

---

# **Oracle9i: PL/SQL Fundamentals**

**Additional Practices**

---

40055GC11  
Production 1.1  
November 2001  
D34070

**ORACLE®**

## **Authors**

Priya Nathan

## **Technical Contributors and Reviewers**

Anna Atkinson

Cesljas Zarco

Chaya Rao

Coley William

Daniel Gabel

Dr. Christoph Burandt

Helen Robertson

Judy Brink

Laszlo Czinkoczki

Laura Pezzini

Linda Boldt

Marco Verbeek

Nagavalli Pataballa

Robert Squires

Sarah Jones

Stefan Lindblad

Sue Onraet

Susan Dee

## **Publisher**

May Lonn Chan-Villareal

**Copyright © Oracle Corporation, 1999, 2000, 2001. All rights reserved.**

This documentation contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited. If this documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

### **Restricted Rights Legend**

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights," as defined in FAR 52.227-14, Rights in Data-General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Education Products, Oracle Corporation, 500 Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

All references to Oracle and Oracle products are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

# Contents

## Preface

## Curriculum Map

### Introduction

- Course Objectives 1-2
- About PL/SQL 1-3
- PL/SQL Environment 1-4
- Benefits of PL/SQL 1-5
- Summary 1-10

### 1 Declaring Variables

- Objectives 1-2
- PL/SQL Block Structure 1-3
- Executing Statements and PL/SQL Blocks 1-4
- Block Types 1-5
- Program Constructs 1-6
- Use of Variables 1-7
- Handling Variables in PL/SQL 1-8
- Types of Variables 1-9
- Using iSQL\*Plus Variables Within PL/SQL Blocks 1-10
- Types of Variables 1-11
- Declaring PL/SQL Variables 1-12
- Guidelines for Declaring PL/SQL Variables 1-13
- Naming Rules 1-14
- Variable Initialization and Keywords 1-15
- Scalar Data Types 1-17
- Base Scalar Data Types 1-18
- Scalar Variable Declarations 1-22
- The %TYPE Attribute 1-23
- Declaring Variables with the %TYPE Attribute 1-24
- Declaring Boolean Variables 1-25
- Composite Data Types 1-26
- LOB Data Type Variables 1-27
- Bind Variables 1-28
- Using Bind Variables 1-30
- Referencing Non-PL/SQL Variables 1-31
- DBMS\_OUTPUT.PUT\_LINE 1-32
- Summary 1-33
- Practice 1 Overview 1-35

## **2 Writing Executable Statements**

- Objectives 2-2
- PL/SQL Block Syntax and Guidelines 2-3
- Identifiers 2-5
- PL/SQL Block Syntax and Guidelines 2-6
- Commenting Code 2-7
- SQL Functions in PL/SQL 2-8
- SQL Functions in PL/SQL: Examples 2-9
- Data Type Conversion 2-10
- Nested Blocks and Variable Scope 2-13
- Identifier Scope 2-15
- Qualify an Identifier 2-16
- Determining Variable Scope 2-17
- Operators in PL/SQL 2-18
- Programming Guidelines 2-20
- Indenting Code 2-21
- Summary 2-22
- Practice 2 Overview 2-23

## **3 Interacting with the Oracle Server**

- Objectives 3-2
- SQL Statements in PL/SQL 3-3
- SELECT Statements in PL/SQL 3-4
- Retrieving Data in PL/SQL 3-7
- Naming Conventions 3-9
- Manipulating Data Using PL/SQL 3-10
- Inserting Data 3-11
- Updating Data 3-12
- Deleting Data 3-13
- Merging Rows 3-14
- Naming Conventions 3-16
- SQL Cursor 3-18
- SQL Cursor Attributes 3-19
- Transaction Control Statements 3-21
- Summary 3-22
- Practice 3 Overview 3-24

## **4 Writing Control Structures**

- Objectives 4-2
- Controlling PL/SQL Flow of Execution 4-3
- IF Statements 4-4
  - Simple IF Statements 4-5
  - Compound IF Statements 4-6
  - IF-THEN-ELSE Statement Execution Flow 4-7
  - IF-THEN-ELSE Statements 4-8
  - IF-THEN-ELSIF Statement Execution Flow 4-9
  - IF-THEN-ELSIF Statements 4-11
- CASE Expressions 4-12
- CASE Expressions: Example 4-13
- Handling Nulls 4-15
- Logic Tables 4-16
- Boolean Conditions 4-17
- Iterative Control: LOOP Statements 4-18
  - Basic Loops 4-19
  - WHILE Loops 4-21
  - FOR Loops 4-23
  - Guidelines While Using Loops 4-26
- Nested Loops and Labels 4-27
- Summary 4-29
- Practice 4 Overview 4-30

## **5 Working with Composite Data Types**

- Objectives 5-2
- Composite Data Types 5-3
  - PL/SQL Records 5-4
  - Creating a PL/SQL Record 5-5
  - PL/SQL Record Structure 5-7
  - The %ROWTYPE Attribute 5-8
  - Advantages of Using %ROWTYPE 5-10
  - The %ROWTYPE Attribute 5-11
- INDEX BY Tables 5-13
  - Creating an INDEX BY Table 5-14
  - INDEX BY Table Structure 5-15
  - Creating an INDEX BY Table 5-16
  - Using INDEX BY Table Methods 5-17
  - INDEX BY Table of Records 5-18
  - Example of INDEX BY Table of Records 5-19
- Summary 5-20
- Practice 5 Overview 5-21

## **6 Writing Explicit Cursors**

Objectives 6-2

About Cursors 6-3

Explicit Cursor Functions 6-4

Controlling Explicit Cursors 6-5

Declaring the Cursor 6-9

Opening the Cursor 6-11

Fetching Data from the Cursor 6-12

Closing the Cursor 6-14

Explicit Cursor Attributes 6-15

The %ISOPEN Attribute 6-16

Controlling Multiple Fetches 6-17

The %NOTFOUND and %ROWCOUNT Attributes 6-18

Example 6-20

Cursors and Records 6-21

Cursor FOR Loops 6-22

Cursor FOR Loops Using Subqueries 6-24

Summary 6-26

Practice 6 Overview 6-27

## **7 Advanced Explicit Cursor Concepts**

Objectives 7-2

Cursors with Parameters 7-3

The FOR UPDATE Clause 7-5

The WHERE CURRENT OF Clause 7-7

Cursors with Subqueries 7-9

Summary 7-10

Practice 7 Overview 7-11

## **8 Handling Exceptions**

Objectives 8-2

Handling Exceptions with PL/SQL 8-3

Handling Exceptions 8-4

Exception Types 8-5

Trapping Exceptions 8-6

Trapping Exceptions Guidelines 8-7

Trapping Predefined Oracle Server Errors 8-8

Predefined Exceptions 8-11  
Trapping Nonpredefined Oracle Server Errors 8-12  
Nonpredefined Error 8-13  
Functions for Trapping Exceptions 8-14  
Trapping User-Defined Exceptions 8-16  
User-Defined Exceptions 8-17  
Calling Environments 8-18  
Propagating Exceptions 8-19  
The RAISE\_APPLICATION\_ERROR Procedure 8-20  
RAISE\_APPLICATION\_ERROR 8-22  
Summary 8-23  
Practice 8 Overview 8-24

**A Practice Solutions**

**B Table Description and Data**

**C REF Cursors**

**Additional Practices**

**Additional Practice Solutions**





---

# **Additional Practices**

---



## **Additional Practices Overview**

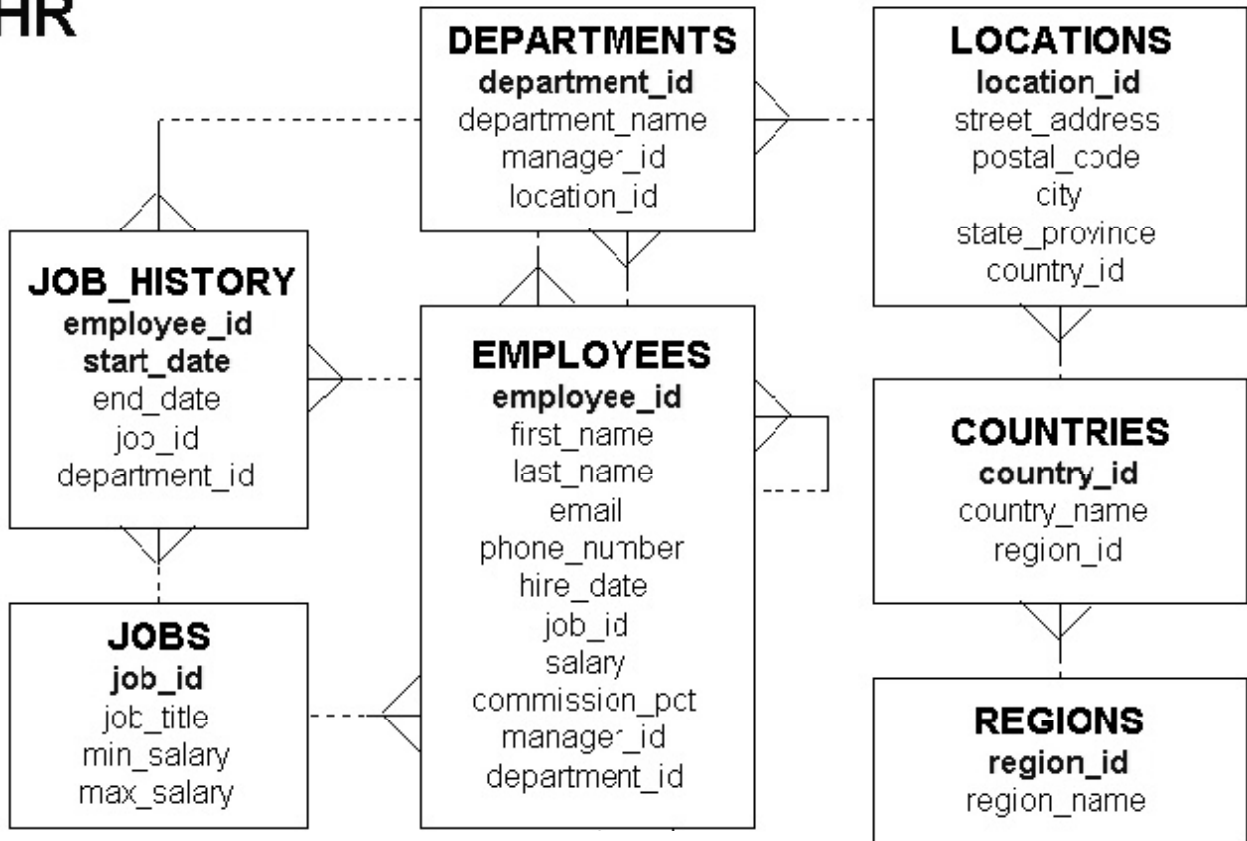
These additional practices are provided as a supplement to the course *Oracle9i: PL/SQL Fundamentals*. In these practices, you apply the concepts that you learned in *Oracle9i: PL/SQL Fundamentals*.

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.

## ENTITY RELATIONSHIP DIAGRAM

Human Resources

**HR**



**Note: These exercises can be used for extra practice when discussing how to declare variables and write executable statements.**

1. Evaluate each of the following declarations. Determine which of them are not legal and explain why.
  - a. DECLARE  
v\_name, v\_dept VARCHAR2(14);
  - b. DECLARE  
v\_test NUMBER(5);
  - c. DECLARE  
V\_MAXSALARY NUMBER(7,2) = 5000;
  - d. DECLARE  
V\_JOINDATE BOOLEAN := SYSDATE;
2. In each of the following assignments, determine the data type of the resulting expression.
  - a. v\_email := v\_firstname || to\_char(v\_empno);
  - b. v\_confirm := to\_date('20-JAN-1999', 'DD-MON-YYYY');
  - c. v\_sal := (1000\*12) + 500
  - d. v\_test := FALSE;
  - e. v\_temp := v\_temp1 < (v\_temp2/ 3);
  - f. v\_var := sysdate;

3. 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;

2

```
v_custid := (v_custid *12) / 10;
```

END;

/

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

- The value of V\_CUSTID at position 1 is:
- The value of V\_CUSTNAME at position 1 is:
- The value of V\_NEW\_CUSTID at position 2 is:
- The value of V\_NEW\_CUSTNAME at position 1 is:
- The value of V\_CUSTID at position 2 is:
- The value of V\_CUSTNAME at position 2 is:

**Note:** These exercises can be used for extra practice when discussing how to interact with the Oracle server and write control structures.

- Write a PL/SQL block to accept a year and check whether it is a leap year. For example, if the year entered is 1990, the output should be "1990 is not a leap year."

**Hint:** The year should be exactly divisible by 4 but not divisible by 100, or it should be divisible by 400.

Test your solution with the following years:

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

```
old 2: V_YEAR NUMBER(4) := &P_YEAR;
new 2: V_YEAR NUMBER(4) := 1990;
1990 is not a leap year
PL/SQL procedure successfully completed.
```

5. a. For the exercises below, you will require a temporary table to store the results. You can either create the table yourself or run the labAp\_05.sql script that will create the table for you. Create a table named TEMP with the following three columns:

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

- b. Write a PL/SQL block that contains two variables, MESSAGE and DATE\_WRITTEN. Declare MESSAGE as VARCHAR2 data type with a length of 35 and DATE\_WRITTEN as DATE data type. Assign the following values to the variables:

Variable	Contents
MESSAGE	'This is my first PL/SQL program'
DATE_WRITTEN	Current date

Store the values in appropriate columns of the TEMP table. Verify your results by querying the TEMP table.

NUM_STORE	CHAR_STORE	DATE_STORE
	This is my first PLSQL Program	24-SEP-01

6. a. Store a department number in a *iSQL\*Plus* substitution variable
- b. Write a PL/SQL block to print the number of people working in that department.

**Hint:** Enable DBMS\_OUTPUT in *iSQL\*Plus* with SET SERVEROUTPUT ON.

```
old 3: V_DEPTNO DEPARTMENTS.department_id%TYPE := &P_DEPTNO;
new 3: V_DEPTNO DEPARTMENTS.department_id%TYPE := 30;
6 employee(s) work for department number 30
PL/SQL procedure successfully completed.
```

7. Write a PL/SQL block to declare a variable called v\_salary to store the salary of an employee. In the executable part of the program, do the following:
  - a. Store an employee name in a *iSQL\*Plus* substitution variable
  - b. Store his or her salary in the variable v\_salary
  - c. If the salary is less than 3,000, give the employee a raise of 500 and display the message '<Employee Name>'s salary updated' in the window.
  - d. If the salary is more than 3,000, print the employee's salary in the format, '<Employee Name> earns .....
  - e. Test the PL/SQL for the following last names:

LAST_NAME	SALARY
Pataballa	4800
Greenberg	12000
Ernst	6000

**Note:** Undefine the variable that stores the employee's name at the end of the script.

8. Write a PL/SQL block to store the salary of an employee in an *iSQL\*Plus* substitution variable. In the executable part of the program do the following:
  - Calculate the annual salary as salary \* 12.
  - Calculate the bonus as indicated below:

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

- Display the amount of the bonus in the window in the following format:  
'The bonus is \$.....'



- Test the PL/SQL for the following test cases:

SALARY	BONUS
5000	2000
1000	1000
15000	2000

**Note: These exercises can be used for extra practice when discussing how to work with composite data types, cursors and handling exceptions.**

9. a.. Write a PL/SQL block to store an employee number, the new department number, and the percentage increase in the salary in *iSQL*\*Plus substitution variables.
- b. Update the department ID of the employee with the new department number, and update the salary with the new salary. Use the EMP table for the updates. Once the update is complete, display the message, 'Update complete' in the window. If no matching records are found, display 'No Data Found'. Test the PL/SQL for the following test cases:

EMPLOYEE_ID	NEW_DEPARTMEN T_ID	% INCREASE	MESSAGE
100	20	2	Updation Complete
10	30	5	No Data found
126	40	3	Updation Complete

10. Create a PL/SQL block to declare a cursor EMP\_CUR to select the employee name, salary, and hire date from the EMPLOYEES table. Process each row from the cursor, and if the salary is greater than 15,000 and the hire date is greater than 01-FEB-1988, display the employee name, salary, and hire date in the window in the format shown in the sample output below:

```
Kochhar earns 17000 and joined the organization on 21-SEP-89
De Haan earns 17000 and joined the organization on 13-JAN-93
PL/SQL procedure successfully completed.
```

11. Create a PL/SQL block to retrieve the last name and department ID of each employee from the EMPLOYEES table for those employees whose EMPLOYEE\_ID is less than 114. From the values retrieved from the EMPLOYEES table, populate two PL/SQL tables, one to store the records of the employee last names and the other to store the records of their department IDs. Using a loop, retrieve the employee name information and the salary information from the PL/SQL tables and display it in the window, using DBMS\_OUTPUT.PUT\_LINE. Display these details for the first 15 employees in the PL/SQL tables.

```
Employee Name: King Department_id: 90
Employee Name: Kochhar Department_id: 90
Employee Name: De Haan Department_id: 90
Employee Name: Hunold Department_id: 60
Employee Name: Ernst Department_id: 60
Employee Name: Austin Department_id: 60
Employee Name: Pataballa Department_id: 60
Employee Name: Lorentz Department_id: 60
Employee Name: Greenberg Department_id: 100
Employee Name: Favier Department_id: 100
Employee Name: Chen Department_id: 100
Employee Name: Sciarra Department_id: 100
Employee Name: Urman Department_id: 100
Employee Name: Popp Department_id: 100
Employee Name: Raphaely Department_id: 30
PL/SQL procedure successfully completed.
```

12. a. Create a PL/SQL block that declares a cursor called DATE\_CUR. Pass a parameter of DATE data type to the cursor and print the details of all employees who have joined after that date.

```
DEFINE P_HIREDATE = 08-MAR-00
```

- b. Test the PL/SQL block for the following hire dates: 08-MAR-00, 25-JUN-97, 28-SEP-98, 07-FEB-99.

```
166 Ande 24-MAR-00
```

```
167 Banda 21-APR-00
```

```
173 Kumar 21-APR-00
```

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

13. Create a PL/SQL block to promote clerks who earn more than 3,000 to the job title SR CLERK and increase their salary by 10%. Use the EMP table for this practice. Verify the results by querying on the EMP table. **Hint:** Use a cursor with FOR UPDATE and CURRENT OF syntax.

14. a. For the exercise below, you will require a table to store the results. You can create the ANALYSIS table yourself or run the labAp\_14a.sql script that creates the table for you. Create a table called ANALYSIS with the following three columns:

Column Name	ENAME	YEARS	SAL
Key Type			
Nulls/Unique			
FK Table			
FK Column			
Datatype	VARCHAR2	Number	Number
Length	20	2	8 , 2

- b. Create a PL/SQL block to populate the ANALYSIS table with the information from the EMPLOYEES table. Use an &SQL\*Plus substitution variable to store an employee's last name.
- c. Query the EMPLOYEES table to find if the number of years that the employee has been with the organization is greater than five, and if the salary is less than 3,500, raise an exception. Handle the exception with an appropriate exception handler that inserts the following values into the ANALYSIS table: employee last name, number of years of service, and the current salary. Otherwise display Not due for a raise in the window. 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	Not due for a raise
Fripp	Not due for a raise
Khoo	Due for a raise



---

## **Additional Practice Solutions**

---



## Additional Practice 1 and 2 Solutions

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

a. DECLARE  
v\_name, v\_dept VARCHAR2(14);

**This is illegal because only one identifier per declaration is allowed.**

b. DECLARE  
v\_test NUMBER(5);

**This is legal.**

c. DECLARE  
V\_MAXSALARY NUMBER(7,2) = 5000;

**This is illegal because the assignment operator is wrong. It should be :=.**

d. DECLARE  
V\_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.**

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

a. v\_email := v\_firstname || to\_char(v\_empno);

**Character string**

b. v\_confirm := to\_date('20-JAN-1999', 'DD-MON-YYYY');

**Date**

c. v\_sal := (1000\*12) + 500

**Number**

d. v\_test := FALSE;

**Boolean**

e. v\_temp := v\_temp1 < (v\_temp2/ 3);

**Boolean**

f. v\_var := sysdate;

**Date**

### Additional Practice 3 Solutions

3. 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 PL/SQL block above and determine the data type and value of each of the following variables, according to the rules of scoping:

- The value of V\_CUSTID at position 1 is:  
**300, and the data type is NUMBER**
- The value of V\_CUSTNAME at position 1 is:  
**Shape up Sports Club Jansports Club, and the data type is VARCHAR2**
- The value of V\_NEW\_CUSTID at position 1 is:  
**500, and the data type is NUMBER (or INTEGER)**
- The value of V\_NEW\_CUSTNAME at position 1 is:  
**Jansports Club, and the data type is VARCHAR2**
- The value of V\_CUSTID at position 2 is:  
**1920, and the data type is NUMBER**
- The value of V\_CUSTNAME at position 2 is:  
**Women Sports Club, and the data type is VARCHAR2**



#### Additional Practice 4 Solutions

4. Write a PL/SQL block to accept a year and check whether it is a leap year. For example, if the year entered is 1990, the output should be “1990 is not a leap year”.

**Hint:** The year should be exactly divisible by 4 but not divisible by 100, or it should be divisible by 400.

Test your solution with the following years:

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

```
SET SERVEROUTPUT ON
DEFINE p_year = 1990
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;
/
SET SERVEROUTPUT OFF
```

## Additional Practice 5 Solutions

5. a. For the exercises below, you will require a temporary table to store the results. You can either create the table yourself or run the labAp\_05.sql script that will create the table for you. Create a table named TEMP with the following three columns:

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

```
CREATE TABLE temp
(num_store NUMBER(7,2),
char_store VARCHAR2(35),
date_store DATE);
```

- b. Write a PL/SQL block that contains two variables, MESSAGE and DATE\_WRITTEN. Declare MESSAGE as VARCHAR2 data type with a length of 35 and DATE\_WRITTEN as DATE data type. Assign the following values to the variables:

Variable	Contents
MESSAGE	This is my first PL/SQL program'
DATE_WRITTEN	Current date

Store the values in appropriate columns of the TEMP table. Verify your results by querying the TEMP table.

```
DECLARE
    MESSAGE VARCHAR2(35);
    DATE_WRITTEN DATE;
BEGIN
    MESSAGE := 'This is my first PLSQL Program';
    DATE_WRITTEN := SYSDATE;
    INSERT INTO temp(CHAR_STORE,DATE_STORE)
    VALUES (MESSAGE,DATE_WRITTEN);
END;
/
SELECT * FROM TEMP;
```

## Additional Practice 6 and 7 Solutions

6. a. Store a department number in a iSQL\*Plus substitution variable

```
DEFINE P_DEPTNO = 30
```

- b. Write a PL/SQL block to print the number of people working in that department.

**Hint:** Enable DBMS\_OUTPUT in iSQL\*Plus with SET SERVEROUTPUT ON.

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
    V_COUNT NUMBER(3);
```

```
    V_DEPTNO DEPARTMENTS.department_id%TYPE := &P_DEPTNO;
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO V_COUNT FROM employees
```

```
    WHERE department_id = V_DEPTNO;
```

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

```
END;
```

```
/
```

```
SET SERVEROUTPUT OFF
```

7. Write a PL/SQL block to declare a variable called v\_salary to store the salary of an employee. In the executable part of the program, do the following:

- a. Store an employee name in a iSQL\*Plus substitution variable

```
SET SERVEROUTPUT ON
```

```
DEFINE P_LASTNAME = Pataballa
```

- b. Store his or her salary in the v\_salary variable

- c. If the salary is less than 3,000, give the employee a raise of 500 and display the message '<Employee Name>'s salary updated' in the window.

- d. If the salary is more than 3,000, print the employee's salary in the format, '<Employee Name> earns .....'

- e. Test the PL/SQL for the last names

**Note:** Undefine the variable that stores the employee's name at the end of the script.

```
DECLARE
```

```
    V_SALARY NUMBER(7,2);
```

```
    V_LASTNAME EMPLOYEES.LAST_NAME%TYPE;
```

```
BEGIN
```

```
    SELECT salary INTO V_SALARY
```

```
    FROM employees
```

```
    WHERE last_name = INITCAP('&P_LASTNAME') FOR UPDATE of salary;
```

## Additional Practice 7 and 8 Solutions

```
V_LASTNAME := INITCAP('&P_LASTNAME');  
IF V_SALARY < 3000 THEN  
    UPDATE employees SET salary = salary + 500  
    WHERE last_name = INITCAP('&P_LASTNAME') ;  
    DBMS_OUTPUT.PUT_LINE (V_LASTNAME || ''s salary updated');  
ELSE  
    DBMS_OUTPUT.PUT_LINE (V_LASTNAME || ' earns ' ||  
        TO_CHAR(V_SALARY));  
    END IF;  
END;  
/  
SET SERVEROUTPUT OFF  
UNDEFINE P_LASTNAME
```

8. Write a PL/SQL block to store the salary of an employee in an *iSQL*\*Plus substitution variable. In the executable part of the program do the following:

- Calculate the annual salary as salary \* 12.
- Calculate the bonus as indicated below:

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

- Display the amount of the bonus in the window in the following format:  
‘The bonus is \$.....’

```
SET SERVEROUTPUT ON  
DEFINE P_SALARY = 5000  
DECLARE  
    V_SALARY    NUMBER(7,2) := &P_SALARY;  
    V_BONUS     NUMBER(7,2);  
    V_ANN_SALARY NUMBER(15,2);
```

## Additional Practice 8 and 9 Solutions

```
BEGIN
    V_ANN_SALARY := V_SALARY * 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;
/
SET SERVEROUTPUT OFF
```

9. a. Write a PL/SQL block to store an employee number, the new department number and the percentage increase in the salary in *iSQL*\*Plus substitution variables.

```
SET SERVEROUTPUT ON
DEFINE P_EMPNO = 100
DEFINE P_NEW_DEPTNO = 10
DEFINE P_PER_INCREASE = 2
```

- b. Update the department ID of the employee with the new department number, and update the salary with the new salary. Use the EMP table for the updates. Once the update is complete, display the message, 'Update complete' in the window. If no matching records are found, display the message, 'No Data Found'. Test the PL/SQL.

```
DECLARE
    V_EMPNO emp.EMPLOYEE_ID%TYPE := &P_EMPNO;
    V_NEW_DEPTNO emp.DEPARTMENT_ID%TYPE := & P_NEW_DEPTNO;
    V_PER_INCREASE NUMBER(7,2) := & P_PER_INCREASE;
BEGIN
    UPDATE emp
    SET department_id = V_NEW_DEPTNO,
        salary = salary + (salary *
V_PER_INCREASE/100)
    WHERE employee_id = V_EMPNO;
    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE ('No Data Found');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Update Complete');
    END IF;
END;
/
SET SERVEROUTPUT OFF
```

## Additional Practice 10 Solutions

10. Create a PL/SQL block to declare a cursor EMP\_CUR to select the employee name, salary, and hire date from the EMPLOYEES table. Process each row from the cursor, and if the salary is greater than 15,000 and the hire date is greater than 01-FEB-1988, display the employee name, salary, and hire date in the window.

```
SET SERVEROUTPUT ON

DECLARE
    CURSOR EMP_CUR IS
        SELECT  last_name,salary,hire_date FROM EMPLOYEES;
        V_ENAME VARCHAR2(25);
        V_SAL    NUMBER(7,2);
        V_HIREDATE DATE;
BEGIN
    OPEN EMP_CUR;
    FETCH EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
    WHILE 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 EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
        END LOOP;
    CLOSE EMP_CUR;
END;
/
SET SERVEROUTPUT OFF
```

## Additional Practice 11 Solutions

11. Create a PL/SQL block to retrieve the last name and department ID of each employee from the EMPLOYEES table for those employees whose EMPLOYEE\_ID is less than 114. From the values retrieved from the EMPLOYEES table, populate two PL/SQL tables, one to store the records of the employee last names and the other to store the records of their department IDs. Using a loop, retrieve the employee name information and the salary information from the PL/SQL tables and display it in the window, using DBMS\_OUTPUT.PUT\_LINE. Display these details for the first 15 employees in the PL/SQL tables.

```
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;
```

```
    V_Tename Table_Ename;
```

```
    V_Tdept Table_dept;
```

```
    i BINARY_INTEGER :=0;
```

```
    CURSOR C_Namedept IS SELECT last_name,department_id from employees
```

```
        WHERE employee_id < 115;
```

```
        V_COUNT NUMBER := 15;
```

```
BEGIN
```

```
    FOR emprec in C_Namedept
```

```
        LOOP
```

```
            i := i +1;
```

```
            V_Tename(i) := emprec.last_name;
```

```
            V_Tdept(i) := emprec.department_id;
```

```
        END LOOP;
```

```
    FOR i IN 1..v_count
```

```
        LOOP
```

```
            DBMS_OUTPUT.PUT_LINE ('Employee Name: ' || V_Tename(i) ||  
                                   ' Department_id: ' || V_Tdept(i));
```

```
        END LOOP;
```

```
END;
```

```
/
```

```
SET SERVEROUTPUT OFF
```

## Additional Practice 12 Solutions

12. a. Create a PL/SQL block that declares a cursor called DATE\_CUR. Pass a parameter of DATE data type to the cursor and print the details of all employees who have joined after that date.

```
SET SERVEROUTPUT ON
```

```
DEFINE P_HIREDATE = 08-MAR-00
```

- b. Test the PL/SQL block for the following hire dates: 08-MAR-00, 25-JUN-97, 28-SEP-98, 07-FEB-99.

```
DECLARE
```

```
CURSOR DATE_CURSOR(JOIN_DATE DATE) IS
```

```
SELECT employee_id,last_name,hire_date FROM employees
```

```
WHERE HIRE_DATE >JOIN_DATE ;
```

```
V_EMPNO    employees.employee_id%TYPE;
```

```
V_ENAME     employees.last_name%TYPE;
```

```
V_HIREDATE  employees.hire_date%TYPE;
```

```
V_DATE employees.hire_date%TYPE := '&P_HIREDATE';
```

```
BEGIN
```

```
OPEN DATE_CURSOR(V_DATE);
```

```
LOOP
```

```
FETCH DATE_CURSOR INTO V_EMPNO,V_ENAME,V_HIREDATE;
```

```
EXIT WHEN DATE_CURSOR%NOTFOUND;
```

```
DBMS_OUTPUT.PUT_LINE (V_EMPNO || ' ' || V_ENAME || ' ' ||  
                        V_HIREDATE);
```

```
END LOOP;
```

```
END;
```

```
/
```

```
SET SERVEROUTPUT OFF;
```



### Additional Practice 13 Solutions

13. Create a PL/SQL block to promote clerks who earn more than 3,000 to SR. CLERK and increase their salary by 10%. Use the EMP table for this practice. Verify the results by querying on the EMP table.

**Hint:** Use a cursor with FOR UPDATE and CURRENT OF syntax.

```
DECLARE
    CURSOR C_Senior_Clerk IS
        SELECT employee_id, job_id FROM emp
        WHERE job_id = 'ST_CLERK' AND salary > 3000
        FOR UPDATE OF job_id;
BEGIN
    FOR V_Emrec IN C_Senior_Clerk
    LOOP
        UPDATE emp
        SET job_id = 'ST_CLERK',
            salary = 1.1 * salary
        WHERE CURRENT OF C_Senior_Clerk;
    END LOOP;
    COMMIT;
END;
/
SELECT * FROM emp;
```

## Additional Practice 14 Solutions

14. a. For the exercise below, you will require a table to store the results. You can create the ANALYSIS table yourself or run the labAp\_14a.sql script that creates the table for you. Create a table called ANALYSIS with the following three columns:

Column Name	ENAME	YEARS	SAL
Key Type			
Nulls/Unique			
FK Table			
FK Column			
Datatype	VARCHAR2	Number	Number
Length	20	2	8,2

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

- b. Create a PL/SQL block to populate the ANALYSIS table with the information from the EMPLOYEES table. Use an iSQL\*Plus substitution variable to store an employee's last name.

```
SET SERVEROUTPUT ON
DEFINE P_ENAME = Austin
```

- c. Query the EMPLOYEES table to find if the number of years that the employee has been with the organization is greater than five, and if the salary is less than 3,500, raise an exception. Handle the exception with an appropriate exception handler that inserts the following values into the ANALYSIS table: employee last name, number of years of service, and the current salary. Otherwise display Not due for a raise in the window. Verify the results by querying the ANALYSIS table. Test the PL/SQL block.

```
DECLARE
  DUE_FOR_RAISE EXCEPTION;
  V_HIREDATE EMPLOYEES.HIRE_DATE%TYPE;
  V_ENAME EMPLOYEES.LAST_NAME%TYPE := INITCAP( '& P_ENAME' );
  V_SAL EMPLOYEES.SALARY%TYPE;
  V_YEARS NUMBER(2);
```

## Additional Practice 14 Solutions (continued)

```
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 DUE_FOR_RAISE;
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Not due for a raise');
    END IF;
EXCEPTION
    WHEN DUE_FOR_RAISE THEN
        INSERT INTO ANALYSIS(ENAME,YEARS,SAL)
        VALUES (V_ENAME,V_YEARS,V_SAL);
END;
/
```

