A
Practice Solutions

Practice 1

Before you begin this practice, ensure that you have seen both the viewlets on *i*SQL*Plus usage.

The `labs` folder is the working directory where you can save your scripts. Ask your instructor for help in locating the `labs` folder for this course. The solutions for all practices are in the `soln` folder.

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

   a.
   ```
   BEGIN
   END;
   ```

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

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

   d.
   ```
   DECLARE
   amount INTEGER(10);
   BEGIN
   DBMS_OUTPUT.PUT_LINE(amount);
   END;
   ```

   *The block in **a** does not execute because the executable section does not have any statements. The block in **b** does not have the mandatory executable section that begins with the **BEGIN** keyword.*

   *The block in **c** has all the necessary parts but the executable section does not have any statements.*

2. Create and execute a simple anonymous block that outputs "Hello World." Execute and save this script as `lab_01_02_soln.sql`.

   a. Start *i*SQL*Plus. Provide login details. The instructor will provide the necessary information.

   b. Type the following code in the workspace.

```
SET SERVEROUTPUT ON
BEGIN
DBMS_OUTPUT.PUT_LINE(' Hello World ');
END;
```

   c. Click the Execute button.

d. You should see the following output:

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

e. Click the Save Script button. Select the folder in which you want to save the file. Enter `lab_01_02_soln.sql` for the file name and click the Save button.

Practice 2

Note: Use *i*SQL*Plus for this practice.

1. Identify valid and invalid identifiers:

a.	`today`	**Valid**
b.	`last_name`	**Valid**
c.	`today's_date`	**Invalid** – character ‘’’ is not allowed
d.	`Number_of_days_in_February_this_year`	**Invalid** – Too long
e.	`Isleap$year`	**Valid**
f.	`#number`	**Invalid –** Cannot start with ‘#’
g.	`NUMBER#`	**Valid**
h.	`number1to7`	**Valid**

2. Identify valid and invalid variable declaration and initialization:

a.	`number_of_copies`	`PLS_INTEGER;`	**Valid**
b.	`PRINTER_NAME`	constant `VARCHAR2(10);`	**Invalid**
c.	`deliver_to`	`VARCHAR2(10)`:=Johnson;	**Invalid**
d.	`by_when`	`DATE`:= `SYSDATE`+1;	**Valid**

*The declaration in **b** is invalid because constant variables must be initialized during declaration.*

*The declaration in **c** is invalid because string literals should be enclosed within single quotes.*

3. Examine the following anonymous block and choose the appropriate statement.

```
SET SERVEROUTPUT ON
DECLARE
fname VARCHAR2(20);
lname VARCHAR2(15) DEFAULT 'fernandez';
BEGIN
DBMS_OUTPUT.PUT_LINE( FNAME ||' ' ||lname);
END;
```

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

   a. The block will execute successfully and print “fernandez.”

4. Create an anonymous block. In *i*SQL*Plus, load the script `lab_01_02_soln.sql`, which you created in exercise 2 of practice 1 by following these instructions:
   Click the Load Script button.
   Browse to select the `lab_01_02_soln.sql` file. Click the Load button. Your workspace will now have the code in the .sql file.

a. Add declarative section to this PL/SQL block. In the declarative section, declare the following variables:
   1. Variable `today` of type `DATE`. Initialize today with `SYSDATE`.

```
DECLARE
today DATE:=SYSDATE;
```

   2. Variable `tomorrow` of type `today`. Use `%TYPE` attribute to declare this variable.

```
tomorrow today%TYPE;
```

b. In the executable section initialize the variable `tomorrow` with an expression, which calculates tomorrow's date (add one to the value in `today`). Print the value of `today` and `tomorrow` after printing "Hello World."

```
BEGIN
tomorrow:=today +1;
DBMS_OUTPUT.PUT_LINE(' Hello World ');
DBMS_OUTPUT.PUT_LINE('TODAY IS : '|| today);
DBMS_OUTPUT.PUT_LINE('TOMORROW IS : ' || tomorrow);
END;
```

c. Execute and save your script as `lab_02_04_soln.sql`. Follow the instructions in step 2 e) of practice 1 to save the file. Sample output is as follows:

```
Hello World
TODAY IS : 12-JAN-04
TOMORROW IS : 13-JAN-04
PL/SQL procedure successfully completed.
```

5. Edit the `lab_02_04_soln.sql` script.

   a. Add code to create two bind variables.
   Create bind variables `basic_percent` and `pf_percent` of type `NUMBER`.

```
VARIABLE basic_percent NUMBER
VARIABLE pf_percent NUMBER
```

   b. In the executable section of the PL/SQL block assign the values 45 and 12 to `basic_percent` and `pf_percent` respectively.

```
:basic_percent:=45;
:pf_percent:=12;
```

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

```
/
PRINT basic_percent
PRINT pf_percent
```

OR

```
PRINT
```

   d. Execute and Save your script as `lab_02_05_soln.sql`. Sample output is as follows:

```
Hello World
TODAY IS : 12-JAN-04
TOMORROW IS : 13-JAN-04
PL/SQL procedure successfully completed.
```

BASIC_PERCENT
45

Next Page

Click the Next Page button.

PF_PERCENT
12

Practice 3

Note: Use *i*SQL*Plus for this practice.

```
DECLARE
 weight    NUMBER(3) := 600;
 message   VARCHAR2(255) := 'Product 10012';
BEGIN
  DECLARE
   weight   NUMBER(3) := 1;
   message   VARCHAR2(255) := 'Product 11001';
1  new_locn  VARCHAR2(50) := 'Europe';
  BEGIN
   weight := weight + 1;
   new_locn := 'Western ' || new_locn;

2  END;
 weight := weight + 1;
 message := message || ' is in stock';
 new_locn := 'Western ' || new_locn;

END;
```

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

   a. The value of `weight` at position 1 is:
   2
   The data type is `NUMBER`.

   b. The value of `new_locn` at position 1 is:
   Western Europe
   The data type is `VARCHAR2`.

   c. The value of `weight` at position 2 is:
   601
   The data type is `NUMBER`.

   d. The value of `message` at position 2 is:
   Product 10012 is in stock.
   The data type is `VARCHAR2`.

   e. The value of `new_locn` at position 2 is:
   Illegal because `new_locn` is not visible outside the subblock.

```
DECLARE
   customer        VARCHAR2(50) := 'Womansport';
   credit_rating   VARCHAR2(50) := 'EXCELLENT';
BEGIN
    DECLARE
       customer  NUMBER(7) := 201;
       name      VARCHAR2(25) := 'Unisports';
    BEGIN
       credit_rating :='GOOD';
       …
    END;
  …
END;
```

2. In the preceding PL/SQL block, determine the values and data types for each of the following cases.

   a. The value of `customer` in the nested block is:
      201
      The data type is `NUMBER`.F

   b. The value of `name` in the nested block is:
      Unisports
      The data type is `VARCHAR2`.

   c. The value of `credit_rating` in the nested block is:
      GOOD
      The data type is `VARCHAR2`.

   d. The value of `customer` in the main block is:
      Womansport
      The data type is `VARCHAR2`.

   e. The value of `name` in the main block is:
      `name` is not visible in the main block and you would see an error.

   f. The value of `credit_rating` in the main block is:
      GOOD
      The data type is `VARCHAR2`.

3. Use the same session that you used to execute the practices in Lesson 2. If you have opened a new session, then execute `lab_02_05_soln.sql`. Edit `lab_02_05_soln.sql`.

   a. Use single line comment syntax to comment the lines that create the bind variables.

```
-- VARIABLE basic_percent NUMBER
-- VARIABLE pf_percent NUMBER
```

   b. Use multiple line comments in the executable section to comment the lines that assign values to the bind variables.

```
/*  :basic_percent:=45;
:pf_percent:=12;  */
```

c. Declare two variables: `fname` of type `VARCHAR2` and size 15, and `emp_sal` of type `NUMBER` and size 10.

```
fname VARCHAR2(15);
emp_sal NUMBER(10);
```

d. Include the following SQL statement in the executable section:

```
SELECT first_name, salary INTO fname, emp_sal
FROM employees WHERE employee_id=110;
```

e. Change the line that prints "Hello World" to print "Hello" and the first name. You can comment the lines that display the dates and print the bind variables, if you want to.

```
DBMS_OUTPUT.PUT_LINE(' Hello '|| fname);
```

f. Calculate the contribution of the employee towards provident fund (PF).
PF is 12% of the basic salary, and the basic salary is 45% of the salary. Use the bind variables for the calculation. Try to use only one expression to calculate the PF. Print the employee's salary and his contribution toward PF.

```
DBMS_OUTPUT.PUT_LINE('YOUR SALARY IS : '||emp_sal);
DBMS_OUTPUT.PUT_LINE('YOUR CONTRIBUTION TOWARDS PF:
'||emp_sal*:basic_percent/100*:pf_percent/100);
```

g. Execute and save your script as `lab_03_03_soln.sql`. Sample output is as follows:

```
Hello John
YOUR SALARY IS : 8200
YOUR CONTRIBUTION TOWARDS PF: 442.8
PL/SQL procedure successfully completed.
```

4. Accept a value at run time using the substitution variable. In this practice, you will modify the script `lab_03_04.sql` to accept user input.
   a. Load the script `lab_03_04.sql` file.
   b. Include the `PROMPT` command to prompt the user with the following message: "Please enter your employee number."

```
ACCEPT empno PROMPT 'Please enter your employee number: '
```

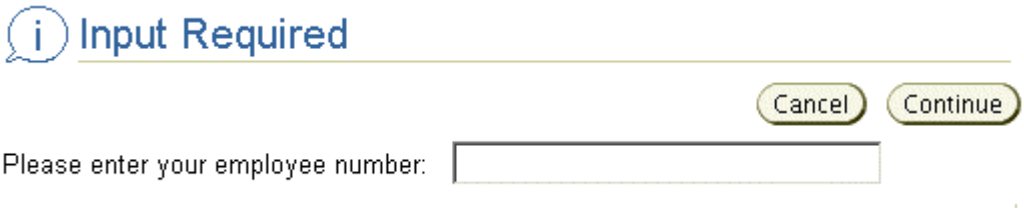
c. Modify the declaration of the `empno` variable to accept the user input.

```
empno NUMBER(6):=&empno;
```

d. Modify the select statement to include the substitution variable `empno`.

```
SELECT first_name, salary INTO fname, emp_sal
FROM employees WHERE employee_id=empno;
```

e. Execute and save this script as `lab_03_04_soln.sql`. Sample output is as follows:



Enter 100 and click the Continue button.

```
Hello Steven
YOUR SALARY IS : 24000
YOUR CONTRIBUTION TOWARDS PF: 1296
PL/SQL procedure successfully completed.
```

5. Execute the script `lab_03_05.sql`. This script creates a table called `employee_details`.
   a. The `employee` and `employee_details` tables have the same data. You will update the data in the `employee_details` table. Do not update or change the data in the `employees` table.
   b. Open the script `lab_03_05b.sql` and observe the code in the file. Note that the code accepts the employee number and the department number from the user.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
ACCEPT emp_id PROMPT 'Please enter your employee number';
ACCEPT emp_deptid PROMPT 'Please enter the department number for which
salary revision is being done';

DECLARE
   emp_authorization NUMBER(5);
   emp_id NUMBER(5):=&emp_id;
   emp_deptid NUMBER(6):=&emp_deptid;
   no_such_employee EXCEPTION;
...
```

   c. You use this as the skeleton script to develop the application, which was discussed in the lesson titled "Introduction."

Practice 4

Note: Use *i*SQL*Plus for this practice.

1. Create a PL/SQL block that selects the maximum department ID in the `departments` table and stores it in the `max_deptno` variable. Display the maximum department ID.
   a. Declare a variable `max_deptno` of type `NUMBER` in the declarative section.

```
SET SERVEROUTPUT ON
DECLARE
   max_deptno    NUMBER;
```

   b. Start the executable section with the keyword `BEGIN` and include a `SELECT` statement to retrieve the maximum `department_id` from the `departments` table.

```
BEGIN
   SELECT MAX(department_id)   INTO max_deptno   FROM    departments;
```

   c. Display `max_deptno` and end the executable block.

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

   d. Execute and save your script as `lab_04_01_soln.sql`. Sample output is as follows:

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

2. Modify the PL/SQL block you created in exercise 1 to insert a new department into the `departments` table.
   a. Load the script `lab_04_01_soln.sql`. Declare two variables:
   `dept_name` of type `departments.department_name`.
   Bind variable `dept_id` of type `NUMBER`.
   Assign 'Education' to `dept_name` in the declarative section.

```
VARIABLE dept_id NUMBER
…
dept_name departments.department_name%TYPE:= 'Education';
```

   b. You have already retrieved the current maximum department number from the `departments` table. Add 10 to it and assign the result to `dept_id`.

```
:dept_id := 10 + max_deptno;
…
```

c. Include an INSERT statement to insert data into the department_name, department_id, and location_id columns of the departments table. Use values in dept_name, dept_id for department_name, department_id and use NULL for location_id.

```
…
INSERT INTO departments (department_id, department_name, location_id)
VALUES (:dept_id,dept_name, NULL);
```

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

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

e. Execute a select statement to check if the new department is inserted. You can terminate the PL/SQL block with "/" and include the SELECT statement in your script.

```
…
/
SELECT * FROM  departments  WHERE  department_id=:dept_id;
```

f. Execute and save your script as lab_04_02_soln.sql. Sample output is as follows:

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

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education		

3. In exercise 2, you set location_id to null. Create a PL/SQL block that updates the location_id to 3000 for the new department. Use the bind variable dept_id to update the row.
   Note: Skip step a if you have not started a new *i*SQL*Plus session for this practice.
   a. If you have started a new *i*SQL*Plus session, delete the department that you have added to the departments table and execute the script lab_04_02_soln.sql.

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

b. Start the executable block with the keyword `BEGIN`. Include the `UPDATE` statement to set the `location_id` to 3000 for the new department. Use the bind variable `dept_id` in your `UPDATE` statement.

```
BEGIN
   UPDATE departments SET location_id=3000 WHERE
   department_id=:dept_id;
```

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

```
END;
/
SELECT * FROM departments WHERE department_id=:dept_id;
```

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

```
DELETE FROM departments WHERE department_id=:dept_id;
```

e. Execute and save your script as `lab_04_03_soln.sql`. Sample output is as follows:

PL/SQL procedure successfully completed.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education		3000

1 row deleted.

4. Load the script `lab_03_05b.sql` to the *i*SQL*Plus workspace.

a. Observe that the code has nested blocks. You will see the declarative section of the outer block. Look for the comment "INCLUDE EXECUTABLE SECTION OF OUTER BLOCK HERE" and start an executable section.

```
BEGIN
```

b. Include a single `SELECT` statement, which retrieves the `employee_id` of the employee working in the "Human Resources" department. Use the `INTO` clause to store the retrieved value in the variable `emp_authorization`.

```
SELECT employee_id into emp_authorization FROM
   employee_details WHERE department_id=(SELECT department_id
   FROM departments WHERE department_name='Human Resources');
```

c. Save your script as `lab_04_04_soln.sql`.

Practice 5

1. Execute the command in the file `lab_05_01.sql` to create the `messages` table. Write a PL/SQL block to insert numbers into the `messages` table.
    a. Insert the numbers 1 to 10, excluding 6 and 8.
    b. Commit before the end of the block.

```
BEGIN
FOR i in 1..10 LOOP
  IF i = 6 or i = 8 THEN
    null;
  ELSE
    INSERT INTO messages(results)
    VALUES (i);
 END IF;
END LOOP;
COMMIT;
END;
/
```

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

```
SELECT * FROM messages;
```

You should see the following output:

RESULTS
1
2
3
4
5
7
9
10

8 rows selected.

2. Execute the script `lab_05_02.sql`. This script creates an `emp` table that is a replica of the `employees` table. It alters the `emp` table to add a new column, `stars`, of `VARCHAR2` data type and size 50. Create a PL/SQL block that inserts an asterisk in the stars column for every $1000 of the employee's salary. Save your script as `lab_05_02_soln.sql`.
    a. Use the `DEFINE` command to define a variable called `empno` and initialize it to 176.

```
SET VERIFY OFF
DEFINE empno = 176
```

b. Start the declarative section of the block and pass the value of empno to the PL/SQL block through an *i*SQL*Plus substitution variable. Declare a variable `asterisk` of type `emp.stars` and initialize it to `NULL`. Create a variable `sal` of type `emp.salary`.

```
DECLARE
  empno        emp.employee_id%TYPE := TO_NUMBER(&empno);
  asterisk     emp.stars%TYPE := NULL;
  sal          emp.salary%TYPE;
```

c. In the executable section, write logic to append an asterisk (*) to the string for every $1000 of the salary. For example, if the employee earns $8000, the string of asterisks should contain eight asterisks. If the employee earns $12500, the string of asterisks should contain 13 asterisks.

```
BEGIN
  SELECT NVL(ROUND(salary/1000), 0)  INTO sal
  FROM emp  WHERE employee_id = empno;

  FOR i IN 1..sal
     LOOP
       asterisk := asterisk ||'*';
     END LOOP;
```

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

```
UPDATE emp SET stars = asterisk
      WHERE employee_id = empno;
      COMMIT;
END;
```

e. Display the row from the `emp` table to verify whether your PL/SQL block has executed successfully.

```
SELECT employee_id,salary, stars
FROM emp WHERE employee_id=&empno;
```

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

EMPLOYEE_ID	SALARY	STARS
176	8600	*********

3. Load the script `lab_04_04_soln.sql`, which you created in exercise 4 of Practice 4.

    a. Look for the comment "INCLUDE SIMPLE IF STATEMENT HERE" and include a simple `IF` statement to check if the values of `emp_id` and `emp_authorization` are the same.

```
IF (emp_id=emp_authorization) THEN
```

    b. Save your script as `lab_05_03_soln.sql`.

Practice 6

1. Write a PL/SQL block to print information about a given country.
    a. Declare a PL/SQL record based on the structure of the `countries` table.
    b. Use the `DEFINE` command to define a variable `countryid`. Assign CA to `countryid`. Pass the value to the PL/SQL block through an *i*SQL*Plus substitution variable.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DEFINE countryid = CA
```

    c. In the declarative section, use the `%ROWTYPE` attribute and declare the variable `country_record` of type `countries`.

```
DECLARE
 country_record countries%ROWTYPE;
```

    d. In the executable section, get all the information from the `countries` table by using `countryid`. Display selected information about the country. Sample output is as follows:

```
BEGIN
 SELECT    *
 INTO country_record
 FROM countries
 WHERE     country_id = UPPER('&countryid');

 DBMS_OUTPUT.PUT_LINE ('Country Id: ' || country_record.country_id ||
  ' Country Name: ' || country_record.country_name
  || ' Region: ' || country_record.region_id);

END;
```

```
Country Id: CA Country Name: Canada Region: 2
PL/SQL procedure successfully completed.
```

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

2. Create a PL/SQL block to retrieve the name of some departments from the `departments` table and print each department name on the screen, incorporating an `INDEX BY` table. Save the script as `lab_06_02_soln.sql`.

   a. Declare an `INDEX BY` table `dept_table_type` of type `departments.department_name`. Declare a variable `my_dept_table` of type `dept_table_type` to temporarily store the name of the departments.

```
SET SERVEROUTPUT ON
DECLARE
   TYPE dept_table_type is table of departments.department_name%TYPE
   INDEX BY PLS_INTEGER;
   my_dept_table   dept_table_type;
```

   b. Declare two variables: `loop_count` and `deptno` of type `NUMBER`. Assign 10 to `loop_count` and 0 to `deptno`.

```
loop_count    NUMBER (2):=10;
deptno        NUMBER (4):=0;
```

   c. Using a loop, retrieve the name of 10 departments and store the names in the `INDEX BY` table. Start with `department_id` 10. Increase `deptno` by 10 for every iteration of the loop. The following table shows the `department_id` for which you should retrieve the `department_name` and store in the `INDEX BY` table.

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

```
BEGIN

  FOR i IN 1..loop_count
  LOOP
      deptno:=deptno+10;
      SELECT department_name
      INTO my_dept_table(i)
      FROM departments
      WHERE department_id = deptno;
  END LOOP;
```

d. Using another loop, retrieve the department names from the `INDEX BY` table and display them.

```
FOR i IN 1..loop_count
  LOOP
    DBMS_OUTPUT.PUT_LINE (my_dept_table(i));
  END LOOP;
END;
```

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

```
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance
PL/SQL procedure successfully completed.
```

3. Modify the block that you created in exercise 2 to retrieve all information about each department from the `departments` table and display the information. Use an `INDEX BY` table of records.

   a. Load the script `lab_06_02_soln.sql`.
   b. You have declared the `INDEX BY` table to be of type `departments.department_name`. Modify the declaration of the `INDEX BY` table, to temporarily store the number, name, and location of all the departments. Use the `%ROWTYPE` attribute.

```
SET SERVEROUTPUT ON
DECLARE
   TYPE dept_table_type is table of departments%ROWTYPE
   INDEX BY PLS_INTEGER;
   my_dept_table dept_table_type;
   loop_count        NUMBER (2):=10;
   deptno            NUMBER (4):=0;
```

c. Modify the select statement to retrieve all department information currently in the `departments` table and store it in the `INDEX BY` table.

```
BEGIN
  FOR i IN 1..loop_count
  LOOP
      deptno := deptno + 10;
      SELECT *
      INTO my_dept_table(i)
      FROM departments
      WHERE department_id = deptno;
  END LOOP;
```

d. Using another loop, retrieve the department information from the `INDEX BY` table and display the information. Sample output is as follows:

```
FOR i IN 1..loop_count
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Department Number: ' ||
my_dept_table(i).department_id
     || ' Department Name: ' || my_dept_table(i).department_name
     || ' Manager Id: '||  my_dept_table(i).manager_id
     || ' Location Id: ' || my_dept_table(i).location_id);
  END LOOP;
END;
```

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

4. Load the script `lab_05_03_soln.sql`.

   a. Look for the comment "DECLARE AN INDEX BY TABLE OF TYPE VARCHAR2(50). CALL IT ename_table_type" and include the declaration.

```
TYPE ename_table_type IS TABLE OF
    VARCHAR2(50) INDEX BY PLS_INTEGER;
```

   b. Look for the comment "DECLARE A VARIABLE ename_table OF TYPE ename_table_type" and include the declaration.

```
ename_table    ename_table_type;
```

   c. Save your script as `lab_06_04_soln.sql`.

Practice 7

1. Create a PL/SQL block that determines the top *n* salaries of the employees.
    a. Execute the script `lab_07_01.sql` to create a new table, `top_salaries`, for storing the salaries of the employees.
    b. Accept a number *n* from the user where *n* represents the number of top *n* earners from the `employees` table. For example, to view the top five salaries, enter 5.
    Note: Use the `DEFINE` command to define a variable `p_num` to provide the value for *n*. Pass the value to the PL/SQL block through an *i*SQL*Plus substitution variable.

```
DELETE FROM top_salaries;
DEFINE p_num = 5
```

    c. In the declarative section, declare two variables: `num` of type `NUMBER` to accept the substitution variable `p_num`, `sal` of type `employees.salary`. Declare a cursor, `emp_cursor` that retrieves the salaries of employees in descending order. Remember that the salaries should not be duplicated.

```
DECLARE
  num          NUMBER(3) := &p_num;
  sal          employees.salary%TYPE;
  CURSOR         emp_cursor IS
    SELECT       distinct salary
    FROM         employees
    ORDER BY     salary DESC;
```

    d. In the executable section, open the loop and fetch top *n* salaries and insert them into `top_salaries` table. You can use a simple loop to operate on the data. Also, try and use `%ROWCOUNT` and `%FOUND` attributes for the exit condition.

```
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO  sal;
  WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
           INSERT INTO top_salaries (salary)
           VALUES (sal);
    FETCH emp_cursor INTO sal;
  END LOOP;
  CLOSE emp_cursor;
END;
```

e. After inserting into the `top_salaries` table, display the rows with a `SELECT` statement. The output shown represents the five highest salaries in the `employees` table.

```
/
SELECT * FROM top_salaries;
```

SALARY
24000
17000
14000
13500
13000

f. Test a variety of special cases, such as *n = 0* or where *n* is greater than the number of employees in the `employees` table. Empty the `top_salaries` table after each test.

2. Create a PL/SQL block that does the following:

a. Use the `DEFINE` command to define a variable `p_deptno` to provide the department ID.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
SET ECHO OFF
DEFINE p_deptno = 10
```

b. In the declarative section, declare a variable `deptno` of type `NUMBER` and assign the value of `p_deptno`.

```
DECLARE
deptno     NUMBER := &p_deptno;
```

c. Declare a cursor, `emp_cursor` that retrieves the `last_name`, `salary`, and `manager_id` of the employees working in the department specified in `deptno`.

```
CURSOR  emp_cursor IS
      SELECT     last_name, salary,manager_id
      FROM       employees
      WHERE      department_id = deptno;
```

d. In the executable section use the cursor `FOR` loop to operate on the data retrieved. If the salary of the employee is less than 5000 and if the manager ID is either 101 or 124, display the message <<*last_name*>> Due for a raise. Otherwise, display the message <<*last_name*>> Not due for a raise.

```
BEGIN
 FOR emp_record IN emp_cursor
  LOOP
   IF emp_record.salary < 5000 AND (emp_record.manager_id=101 OR
emp_record.manager_id=124) THEN
      DBMS_OUTPUT.PUT_LINE (emp_record.last_name || ' Due for a raise');
   ELSE
      DBMS_OUTPUT.PUT_LINE (emp_record.last_name || ' Not Due for a
raise');
   END IF;
  END LOOP;
END;
```

e. Test the PL/SQL block for the following cases:

Department ID	Message
10	Whalen Due for a raise
20	Hartstein Not Due for a raise Fay Not Due for a raise
50	Weiss Not Due for a raise Fripp Not Due for a raise Kaufling Not Due for a raise Vollman Not Due for a raise Mourgas Not Due for a raise . . . . . . Rajs Due for a raise
80	Russel Not Due for a raise Partners Not Due for a raise Errazuriz Not Due for a raise Cambrault Not Due for a raise . . . . . .

3. Write a PL/SQL block, which declares and uses cursors with parameters.
   In a loop, use a cursor to retrieve the department number and the department name from the `departments` table for a department whose `department_id` is less than 100. Pass the department number to another cursor as a parameter to retrieve from the `employees` table the details of employee last name, job, hire date, and salary of those employees whose `employee_id` is less than 120 and who work in that department.

a. In the declarative section declare a cursor `dept_cursor` to retrieve `department_id`, `department_name` for those departments with `department_id` less than 100. Order by `department_id`.

```
SET SERVEROUTPUT ON
DECLARE
      CURSOR dept_cursor IS
         SELECT department_id,department_name
         FROM    departments
         WHERE department_id < 100
         ORDER BY    department_id;
```

b. Declare another cursor `emp_cursor` that takes the department number as parameter and retrieves `last_name`, `job_id`, `hire_date`, and `salary` of those employees with `employee_id` of less than 120 and who work in that department.

```
CURSOR emp_cursor(v_deptno NUMBER) IS
        SELECT last_name,job_id,hire_date,salary
        FROM    employees
        WHERE   department_id = v_deptno
        AND employee_id < 120;
```

c. Declare variables to hold the values retrieved from each cursor. Use the `%TYPE` attribute while declaring variables.

```
current_deptno departments.department_id%TYPE;
current_dname  departments.department_name%TYPE;
ename          employees.last_name%TYPE;
job            employees.job_id%TYPE;
hiredate       employees.hire_date%TYPE;
sal            employees.salary%TYPE;
```

d. Open the `dept_cursor`, use a simple loop and fetch values into the variables declared. Display the department number and department name.

```
BEGIN
OPEN dept_cursor;
       LOOP
            FETCH dept_cursor INTO current_deptno,current_dname;
            EXIT WHEN dept_cursor%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE ('Department Number : ' ||
current_deptno ||  '  Department Name : ' || current_dname);
```

e. For each department, open the `emp_cursor` by passing the current department number as a parameter. Start another loop and fetch the values of `emp_cursor` into variables and print all the details retrieved from the `employees` table.
Note: You may want to print a line after you have displayed the details of each department. Use appropriate attributes for the exit condition. Also check if a cursor is already open before opening the cursor.

```
IF emp_cursor%ISOPEN THEN
      CLOSE emp_cursor;
END IF;
  OPEN emp_cursor (current_deptno);
  LOOP
    FETCH emp_cursor INTO  ename,job,hiredate,sal;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (ename || '    ' ||  job || '   '  ||  hiredate
|| '   ' ||  sal);
  END LOOP;
 DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------
-------------------------------------');
  CLOSE emp_cursor;
```

f. Close all the loops and cursors, and end the executable section. Execute the script.

```
END LOOP;
     CLOSE dept_cursor;
END;
```

The sample output is as follows:

```
Department Number : 10 Department Name : Administration
-------------------------------------------------------------------------------
Department Number : 20 Department Name : Marketing
-------------------------------------------------------------------------------
Department Number : 30 Department Name : Purchasing
Raphaely PU_MAN 07-DEC-94 11000
Khoo PU_CLERK 18-MAY-95 3100
Baida PU_CLERK 24-DEC-97 2900
Tobias PU_CLERK 24-JUL-97 2800
Himuro PU_CLERK 15-NOV-98 2600
Colmenares PU_CLERK 10-AUG-99 2500
-------------------------------------------------------------------------------
Department Number : 40 Department Name : Human Resources
-------------------------------------------------------------------------------
Department Number : 50 Department Name : Shipping
-------------------------------------------------------------------------------
Department Number : 60 Department Name : IT
Hunold IT_PROG 03-JAN-90 9000
Ernst IT_PROG 21-MAY-91 6000
Austin IT_PROG 25-JUN-97 4800
Pataballa IT_PROG 05-FEB-98 4800
Lorentz IT_PROG 07-FEB-99 4200
-------------------------------------------------------------------------------
Department Number : 70 Department Name : Public Relations
-------------------------------------------------------------------------------
Department Number : 80 Department Name : Sales
-------------------------------------------------------------------------------
Department Number : 90 Department Name : Executive
King AD_PRES 17-JUN-87 24000
Kochhar AD_VP 21-SEP-89 17000
De Haan AD_VP 13-JAN-93 17000
-------------------------------------------------------------------------------
PL/SQL procedure successfully completed.
```

4. Load the script `lab_06_04_soln.sql`.

   a. Look for the comment "DECLARE A CURSOR CALLED emp_records TO HOLD salary, first_name, and last_name of employees" and include the declaration. Create the cursor such that it retrieves the `salary`, `first_name`, and `last_name` of employees in the department specified by the user (substitution variable `emp_deptid`). Use the `FOR UPDATE` clause.

```
CURSOR emp_records IS SELECT salary,first_name,last_name
FROM employee_details WHERE department_id=emp_deptid
FOR UPDATE;
```

   b. Look for the comment "INCLUDE EXECUTABLE SECTION OF INNER BLOCK HERE" and start the executable block.

```
BEGIN
```

   c. Only employees working in the departments with `department_id` 20, 60, 80,100, and 110 are eligible for raises this quarter. Check if the user has entered any of these department IDs. If the value does not match, display the message "SORRY, NO SALARY REVISIONS FOR EMPLOYEES IN THIS DEPARTMENT." If the value matches, open the cursor `emp_records`.

```
IF (emp_deptid NOT IN (20,60,80,100,110)) THEN
DBMS_OUTPUT.PUT_LINE ('SORRY, NO SALARY REVISIONS FOR
EMPLOYEES IN THIS DEPARTMENT');
 ELSE
   OPEN emp_records;
```

   d. Start a simple loop and fetch the values into `emp_sal`, `emp_fname`, and `emp_lname`. Use `%NOTFOUND` for the exit condition.

```
LOOP
  FETCH emp_records INTO emp_sal,emp_fname,emp_lname;
  EXIT WHEN emp_records%NOTFOUND;
```

   e. Include a `CASE` expression. Use the following table as reference for the conditions in the `WHEN` clause of the `CASE` expression.
   Note: In your `CASE` expressions use the constants such as `c_range1`, `c_hike1` that are already declared.

salary	**Hike percentage**
< 6500	20
> 6500 < 9500	15
> 9500 <12000	8
>12000	3

For example, if the salary of the employee is less than 6500, then increase the salary by 20 percent. In every WHEN clause, concatenate the first_name and last_name of the employee and store it in the INDEX BY table. Increment the value in variable i so that you can store the string in the next location. Include an UPDATE statement with the WHERE CURRENT OF clause.

```
CASE
     WHEN   emp_sal<c_range1 THEN
     ename_table(i):=emp_fname||' '||emp_lname;
     i:=i+1;
     UPDATE employee_details SET salary=emp_sal +  (emp_sal*c_hike1)
     WHERE CURRENT OF emp_records;
     WHEN   emp_sal<c_range2 THEN
      ename_table(i):=emp_fname||' '||emp_lname;
      i:=i+1;
      UPDATE employee_details SET salary=emp_sal+(emp_sal*c_hike2)
      WHERE CURRENT OF emp_records;
     WHEN (emp_sal<c_range3) THEN
      ename_table(i):=emp_fname||' '||emp_lname;
      i:=i+1;
      UPDATE employee_details SET salary=emp_sal+(emp_sal*c_hike3)
      WHERE CURRENT OF emp_records;
     ELSE
     ename_table(i):=emp_fname||' '||emp_lname;
     i:=i+1;
      UPDATE employee_details SET salary=emp_sal+(emp_sal*c_hike4)
      WHERE CURRENT OF emp_records;
     END CASE;
```

f. Close the loop. Use the %ROWCOUNT attribute and print the number of records that were modified. Close the cursor.

```
END LOOP;

DBMS_OUTPUT.PUT_LINE ('NUMBER OF RECORDS MODIFIED :
'||emp_records%ROWCOUNT);
CLOSE emp_records;
```

g. Include a simple loop to print the names of all the employees whose salaries were revised.
Note: You already have the names of these employees in the INDEX BY table. Look for the comment "CLOSE THE INNER BLOCK" and include an END IF statement and an END statement.

```
DBMS_OUTPUT.PUT_LINE ('The following employees'' salaries are updated');
 FOR i IN ename_table.FIRST..ename_table.LAST
  LOOP
     DBMS_OUTPUT.PUT_LINE(ename_table(i));
  END LOOP;
END IF;
END;
```

h. Save your script as `lab_07_04_soln.sql`.

Practice 8

1. The purpose of this example is to show the usage of predefined exceptions. Write a PL/SQL block to select the name of the employee with a given salary value.

    a. Delete all the records in the `messages` table. Use the `DEFINE` command to define a variable `sal` and initialize it to 6000.

```
DELETE FROM MESSAGES;
SET VERIFY OFF
DEFINE sal = 6000
```

    b. In the declarative section declare two variables: `ename` of type `employees.last_name` and `emp_sal` of type `employees.salary`. Pass the value of the substitution variables to `emp_sal`.

```
DECLARE
  ename    employees.last_name%TYPE;
  emp_sal  employees.salary%TYPE := &sal;
```

    c. In the executable section retrieve the last names of employees whose salaries are equal to the value in `emp_sal`.
    Note: Do not use explicit cursors.
    If the salary entered returns only one row, insert into the `messages` table the employee's name and the salary amount.

```
BEGIN
  SELECT   last_name
  INTO     ename
  FROM     employees
  WHERE    salary = emp_sal;
  INSERT INTO messages (results)
  VALUES (ename || ' - ' || emp_sal);
```

    d. If the salary entered does not return any rows, handle the exception with an appropriate exception handler and insert into the `messages` table the message "No employee with a salary of <*salary*>."

```
EXCEPTION
  WHEN no_data_found THEN
    INSERT INTO messages (results)
    VALUES ('No employee with a salary of '|| TO_CHAR(emp_sal));
```

    e. If the salary entered returns more than one row, handle the exception with an appropriate exception handler and insert into the `messages` table the message "More than one employee with a salary of <*salary*>."

```
WHEN too_many_rows THEN
    INSERT INTO messages (results)
```

```
    VALUES ('More than one employee with a salary of '||
             TO_CHAR(emp_sal));
```

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

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

g. Display the rows from the `messages` table to check whether the PL/SQL block has executed successfully. Sample output is as follows:

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

RESULTS
More than one employee with a salary of 6000

2. The purpose of this example is to show how to declare exceptions with a standard Oracle Server error. Use the Oracle server error ORA-02292 (integrity constraint violated – child record found).

   a. In the declarative section declare an exception `childrecord_exists`. Associate the declared exception with the standard Oracle server error –02292.

```
SET SERVEROUTPUT ON
DECLARE
   childrecord_exists EXCEPTION;
   PRAGMA EXCEPTION_INIT(childrecord_exists, -02292);
```

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

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

   c. Include an exception section to handle the `childrecord_exists` exception and display the appropriate message. Sample output is as follows:

```
EXCEPTION
   WHEN childrecord_exists THEN
   DBMS_OUTPUT.PUT_LINE(' Cannot delete this department. There are
employees in this department (child records exist.)');
END;
```

```
Deleting department 40.....
Cannot delete this department. There are employees in this department
(child records exist.)
PL/SQL procedure successfully completed.
```

3. Load the script `lab_07_04_soln.sql`.
    a. Observe the declarative section of the outer block. Note that the `no_such_employee` exception is declared.
    b. Look for the comment "RAISE EXCEPTION HERE." If the value of `emp_id` is not between 100 and 206, then raise the `no_such_employee` exception.

```
  IF (emp_id NOT BETWEEN 100 AND 206) THEN
   RAISE no_such_employee;
  END IF;
```

    c. Look for the comment "INCLUDE EXCEPTION SECTION FOR OUTER BLOCK" and handle the exceptions `no_such_employee` and `too_many_rows`. Display appropriate messages when the exceptions occur. The `employees` table has only one employee working in the `HR` department and therefore the code is written accordingly. The `too_many_rows` exception is handled to indicate that the select statement retrieves more than one employee working in the `HR` department.

```
EXCEPTION
  WHEN no_such_employee THEN
  DBMS_OUTPUT.PUT_LINE ('NO EMPLOYEE EXISTS WITH THE
  GIVEN EMPLOYEE NUMBER: PLEASE CHECK');

  WHEN TOO_MANY_ROWS THEN
  DBMS_OUTPUT.PUT_LINE  (' THERE IS MORE THAN ONE
  EMPLOYEE IN THE HR DEPARTMENT. ');
```

    d. Close the outer block.

```
END;
```

    e. Save your script as `lab_08_03_soln.sql`.
    f. Execute the script. Enter the employee number and the department number and observe the output. Enter different values and check for different conditions. The sample output for employee ID 203 and department ID 100 is as follows:

```
NUMBER OF RECORDS MODIFIED : 6
The following employees' salaries are updated
Nancy Greenberg
Daniel Faviet
John Chen
Ismael Sciarra
Jose Manuel Urman
Luis Popp
PL/SQL procedure successfully completed.
```

Practice 9

1. In *i*SQL*Plus, load the script `lab_02_04_soln.sql` that you created for exercise 4 of practice 2.
    a. Modify the script to convert the anonymous block to a procedure called `greet`.

```
CREATE PROCEDURE greet IS
   today DATE:=SYSDATE;
   tomorrow today%TYPE;
...
```

    b. Execute the script to create the procedure.
    c. Save this script as `lab_09_01_soln.sql`.
    d. Click the Clear button to clear the workspace.
    e. Create and execute an anonymous block to invoke the procedure `greet`. Sample output is as follows:

```
BEGIN
 greet;
END;
```

```
Hello World
TODAY IS : 20-JAN-04
TOMORROW IS : 21-JAN-04
PL/SQL procedure successfully completed.
```

2. Load the script `lab_09_01_soln.sql`.
    a. Drop the procedure `greet` by issuing the following command:

```
DROP PROCEDURE greet
```

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

```
CREATE PROCEDURE greet(name VARCHAR2) IS
   today DATE:=SYSDATE;
   tomorrow today%TYPE;
```

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

```
BEGIN
   tomorrow:=today +1;
   DBMS_OUTPUT.PUT_LINE(' Hello '|| name);
```

    d. Save your script as `lab_09_02_soln.sql`.
    e. Execute the script to create the procedure.

f. Create and execute an anonymous block to invoke the procedure `greet` with a parameter. Sample output is as follows:

```
BEGIN
 greet('Neema');
END;
```

```
Hello Neema
TODAY IS : 20-JAN-04
TOMORROW IS : 21-JAN-04
PL/SQL procedure successfully completed.
```