PLSQL Assignments

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

- a. DECLARE

  v_id NUMBER(4);
  It is legal since id length can be of size of or less than 4.

- b. DECLARE

  v_x, v_y, v_z VARCHAR2(10);
  Multiple variable declaration is not allowed in pl sql.

- c. DECLARE

  v_birthdate DATE NOT NULL;
  incorrect, because must assign a value

- d. DECLARE

  V_in_stock BOOLEAN := 1;
  incorrect, boolean can be declared as true,false or null

2. In each of the following assignments, indicate whether the statement is valid and what the valid data type of the result will be.

a. v_days_to_go := v_due_date - SYSDATE;

It is correct only because this is going to return the numeric value of no of days to go.

b. v_sender := USER || ': ' || TO_CHAR(v_dept_no);

Incorrect, because sender can be a non-string value

c. v_sum := $100,000 + $250,000;

Incorrect, because $ is not allowed. Any currency data type not found for pl/sql statement.

d. v_flag := TRUE;

Correct only

e. v_n1 := v_n2 > (2 * v_n3);

Correct, it will return True/False as here we are comparing the values as Boolean.

f. v_value := NULL;

Unable to assign any NULL value for the given variable.

3. Create an anonymous block to output the phrase “My PL/SQL Block Works” to the screen.

```
DECLARE
V_MESSAGE VARCHAR(25):='My PL/SQL Block Works';
BEGIN
dbms_output.put_line(V_MESSAGE);
END;
/
```

4. Create a block that declares two variables. Assign the value of these PL/SQL variables to iSQL*Plus host variables and print the results of the PL/SQL variables to the screen. Execute your PL/SQL block. Save your PL/SQL block in a file named p1q4.sql, by clicking the Save Script button. Remember to save the script with a .sql extension.

V_CHAR Character (variable length) V_NUM Number Assign values to these variables as follows:

Variable Value -------- ----------

V_CHAR The literal '42 is the answer'

V_NUM The first two characters from V_CHAR.

```
DECLARE
V_CHAR CHAR(20):='42 is the answer';
V_NUM NUMBER(5):=SUBSTR(V_CHAR,1,2);
BEGIN
dbms_output.put_line(V_CHAR);
dbms_output.put_line(V_NUM);
END;
/
```

5. PL/SQL Block

```
DECLARE
v_weight NUMBER(3) := 600;
v_message VARCHAR2(255) := 'Product 10012';
BEGIN
DECLARE
v_weight NUMBER(3) := 1;
v_message VARCHAR2(255) := 'Product 11001';
v_new_locn VARCHAR2(50) := 'Europe';
BEGIN
v_weight := v_weight + 1;
v_new_locn := 'Western ' || v_new_locn;
END;
v_weight := v_weight + 1;
v_message := v_message || ' is in stock';
v_new_locn := 'Western ' || v_new_locn;
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.

a. The value of V_WEIGHT at position 1 is: **1**

b. The value of V_NEW_LOCN at position 1 is: **Western Europe**

c. The value of V_WEIGHT at position 2 is: **601**

d. The value of V_MESSAGE at position 2 is: **Product 10012 is in stock**

e. The value of V_NEW_LOCN at position 2 is: **Western Hello**

SQL Worksheet

```
v_weight NUMBER(3) := 600;
v_message VARCHAR2(255) := 'Product 10012';
v_new_locn VARCHAR(300):='Hello';
BEGIN
DECLARE
v_weight NUMBER(3) := 1;
v_message VARCHAR2(255) := 'Product 11001';
v_new_locn VARCHAR2(50) := 'Europe';

BEGIN
v_weight := v_weight + 1;
v_new_locn := 'Western ' || v_new_locn;
dbms_output.put_line(v_weight);
dbms_output.put_line(v_message);
dbms_output.put_line(v_new_locn);
--1
END;
v_weight := v_weight + 1;
v_message := v_message || ' is in stock';
v_new_locn := 'Western ' || v_new_locn;
dbms_output.put_line(v_weight);
dbms_output.put_line(v_message);
dbms_output.put_line(v_new_locn);
--2
END;
/
```

```
Statement processed.
2
Product 11001
Western Europe
601
Product 10012 is in stock
Western Hello
```

6. DECLARE

v_customer VARCHAR2(50) := 'Womansport';

v_credit_rating VARCHAR2(50) := 'EXCELLENT';

BEGIN

DECLARE

v_customer NUMBER(7) := 201;

v_name VARCHAR2(25) := 'Unisports';

BEGIN

v_customer v_name v_credit_rating

END;

v_customer v_name v_credit_rating

END;

/

Suppose you embed a subblock within a block, as shown above. You declare two variables, V_CUSTOMER and V_CREDIT_RATING, in the main block. You also declare two variables, V_CUSTOMER and V_NAME, in the subblock. Determine the values and data types for each of the following cases.

a. The value of V_CUSTOMER in the subblock is:

b. The value of V_NAME in the subblock is:

c. The value of V_CREDIT_RATING in the subblock is:

d. The value of V_CUSTOMER in the main block is:

e. The value of V_NAME in the main block is:

f. The value of V_CREDIT_RATING in the main block is:

SQL Worksheet

```
DECLARE
v_customer VARCHAR2(50) := 'Womansport';
v_credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
DECLARE
v_customer NUMBER(7) := 201;
v_name VARCHAR2(25) := 'Unisports';
BEGIN
--v_customer v_name v_credit_rating   SUBBLOCK
dbms_output.put_line(v_customer);
dbms_output.put_line(v_name);
dbms_output.put_line(v_credit_rating);
dbms_output.put_line('--------');
END;
--v_customer v_name v_credit_rating MAINBLOCK
dbms_output.put_line(v_customer);
--dbms_output.put_line(v_name); V_NAME NOT ACCESSIBLE OUTSIDE SUB BLOCK
dbms_output.put_line(v_credit_rating);
END;
/
--CONCLUSION
--OUTER BLOCK CANNOT GET THE VALUES FROM INNER BLOCK
--INNER BLOCK CAN GET THE VALUES FROM THE OUTER BLOCK
```

```
Statement processed.
201
Unisports
EXCELLENT
--------
Womansport
EXCELLENT
```

7. Create and execute a PL/SQL block that accepts two numbers through iSQL*Plus substitution variables.

a. Use the DEFINE command to provide the two values.

DEFINE p_num1 = 2

DEFINE p_num2 = 4

b. Pass the two values defined in step a above, to the PL/SQL block through iSQL*Plus substitution variables. The first number should be divided by the second number and have the second number added to the result. The result should be stored in a PL/SQL variable and printed on the screen.

Note: SET VERIFY OFF in the PL/SQL block.

SQL Worksheet

```
DECLARE
v_tot_result number;
v_result number;
p_num1 number:=2;
p_num2 number:=4;
BEGIN
V_result:=p_num1/p_num2;
V_tot_result:=V_result+p_num2;
DBMS_OUTPUT.PUT_LINE(V_RESULT);
DBMS_OUTPUT.PUT_LINE(V_TOT_RESULT);
END;
```

```
Statement processed.
.5
4.5
```

8. Build a PL/SQL block that computes the total compensation for one year.

a. The annual salary and the annual bonus percentage values are defined using the DEFINE command.

b. Pass the values defined in the above step to the PL/SQL block through iSQL*Plus substitution variables. The bonus must be converted from a whole number to a decimal (for example, 15 to .15). If the salary is null, set it to zero before computing the total compensation. Execute the PL/SQL block. Reminder: Use the NVL function to handle null values.

Note: Total compensation is the sum of the annual salary and the annual bonus.

To test the NVL function, set the DEFINE variable equal to NULL.

DEFINE p_salary = 50000

DEFINE p_bonus = 1

SQL Worksheet

```
DECLARE
v_g_total NUMBER;
p_salary NUMBER:=50000;
p_bonus NUMBER:=10;
begin
 v_g_total:=nvl(p_salary,0)+nvl(p_salary,0)*(p_bonus/100);
 DBMS_OUTPUT.PUT_LINE('G_TOTAL: ' || V_G_TOTAL);
END;
/
```

```
Statement processed.
G_TOTAL: 55000
```

9. Create a PL/SQL block that selects the maximum department number in the DEPARTMENTS table and stores it in an iSQL*Plus variable. Print the results to the screen. Save your PL/SQL block in a file named p3q1.sql. by clicking the Save Script button. Save the script with a .sql extension.

CODE:

```
select * from dept;

Declare

 v_max_deptno number;

 max_deptno number;

begin

 select max(deptno) into v_max_deptno from dept;

 max_deptno:=v_max_deptno;

 dbms_output.put_line('The maximum department number in the DEPARTMENTS table:
' || max_deptno);

End;
```

SQL Worksheet

```
select * from dept;
declare
 v_max_deptno number;
 max_deptno number;
begin
 select max(deptno) into v_max_deptno from dept;
 max_deptno:=v_max_deptno;
 dbms_output.put_line('The maximum department number in the DEPARTMENTS table: ' || max_deptno);
end;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Download CSV

4 rows selected.

Statement processed.

The maximum department number in the DEPARTMENTS table: 40

10. Modify the PL/SQL block you created in exercise 1 to insert a new department into the DEPARTMENTS table. Save the PL/SQL block in a file named p3q2.sql by clicking the Save Script button. Save the script with a .sql extension.

a. Use the DEFINE command to provide the department name. Name the new department Education.

SQL Worksheet

```sql


DECLARE
 v_deptno number:=50;
 v_dname varchar2(30):='EDUCATION';
BEGIN
INSERT INTO dept(deptno,dname) VALUES(v_deptno,v_dname);
END;
/
SELECT * FROM DEPT;
```

Statement processed.

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	EDUCATION	-

Download CSV

5 rows selected.

b. Pass the value defined for the department name to the PL/SQL block through a iSQL*Plus substitution variable. Rather than printing the department number retrieved from exercise 1, add 10 to it and use it as the department number for the new department.

c. Leave the location number as null for now.

d. Execute the PL/SQL block.

e. Display the new department that you created.

11. Create a PL/SQL block that updates the location ID for the new department that you added in the previous practice. Save your PL/SQL block in a file named p3q3.sql by clicking the Save Script button. Save the script with a .sql extension.

a. Use an iSQL*Plus variable for the department ID number that you added in the previous practice.

b. Use the DEFINE command to provide the location ID. Name the new location ID 1700.

DEFINE p_deptno = 280

DEFINE p_loc = 1700

c. Pass the value to the PL/SQL block through a iSQL*Plus substitution variable. Test the PL/SQL block.

d. Display the department that you updated.

```
DECLARE
   department_number NUMBER;
   v_loc VARCHAR2(20) := 'Chennai;
BEGIN
   UPDATE dept
   SET loc = v_loc
   WHERE deptno = (50);
END;
/
SELECT * FROM dept;
```

12. Create a PL/SQL block that deletes the department that you created in exercise 2. Save the PL/SQL block in a file named p3q4.sql. by clicking the Save Script button. Save the script with a .sql extension.

– Use the DEFINE command to provide the department ID.

DEFINE p_deptno=280

b. Pass the value to the PL/SQL block through a iSQL*Plus substitution variable. Print to the screen the number of rows affected.

c. Test the PL/SQL block.

d. Confirm that the department has been deleted.

```
DECLARE

   department_number NUMBER := 50;

BEGIN

   DELETE FROM dept

   WHERE deptno = 50;

END;

/

SELECT * FROM dept;
```

13. 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.

c. Select from the MESSAGES table to verify that your PL/SQL block worked.

SQL Worksheet

```
create table message(result varchar(100));
DECLARE
v_counter NUMBER;
BEGIN
            FOR v_counter IN 1..10 LOOP
                IF v_counter NOT IN (6, 8)
                     THEN
                        INSERT INTO message (result) VALUES (v_counter);
                 END IF;
             END LOOP;
END;
/
select * from message;
```

```
Table created.

Statement processed.
```

RESULT
1
2
3
4
5
7
9
10

14. Create a PL/SQL block that computes the commission amount for a given employee based on the employee's salary.

a. Use the DEFINE command to provide the employee ID. Pass the value to the PL/SQL block through a iSQL*Plus substitution variable.

DEFINE p_empno = 100

b. If the employee's salary is less than $5,000, display the bonus amount for the employee as 10% of the salary.

c. If the employee's salary is between $5,000 and $10,000, display the bonus amount for the employee as 15% of the salary.

d. If the employee's salary exceeds $10,000, display the bonus amount for the employee as 20% of the salary.

e. If the employee's salary is NULL, display the bonus amount for the employee as 0.

f. Test the PL/SQL block for each case using the following test cases, and check each bonus amount.

Note: Include SET VERIFY OFF in your solution.

1.
```
DECLARE
   v_empid NUMBER;
   v_total NUMBER;
BEGIN
   v_empid := &empID;
   SELECT sal
   FROM emp
   WHERE empno = v_empid;
   IF sal<5000 THEN
      v_total = 1.1 * sal;
   ELSIF sal>5000 AND sal<10000 THEN
      v_total = 1.15 * sal;
   ELSIF sal>10000 THEN
      v_total = 1.2 * sal;
   ELSE
      v_total = 0;
   END IF;
   dbms_output.put_line(v_total);
END;
```

15. Create an EMP table that is a replica of the EMPLOYEES table. You can do this by executing the script lab04_3.sql. Add a new column, STARS, of VARCHAR2 data type and length of 50 to the EMP table for storing asterisk (*).

CREATE TABLE employee

AS (SELECT * FROM EMP);

ALTER TABLE employee

ADD COLUMN STARS VARCHAR2(50);

16. Create a PL/SQL block that rewards an employee by appending an asterisk in the STARS column for every $1000 of the employee's salary. Save your PL/SQL block in a file called p4q4.sql by clicking on the Save Script button. Remember to save the script with a .sql extension.

– Use the DEFINE command to provide the employee ID. Pass the value to the PL/SQL block through a iSQL*Plus substitution variable.

DEFINE p_empno=104

b. Initialize a v_asterisk variable that contains a NULL.

c. Append an asterisk to the string for every $1000 of the salary amount. For example, if the employee has a salary amount of $8000, the string of asterisks should contain eight asterisks. If the employee has a salary amount of $12500, the string of asterisks should contain 13 asterisks.

d. Update the STARS column for the employee with the string of asterisks.

e. Commit.

f. Test the block for the following values: DEFINE p_empno=174 DEFINE p_empno=176

g. Display the rows from the EMP table to verify whether your PL/SQL block has executed successfully.

Note: SET VERIFY OFF in the PL/SQL block

```
DECLARE
  v_sal emp.sal%TYPE;
```

```
    v_empid varchar2(50);
    v_stars varchar2(50);
    v_count NUMBER;
BEGIN
    v_empid := &empID;
    SELECT sal
    INTO v_sal
    FROM emp
    WHERE empno = v_empid;
    FOR v_count IN 0..(v_sal/1000)
        v_stars := v_stars + '*';
    END FOR;
    dbms_output.put_line(v_stars);
END;
```

17. Run the command in the script lab06_1.sql to create a new table for storing the salaries of the employees.

CREATE TABLE top_dogs

(salary NUMBER(8,2));

SQL Worksheet

```
CREATE TABLE SALARY AS SELECT SAL FROM EMP;
Declare
   CURSOR EMP_SAL is
   --CREATE TABLE SALARY AS SELECT SAL FROM EMP;
   SELECT SAL FROM SALARY;
   E_SAL EMP_SAL%rowtype;
BEGIN
   OPEN EMP_SAL;
   LOOP
     FETCH EMP_SAL into E_SAL;
     EXIT WHEN EMP_SAL%notfound;
      DBMS_OUTPUT.put_line('SALARY: '|| E_SAL.SAL);
   END LOOP;
   EXCEPTION
      WHEN NO_DATA_FOUND
      THEN
      DBMS_OUTPUT.put_line ('No such customer exists with this custid-->');
      CLOSE EMP_SAL;
END;
/
```

```
Statement processed.
SALARY: 5000
SALARY: 2850
SALARY: 2450
SALARY: 2975
SALARY: 1250
SALARY: 1600
SALARY: 1500
SALARY: 950
SALARY: 1250
```

18. Create a PL/SQL block that determines the top employees with respect to salaries.

a. 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 earners, enter 5.

Note: Use the DEFINE command to provide the value for n. Pass the value to the PL/SQL block through a iSQL*Plus substitution variable.

b. In a loop use the iSQL*Plus substitution parameter created in step 1 and gather the salaries of the top n people from the EMPLOYEES table. There should be no duplication in the salaries. If two employees earn the same salary, the salary should be picked up only once.

c. Store the salaries in the TOP_DOGS table.

d. 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_DOGS table after each test. The output shown represents the five highest salaries in the EMPLOYEES table.

```
declare
 cursor emp_sal is select distinct sal from emp order by sal desc;

r emp.sal%type;
begin
 open emp_sal;
loop
 fetch emp_sal into r;
dbms_output.put_line('THE TOP SALARY WISE: '||r);
insert into top_dogs(salary) values(r);
exit when emp_sal%rowcount>4;
end loop;
close emp_sal;
end;
/
show errors;
```

```
Statement processed.
THE TOP SALARY WISE: 9999
THE TOP SALARY WISE: 3000
THE TOP SALARY WISE: 2975
THE TOP SALARY WISE: 2850
THE TOP SALARY WISE: 2450
```

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

a. Use the DEFINE command to provide the department ID. Pass the value to the PL/SQL block through a iSQL*Plus substitution variable.

b. In a PL/SQL block, retrieve the last name, salary, and MANAGER ID of the employees working in that department.

c. 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.

Note: SET ECHO OFF to avoid displaying the PL/SQL code every time you execute the script.

CODE:

DECLARE

```
myvar NUMBER(3):=10;

CURSOR C1 IS

select MGR,ENAME,SAL from Emp where DEPTNO=10;

empSalary emp.sal%type;

empMgr EMP.MGR%type;

emplname EMP.ENAME%type;

BEGIN

OPEN C1;

LOOP

FETCH C1 INTO empMgr,emplname,empSalary;

EXIT WHEN C1%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(empMgr||' '||emplname||' '||empSalary);

END LOOP;

CLOSE C1;

END;

DECLARE

CURSOR C1 IS SELECT MGR,ENAME,SAL FROM EMP;

empSalary emp.sal%type;

empMgr EMP.MGR%type;

emplname EMP.ENAME%type;

BEGIN

OPEN C1;

LOOP

FETCH C1 INTO empMgr,emplname,empSalary;

IF EMPSALARY<5000 THEN
```

```
IF EMPMGR in (101,124)

THEN

DBMS_OUTPUT.PUT_LINE(emplname||' Due for a raise');

ELSE

DBMS_OUTPUT.PUT_LINE(emplname||' Not Due for a raise');

END IF;

ELSE

DBMS_OUTPUT.PUT_LINE(emplname||' Not Due for a raise');

END IF;

EXIT WHEN C1%NOTFOUND;

END LOOP;

CLOSE C1;

END;

/
```

20. Write a PL/SQL block to select the name of the employee with a given salary value.

A. Use the DEFINE command to provide the salary.

B.Pass the value to the PL/SQL block through a iSQL*Plus substitution variable. 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>.”

c. 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>.”

d. If the salary entered returns only one row, insert into the MESSAGES table the employee’s name and the salary amount.

e. Handle any other exception with an appropriate exception handler and insert into the MESSAGES table the message “Some other error occurred.”

f. Test the block for a variety of test cases. Display the rows from the MESSAGES table to check whether the PL/SQL block has executed successfully. Some sample output is shown below.

CODE:B

```
DECLARE

inpvar NUMBER(8):=1500;

empname EMP.ENAME%type;

BEGIN

select ename INTO empname FROM Emp where sal=1500;

DBMS_OUTPUT.PUT_LINE('Employee with Salary '||empname||' '||inpvar);

EXCEPTION

WHEN TOO_MANY_ROWS THEN

DBMS_OUTPUT.PUT_LINE (' Your select statement retrieved multiple rows. Consider using a cursor.');

END;

/
```

D:

```
DECLARE

inpvar NUMBER(8):=1500;

empname EMP.ENAME%type;

BEGIN

select ENAME INTO empname FROM Emp where sal=inpvar;

DBMS_OUTPUT.PUT_LINE('Employee with Salary '||empname||'  '||inpvar);

EXCEPTION

WHEN NO_DATA_FOUND THEN
```

```
DBMS_OUTPUT.PUT_LINE ('No employee with a salary of '||inpvar);

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE ('Some other exception occurred');

END;
```

21. Modify the code in p3q3.sql to add an exception handler.

– Use the DEFINE command to provide the department ID and department location. Pass the values to the PL/SQL block through a iSQL*Plus substitution variables.

b. Write an exception handler for the error to pass a message to the user that the specified department does not exist. Use a bind variable to pass the message to the user.

c. Execute the PL/SQL block by entering a department that does not exist.

```
DECLARE

inpvar1 NUMBER(8):=50;

empname EMP.ENAME%type;

BEGIN

select ENAME INTO empname FROM Emp where deptno=inpvar1;

DBMS_OUTPUT.PUT_LINE('Employee with Salary '||empname||inpvar1);

EXCEPTION

WHEN NO_DATA_FOUND THEN

DBMS_OUTPUT.PUT_LINE ('DOES NOT EXIST DEPT');
```

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE ('Some other exception occurred');

END;

/

22. Write a PL/SQL block that prints the number of employees who earn plus or minus $100 of the salary value set for an iSQL*Plus substitution variable. Use the DEFINE command

to provide the salary value. Pass the value to the PL/SQL block through a iSQL*Plus

substitution variable.

a. If there is no employee within that salary range, print a message to the user indicating that is the case. Use an exception for this case.

b. If there are one or more employees within that range, the message should indicate how many employees are in that salary range.

select * from emp;

DECLARE

sal NUMBER(4):=1200;

EmpNo NUMBER(5);

BEGIN

SELECT count(*) INTO EmpNo FROM EMP where SAL =(1200-100) or SAL=(1200+100);

DBMS_OUTPUT.PUT_LINE(EmpNo);

EN**D;**

c. Handle any other exception with an appropriate exception handler. The message should indicate that some other error occurred.

DEFINE p_sal = 7000

DEFINE p_sal = 2500

DEFINE p_sal = 6500

declare

```
NO_DATA_FOUNDING EXCEPTION;

sal NUMBER(5):=1200;

Empno NUMBER(3);

begin

SELECT count(*) INTO Empno FROM EMP where SAL BETWEEN (1200-100) and (1200+100);

if EMPNO =0 then

RAISE NO_DATA_FOUNDING;

ELSE

DBMS_OUTPUT.PUT_LINE(EmpNo||' Employees are in the salary range');

END IF;

EXCEPTION

WHEN NO_DATA_FOUNDING THEN

DBMS_OUTPUT.PUT_LINE('There is no employee in that salary range');

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('SOME OTHER ERRORS');

END;
```

23. Create and invoke the ADD_JOB procedure and consider the results.

a. Create a procedure called ADD_JOB to insert a new job into the JOBS table. Provide the ID and title of the job, using two parameters.

b. Compile the code, and invoke the procedure with IT_DBA as job ID and Database Administrator as job title. Query the JOBS table to view the results.

c. Invoke your procedure again, passing a job ID of ST_MAN and a job title of Stock Manager. What happens and why?

```
CREATE OR REPLACE PROCEDURE ADD_JOB(
   jobid IN VARCHAR(20),
   jobtitle IN VARCHAR(20))
BEGIN
   INSERT INTO jobs VALUES(jobid,jobtitle);
END;
/
DECLARE
   jobid VARCHAR(20) := 'IT_DBA';
   jobtitle VARCHAR(20) := 'Database Administrator';
BEGIN
   ADD_JOB(jobid,jobtitle);
   SELECT * FROM jobs;
END;
```

24. Create a procedure called UPD_JOB to modify a job in the JOBS table.

a. Create a procedure called UPD_JOB to update the job title. Provide the job ID and a new title, using two parameters. Include the necessary exception handling if no update occurs.

b. Compile the code; invoke the procedure to change the job title of the job ID IT_DBA to Data Administrator. Query the JOBS table to view the results.

Also check the exception handling by trying to update a job that does not exist (you can use job ID IT_WEB and job title Web Master).

```
CREATE OR REPLACE PROCEDURE UPD_JOB(
   jobid IN VARCHAR(20),
   jobtitle IN VARCHAR(20))
BEGIN
   UPDATE jobs
   SET (JOB_TITLE = jobtitle)
```

```
    WHERE JOB_ID = jobid;
END;
/
DECLARE
    jobid VARCHAR(20) := 'IT_DBA';
jobtitle VARCHAR(20) := 'DB MAN';
BEGIN
    UPD_JOB(jobid,jobtitle);
    SELECT * FROM jobs;
END;
```

25. Create a procedure called DEL_JOB to delete a job from the JOBS table.

a. Create a procedure called DEL_JOB to delete a job. Include the necessary exception handling if no job is deleted.

b. Compile the code; invoke the procedure using job ID IT_DBA. Query the JOBS table to view the results.

Also, check the exception handling by trying to delete a job that does not exist (use job ID IT_WEB). You should get the message you used in the exception-handling section of the procedure as output.

1.
```
CREATE OR REPLACE PROCEDURE DEL_JOB(
    jobid IN VARCHAR(20))
BEGIN
    DELETE FROM jobs
    WHERE JOB_ID = jobid;
END;
/
DECLARE
    jobid VARCHAR(20) := 'IT_DBA';
BEGIN
    DEL_JOB(jobid,jobtitle);
    SELECT * FROM jobs;
END;
```

26. Create a procedure called QUERY_EMP to query the EMPLOYEES table, retrieving the salary and job ID for an employee when provided with the employee ID.

a. Create a procedure that returns a value from the SALARY and JOB_ID columns for a specified employee ID.

Use host variables for the two OUT parameters salary and job ID.

b. Compile the code, invoke the procedure to display the salary and job ID for employee ID 120.

c. Invoke the procedure again, passing an EMPLOYEE_ID of 300. What happens and why?

```
CREATE OR REPLACE PROCEDURE QUERY_EMP(
   empid IN VARCHAR2(20),
   salary OUT NUMBER,
   jobid OUT VARCHAR2(20))
BEGIN
   SELECT sal, job_id
   INTO salary, jobid
   FROM emp;
EXCEPTION
   WHEN NO_ROWS_FOUND THEN
      dbms_output.put_line('Wrong employee ID');
END;
/
DECLARE
   empid VARCHAR(20) := 102;
   salary NUMBER;
   jobid VARCHAR2(20);
BEGIN
   DEL_JOB(empid, salary, jobid);
   dbms_output.put_line(salary || ' ' || jobid);
END;
```

27. Create a package specification and body called JOB_PACK. (You can save the package body and specification in two separate files.) This package contains your ADD_JOB, UPD_JOB, and DEL_JOB procedures, as well as your Q_JOB function.

Note: Use the code in your previously saved script files when creating the package.

a. Make all the constructs public.

Note: Consider whether you still need the stand-alone procedures and functions you just

packaged.

b. Invoke your ADD_JOB procedure by passing values IT_SYSAN and SYSTEMS ANALYST as parameters.

c. Query the JOBS table to see the result.

1.
```
CREATE OR REPLACE PACKAGE JOB_PACK IS
   PROCEDURE ADD_JOB(jobid IN VARCHAR(20),jobtitle IN VARCHAR(20));
   PROCEDURE UPD_JOB(jobid IN VARCHAR(20));
   PROCEDURE DEL_JOB(jobid IN VARCHAR(20));
END JOB_PACK;
/

CREATE OR REPLACE PACKAGE BODY JOB_PACK IS
   CREATE OR REPLACE PROCEDURE ADD_JOB(
      jobid IN VARCHAR(20),
      jobtitle IN VARCHAR(20))
   BEGIN
      INSERT INTO jobs VALUES(jobid,jobtitle);
   END ADD_JOB;

   CREATE OR REPLACE PROCEDURE UPD_JOB(
      jobid IN VARCHAR(20),
      jobtitle IN VARCHAR(20))
   BEGIN
      UPDATE jobs

  SET (JOB_TITLE = jobtitle)
```

```
      WHERE JOB_ID = jobid;
   END UPD_JOB;

   CREATE OR REPLACE PROCEDURE DEL_JOB(

 jobid IN VARCHAR(20))

   BEGIN
      DELETE FROM jobs
      WHERE JOB_ID = jobid;
   END DEL_JOB;
END JOB_PACK;
/

DECLARE
   jobid VARCHAR(20) := 'IT_DBA';
   jobtitle VARCHAR(20) := 'Database Administrator';
BEGIN
   JOB_PACK.ADD_JOB(jobid, jobtitle);
   SELECT * FROM jobs;
END;
```

28. Create and invoke a package that contains private and public constructs.

a. Create a package specification and package body called EMP_PACK that contains your NEW_EMP procedure as a public construct, and your VALID_DEPTID function as a private construct. (You can save the specification and body into separate files.)

b. Invoke the NEW_EMP procedure, using 15 as a department number. Because the department ID 15 does not exist in the DEPARTMENTS table, you should get an error message as specified in the exception handler of your procedure.

c. Invoke the NEW_EMP procedure, using an existing department ID 80.

29. Create a package called CHK_PACK that contains the procedures CHK_HIREDATE and CHK_DEPT_MGR. Make both constructs public. (You can save the specification and body into separate files.) The procedure CHK_HIREDATE checks whether an employee's hire date is within the following range: [SYSDATE - 50 years, SYSDATE + 3 months].

Note:

• If the date is invalid, you should raise an application error with an appropriate message indicating why the date value is not acceptable.

• Make sure the time component in the date value is ignored.

• Use a constant to refer to the 50 years boundary.

• A null value for the hire date should be treated as an invalid hire date.

The procedure CHK_DEPT_MGR checks the department and manager combination for a given employee. The CHK_DEPT_MGR procedure accepts an employee ID and a manager ID. The procedure checks that the manager and employee work in the same department. The procedure also checks that the job title of the manager ID provided is

MANAGER.

Note: If the department ID and manager combination is invalid, you should raise an

application error with an appropriate message.

a. Test the CHK_HIREDATE procedure with the following command:

EXECUTE chk_pack.chk_hiredate('01-JAN-47')

What happens, and why?

b. Test the CHK_HIREDATE procedure with the following command:

EXECUTE chk_pack.chk_hiredate(NULL)

What happens, and why?

c. Test the CHK_DEPT_MGR procedure with the following command:

EXECUTE chk_pack.chk_dept_mgr(117,100)

What happens, and why?

```
CREATE OR REPLACE PACKAGE JOB_PACK IS
   PROCEDURE NEW_EMP(empid IN NUMBER, empname IN VARCHAR2(20),
deptidNUMBER);
   EXCEPTION omega_exception;
END JOB_PACK;
/

CREATE OR REPLACE PACKAGE BODY JOB_PACK IS

   CREATE OR REPLACE PROCEDURE VALID_DEPTID(deptid NUMBER, isVALID OUT
VARCHAR2(20)) IS
      IF deptid NOT IN (SELECT deptno FROM dept) THEN
         isVALID := 'Not Valid';
      ELSE
         isVALID := 'Valid';
      END IF;
   END VALID_DEPTID;
```

```
   CREATE OR REPLACE PROCEDURE NEW_EMP(empid IN NUMBER, empname IN
VARCHAR2(20), deptid NUMBER) IS
      v_temp VARCHAR(20);
      VALID_DEPtID(deptid, v_temp);
      IF v_temp = 'Valid' THEN
         INSERT INTO emp(empno, ename, deptno) VALUES(empid, empname,
deptid);
      ELSE
         RAISE omega_exception;
   EXCEPTION
   WHEN omega_exception THEN
      dbms_output.put_line('Department doesnt exist');
   END;
/
```

Triggers

30. Changes to data are allowed on tables only during normal office hours of 8:45 a.m. until 5:30 p.m., Monday through Friday.

Create a stored procedure called SECURE_DML that prevents the DML statement from executing outside of normal office hours, returning the message, "You may only make changes during normal office hours."

```
CREATE OR REPLACE TRIGGER secure_emp
   BEFORE INSERT ON emp
   BEGIN
      IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
         (TO_CHAR(SYSDATE,'HH24:MI') NOT BETWEEN '08:45' AND '17:30')
      THEN
         RAISE APPLICATION ERROR (-20500, 'You cannot insert now');
      END IF
   END;
/
```

31. a. Create a statement trigger on the JOBS table that calls the above procedure.

b. Test the procedure by temporarily modifying the hours in the procedure and attempting to insert a new record into the JOBS table. (Example: replace 08:45 with 16:45; This attempt results in an error message)

After testing, reset the procedure hours as specified in question 1 and recreate the procedure as in question 1 above.

32. Employees should receive an automatic increase in salary if the minimum salary for a job is increased. Implement this requirement through a trigger on the JOBS table.

a. Create a stored procedure named UPD_EMP_SAL to update the salary amount. This procedure accepts two parameters: the job ID for which salary has to be updated, and the new minimum salary for this job ID. This procedure is executed from the trigger on the JOBS table.

b. Create a row trigger named UPDATE_EMP_SALARY on the JOBS table that invokes the procedure UPD_EMP_SAL, when the minimum salary in the JOBS table is updated for a specified job ID.

c. Query the EMPLOYEES table to see the current salary for employees who are programmers.

1.
```
CREATE OR REPLACE PROCEDURE UPD_EMP_SAL(jobid NUMBER, minsal NUMBER)
BEGIN
   UPDATE jobs
   SET salary = minsal
   WHERE JOB_ID = jobid;
END;
/

CREATE OR REPLACE TRIGGER update_emp_salary
   INSTEAD OF INSERT OR UPDATE OF salary ON jobs
   FOR EACH ROW
   BEGIN
      UPD_EMP_SAL(NEW.JOB_ID, NEW.SALARY);
   END;
/
```