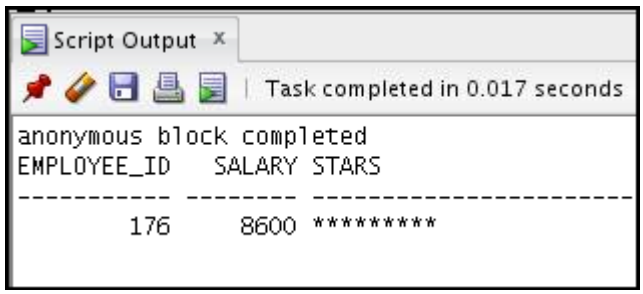
   c. Update the `stars` column for the employee with the string of asterisks. Commit before the end of the block.

```
   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:



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 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:

```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```

   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**
50	**Shipping**
60	**IT**
70	**Public Relations**
80	**Sales**

90	Executive
100	Finance

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:

```
anonymous block completed
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance
```

3. Modify the block that you created in Practice 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:

```
anonymous block completed
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:

```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```

   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:

```
anonymous block completed
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance
```

3. Modify the block that you created in Practice 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 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:

```
anonymous block completed
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 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 the 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 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.

   Note
   - 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:

```
anonymous block completed
Department Number : 10  Department Name : Administration
-------------------------------------------------------------------------------
Department Number : 20  Department Name : Marketing
-------------------------------------------------------------------------------
Department Number : 30  Department Name : Purchasing
Raphaely    PU_MAN   07-DEC-02   11000
Khoo    PU_CLERK   18-MAY-03   3100
Baida    PU_CLERK   24-DEC-05   2900
Tobias    PU_CLERK   24-JUL-05   2800
Himuro    PU_CLERK   15-NOV-06   2600
Colmenares    PU_CLERK   10-AUG-07   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-06   9000
Ernst    IT_PROG   21-MAY-07   6000
Austin    IT_PROG   25-JUN-05   4800
Pataballa    IT_PROG   05-FEB-06   4800
Lorentz    IT_PROG   07-FEB-07   4200
-------------------------------------------------------------------------------
Department Number : 70  Department Name : Public Relations
-------------------------------------------------------------------------------
Department Number : 80  Department Name : Sales
-------------------------------------------------------------------------------
Department Number : 90  Department Name : Executive
King    AD_PRES   17-JUN-03   24000
Kochhar    AD_VP   21-SEP-05   17000
De Haan    AD_VP   13-JAN-01   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 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 the `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 the 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 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.

Note

- 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:

```
anonymous block completed
Department Number : 10  Department Name : Administration
--------------------------------------------------------------------------------
Department Number : 20  Department Name : Marketing
--------------------------------------------------------------------------------
Department Number : 30  Department Name : Purchasing
Raphaely    PU_MAN   07-DEC-02   11000
Khoo    PU_CLERK   18-MAY-03   3100
Baida    PU_CLERK   24-DEC-05   2900
Tobias    PU_CLERK   24-JUL-05   2800
Himuro    PU_CLERK   15-NOV-06   2600
Colmenares    PU_CLERK   10-AUG-07   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-06   9000
Ernst    IT_PROG   21-MAY-07   6000
Austin    IT_PROG   25-JUN-05   4800
Pataballa    IT_PROG   05-FEB-06   4800
Lorentz    IT_PROG   07-FEB-07   4200
--------------------------------------------------------------------------------
Department Number : 70  Department Name : Public Relations
--------------------------------------------------------------------------------
Department Number : 80  Department Name : Sales
--------------------------------------------------------------------------------
Department Number : 90  Department Name : Executive
King    AD_PRES   17-JUN-03   24000
Kochhar    AD_VP   21-SEP-05   17000
De Haan    AD_VP   13-JAN-01   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 `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 and 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.

```
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.

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 `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 and 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 into the `MESSAGES` table the employee's name and the salary amount.
   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 into the `MESSAGES` table the message "No employee with a salary of <*salary*>."
5. If the salary entered returns multiple rows, handle the exception with an appropriate exception handler and insert into the `MESSAGES` table the message "More than one employee with a salary of <*salary*>."
6. Handle any other exception with an appropriate exception handler and insert into the `MESSAGES` table the message "Some other error occurred."
7. Display the rows from the `MESSAGES` table to check whether the PL/SQL block has executed successfully. The output is as follows:

```
RESULTS
--------------------------------------------------
More than one employee with a salary of 6000


1 rows selected
```

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

```
RESULTS
--------------------------------------------------
More than one employee with a salary of 6000
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 into the `MESSAGES` table the employee's name and the salary amount.
   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 into the `MESSAGES` table the message "No employee with a salary of <*salary*>."

```
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 into the `MESSAGES` table the message "More than one employee with a salary of <*salary*>."

```
WHEN too_many_rows THEN
    INSERT INTO messages (results)
    VALUES ('More than one employee with a salary of '||
             TO_CHAR(v_emp_sal));
```

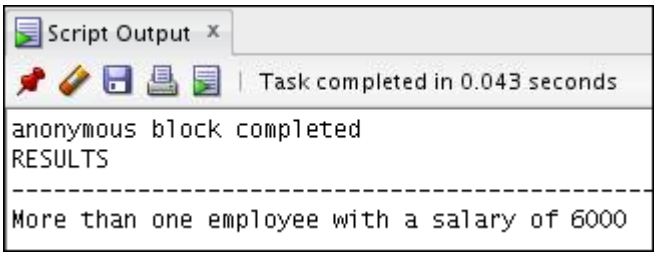
6. Handle any other exception with an appropriate exception handler and insert into the `MESSAGES` table the message "Some other error occurred."

```
WHEN others THEN
    INSERT INTO messages (results)
    VALUES ('Some other error occurred.');
END;
```

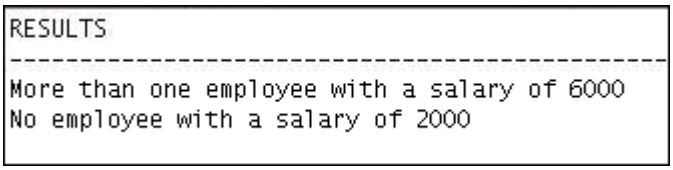
7. Display the rows from the `MESSAGES` table to check whether the PL/SQL block has executed successfully.

```
/
SELECT * FROM messages;
```

The output is as follows:



8. 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.
3. Include an exception section to handle the `e_childrecord_exists` exception and display the appropriate message.

The sample output is as follows:

```
anonymous block 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:

```
anonymous block completed
 Deleting department 40........
 Cannot delete this department. There are employees in this department (child records exist.)
```

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 you execute the following code before starting this practice:

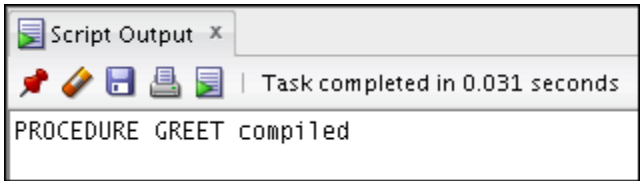
```
DROP table dept;
DROP procedure add_dept;
DROP function check_sal;
```

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

1. Open `sol_03.sql` script from the /home/oracle/labs/plsf/soln/ folder. Copy the code under task 4 into a new worksheet.

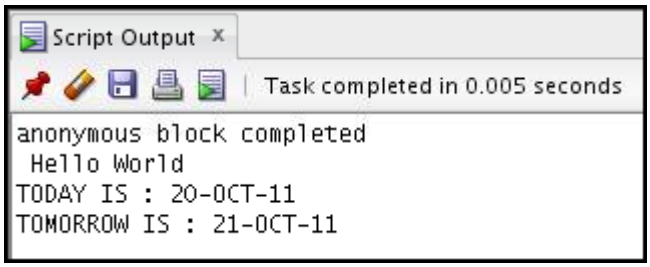
```
SET SERVEROUTPUT ON

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.)

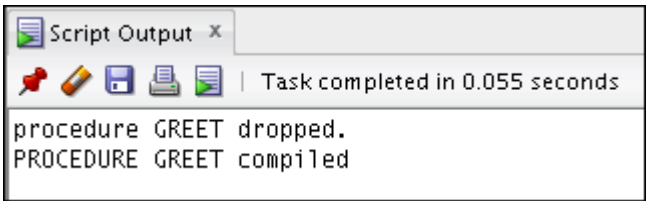
The output should be similar to the following:



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

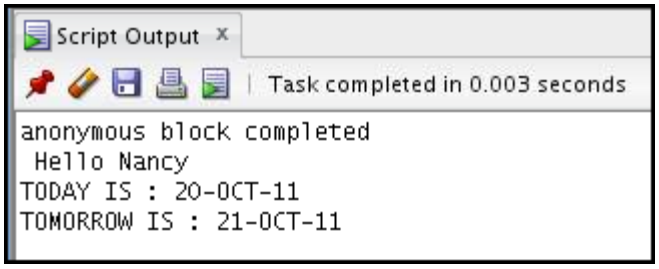
```
DROP PROCEDURE greet;
```

   b. Modify the procedure to accept an argument of type `VARCHAR2`. Call the argument `p_name`.
   c. Print `Hello` *<name>* (that is, the contents of the argument) instead of printing `Hello World`.
   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.

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.

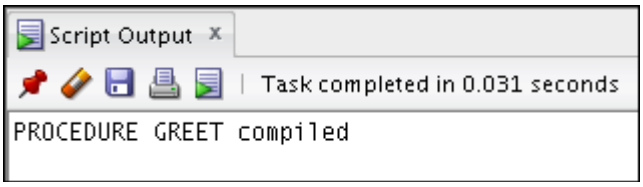
1. Open `sol_03.sql` script from the /home/oracle/labs/plsf/soln/ folder. Copy the code under task 4 into a new worksheet.

```
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 today%TYPE;
...
```

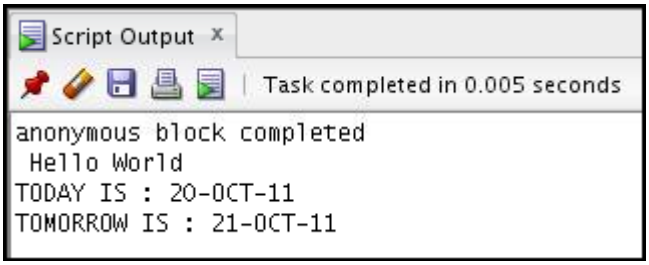
   b. Execute the script to create the procedure. The output results should be as follows:



   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;
```

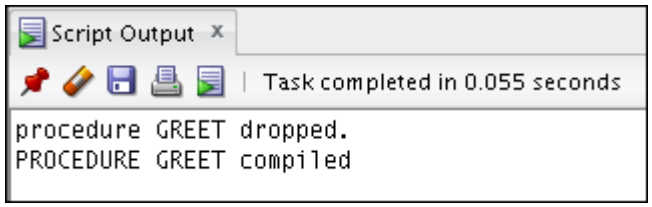
   b. Modify the procedure to accept an argument of type `VARCHAR2`. Call the argument `p_name`.

```
CREATE PROCEDURE greet(p_name VARCHAR2) IS
   V_today DATE:=SYSDATE;
   V_tomorrow today%TYPE;
```

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

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

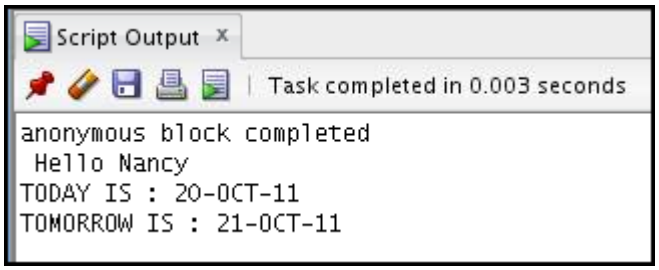
   d. Save your script as `lab_10_02_soln.sql`.
   e. Execute the script to create the procedure. The output results should be as follows:



   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:



Additional Practices and Solutions for Lesson 1

Chapter 11

Practices for Lesson 1

There are no practices for this lesson.

Additional Practices and Solutions for Lesson 2

Chapter 12

Additional Practices for Lesson 2

Overview

These additional practices are provided as a supplement to the *Oracle Database: PL/SQL Fundamentals* course. In these practices, you apply the concepts that you learned in the course.

These additional practices provide supplemental practice in declaring variables, writing executable statements, interacting with the Oracle Server, writing control structures, and working with composite data types, cursors, and handle exceptions. The tables used in this portion of the additional practices include `employees`, `jobs`, `job_history`, and `departments`.

Practice 2: Evaluating Declarations

Overview

These paper-based exercises are used for extra practice in declaring variables and writing executable statements.

Evaluate each of the following declarations. Determine which of them are not legal and explain why.

```
1. DECLARE
   name,dept    VARCHAR2(14);
2. DECLARE
   test         NUMBER(5);
3. DECLARE
   MAXSALARY    NUMBER(7,2) = 5000;
4. DECLARE
   JOINDATE     BOOLEAN := SYSDATE;
```

Solution 2: Evaluating Declarations

Evaluate each of the following declarations. Determine which of them are not legal and explain why.

1.
```
DECLARE
name,dept     VARCHAR2(14);
```
This is illegal because only one identifier per declaration is allowed.

2.
```
DECLARE
test          NUMBER(5);
```
This is legal.

3.
```
DECLARE
MAXSALARY     NUMBER(7,2) = 5000;
```
This is illegal because the assignment operator is wrong. It should be :=.

4.
```
DECLARE
JOINDATE      BOOLEAN := SYSDATE;
```
This is illegal because there is a mismatch in the data types. A Boolean data type cannot be assigned a date value. The data type should be date.

Additional Practices and Solutions for Lesson 3

Chapter 13

Practice 3: Evaluating Expressions

In each of the following assignments, determine the data type of the resulting expression.

```
1. email := firstname || to_char(empno);

2. confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');

3. sal := (1000*12) +  500

4. test := FALSE;

5. temp := temp1 < (temp2/ 3);

6. var := sysdate;
```

Solution 3: Evaluating Expressions

In each of the following assignments, determine the data type of the resulting expression.

1. `email := firstname || to_char(empno);`

   Character string

2. `confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');`

   Date

3. `sal := (1000*12) +  500`

   Number

4. `test := FALSE;`

   Boolean

5. `temp := temp1 < (temp2/ 3);`

   Boolean

6. `var := sysdate;`

   Date

Additional Practices and Solutions for Lesson 4

Chapter 14

Practice 4: Evaluating Executable Statements

In this paper-based exercise, you evaluate the PL/SQL block, and then answer the questions that follow by determining the data type and value of each variable, according to the rules of scoping.

```
DECLARE
      v_custid    NUMBER(4) := 1600;
      v_custname VARCHAR2(300) := 'Women Sports Club';
      v_ new_custid   NUMBER(3) := 500;
BEGIN
DECLARE
      v_custid     NUMBER(4) := 0;
      v_custname  VARCHAR2(300) := 'Shape up Sports Club';
      v_new_custid  NUMBER(3) := 300;
      v_new_custname  VARCHAR2(300) := 'Jansports Club';
BEGIN
      v_custid := v_new_custid;
      v_custname := v_custname || ' ' ||  v_new_custname;
1 ----------►
END;
      v_custid := (v_custid *12) / 10;
2 ----------►
END;
```

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

1. `v_custid` at position 1:

2. `v_custname` at position 1:

3. `v_new_custid` at position 1:

4. `v_new_custname` at position 1:

5. `v_custid` at position 2:

6. `v_custname` at position 2:

Solution 4: Evaluating Executable Statements

Evaluate the following PL/SQL block. Then, answer the questions that follow by determining the data type and value of each of the following variables, according to the rules of scoping.

```
DECLARE
      v_custid    NUMBER(4) := 1600;
      v_custname VARCHAR2(300) := 'Women Sports Club';
      v_ new_custid   NUMBER(3) := 500;
BEGIN
DECLARE
      v_custid     NUMBER(4) := 0;
      v_custname  VARCHAR2(300) := 'Shape up Sports Club';
      v_new_custid  NUMBER(3) := 300;
      v_new_custname  VARCHAR2(300) := 'Jansports Club';
BEGIN
      v_custid := v_new_custid;
      v_custname := v_custname || ' ' ||  v_new_custname;
1 ----------►
END;
      v_custid := (v_custid *12) / 10;
2 ----------►
END;
```

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

1. `v_custid` at position 1:

   500, and the data type is `NUMBER`.

2. `v_custname` at position 1:

   Shape up Sports Club Jansports Club, and the data type is `VARCHAR2`.

3. `v_new_custid` at position 1:

   300, and the data type is `NUMBER` (or `INTEGER`).

4. `v_new_custname` at position 1:

   Jansports Club, and the data type is `VARCHAR2`.

5. `v_custid` at position 2:

   1920, and the data type is `NUMBER`.

6. `v_custname` at position 2:

   Women Sports Club, and the data type is `VARCHAR2`.

Additional Practices and Solutions for Lesson 5

Chapter 15

Practice 5-1: Using SQL Statements within a PL/SQL

For this exercise, a temporary table is required to store the results.

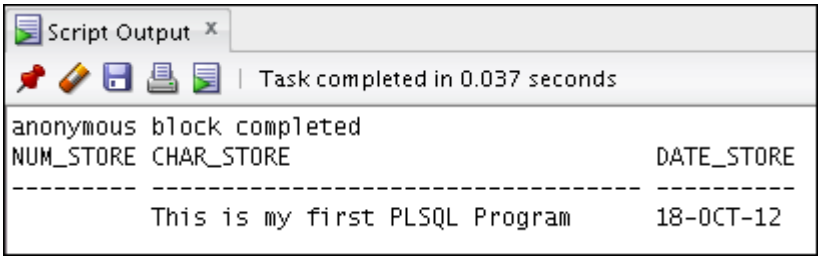
1. Run the `lab_ap_05.sql` script that creates the table described here:

Column Name	NUM_STORE	CHAR_STORE	DATE_STORE
Key Type			
Nulls/Unique			
FK Table			
FK Column			
Data Type	Number	VARCHAR2	Date
Length	7,2	35	

2. Write a PL/SQL block that performs the following:
   a. Declares two variables and assigns the following values to these variables:

Variable	Data type	Contents
V_MESSAGE	VARCHAR2(35)	This is my first PL/SQL program
V_ DATE_WRITTEN	DATE	Current date

   b. Stores the values from these variables in the appropriate `TEMP` table columns
3. Verify your results by querying the `TEMP` table. The output results should appear as follows:



Solution 5-1: Using SQL Statements within a PL/SQL

For this exercise, a temporary table is required to store the results.

1. Run the `lab_ap_05.sql` script that creates the table described here:

Column Name	NUM_STORE	CHAR_STORE	DATE_STORE
Key Type			
Nulls/Unique			
FK Table			
FK Column			
Data Type	Number	VARCHAR2	Date
Length	7,2	35	

2. Write a PL/SQL block that performs the following:
   a. Declares two variables and assigns the following values to these variables:

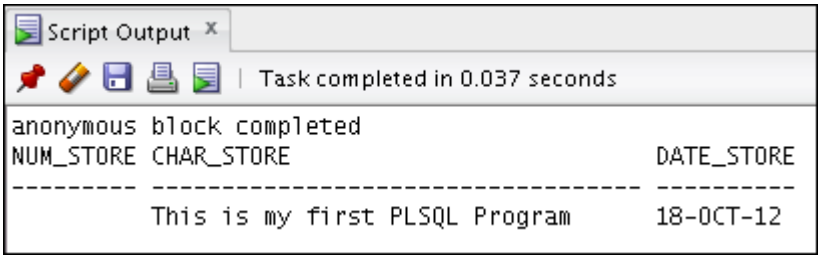
Variable	Data type	Contents
V_MESSAGE	VARCHAR2(35)	This is my first PL/SQL program
V_ DATE_WRITTEN	DATE	Current date

   b. Stores the values from these variables in the appropriate `TEMP` table columns

```
DECLARE
  V_MESSAGE VARCHAR2(35);
  V_DATE_WRITTEN DATE;
BEGIN
  V_MESSAGE := 'This is my first PLSQL Program';
  V_DATE_WRITTEN := SYSDATE;
  INSERT INTO temp(CHAR_STORE,DATE_STORE)
    VALUES (V_MESSAGE,V_DATE_WRITTEN);
END;
/
```

3. Verify your results by querying the `TEMP` table. The output results should look similar to the following:

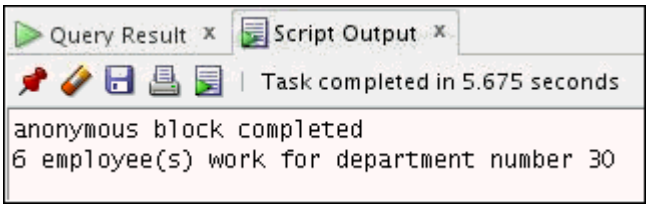
```
SELECT * FROM TEMP;
```



Practice 5-2: Using SQL Statements within a PL/SQL

In this exercise, you use data from the `employees` table.

1. Write a PL/SQL block to determine how many employees work for a specified department. The PL/SQL block should:
   - Use a substitution variable to store a department number
   - Print the number of people working in the specified department
2. When the block is run, a substitution variable window appears. Enter a valid department number and click OK. The output results should look similar to the following:



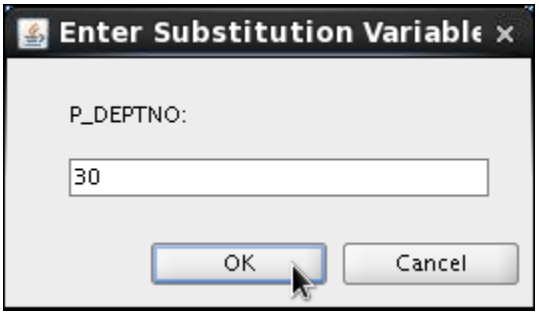
Solution 5-2: Using SQL Statements within a PL/SQL

In this exercise, you use data from the `employees` table.

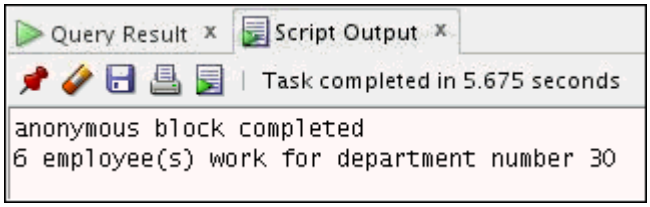
1. Write a PL/SQL block to determine how many employees work for a specified department. The PL/SQL block should:
   - Use a substitution variable to store a department number
   - Print the number of people working in the specified department

```
SET SERVEROUTPUT ON;
DECLARE
   V_HOWMANY NUMBER(3);
   V_DEPTNO DEPARTMENTS.department_id%TYPE := &P_DEPTNO;
BEGIN
   SELECT COUNT(*) INTO V_HOWMANY FROM employees
   WHERE department_id = V_DEPTNO;
   DBMS_OUTPUT.PUT_LINE (V_HOWMANY  || ' employee(s)
     work for department number ' ||V_DEPTNO);
END;
/
```

2. When the block is run, a substitution variable window appears. Enter a valid department number and click OK.



The output results should look similar to the following:



Additional Practices and Solutions for Lesson 6

Chapter 16

Practice 6-1: Writing Control Structures

In these practices, you use control structures to direct the logic of program flow.

1. Write a PL/SQL block to accept a year input and check whether it is a leap year.
   Hint: The year should be exactly divisible by 4 but not divisible by 100, or it should be divisible by 400.
2. Test your solution by using the following table. For example, if the year entered is 1990, the output should be "1990 is not a leap year."

1990	Not a leap year
2000	Leap year
1996	Leap year
1886	Not a leap year
1992	Leap year
1824	Leap year

Solution 6-1: Writing Control Structures

1. Write a PL/SQL block to accept a year input and check whether it is a leap year.
   Hint: The year should be exactly divisible by 4 but not divisible by 100, or it should be divisible by 400.

```
SET SERVEROUTPUT ON;
DECLARE
   v_YEAR NUMBER(4) := &P_YEAR;
   v_REMAINDER1 NUMBER(5,2);
   v_REMAINDER2 NUMBER(5,2);
   v_REMAINDER3 NUMBER(5,2);
BEGIN
   v_REMAINDER1 := MOD(v_YEAR,4);
   v_REMAINDER2 := MOD(v_YEAR,100);
   v_REMAINDER3 := MOD(v_YEAR,400);
   IF ((v_REMAINDER1 = 0 AND v_REMAINDER2 <> 0 ) OR
       v_REMAINDER3 = 0) THEN
      DBMS_OUTPUT.PUT_LINE(v_YEAR || ' is a leap year');
   ELSE
      DBMS_OUTPUT.PUT_LINE(v_YEAR || ' is not a leap
      year');
   END IF;
END;
/
```

2. Test your solution by using the following table. For example, if the year entered is 1990, the output should be “1990 is not a leap year.”

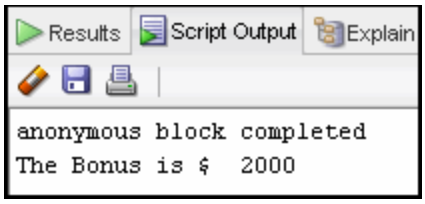
1990	Not a leap year
2000	Leap year
1996	Leap year
1886	Not a leap year
1992	Leap year
1824	Leap year

Practice 6-2: Writing Control Structures

1. Write a PL/SQL block to store the monthly salary of an employee in a substitution variable. The PL/SQL block should:
   - Calculate the annual salary as salary * 12
   - Calculate the bonus as indicated in the following table:

Annual Salary	Bonus
>= 20,000	2,000
19,999–10,000	1,000
<= 9,999	500

   - Display the amount of the bonus in the Script Output window in the following format:



2. Test the PL/SQL for the following test cases:

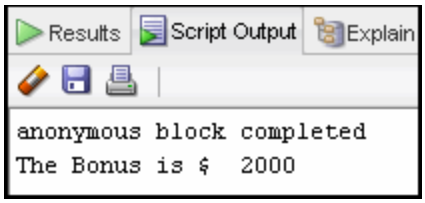
Monthly Salary	Bonus
3000	2000
1200	1000
800	500

Solution 6-2: Writing Control Structures

1. Write a PL/SQL block to store the monthly salary of an employee in a substitution variable. The PL/SQL block should:
   - Calculate the annual salary as salary * 12
   - Calculate the bonus as indicated in the following table:

Annual Salary	Bonus
>= 20,000	2,000
19,999–10,000	1,000
<= 9,999	500

   - Display the amount of the bonus in the Script Output window in the following format:




```
SET SERVEROUTPUT ON;
DECLARE
   V_SAL         NUMBER(7,2) := &B_SALARY;
   V_BONUS       NUMBER(7,2);
   V_ANN_SALARY NUMBER(15,2);
BEGIN
   V_ANN_SALARY := V_SAL * 12;
   IF V_ANN_SALARY >= 20000 THEN
      V_BONUS := 2000;
   ELSIF V_ANN_SALARY <= 19999 AND V_ANN_SALARY >=10000 THEN
      V_BONUS := 1000;
   ELSE
      V_BONUS := 500;
   END IF;
   DBMS_OUTPUT.PUT_LINE ('The Bonus is $  ' ||
     TO_CHAR(V_BONUS));
END;
/
```

2. Test the PL/SQL for the following test cases:

Monthly Salary	Bonus
3000	2000
1200	1000
800	500

Additional Practices and Solutions for Lesson 7: Working with Composite Data Types

Chapter 17

Additional Practices for Lessons Titled "Working with Composite Data Types" and "Using Explicit Cursors"

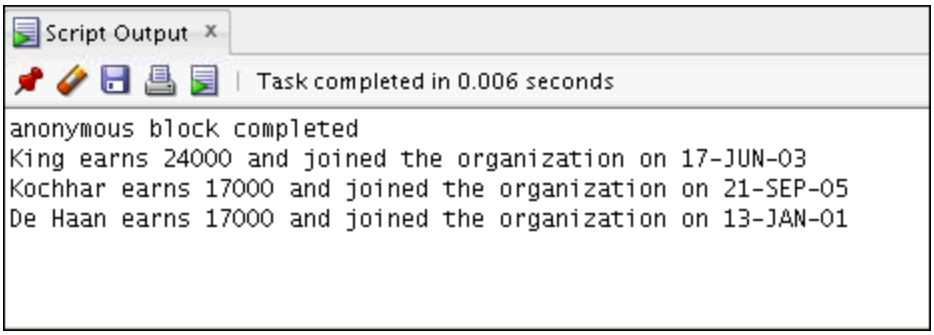
Overview

In the following exercises, you practice using associative arrays (this topic is covered in the lesson titled "Working with Composite Data Types") and explicit cursors (this topic is covered in the lesson titled "Using Explicit Cursors"). In the first exercise, you define and use an explicit cursor to fetch data. In the second exercise, you combine the use of associative arrays with an explicit cursor to output data that meets a certain criteria.

Practice 7/8-1: Fetching Data with an Explicit Cursor

In this practice, you create a PL/SQL block to perform the following:

1. Declare a cursor named `EMP_CUR` to select the employee's last name, salary, and hire date from the `EMPLOYEES` table.
2. Process each row from the cursor, and if the salary is greater than 15,000 and the hire date is later than 01-FEB-1988, display the employee name, salary, and hire date in the format shown in the following sample output:



Solution 7/8-1: Fetching Data with an Explicit Cursor

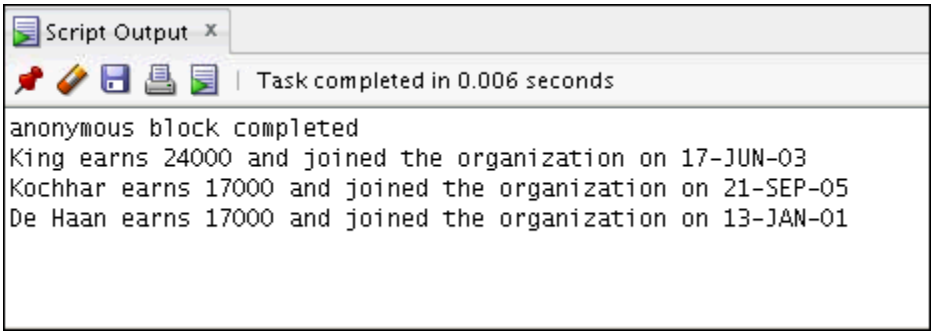
In this practice, you create a PL/SQL block to perform the following:

1. Declare a cursor named `EMP_CUR` to select the employee's last name, salary, and hire date from the `EMPLOYEES` table.

```
SET SERVEROUTPUT ON;
DECLARE
  CURSOR C_EMP_CUR IS
    SELECT  last_name,salary,hire_date FROM EMPLOYEES;
  V_ENAME VARCHAR2(25);
  v_SAL   NUMBER(7,2);
  V_HIREDATE DATE;
```

2. Process each row from the cursor, and if the salary is greater than 15,000 and the hire date is later than 01-FEB-1988, display the employee name, salary, and hire date in the format shown in the following sample output:

```
BEGIN
  OPEN C_EMP_CUR;
  FETCH C_EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
  WHILE C_EMP_CUR%FOUND
  LOOP
    IF V_SAL > 15000 AND V_HIREDATE >=
       TO_DATE('01-FEB-1988','DD-MON-YYYY') THEN
         DBMS_OUTPUT.PUT_LINE (V_ENAME || ' earns '
         || TO_CHAR(V_SAL)|| ' and joined the organization on '
         || TO_DATE(V_HIREDATE,'DD-Mon-YYYY'));
    END IF;
    FETCH C_EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
  END LOOP;
  CLOSE C_EMP_CUR;
END;
/
```

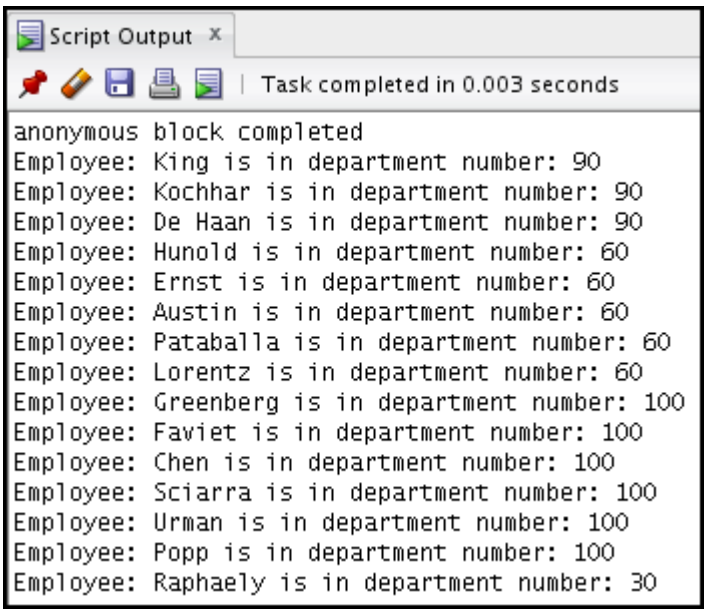


Practice 7/8-2: Using Associative Arrays and Explicit Cursors

In this practice, you create a PL/SQL block to retrieve and output the last name and department ID of each employee from the `EMPLOYEES` table for those employees whose `EMPLOYEE_ID` is less than 115.

In the PL/SQL block, use a cursor `FOR` loop strategy instead of the `OPEN` / `FETCH` / `CLOSE` cursor methods used in the previous practice.

1. In the declarative section:
   - Create two associative arrays. The unique key column for both arrays should be of the `BINARY INTEGER` data type. One array holds the employee's last name and the other holds the department ID.
   - Declare a cursor that selects the last name and department ID for employees whose ID is less than 115
   - Declare the appropriate counter variable to be used in the executable section
2. In the executable section, use a cursor `FOR` loop (covered in the lesson titled "Using Explicit Cursors") to access the cursor values, assign them to the appropriate associative arrays, and output those values from the arrays. The correct output should return 15 rows, in the following format:



Solution 7/8-2: Using Associative Arrays and Explicit Cursors

In this practice, you create a PL/SQL block to retrieve and output the last name and department ID of each employee from the EMPLOYEES table for those employees whose EMPLOYEE_ID is less than 115.

In the PL/SQL block, use a cursor FOR loop strategy instead of the OPEN / FETCH / CLOSE cursor methods used in the previous practice.

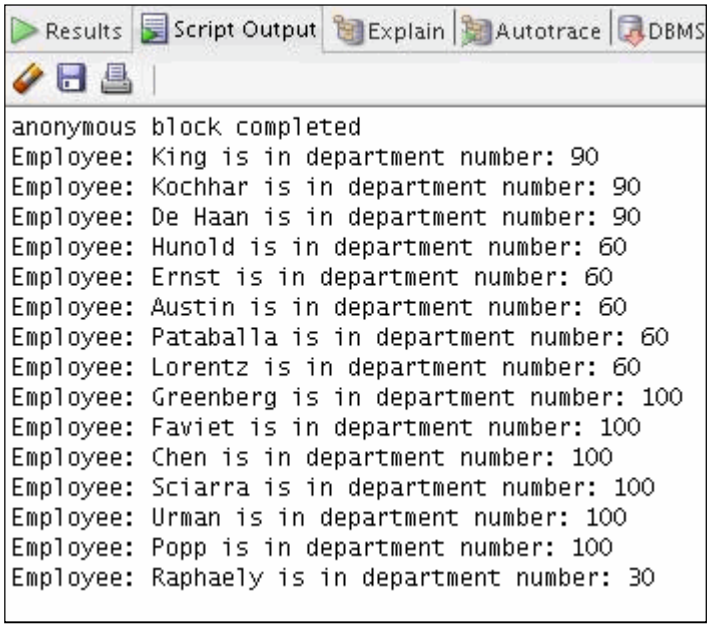
1. In the declarative section:
   - Create two associative arrays. The unique key column for both arrays should be of the BINARY INTEGER data type. One array holds the employee's last name and the other holds the department ID.
   - Declare a counter variable to be used in the executable section
   - Declare a cursor that selects the last name and department ID for employees whose ID is less than 115

```
SET SERVEROUTPUT ON;
DECLARE
      TYPE Table_Ename IS table of employees.last_name%TYPE
            INDEX BY BINARY_INTEGER;
      TYPE Table_dept IS table of employees.department_id%TYPE
            INDEX BY BINARY_INTEGER;
      Tename Table_Ename;
      Tdept  Table_dept;
      i BINARY_INTEGER :=0;
      CURSOR Namedept IS SELECT last_name,department_id
  FROM employees WHERE employee_id < 115;
```

2. In the executable section, use a cursor FOR loop (covered in the lesson titled "Using Explicit Cursors") to access the cursor values, assign them to the appropriate associative arrays, and output those values from the arrays.

```
BEGIN
  FOR emprec in Namedept
  LOOP
    i         := i +1;
    Tename(i) := emprec.last_name;
     Tdept(i)  := emprec.department_id;
    DBMS_OUTPUT.PUT_LINE ('Employee: ' || Tename(i) ||
' is in department number: ' || Tdept(i));
  END LOOP;
END;
/
```

The correct output should return 15 rows, similar to the following:



Additional Practices and Solutions for Lesson 8: Using Explicit Cursors

Chapter 18

Practices for Lesson 8

Practices Overview

Practices of this lesson are included in Practice 7.

Additional Practices and Solutions for Lesson 9: Handling Exceptions

Chapter 19

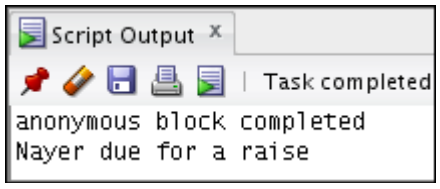
Practice 9-1: Handling Exceptions

For this exercise, you must first create a table to store some results. Run the `lab_ap_09.sql` script that creates the table for you. The script looks like the following:

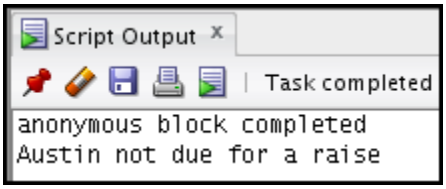
```
CREATE TABLE analysis
       (ename Varchar2(20), years Number(2), sal Number(8,2)
     );
```

In this practice, you write a PL/SQL block that handles an exception, as follows:

1. Declare variables for the employee last name, salary, and hire date. Use a substitution variable for the employee last name. Then, query the `EMPLOYEES` table for the `last_name`, `salary`, and `hire_date` of the specified employee.
2. If the employee has been with the organization for more than five years, and if that employee's salary is less than 3,500, raise an exception. In the exception handler, perform the following:
   - Output the following information: employee last name and the message "due for a raise," similar to the following:

   - Insert the last name, years of service, and salary into the `ANALYSIS` table.
3. If there is no exception, output the employee last name and the message "not due for a raise," similar to the following:

4. Verify the results by querying the `ANALYSIS` table. Use the following test cases to test the PL/SQL block.

LAST_NAME	MESSAGE
Austin	Not due for a raise
Nayer	Due for a raise
Fripp	Not due for a raise
Khoo	Due for a raise

Solution 9-1: Handling Exceptions

For this exercise, you must first create a table to store some results. Run the `lab_ap_09.sql` script that creates the table for you. The script looks similar to the following:

```
CREATE TABLE analysis
       (ename Varchar2(20), years Number(2), sal Number(8,2)
  );
```

In this practice, you write a PL/SQL block that handles an exception, as follows:

1. Declare variables for the employee last name, salary, and hire date. Use a substitution variable for the employee last name. Then, query the `EMPLOYEES` table for the `last_name`, `salary`, and `hire_date` of the specified employee.
2. If the employee has been with the organization for more than five years, and if that employee's salary is less than 3,500, raise an exception. In the exception handler, perform the following:
   - Output the following information: employee last name and the message "due for a raise."
   - Insert the employee name, years of service, and salary into the `ANALYSIS` table.
3. If there is no exception, output the employee last name and the message "not due for a raise."

```
SET SERVEROUTPUT ON;
DECLARE
   E_DUE_FOR_RAISE EXCEPTION;
   V_HIREDATE EMPLOYEES.HIRE_DATE%TYPE;
   V_ENAME EMPLOYEES.LAST_NAME%TYPE := INITCAP( '& B_ENAME');
   V_SAL EMPLOYEES.SALARY%TYPE;
   V_YEARS NUMBER(2);
BEGIN
   SELECT SALARY,HIRE_DATE,MONTHS_BETWEEN(SYSDATE,hire_date)/12
   YEARS
   INTO V_SAL,V_HIREDATE,V_YEARS
   FROM employees WHERE last_name = V_ENAME;
        IF V_SAL < 3500 AND V_YEARS > 5 THEN
             RAISE E_DUE_FOR_RAISE;
        ELSE
             DBMS_OUTPUT.PUT_LINE (V_ENAME|| ' not due for a
   raise');
        END IF;
   EXCEPTION
        WHEN E_DUE_FOR_RAISE THEN
        BEGIN
             DBMS_OUTPUT.PUT_LINE (V_ENAME || ' due for a raise');
             INSERT INTO ANALYSIS(ENAME,YEARS,SAL)
             VALUES (V_ENAME,V_YEARS,V_SAL);
```

```
        END;
END;
/
```

4. Verify the results by querying the `ANALYSIS` table. Use the following test cases to test the PL/SQL block.

LAST_NAME	MESSAGE
Austin	Not due for a raise
Nayer	Due for a raise
Fripp	Not due for a raise
Khoo	Due for a raise

```
SELECT * FROM analysis;
```

	ENAME	YEARS	SAL
1	Nayer	14	3200
2	Khoo	16	3100