

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

- a. DECLARE  
    v\_id                               NUMBER(4);
- b. DECLARE  
    v\_x, v\_y, v\_z                    VARCHAR2(10);
- c. DECLARE  
    v\_birthdate                      DATE NOT NULL;
- d. DECLARE  
    v\_in\_stock                        BOOLEAN := 1;

b is not valid because we cannot declare multiple variables together.

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;
- b. v\_sender := USER || ': ' || TO\_CHAR(v\_dept\_no);
- c. v\_sum := \$100,000 + \$250,000;
- d. v\_flag := TRUE;
- e. v\_n1 := v\_n2 > (2 \* v\_n3);
- f. v\_value := NULL;

- a – valid, return type -DATE
- b – valid, return type -String
- c – valid, return type -Money
- d – valid, return type - Boolean
- e – valid, return type - Boolean

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

G_MESSAGE
My PL/SQL Block Works

```
BEGIN
  dbms_output.put_line('My PL/SQL Block works');
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

G_CHAR	
42 is the answer	

  

G_NUM	
	42

## 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;

1

END;

v\_weight := v\_weight + 1;

v\_message := v\_message || ' is in stock';

v\_new\_locn := 'Western ' || v\_new\_locn;

2

END;

/

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

- The value of V\_WEIGHT at position 1 is:
- The value of V\_NEW\_LOCN at position 1 is:
- The value of V\_WEIGHT at position 2 is:
- The value of V\_MESSAGE at position 2 is:
- The value of V\_NEW\_LOCN at position 2 is:

- a – 2
- b – Western Europe
- c – 601
- d – Product 10012 is in stock
- e – Western

```

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:

- a – 201;
- b – Unisports
- c – Excellent
- d – Womansport
- e – doesn't exist
- f – 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.

```
DECLARE
  v_total NUM;
  p_num1 := 2;
  p_num2 := 4;
BEGIN
  v_total := (p_num1/p_num2)+p_num2;
  dbms_output.put_line(v_total);
END;
```

- 8 . Build a PL/SQL block that computes the total compensation for one year.
- The annual salary and the annual bonus percentage values are defined using the DEFINE command.
  - 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 = 10
```

PL/SQL procedure successfully completed.

G_TOTAL	
	55000

```
DECLARE
  v_total NUM;
  p_salary NUM := 50000;
  p_bonus NUM := 3500;
BEGIN
  v_total = nvl(p_salary,0) + (nvl(p_bonus,0)/100);
  dbms_output.put_line(v_total);
END;
```

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.

G_MAX_DEPTNO
270

```
DECLARE
    department_number NUM;
BEGIN
    SELECT MAX(deptno)
    INTO department_number
    FROM dept;
    dbms_output.put_line(department_number);
END;
```

- 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.
- Use the DEFINE command to provide the department name. Name the new department Education.
  - Pass the value defined for the department name to the PL/SQL block through a &SQL\*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.
  - Leave the location number as null for now.
  - Execute the PL/SQL block.
  - Display the new department that you created.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education		

```
DECLARE
    department_number NUM;
    v_deptname VARCHAR2(20) := 'Education';
BEGIN
    SELECT MAX(deptno)
    INTO department_number
    FROM dept;
    INSERT INTO dept(deptno, dname)
    VALUES(department_number+10,v_deptname);
END;
```

```

/
SELECT * FROM dept;

```

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.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education		1700

```

DECLARE
  department_number NUM;
  v_loc VARCHAR2(20) := 'New york';
BEGIN
  UPDATE dept
  SET loc = v_loc
  WHERE deptno = (30);
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.

G_RESULT
1 row(s) deleted.

- d. Confirm that the department has been deleted.
 

```

no rows selected

```

```

DECLARE
    department_number NUMBER := 30;
BEGIN
    DELETE FROM dept
    WHERE deptno = 30;
END;
/
SELECT * FROM dept;

```

13. Create the `MESSAGES` table. Write a PL/SQL block to insert numbers into the `MESSAGES` table.
- Insert the numbers 1 to 10, excluding 6 and 8.
  - Commit before the end of the block.
  - Select from the `MESSAGES` table to verify that your PL/SQL block worked.

RESULTS	
1	
2	
3	
4	
5	
7	
9	
10	

8 rows selected.

```

CREATE TABLE MESSAGES(RESULT NUM);
DECLARE
    v_count NUM;
BEGIN
    FOR v_count IN 1..10 LOOP
        IF v_count NOT IN (6,8) THEN
            INSERT INTO MESSAGES(result) VALUES (v_count);
        END IF;
    END LOOP;
END;
/
SELECT * FROM MESSAGES;

```

14. Create a PL/SQL block that computes the commission amount for a given employee based on the employee's salary.
- 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`
  - If the employee's salary is less than \$5,000, display the bonus amount for the employee as 10% of the salary.
  - If the employee's salary is between \$5,000 and \$10,000, display the bonus amount for the employee as 15% of the salary.
  - If the employee's salary exceeds \$10,000, display the bonus amount for the employee as 20% of the salary.
  - If the employee's salary is `NULL`, display the bonus amount for the employee as 0.
  - 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.

Employee Number	Salary	Resulting Bonus
100	24000	4800
149	10500	2100
178	7000	1050

```
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 is );
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 &SQL\*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

EMPLOYEE_ID	SALARY	STARS
104	6000	*****
174	11000	*****
176	8600	*****

```
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 is);
END;

```

## Cursors

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));

```

```

CREATE TABLE top_dogs
AS SELECT sal FROM emp;

```

18. Create a PL/SQL block that determines the top employees with respect to salaries.
- 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  $\&n$  substitution variable.
  - In a loop use the  $\&n$  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.
  - Store the salaries in the TOP\_DOGS table.
  - 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.

SALARY	
	24000
	17000
	14000
	13500
	13000

```

DECLARE
v_emp NUMBER;
v_sal emp.sal%TYPE;
CURSOR cuntr IS

```

```

SELECT DISTINCT sal
FROM emp
ORDER BY sal DESC;

```

```

BEGIN
  v_emp = &v_emp;
  open cuntr;
  LOOP
    FETCH cuntr INTO v_sal;
    dbms_output.put_line(v_sal);
    INSERT INTO top_dogs(salary) VALUES (v_sal);
    EXIT WHEN cuntr%ROWCOUNT>v_emp;
  END LOOP;
END;
/

```

19. Create a PL/SQL block that does the following:
- Use the `DEFINE` command to provide the department ID. Pass the value to the PL/SQL block through a `&iSQL*Plus` substitution variable.
  - In a PL/SQL block, retrieve the last name, salary, and `MANAGER` ID of the employees working in that department.
  - 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.

d. 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 Due for a raise Kaufling Due for a raise Vollman Due for a raise Mourgas 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 . . .

```

DECLARE
  v_empid NUMBER;

```

```

v_name VARCHAR2(20);
v_salary NUMBER;
v_manager NUMBER;
BEGIN
v_empid := &empID;
SELECT ename, sal, mgr
INTO v_name, v_salary, v_manager
FROM emp
WHERE empno = v_empid;
IF v_salary < 5000 AND v_manager IN (101,124) THEN\
    dbms_output.put_line(v_name || ' eligible for due of a raise');
ELSE
    dbms_output.put_line(v_name || ' not eligiblr for due of a raise');
END IF;

```

## Exceptions

20. Write a PL/SQL block to select the name of the employee with a given salary value.
- Use the `DEFINE` command to provide the salary.
  - Pass the value to the PL/SQL block through a `&SQL*`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>`.”
  - 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>`.”
  - If the salary entered returns only one row, insert into the `MESSAGES` table the employee’s name and the salary amount.
  - Handle any other exception with an appropriate exception handler and insert into the `MESSAGES` table the message “Some other error occurred.”
  - 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.

RESULTS
More than one employee with a salary of 6000
No employee with a salary of 5000
More than one employee with a salary of 7000
No employee with a salary of 2000

```

DECLARE
v_ename VARCHAR2(20);
CURSOR empcursor IS
SELECT ename

```

```

        FROM emp
        WHERE v_salary = 50000;
BEGIN
    OPEN empcursor;
    LOOP
        FETCH empcursor INTO v_ename;
        dbms_output.put_line(v_ename is);
    END LOOP;
EXCEPTION
    WHEN NO_ROWS_FOUND THEN
        dbms_output.put_line('This salary doesnot exist');
    WHEN others THEN
        dbms_output.put_line('SomeError!');
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.

G_MESSAGE
Department 200 is an invalid department

```

DECLARE
    v_deptid NUMBER;
    v_dname VARCHAR2(20);
BEGIN
    v_deptid := &deptID;
    SELECT dname
    INTO v_dname
    FROM dept
    WHERE deptno = v_deptid;
    dbms_output.put_line(v_dname);
EXCEPTION
    WHEN NO_ROWS_FOUND THEN
        dbms_output.put_line('Wrong department ID given');
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.

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

#### G\_MESSAGE

There is/are 4 employee(s) with a salary between 6900 and 7100

#### G\_MESSAGE

There is/are 12 employee(s) with a salary between 2400 and 2600

#### G\_MESSAGE

There is/are 3 employee(s) with a salary between 6400 and 6600

```
DECLARE
  v_emp VARCHAR2(20);
  v_sal NUMBER;
  CURSOR container IS
    SELECT ename
    FROM emp
    WHERE sal IN (5000 + 100, 5000-100);
BEGIN
  OPEN container;
  LOOP
    FETCH holder INTO v_emp;
    dbms_output.put_line(v_emp);
  END LOOP;
EXCEPTION
  WHEN NO_ROWS_FOUND THEN
    dbms_output.put_line('No employees exist with this salary');
  WHEN OTHERS THEN
    dbms_output.put_line(' SOME ERROR!');
END;
```

## Procedures

Save your subprograms as .sql files, using the Save Script button.

Remember to set the SERVEROUTPUT ON if you set it off previously.

23. Create and invoke the ADD\_JOB procedure and consider the results.
- 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.
  - 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.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		

- 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 A_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  
  A_job(jobid,jobtitle);  
  SELECT * FROM jobs;  
END;
```

24. Create a procedure called UPD\_JOB to modify a job in the JOBS table.
- 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.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		

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 U_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
  U_job(jobid,jobtitle);
  SELECT * FROM jobs;
END;
```

25. Create a procedure called DEL\_JOB to delete a job from the JOBS table.
- Create a procedure called DEL\_JOB to delete a job. Include the necessary exception handling if no job is deleted.
  - Compile the code; invoke the procedure using job ID IT\_DBA. Query the JOBS table to view the results.
- no rows selected

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.

```
CREATE OR REPLACE PROCEDURE D_job(
  jobid IN VARCHAR(20))
BEGIN
  DELETE FROM jobs
  WHERE JOB_ID = jobid;
END;
/
DECLARE
  jobid VARCHAR(20) := 'IT_DBA';
```



```
BEGIN
  D_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.
- 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.
  - Compile the code, invoke the procedure to display the salary and job ID for employee ID 120.

G_SAL
8000

G_JOB
ST_MAN

- 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 GIVEN');
END;
/
DECLARE
  empid VARCHAR(20) := 102;
  salary NUMBER;
  jobid VARCHAR2(20);
BEGIN
  DEL_JOB(empid, salary, jobid);
  dbms_output.put_line(salary || ' ' || jobid);
END;
```

## Packages

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.

```
CREATE OR REPLACE PACKAGE JOB_PACK IS
  PROCEDURE A_job(jobid IN VARCHAR(20),jobtitle IN VARCHAR(20));
  PROCEDURE U_job(jobid IN VARCHAR(20));
  PROCEDURE D_job(jobid IN VARCHAR(20));
END JOB_PACK;
/
```

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

```
  CREATE OR REPLACE PROCEDURE U_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 D_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.
- 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.)
  - 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.
  - Invoke the `NEW_EMP` procedure, using an existing department ID 80.

```

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;
/

```

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.

- Test the `CHK_HIREDATE` procedure with the following command:  
**EXECUTE chk\_pack.chk\_hiredate('01-JAN-47')**  
 What happens, and why?
- Test the `CHK_HIREDATE` procedure with the following command:  
**EXECUTE chk\_pack.chk\_hiredate(NULL)**  
 What happens, and why?
- Test the `CHK_DEPT_MGR` procedure with the following command:  
**EXECUTE chk\_pack.chk\_dept\_mgr(117,100)**  
 What happens, and why?

## 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 date_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 may only make changes during
normal office hours."
);

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.

LAST_NAME	FIRST_NAME	SALARY
Austin	David	5280
Hunold	Alexander	9000
Ernst	Bruce	6000
Pataballa	Valli	5280
Lorentz	Diana	4620

- d. Increase the minimum salary for the Programmer job from 4,000 to 5,000.
- e. Employee Lorentz (employee ID 107) had a salary of less than 4,500. Verify that her salary has been increased to the new minimum of 5,000.

LAST_NAME	FIRST_NAME	SALARY
Lorentz	Diana	5000

```
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;
/
```