Oracle Database 10*g*: PL/SQL Fundamentals

Volume 2 • Additional Practices

D17112GC21
Edition 2.1
December 2006
D48245

ORACLE

Authors

Tulika Srivastava
Sunitha Patel

Technical Contributors and Reviewers

Chaitanya Koratamaddi
Christoph Burandt
Zarko Cesljas
Dairy Chan
Isabelle Cornu
Kathryn Cunningham
Burt Demchick
Joel Goodman
Jonathan Grove
Jessie Ho
Craig Hollister
Alison Holloway
Bryn Llewellyn
Malika Marghadi
Hildegard Mayr
Nancy Greenberg
Miyuki Osato
Nagavalli Pataballa
Srinivas Putrevu
Bryan Roberts
Helen Robertson
Grant Spencer
Lex Van Der Werff

Editors

Richard Wallis
Arijit Ghosh

Graphic Designer

Steve Elwood

Publishers

Nita Brozowski
Srividya Rameshkumar

Contents

Additional Practices

Additional Practices Overview

These additional practices are provided as a supplement to the course *Oracle Database 10g: PL/SQL Fundamentals*. In these practices, you apply the concepts that you learned in *Oracle Database 10g: 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`.

Additional Practice 1 and 2

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
       name,dept    VARCHAR2(14);
   ```

   b.
   ```
   DECLARE
       test         NUMBER(5);
   ```

   c.
   ```
   DECLARE
       MAXSALARY          NUMBER(7,2) = 5000;
   ```

   d.
   ```
   DECLARE
       JOINDATE           BOOLEAN := SYSDATE;
   ```

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

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

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

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

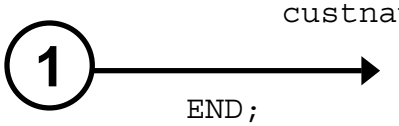
   d. `test := FALSE;`

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

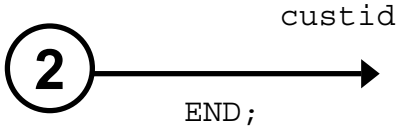
   f. `var := sysdate;`

Additional Practice 3

3.
```
DECLARE
      custid     NUMBER(4) := 1600;
      custname  VARCHAR2(300) := 'Women Sports Club';
      new_custid     NUMBER(3) := 500;
BEGIN
DECLARE
      custid      NUMBER(4) := 0;
      custname  VARCHAR2(300) := 'Shape up Sports Club';
      new_custid  NUMBER(3) := 300;
      new_custname  VARCHAR2(300) := 'Jansports Club';
BEGIN
      custid := new_custid;
      custname := custname || ' ' ||  new_custname;
```




```
END;
      custid := (custid *12) / 10;
```




```
END;
      /
```

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

a. The value of `CUSTID` at position 1 is:
b. The value of `CUSTNAME` at position 1 is:
c. The value of `NEW_CUSTID` at position 2 is:
d. The value of `NEW_CUSTNAME` at position 1 is:
e. The value of `CUSTID` at position 2 is:
f. The value of `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.

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.

Additional Practice 4 and 5

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: YEAR NUMBER(4) := &P_YEAR;
new 2: 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 `lab_ap_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			
Data Type	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	19-FEB-04

Additional Practice 6 and 7

6. a. Store a department number in an *i*SQL*Plus substitution variable.
   b. Write a PL/SQL block to print the number of people working in that department.
   Hint: Enable `DBMS_OUTPUT` in *i*SQL*Plus with `SET SERVEROUTPUT ON`.

```
old 3: DEPTNO DEPARTMENTS.department_id%TYPE :=
&P_DEPTNO;
new 3: 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 `sal` to store the salary of an employee.
   In the executable part of the program, do the following:
   a. Store an employee name in an *i*SQL*Plus substitution variable.
   b. Store his or her salary in the `sal` 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 block 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.

Additional Practice 8 and 9

8. Write a PL/SQL block to store the salary of an employee in an *i*SQL*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. Execute the script `lab_ap_09_a.sql` to create a temporary tabled called `emp`. Write a PL/SQL block to store an employee number, the new department number, and the percentage increase in the salary in *i*SQL*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. After 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 block for the following test cases:

EMPLOYEE_ID	NEW_DEPARTMENT_ID	% INCREASE	MESSAGE
100	20	2	Update Complete
10	30	5	No Data found
126	40	3	Update Complete

Additional Practice 10 and 11

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: Faviet 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.
    ```

Additional Practice 12, 13, and 14

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 the 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. Execute the script `lab_ap_09_a.sql` to re-create the `emp` table. Create a PL/SQL block to promote clerks who earn more than 3,000 to the job title `SR CLERK` and increase their salaries 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 `lab_ap_14_a.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			
Data Type	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 *i*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

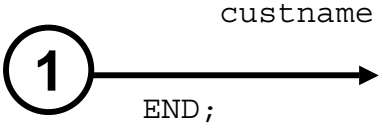
Additional Practice Solutions

Additional Practice 1 and 2: Solutions

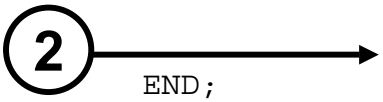
1. Evaluate each of the following declarations. Determine which of them are *not* legal and explain why.
   a.
   ```
   DECLARE
       name,dept    VARCHAR2(14);
   ```
   This is illegal because only one identifier per declaration is allowed.
   b.
   ```
   DECLARE
       test         NUMBER(5);
   ```
   This is legal.
   c.
   ```
   DECLARE
       MAXSALARY          NUMBER(7,2) = 5000;
   ```
   This is illegal because the assignment operator is wrong. It should be :=.
   d.
   ```
   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.
2. In each of the following assignments, determine the data type of the resulting expression.
   a. `email := firstname || to_char(empno);`
   Character string
   b. `confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');`
   Date
   c. `sal := (1000*12) + 500`
   Number
   d. `test := FALSE;`
   Boolean
   e. `temp := temp1 < (temp2/ 3);`
   Boolean
   f. `var := sysdate;`
   Date

Additional Practice 3: Solutions

3.
```
DECLARE
     custid    NUMBER(4) := 1600;
     custname     VARCHAR2(300) := 'Women Sports Club';
     new_custid    NUMBER(3) := 500;
BEGIN
DECLARE
    custid      NUMBER(4) := 0;
    custname  VARCHAR2(300) := 'Shape up Sports Club';
    new_custid  NUMBER(3) := 300;
    new_custname VARCHAR2(300) := 'Jansports Club';
BEGIN
    custid := new_custid;
    custname := custname || ' ' ||  new_custname;
```




```
END;
    custid := (custid *12) / 10;
```




```
END;
     /
```

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

a. The value of CUSTID at position 1 is:
   300, and the data type is NUMBER

b. The value of CUSTNAME at position 1 is:
   Shape up Sports Club Jansports Club, and the data type is VARCHAR2

c. The value of NEW_CUSTID at position 1 is:
   500, and the data type is NUMBER (or INTEGER)

d. The value of NEW_CUSTNAME at position 1 is:
   Jansports Club, and the data type is VARCHAR2

e. The value of CUSTID at position 2 is:
   1920, and the data type is NUMBER

f. The value of 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
DECLARE
    YEAR NUMBER(4) := &P_YEAR;
    REMAINDER1 NUMBER(5,2);
    REMAINDER2 NUMBER(5,2);
    REMAINDER3 NUMBER(5,2);
BEGIN
    REMAINDER1 := MOD(YEAR,4);
    REMAINDER2 := MOD(YEAR,100);
    REMAINDER3 := MOD(YEAR,400);
    IF ((REMAINDER1 = 0 AND REMAINDER2 <> 0 )
                OR REMAINDER3 = 0) THEN
         DBMS_OUTPUT.PUT_LINE(YEAR || ' is a leap year');
    ELSE
         DBMS_OUTPUT.PUT_LINE (YEAR || ' is not a leap year');
    END IF;
END;
/
SET SERVEROUTPUT OFF
```

Additional Practice 5: Solutions

5. a. For the following exercises, you will require a temporary table to store the results. You can either create the table yourself or run the `lab_ap_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			
Data Type	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.

```
SET SERVEROUTPUT ON
    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 Practices 6 and 7 Solutions

6. a. Store a department number in a *i*SQL*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 *i*SQL*Plus with `SET SERVEROUTPUT ON`.

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

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

   a. Store an employee name in an *i*SQL*Plus substitution variable:

```
SET SERVEROUTPUT ON
   DEFINE P_LASTNAME = Pataballa
```

   b. Store his or her salary in the `sal` 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 block for the 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.

Additional Practices 7 and 8: Solutions

```
DECLARE
        SAL NUMBER(7,2);
        LASTNAME EMPLOYEES.LAST_NAME%TYPE;
BEGIN
        SELECT salary INTO SAL
        FROM employees
        WHERE last_name = INITCAP('&&P_LASTNAME') FOR UPDATE of
        salary;

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

8. Write a PL/SQL block to store the salary of an employee in an *i*SQL*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

Additional Practices 8 and 9: Solutions

```
SET SERVEROUTPUT ON
   DEFINE P_SALARY = 5000
   DECLARE
    SAL  NUMBER(7,2) := &P_SALARY;
    BONUS   NUMBER(7,2);
    ANN_SALARY NUMBER(15,2);
BEGIN
   ANN_SALARY := SAL * 12;
   IF ANN_SALARY >= 20000 THEN
      BONUS := 2000;
   ELSIF ANN_SALARY <= 19999 AND ANN_SALARY >=10000 THEN
      BONUS := 1000;
   ELSE
      BONUS := 500;
   END IF;
   DBMS_OUTPUT.PUT_LINE ('The Bonus is $  ' ||
   TO_CHAR(BONUS));
END;
/
SET SERVEROUTPUT OFF
```

9. a. Execute the script `lab_ap_09_a.sql` to create a temporary table called `emp`. Write a PL/SQL block to store an employee number, the new department number, and the percentage increase in the salary in *i*SQL*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. After 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 block for the following test cases.

EMPLOYEE_ID	NEW_DEPARTMENT_ID	% INCREASE	MESSAGE
100	20	2	Update Complete
10	30	5	No Data found
126	40	3	Update Complete

```
DECLARE
   EMPNO emp.EMPLOYEE_ID%TYPE := &P_EMPNO;
   NEW_DEPTNO emp.DEPARTMENT_ID%TYPE := & P_NEW_DEPTNO;
   PER_INCREASE NUMBER(7,2) := & P_PER_INCREASE;
BEGIN
    UPDATE emp
    SET department_id = NEW_DEPTNO,
    salary  = salary + (salary * PER_INCREASE/100)
    WHERE employee_id = 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
```

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;
    ENAME VARCHAR2(25);
    SAL   NUMBER(7,2);
    HIREDATE DATE;
BEGIN
   OPEN EMP_CUR;
   FETCH EMP_CUR INTO ENAME,SAL,HIREDATE;
   WHILE EMP_CUR%FOUND
   LOOP
   IF SAL > 15000 AND HIREDATE >=  TO_DATE('01-FEB-
   1988','DD-MON-
        YYYY') THEN
   DBMS_OUTPUT.PUT_LINE (ENAME ||  ' earns ' ||
   TO_CHAR(SAL)||  `
        and joined the organization on ' ||
   TO_DATE(HIREDATE,'DD-
        Mon-YYYY'));
   END IF;
```

```
FETCH EMP_CUR INTO ENAME,SAL,HIREDATE;
   END LOOP;
CLOSE EMP_CUR;
END;
/
SET SERVEROUTPUT OFF
```

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;
   Tename  Table_Ename;
   Tdept  Table_dept;
   i  BINARY_INTEGER :=0;
   CURSOR Namedept IS SELECT last_name,department_id from
   employees WHERE employee_id < 115;
    TRACK  NUMBER := 15;
BEGIN
   FOR emprec in Namedept
   LOOP
      i := i +1;
      Tename(i) := emprec.last_name;
      Tdept(i) := emprec.department_id;
   END LOOP;
```

```
    FOR i IN 1..TRACK
     LOOP
        DBMS_OUTPUT.PUT_LINE ('Employee Name: ' ||
   Tename(i) || ' Department_id: ' || Tdept(i));
     END LOOP;
END;
/
SET SERVEROUTPUT OFF
```

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 the 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 ;
   EMPNO   employees.employee_id%TYPE;
   ENAME   employees.last_name%TYPE;
    HIREDATE employees.hire_date%TYPE;
   HDATE employees.hire_date%TYPE :=   '&P_HIREDATE';
BEGIN
   OPEN DATE_CURSOR(HDATE);
   LOOP
     FETCH DATE_CURSOR INTO EMPNO,ENAME,HIREDATE;
      EXIT WHEN DATE_CURSOR%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE (EMPNO || ' ' || ENAME || ' ' ||
         HIREDATE);
      END LOOP;
 END;
/
SET SERVEROUTPUT OFF;
```

Additional Practice 13: Solutions

13. Execute the script `lab_ap_09_a.sql` to re-create the `emp` table. Create a PL/SQL block to promote clerks who earn more than 3,000 to SR CLERK and increase their salaries 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 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 Emrec IN Senior_Clerk
   LOOP
    UPDATE emp
    SET job_id = 'SR_CLERK',
    salary = 1.1 * salary
    WHERE CURRENT OF Senior_Clerk;
   END LOOP;
   COMMIT;
END;
/
SELECT * FROM emp;
```

Additional Practice 14: Solutions

14. a. For the following exercise, you will require a table to store the results. You can create the `analysis` table yourself or run the `lab_ap_14_a.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			
Data Type	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 *i*SQL*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. 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 14: Solutions (continued)

```
DECLARE
    DUE_FOR_RAISE EXCEPTION;
    HIREDATE EMPLOYEES.HIRE_DATE%TYPE;
    ENAME EMPLOYEES.LAST_NAME%TYPE :=  INITCAP( '&    P_ENAME');
    SAL EMPLOYEES.SALARY%TYPE;
    YEARS NUMBER(2);
BEGIN
    SELECT LAST_NAME,SALARY,HIRE_DATE
    INTO  ENAME,SAL,HIREDATE
    FROM employees WHERE last_name =  ENAME;
    YEARS := MONTHS_BETWEEN(SYSDATE,HIREDATE)/12;
    IF SAL < 3500 AND 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 (ENAME, YEARS, SAL);
END;
/
```