

# Oracle Database: PL/SQL Fundamentals

## Additional Practices

D64254GC11

Edition 1.1

March 2012

D76336

**ORACLE**

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

#### **Disclaimer**

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

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

#### **Restricted Rights Notice**

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

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

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

#### **Trademark Notice**

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

#### **Authors**

Prathima Trivedi, Brian Pottle

#### **Technical Contributors and Reviewers**

Diganta Choudhury, Supriya Anant, Krishnachitta, Bryan Roberts, Kimseong Loh, Laszlo Czinkoczki, Ashita Dhir, Peter Driver, Brent Dayley, Gerlinde Frenzen, Nancy Greenberg, Swarnapriya Shridhar, Manish Pawar, Tim Lenlanc, Yanti Chang, Lex Van Der Werff, Hilda Simon

#### **Editors**

Richard Wallis, Smita Kommini

#### **Graphic Designer**

Rajiv Chandrabhanu

#### **Publishers**

Syed Imtiaz Ali, Sumesh Koshy

This book was published using: **oracle***tutor*

# Table of Contents

---

|  |            |
|--|------------|
| <b>Additional Practices and Solutions for Lesson 1</b> .....       | <b>1-1</b> |
| Practices for Lesson 1.....  | 1-2        |
| <b>Additional Practices and Solutions for Lesson 2</b> .....       | <b>2-1</b> |
| Additional Practices for Lesson 2.....                             | 2-2        |
| Practice 2: Evaluating Declarations.....                           | 2-3        |
| Solution 2: Evaluating Declarations.....                           | 2-4        |
| <b>Additional Practices and Solutions for Lesson 3</b> .....       | <b>3-1</b> |
| Practice 3: Evaluating Expressions.....                            | 3-2        |
| Solution 3: Evaluating Expressions.....                            | 3-3        |
| <b>Additional Practices and Solutions for Lesson 4</b> .....       | <b>4-1</b> |
| Practice 4: Evaluating Executable Statements.....                  | 4-2        |
| Solution 4: Evaluating Executable Statements.....                  | 4-3        |
| <b>Additional Practices and Solutions for Lesson 5</b> .....       | <b>5-1</b> |
| Practice 5-1: Interacting with the Oracle Server.....              | 5-2        |
| Solution 5-1: Interacting with the Oracle Server.....              | 5-3        |
| Practice 5-2: Interacting with the Oracle Server.....              | 5-4        |
| Solution 5-2: Interacting with the Oracle Server.....              | 5-5        |
| <b>Additional Practices and Solutions for Lesson 6</b> .....       | <b>6-1</b> |
| Practice 6-1: Writing Control Structures.....                      | 6-2        |
| Solution 6-1: Writing Control Structure.....                       | 6-3        |
| Practice 6-2: Writing Control Structures.....                      | 6-4        |
| Solution 6-2: Writing Control Structures.....                      | 6-5        |
| <b>Additional Practices and Solutions for Lesson 7</b> .....       | <b>7-1</b> |
| Additional Practices for Lesson 7 and 8.....                       | 7-2        |
| Practice 7/8-1: Fetching Data with an Explicit Cursor.....         | 7-3        |
| Solution 7/8-1: Fetching Data with an Explicit Cursor.....         | 7-4        |
| Practice 7/8-2: Using Associative Arrays and Explicit Cursors..... | 7-5        |
| Solution 7/8-2: Using Associative Arrays and Explicit Cursors..... | 7-6        |
| <b>Additional Practices and Solutions for Lesson 8</b> .....       | <b>8-1</b> |
| Practices for Lesson 8.....  | 8-2        |
| <b>Additional Practices and Solutions for Lesson 9</b> .....       | <b>9-1</b> |
| Practice 9-1: Handling Exceptions.....                             | 9-2        |
| Solution 9-1: Handling Exceptions.....                             | 9-3        |

Tran Van Binh (tranbinh48ca@gmail.com) has a non-transferable  
license to use this Student Guide.

# **Additional Practices and Solutions for Lesson 1**

## **Chapter 1**

## Practices for Lesson 1

---

### Practices Overview

There are no practices for this lesson.

## **Additional Practices and Solutions for Lesson 2**

### **Chapter 2**

## 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 3**

## 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**

Tran Van Binh (tranbinh48ca@gmail.com) has a non-transferable license to use this Student Guide.

## **Additional Practices and Solutions for Lesson 4**

### **Chapter 4**

## 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:  
**300, 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.**

Tran Van Binh (tranbinh48ca@gmail.com) has a non-transferable license to use this Student Guide.

## **Additional Practices and Solutions for Lesson 5**

### **Chapter 5**

## Practice 5-1: Interacting with the Oracle Server

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         |            |

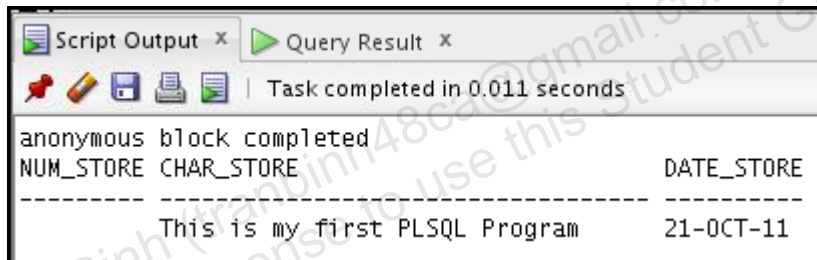
2. Write a PL/SQL block that performs the following:

- 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:



| NUM_STORE | CHAR_STORE                     | DATE_STORE |
|-----------|--------------------------------|------------|
|           | This is my first PLSQL Program | 21-OCT-11  |

## Solution 5-1: Interacting with the Oracle Server

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)**

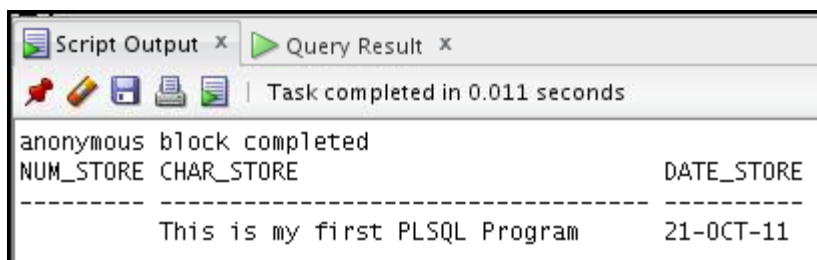
**VALUES (V\_MESSAGE, V\_DATE\_WRITTEN);**

**END;**

**/**

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

**SELECT \* FROM TEMP;**



The screenshot shows a window titled 'Script Output' and 'Query Result'. It displays the message 'Task completed in 0.011 seconds' and 'anonymous block completed'. Below this, the query results are shown in a table format with columns NUM\_STORE, CHAR\_STORE, and DATE\_STORE. The data row shows 'This is my first PLSQL Program' in CHAR\_STORE and '21-OCT-11' in DATE\_STORE.

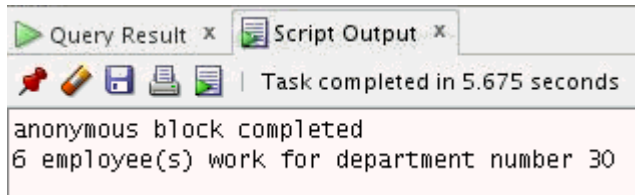
|           |                                |            |
|-----------|--------------------------------|------------|
| NUM_STORE | CHAR_STORE                     | DATE_STORE |
|           | This is my first PLSQL Program | 21-OCT-11  |

## Practice 5-2: Interacting with the Oracle Server

---

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: Interacting with the Oracle Server

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
```

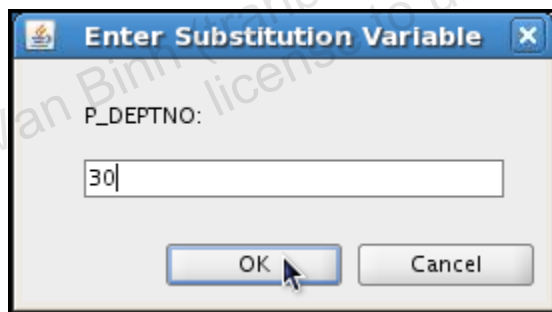
```
    WHERE department_id = V_DEPTNO;
```

```
    DBMS_OUTPUT.PUT_LINE (V_HOWMANY || ' employee(s)  
    work for department number ' || V_DEPTNO);
```

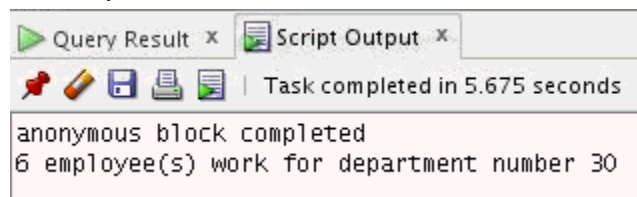
```
END;
```

```
/
```

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:



Tran Van Binh (tranbinh48ca@gmail.com) has a non-transferable license to use this Student Guide.



## **Additional Practices and Solutions for Lesson 6**

### **Chapter 6**

## 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 Structure

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       |

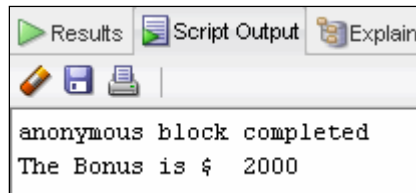
## Practice 6-2: Writing Control Structures

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   |

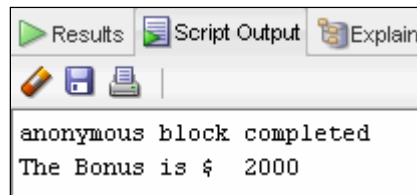
## Solution 6-2: Writing Control Structures

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   |

Tran Van Binh (tranbinh48ca@gmail.com) has a non-transferable license to use this Student Guide.

## **Additional Practices and Solutions for Lesson 7**

### **Chapter 7**

## Additional Practices for Lesson 7 and 8

---

### Overview

In the following exercises, you practice using associative arrays (this topic is covered in Lesson 7) and explicit cursors (this topic is covered in Lesson 8). 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.

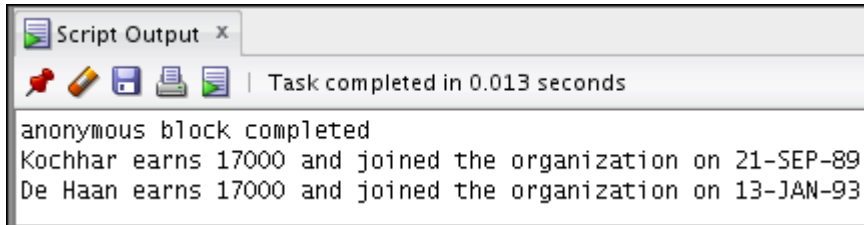


## Practice 7/8-1: Fetching Data with an Explicit Cursor

---

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:



```
Script Output x
Task completed in 0.013 seconds
anonymous block completed
Kochhar earns 17000 and joined the organization on 21-SEP-89
De Haan earns 17000 and joined the organization on 13-JAN-93
```

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

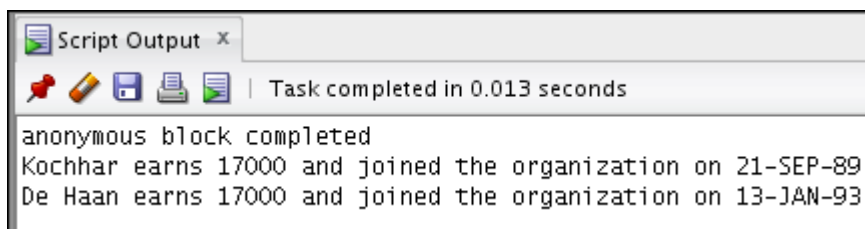
In this practice, you create a PL/SQL block to perform the following:

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;
```

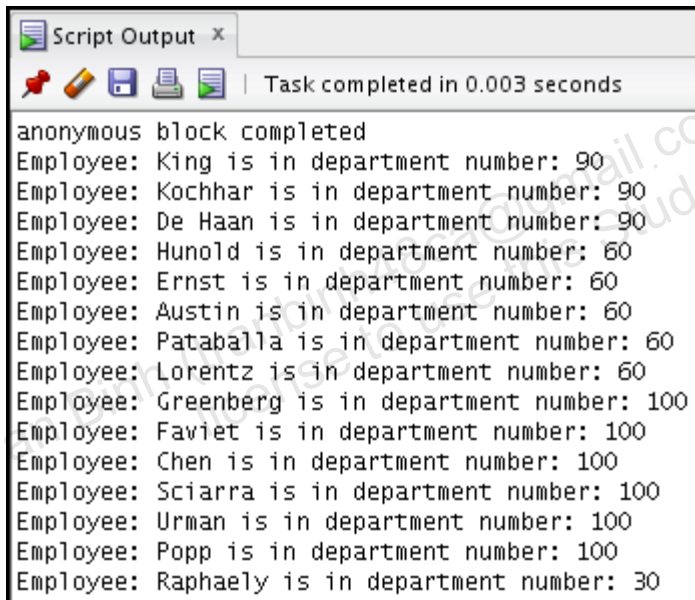


## Practice 7/8-2: Using Associative Arrays and Explicit Cursors

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 Lesson 8) 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:



```
Script Output x
Task completed in 0.003 seconds

anonymous block completed
Employee: King is in department number: 90
Employee: Kochhar is in department number: 90
Employee: De Haan is in department number: 90
Employee: Hunold is in department number: 60
Employee: Ernst is in department number: 60
Employee: Austin is in department number: 60
Employee: Pataballa is in department number: 60
Employee: Lorentz is in department number: 60
Employee: Greenberg is in department number: 100
Employee: Faviet is in department number: 100
Employee: Chen is in department number: 100
Employee: Sciarra is in department number: 100
Employee: Urman is in department number: 100
Employee: Popp is in department number: 100
Employee: Raphaely is in department number: 30
```

## 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 Lesson 8) 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) ||
```

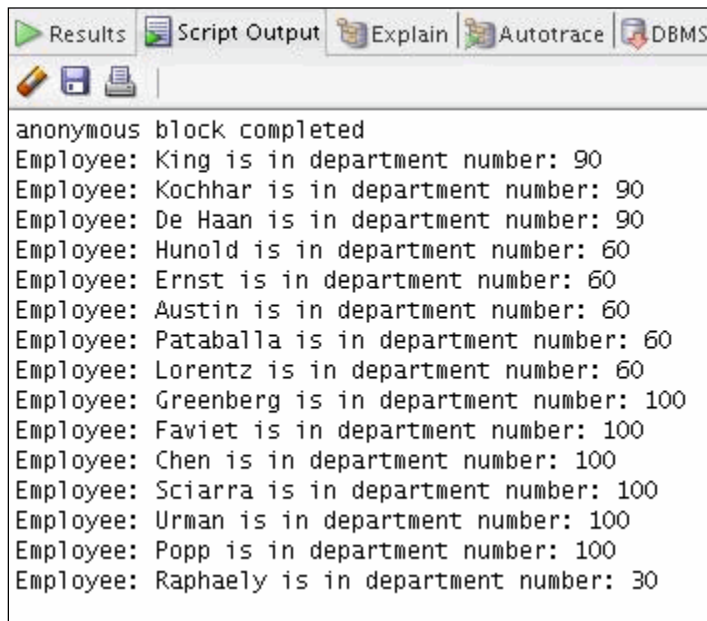
```
        ' is in department number: ' || Tdept(i));
```

```
    END LOOP;
```

```
END;
```

```
/
```

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



The screenshot shows the 'Results' tab in SQL Developer. The output consists of 15 rows, each representing an employee and their department number. The first row is 'anonymous block completed'. The subsequent 14 rows are employee records.

| Employee                  | Department Number |
|---------------------------|-------------------|
| anonymous block completed |                   |
| Employee: King            | 90                |
| Employee: Kochhar         | 90                |
| Employee: De Haan         | 90                |
| Employee: Hunold          | 60                |
| Employee: Ernst           | 60                |
| Employee: Austin          | 60                |
| Employee: Pataballa       | 60                |
| Employee: Lorentz         | 60                |
| Employee: Greenberg       | 100               |
| Employee: Faviet          | 100               |
| Employee: Chen            | 100               |
| Employee: Sciarra         | 100               |
| Employee: Urman           | 100               |
| Employee: Popp            | 100               |
| Employee: Raphaely        | 30                |

Tran Van Binh (tranbinh48ca@gmail.com) has a non-transferable license to use this Student Guide.

## **Additional Practices and Solutions for Lesson 8**

### **Chapter 8**

## Practices for Lesson 8

---

### Practices Overview

Practices of this lesson are included with lesson 7.



## **Additional Practices and Solutions for Lesson 9**

### **Chapter 9**

## Practice 9-1: Handling Exceptions

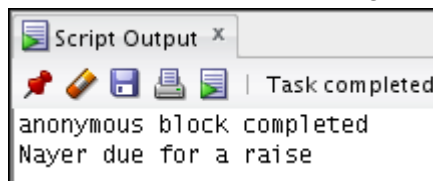
For this exercise, you must first create a table to store some results. Run the `lab_ap_09.sql` script that creates the table for you. The script looks like the following:

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

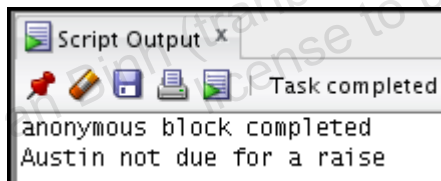
In this practice, you write a PL/SQL block that handles an exception, as follows:

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 LAST_NAME, SALARY, HIRE_DATE
  INTO V_ENAME, V_SAL, V_HIREDATE
  FROM employees WHERE last_name = V_ENAME;
  V_YEARS := MONTHS_BETWEEN(SYSDATE, V_HIREDATE) / 12;
  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 |