Lab1

1.1: Get the details of all the database objects and their types created by the current

user.

DECLARE

CURSOR ts IS

SELECT OBJECT\_NAME,OBJECT\_TYPE

FROM USER\_OBJECTS;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the database objects and their type created by the current user is:');

DBMS\_OUTPUT.PUT\_LINE('Object name' || chr(9) || chr(9) || 'Object type');

FOR I IN ts

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.OBJECT\_NAME || chr(9) || chr(9) || I.OBJECT\_TYPE);

END LOOP;

END;

1.2 Get the details of all the table names owned by current userDECLARE

CURSOR ts IS

SELECT TABLE\_NAME

FROM USER\_TABLES;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the tables created by the current user are:');

DBMS\_OUTPUT.PUT\_LINE('Object name');

FOR I IN ts

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.TABLE\_NAME);

END LOOP;

END;

1.3 Get the details of table names and corresponding column names

DECLARE

CURSOR utc IS

SELECT TABLE\_NAME,COLUMN\_NAME

FROM USER\_TAB\_COLUMNS;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the table names and the corresponding column names :');

DBMS\_OUTPUT.PUT\_LINE('Table Name' || chr(9) || chr(9) || 'Column Name');

FOR I IN utc

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.TABLE\_NAME || chr(9) || chr(9) || I.COLUMN\_NAME);

END LOOP;

END;

1.4 Get the details of column names and corresponding constraint names

DECLARE

CURSOR ucc IS

SELECT COLUMN\_NAME,CONSTRAINT\_NAME

FROM USER\_CONS\_COLUMNS;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the column names and the corresponding constraint names :');

DBMS\_OUTPUT.PUT\_LINE('Column Name' || chr(9) || chr(9) || 'Constraint Name');

FOR I IN ucc

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.COLUMN\_NAME || chr(9) || chr(9) || I.CONSTRAINT\_NAME);

END LOOP;

END;

1.5: Get the details of the constraints and corresponding table name.

DECLARE

CURSOR ucc IS

SELECT CONSTRAINT\_NAME,TABLE\_NAME

FROM USER\_CONS\_COLUMNS;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the constraint names and the corresponding table names :');

DBMS\_OUTPUT.PUT\_LINE('Constraint Name' || chr(9) || chr(9) || 'Table Name');

FOR I IN ucc

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.CONSTRAINT\_NAME || chr(9) || chr(9) || I.TABLE\_NAME);

END LOOP;

CLOSE ucc;

END;

1.6: Get the details of all the View names and corresponding Text of the same.

DECLARE

CURSOR uv IS

SELECT VIEW\_NAME,TEXT

FROM USER\_VIEWS;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the view names and the corresponding text :');

DBMS\_OUTPUT.PUT\_LINE('View Name' || chr(9) || chr(9) || 'Text');

FOR I IN uv

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.VIEW\_NAME || chr(9) || chr(9) || I.TEXT);

END LOOP;

END;

1.7: Get the details of all the Sequence names and their last numbers reached so far.

DECLARE

CURSOR us IS

SELECT SEQUENCE\_NAME,LAST\_NUMBER

FROM USER\_SEQUENCES;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the view names and the corresponding text :');

DBMS\_OUTPUT.PUT\_LINE('Sequence Name' || chr(9) || chr(9) || 'Last Value');

FOR I IN us

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.SEQUENCE\_NAME || chr(9) || chr(9) || I.LAST\_NUMBER);

END LOOP;

END;

1.8: Get the details of all the Synonym names and their parent object names.

DECLARE

CURSOR us IS

SELECT SYNONYM\_NAME,TABLE\_OWNER

FROM USER\_SYNONYMS;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Details of the view names and the corresponding text :');

DBMS\_OUTPUT.PUT\_LINE('Synonym Name' || chr(9) || chr(9) || 'Owner Name');

FOR I IN us

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.SYNONYM\_NAME || chr(9) || chr(9) || I.TABLE\_OWNER);

END LOOP;

END;

1.9: Get the list of all the Index names

DECLARE

CURSOR ui IS

SELECT INDEX\_NAME

FROM USER\_INDEXES;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('List of index names :');

DBMS\_OUTPUT.PUT\_LINE('Index Name');

FOR I IN ui

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.INDEX\_NAME);

END LOOP;

END;

Lab:2

2.1.Identify the problems(if any) in the below declarations:

DECLARE

V\_Sample1 NUMBER(2);

V\_Sample2 CONSTANT NUMBER(2) := 5 ;

V\_Sample3 NUMBER(2) NOT NULL := 5;

V\_Sample4 NUMBER(2) := 50;

V\_Sample5 NUMBER(2) DEFAULT 25;

BEGIN

dbms\_output.put\_line(V\_Sample2);

dbms\_output.put\_line(V\_Sample3);

dbms\_output.put\_line(V\_Sample4);

dbms\_output.put\_line(V\_Sample5);

END;

2.2.The following PL/SQL block is incomplete.

Modify the block to achieve requirements as stated in the comments in the block.

DECLARE --outer block

var\_num1 NUMBER := 5;

BEGIN

DECLARE --inner block

var\_num1 NUMBER := 10;

BEGIN

--DBMS\_OUTPUT.PUT\_LINE('Value for var\_num1:' ||var\_num1);

--Can outer block variable (var\_num1) be printed here.If Yes,Print the same.

dbms\_output.put\_line(OUTERBLOCK.var\_num1);

--DBMS\_OUTPUT.PUT\_LINE('Value for var\_num1 inner block:' ||var\_num1);

END;

--DBMS\_OUTPUT.PUT\_LINE('Value for var\_num1 outer block:' ||var\_num1);

--Can inner block variable(var\_num1) be printed here.If Yes,Print the same.

END;

2.3. Write a PL/SQL block to retrieve all staff (code, name, salary) under specific

department number and display the result. (Note: The Department\_Code will be accepted

from user. Cursor to be used.**)**

DECLARE

n\_dept\_code STAFF\_MASTERS.DEPT\_CODE%TYPE;

CURSOR staff IS

SELECT STAFF\_MASTERS.STAFF\_CODE,STAFF\_MASTERS.STAFF\_NAME,STAFF\_MASTERS.STAFF\_SAL

FROM STAFF\_MASTERS

WHERE STAFF\_MASTERS.DEPT\_CODE = n\_dept\_code;

BEGIN

n\_dept\_code := &dept\_code;

FOR I IN staff

LOOP

dbms\_output.put\_line('Staff code = ' || I.STAFF\_CODE || ' ,Staff name = ' || I.STAFF\_NAME || ' ,Staff salary = ' || I.STAFF\_SAL);

END LOOP;

END;

2.4. Write a PL/SQL block to increase the salary by 30 % or 5000 whichever minimum for

a given Department\_Code.

DECLARE

n\_staff\_newsal STAFF\_MASTERS.STAFF\_SAL%TYPE;

n\_dept\_code STAFF\_MASTERS.DEPT\_CODE%TYPE;

CURSOR staff IS

SELECT STAFF\_MASTERS.STAFF\_SAL

FROM STAFF\_MASTERS

WHERE STAFF\_MASTERS.DEPT\_CODE = n\_dept\_code;

BEGIN

n\_dept\_code := &dept\_code;

FOR I IN staff

LOOP

n\_staff\_newsal := I.STAFF\_SAL \* 0.3;

dbms\_output.put\_line('SALARY Before updation \* 0.3 : ' || n\_staff\_newsal);

IF n\_staff\_newsal > 5000 THEN

n\_staff\_newsal := 5000;

END IF;

dbms\_output.put\_line('SALARY Before updation : ' || I.STAFF\_SAL);

UPDATE STAFF\_MASTERS set STAFF\_SAL = STAFF\_SAL + n\_staff\_newsal;

dbms\_output.put\_line('SALARY after updation: ' || (I.STAFF\_SAL + n\_staff\_newsal));

END LOOP;

END;

2.5. Write a PL/SQL block to generate the following report for a given Department code

Student\_Code Sudent\_Name Subject1 Subject2 Subject3 Total Percentage

Grade

Note: Display suitable error massage if wrong department code has entered and if there

is no student in the given department.

For Grade:

Student should pass in each subject individually (pass marks 60).

Percent >= 80 then grade= A

Percent >= 70 and < 80 then grade= B

Percent >= 60 and < 70 then grade= C

Else D

DECLARE

n\_dept\_code STUDENT\_MASTERS.DEPT\_CODE%TYPE;

n\_total NUMBER(5);

n\_average NUMBER(5);

c\_grade VARCHAR(2);

count\_student NUMBER;

invalid\_dept EXCEPTION;

CURSOR student IS

SELECT STUDENT\_MASTERS.STUDENT\_CODE,STUDENT\_MASTERS.STUDENT\_NAME,STUDENT\_MARKS.SUBJECT1,STUDENT\_MARKS.SUBJECT2,STUDENT\_MARKS.SUBJECT3

FROM STUDENT\_MASTERS,STUDENT\_MARKS

WHERE STUDENT\_MASTERS.STUDENT\_CODE = STUDENT\_MARKS.STUDENT\_CODE

AND

DEPT\_CODE = n\_dept\_code;

BEGIN

n\_dept\_code := &dept\_code;

SELECT COUNT(\*)

INTO count\_student

FROM STAFF\_MASTERS

WHERE DEPT\_CODE = n\_dept\_code;

IF count\_student = 0 THEN

RAISE invalid\_dept;

END IF;

FOR I IN student

LOOP

n\_total := I.SUBJECT1 + I.SUBJECT2 + I.SUBJECT3;

n\_average := ROUND((n\_total/3),0);

CASE

WHEN n\_average >= 80 THEN c\_grade := 'A';

WHEN n\_average >= 70 AND n\_average < 80 THEN c\_grade := 'B';

WHEN n\_average >= 60 AND n\_average < 70 THEN c\_grade := 'C';

ELSE c\_grade := 'D';

END CASE;

IF (I.SUBJECT1 < 60) OR (I.SUBJECT2 < 60) OR (I.SUBJECT3 < 60) THEN

c\_grade := 'D';

END IF;

EXIT WHEN student%NOTFOUND;

dbms\_output.put\_line('Student Code' || chr(9) ||'Student name' || chr(9) ||'Subject1' || chr(9) ||'Subject2' || chr(9) ||'Subject3' || chr(9) ||'Grade');

dbms\_output.put\_line(I.STUDENT\_CODE || chr(9) || I.STUDENT\_NAME || chr(9) || I.SUBJECT1 || chr(9) || I.SUBJECT2 || chr(9) || I.SUBJECT3 || chr(9) || c\_grade);

END LOOP;

EXCEPTION

WHEN invalid\_dept THEN

DBMS\_OUTPUT.PUT\_LINE('Invalid dept\_code was entered!!!');

END;

2.6. Write a PL/SQL block to retrieve the details of the staff belonging to a particular

department. Department code should be passed as a parameter to the cursor.

DECLARE

n\_dept\_code STAFF\_MASTERS.DEPT\_CODE%TYPE;

CURSOR staff\_cur(v\_dept\_code NUMBER) IS

SELECT \*

FROM staff\_masters

WHERE DEPT\_CODE = v\_dept\_code;

staff\_rec STAFF\_MASTERS%ROWTYPE;

BEGIN

n\_dept\_code := &dept\_code;

FOR I IN staff\_cur(n\_dept\_code)

LOOP

dbms\_output.put\_line('Staff code: ' || I.staff\_code);

dbms\_output.put\_line('Staff Name: ' || I.staff\_name);

dbms\_output.put\_line('Staff Designation code: ' || I.design\_code);

dbms\_output.put\_line('Staff Department code: ' || I.dept\_code);

dbms\_output.put\_line('Staff DOB: ' || I.staff\_dob);

dbms\_output.put\_line('Staff Hiredate: ' || I.hiredate);

dbms\_output.put\_line('Staff Manager Code: ' || I.mgr\_code);

dbms\_output.put\_line('Staff Salary: ' || I.staff\_sal);

dbms\_output.put\_line('Staff Address: ' || I.staff\_address);

END LOOP;

END;

.

/

LAB 3

--3.3:

DECLARE

V\_SAL STAFF\_MASTERS.STAFF\_SAL%TYPE;

V\_BONUS V\_SAL%TYPE;

CURSOR emp IS

SELECT STAFF\_SAL

FROM STAFF\_MASTERS

WHERE MGR\_CODE=100006;

BEGIN

OPEN emp;

LOOP

FETCH emp into V\_SAL;

EXIT WHEN emp%NOTFOUND;

V\_BONUS:=2\*V\_SAL;

DBMS\_OUTPUT.PUT\_LINE('STAFF SALARY IS ' || V\_SAL);

DBMS\_OUTPUT.PUT\_LINE('STAFF BONUS IS ' || V\_BONUS);

END LOOP;

CLOSE emp;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('GIVEN CODE IS NOT VALID.ENTER VALID CODE');

END;

3.4.Predict the output of the following block ? What corrections would be needed to make it

more efficient?

DECLARE

BEGIN

DECLARE

fname STAFF\_MASTERS.staff\_name%TYPE;

BEGIN

SELECT staff\_name

INTO fname

FROM STAFF\_MASTERS

WHERE 1=2;

DBMS\_OUTPUT.PUT\_LINE('This statement will print');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Some inner block error');

END;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No data found in fname');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Some outer block error');

END;

3.6: Write a PL/SQL program to check for the commission for an employee no 7369. If no

commission exists, then display the error message. Use Exceptions.DECLARE

n\_sal STAFF\_MASTERS.STAFF\_SAL%TYPE;

BEGIN

SELECT STAFF\_SAL

INTO n\_sal

FROM STAFF\_MASTERS

WHERE STAFF\_CODE = 7369;

DBMS\_OUTPUT.PUT\_LINE(n\_sal);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No staff with employee number 7369 exists in the database.');

END;

3.7: Write a PL/SQL block to drop any user defined table.

DECLARE

table\_name VARCHAR(30);

BEGIN

table\_name := '&table\_name';

execute immediate 'DROP TABLE ' || table\_name;

DBMS\_OUTPUT.PUT\_LINE('Table dropped successfully.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('No such table exists.');

END;

Lab4

4.1 Write a PL/SQL block to find the maximum salary of the staff in the given department.

Note: Department code should be passed as parameter to the cursor.  
DECLARE

n\_count NUMBER(2);

n\_dept\_code STAFF\_MASTERS.DEPT\_CODE%TYPE;

n\_max\_salary STAFF\_MASTERS.STAFF\_SAL%TYPE;

--User defined exception

ex\_invalid\_dept\_id EXCEPTION;

BEGIN

n\_dept\_code := &dept\_code;

SELECT COUNT(\*)

INTO n\_count

FROM STAFF\_MASTERS

WHERE DEPT\_CODE = n\_dept\_code;

IF n\_count < 0 THEN

RAISE ex\_invalid\_dept\_id;

ELSE

SELECT MAX(STAFF\_SAL)

INTO n\_max\_salary

FROM STAFF\_MASTERS

WHERE DEPT\_CODE = n\_dept\_code

GROUP BY DEPT\_CODE;

dbms\_output.put\_line('Maximum Salary for an employee with department code of ' || n\_dept\_code || ' is ' || n\_max\_salary);

END IF;

EXCEPTION

WHEN ex\_invalid\_dept\_id THEN

dbms\_output.put\_line('No Such department id exists.');

WHEN OTHERS THEN

dbms\_output.put\_line('Error Occurred');

END;

4.2. Write a function to compute age. The function should accept a date and return age in

yearsDECLARE

c number;

c1 number;

c2 number;

FUNCTION dateofbirth(x in VARCHAR2)

RETURN number

IS

months number := 1;

BEGIN

months := ROUND(MONTHS\_BETWEEN(SYSDATE(),x));

RETURN months;

END;

BEGIN

c := dateofbirth('08-APR-96');

c1 := ROUND(c / 12,0);

c2 := c - (c1 \* 12);

dbms\_output.put\_line('Age is ' || c1 || ' years and ' || c2 || ' months.');

END;

4.3. Write a procedure that accept staff code and update staff name to Upper case. If the

staff name is null raise a user defined exception.

DECLARE

n\_staff\_code STAFF\_MASTERS.STAFF\_CODE%TYPE;

invalid\_name EXCEPTION;

S\_NAME STAFF\_MASTERS.STAFF\_NAME%TYPE;

PROCEDURE upper\_name (scode IN number) IS

BEGIN

SELECT STAFF\_NAME

INTO S\_NAME

FROM STAFF\_MASTERS

WHERE STAFF\_CODE = scode;

IF S\_NAME IS NULL THEN

RAISE invalid\_name;

ELSE

UPDATE staff\_masters

SET staff\_name = UPPER(staff\_name)

WHERE staff\_code = scode;

END IF;

END;

BEGIN

n\_staff\_code := &staff\_code;

upper\_name(n\_staff\_code);

dbms\_output.put\_line('Staff name updated successfully');

EXCEPTION

WHEN invalid\_name THEN

DBMS\_OUTPUT.PUT\_LINE('Staff name was NULL!!!');

END;

4.4 Write a procedure to find the manager of a staff. Procedure should return the

following – Staff\_Code, Staff\_Name, Dept\_Code and Manager Name.

DECLARE

n\_staff\_code STAFF\_MASTERS.STAFF\_CODE%TYPE;

v\_staff\_name STAFF\_MASTERS.STAFF\_NAME%TYPE;

n\_dept\_code STAFF\_MASTERS.DEPT\_CODE%TYPE;

v\_mgr\_name STAFF\_MASTERS.STAFF\_NAME%TYPE;

PROCEDURE manager(scode IN OUT NUMBER, sname OUT VARCHAR, dcode OUT NUMBER, mname OUT VARCHAR) IS

BEGIN

SELECT e1.staff\_name,e1.dept\_code,e2.staff\_name

INTO sname,dcode,mname

FROM staff\_masters e1 , staff\_masters e2

WHERE e1.mgr\_code = e2.staff\_code

AND e1.staff\_code = scode;

END;

BEGIN

n\_staff\_code := &staff\_code;

manager(n\_staff\_code,v\_staff\_name,n\_dept\_code,v\_mgr\_name);

dbms\_output.put\_line('Staff Code:' || n\_staff\_code);

dbms\_output.put\_line('Staff Name:' || v\_staff\_name);

dbms\_output.put\_line('Department Code:' || n\_dept\_code);

dbms\_output.put\_line('Manager Name:' || v\_mgr\_name);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Staff Code not Valid!!');

END;

4.5. Write a function to compute the following. Function should take Staff\_Code and

return the cost to company.

DA = 15% Salary, HRA= 20% of Salary, TA= 8% of Salary.

Special Allowance will be decided based on the service in the company.

< 1 Year Nil

>=1 Year< 2 Year 10% of Salary

>=2 Year< 4 Year 20% of Salary

>4 Year 30% of Salary

DECLARE

n\_staff\_code STAFF\_MASTERS.STAFF\_CODE%TYPE;

c NUMBER;

n\_total\_sal STAFF\_MASTERS.STAFF\_SAL%TYPE;

FUNCTION calculate\_salary(scode IN NUMBER)

RETURN NUMBER

IS

d\_hiredate STAFF\_MASTERS.HIREDATE%TYPE;

n\_staff\_sal STAFF\_MASTERS.STAFF\_SAL%TYPE;

months NUMBER := 1;

year1 NUMBER;

n\_total\_sal STAFF\_MASTERS.STAFF\_SAL%TYPE;

BEGIN

SELECT staff\_sal,hiredate

INTO n\_staff\_sal,d\_hiredate

FROM staff\_masters

WHERE staff\_code = scode;

months := ROUND(MONTHS\_BETWEEN(SYSDATE(),d\_hiredate));

year1 := Round(months/12,0);

n\_total\_sal := 0.43 \* n\_staff\_sal;

IF (year1 >= 1 AND year1 < 2) THEN

n\_total\_sal := n\_total\_sal + n\_staff\_sal \* 0.1;

ELSIF (year1 >= 2 AND year1 < 4) THEN

n\_total\_sal := n\_total\_sal + n\_staff\_sal \* 0.2;

ELSIF (year1 > 4) THEN

n\_total\_sal := n\_total\_sal + n\_staff\_sal \* 0.3;

END IF;

RETURN n\_total\_sal;

END;

BEGIN

n\_staff\_code := &staff\_code;

c := calculate\_salary(n\_staff\_code);

dbms\_output.put\_line('Cost to Company is ' || c);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Staff Code not Valid!!');

END;

4.6. Write a procedure that displays the following information of all staff

Staff\_Name Department Name Designation Salary Status

Note: - Status will be (Greater, Lesser or Equal) respective to average salary of their own

department. Display an error message Staff\_Master table is empty if there is no matching

record.

CREATE OR REPLACE PROCEDURE status\_calculator10 IS

v\_staff\_name STAFF\_MASTERS.STAFF\_NAME%TYPE;

v\_dept\_name DEPARTMENT\_MASTERS.DEPT\_NAME%TYPE;

v\_design\_name DESIGNATION\_MASTERS.DESIGN\_NAME%TYPE;

n\_staff\_sal STAFF\_MASTERS.STAFF\_SAL%TYPE;

n\_average\_sal STAFF\_MASTERS.STAFF\_SAL%TYPE;

v\_status VARCHAR2(10);

CURSOR staff IS

SELECT A.STAFF\_NAME,B.DEPT\_NAME,C.DESIGN\_NAME,A.STAFF\_SAL,D.average,

CASE

WHEN A.STAFF\_SAL > D.average THEN 'GREATER'

WHEN A.STAFF\_SAL = D.average THEN 'EQUAL'

ELSE 'LESSER'

END STATUS

FROM STAFF\_MASTERS A,DEPARTMENT\_MASTERS B,DESIGNATION\_MASTERS C,

(SELECT DEPT\_CODE AS code,AVG(STAFF\_SAL) AS average

FROM STAFF\_MASTERS

GROUP BY DEPT\_CODE) D

WHERE A.DEPT\_CODE = B.DEPT\_CODE

AND A.DESIGN\_CODE = C.DESIGN\_CODE

AND D.code = A.DEPT\_CODE;

BEGIN

OPEN staff;

LOOP

FETCH staff INTO v\_staff\_name,v\_dept\_name,v\_design\_name,n\_staff\_sal,n\_average\_sal,v\_status;

EXIT WHEN staff%NOTFOUND;

dbms\_output.put\_line(v\_staff\_name||chr(9)||v\_dept\_name||chr(9)||v\_design\_name||chr(9)||n\_staff\_sal||chr(9)||n\_average\_sal||chr(9)||v\_status);

END LOOP;

CLOSE staff;

END;

4.7. Write a procedure that accept Staff\_Code and update the salary and store the old

salary details in Staff\_Master\_Back (Staff\_Master\_Back has the same structure without

any constraint) table.

Exp < 2 then no Update

Exp > 2 and < 5 then 20% of salary

Exp > 5 then 25% of salary

CREATE OR REPLACE PROCEDURE update\_salary(scode IN NUMBER) IS

staff STAFF\_MASTERS%ROWTYPE;

num1 NUMBER;

months NUMBER := 1;

year1 NUMBER;

BEGIN

SELECT \*

INTO staff

FROM staff\_masters

WHERE staff\_code = scode;

months := ROUND(MONTHS\_BETWEEN(SYSDATE(),staff.hiredate));

year1 := ROUND(months/12,0);

num1 := 1;

IF year1 > 2 AND year1 < 5 THEN

num1 := num1 \* 1.2;

ELSIF year1 > 5 THEN

num1 := num1 \* 1.25;

END IF;

IF (num1 != 1) THEN

INSERT INTO staff\_master\_back

SELECT \*

FROM staff\_masters

WHERE staff\_code = scode;

UPDATE staff\_masters

SET staff\_sal = staff\_sal \* num1

WHERE staff\_code = scode;

dbms\_output.put\_line('Salary updated successfully');

END IF;

END;

4.8. Create a procedure that accepts the book code as parameter from the user. Display

the details of the students/staff that have borrowed that book and has not returned the

same. The following details should be displayed

Student/Staff Code Student/Staff Name Issue Date Designation Expected Ret\_Date

CREATE OR REPLACE PROCEDURE books\_not\_returned(bcode IN NUMBER) IS

n\_code NUMBER;

v\_name VARCHAR2(50);

idate DATE;

designation DESIGNATION\_MASTERS.DESIGN\_NAME%TYPE := NULL;

return\_date DATE;

CURSOR staff IS

SELECT B.STUDENT\_CODE AS CODE,B.STUDENT\_NAME AS NAME,A.BOOK\_ISSUE\_DATE,A.BOOK\_EXPECTED\_RETURN\_DATE

FROM BOOK\_TRANSACTIONS A,STUDENT\_MASTERS B

WHERE A.STUDENT\_CODE = B.STUDENT\_CODE

AND A.BOOK\_ACTUAL\_RETURN\_DATE IS NULL

AND A.BOOK\_CODE = bcode;

CURSOR student IS

SELECT B.STAFF\_CODE AS CODE,B.STAFF\_NAME AS NAME,C.DESIGN\_NAME,A.BOOK\_ISSUE\_DATE,A.BOOK\_EXPECTED\_RETURN\_DATE

FROM BOOK\_TRANSACTIONS A,STAFF\_MASTERS B,DESIGNATION\_MASTERS C

WHERE A.STAFF\_CODE = B.STAFF\_CODE

AND B.DESIGN\_CODE = C.DESIGN\_CODE

AND A.BOOK\_ACTUAL\_RETURN\_DATE IS NULL

AND A.BOOK\_CODE = bcode;

BEGIN

dbms\_output.put\_line('Student/Staff Code'||chr(9)||'Student/Staff Name'||chr(9)||'Issue Date'||chr(9)||'Designation'||chr(9)||'Expected Return Date');

OPEN staff;

LOOP

FETCH staff INTO n\_code,v\_name,idate,return\_date;

EXIT WHEN staff%NOTFOUND;

dbms\_output.put\_line(n\_code||chr(9)||v\_name||chr(9)||idate||chr(9)||designation||chr(9)||return\_date);

END LOOP;

CLOSE staff;

OPEN student;

LOOP

FETCH student INTO n\_code,v\_name,idate,designation,return\_date;

EXIT WHEN student%NOTFOUND;

dbms\_output.put\_line(n\_code||chr(9)||v\_name||chr(9)||idate||chr(9)||designation||chr(9)||return\_date);

END LOOP;

CLOSE student;

END;

4.9. Write a package which will contain a procedure and a function.

Function: This function will return years of experience for a staff. This function will take

the hiredate of the staff as an input parameter. The output will be rounded to the nearest

year (1.4 year will be considered as 1 year and 1.5 year will be considered as 2 year).

Procedure: Capture the value returned by the above function to calculate the additional

allowance for the staff based on the experience.

Additional Allowance = Year of experience x 3000

Calculate the additional allowance and store Staff\_Code, Date of Joining, and Experience

in years and additional allowance in Staff\_Allowance table.

CREATE OR REPLACE PACKAGE staff\_package

AS

FUNCTION EXPERIENCE(V\_HIREDATE IN STAFF\_MASTERS.HIREDATE%TYPE) RETURN NUMBER;

PROCEDURE ALLOWANCE(S\_CODE IN NUMBER);

END;

CREATE OR REPLACE PACKAGE BODY staff\_package

AS

FUNCTION EXPERIENCE(V\_HIREDATE IN STAFF\_MASTERS.HIREDATE%TYPE)

RETURN NUMBER

IS

c\_expr NUMBER;

BEGIN

c\_expr := ROUND(MONTHS\_BETWEEN(SYSDATE,V\_HIREDATE)/12,0);

RETURN c\_expr;

END EXPERIENCE;

PROCEDURE ALLOWANCE(S\_CODE IN NUMBER)

IS

C\_EXPR NUMBER;

ADDITIONAL\_ALLOWANCE NUMBER;

V\_HIREDATE STAFF\_MASTERS.HIREDATE%TYPE;

BEGIN

SELECT HIREDATE

INTO V\_HIREDATE

FROM STAFF\_MASTERS

WHERE STAFF\_CODE = S\_CODE;

C\_EXPR := staff\_package.EXPERIENCE(V\_HIREDATE);

DBMS\_OUTPUT.PUT\_LINE(C\_EXPR);

ADDITIONAL\_ALLOWANCE := C\_EXPR \* 3000;

DBMS\_OUTPUT.PUT\_LINE(ADDITIONAL\_ALLOWANCE);

INSERT INTO STAFF\_ALLOWANCE VALUES(S\_CODE,V\_HIREDATE,C\_EXPR,ADDITIONAL\_ALLOWANCE);

END ALLOWANCE;

END staff\_package;

4.10. Write a procedure to insert details into Book\_Transaction table. Procedure should

accept the book code and staff/student code. Date of issue is current date and the

expected return date should be 10 days from the current date. If the expected return date

falls on Saturday or Sunday, then it should be the next working day.

CREATE OR REPLACE PROCEDURE BOOK\_TRANSACTION\_PROCEDURE(B\_CODE IN NUMBER,S\_CODE IN NUMBER)

AS

COUNT\_STAFF NUMBER;

COUNT\_STUDENT NUMBER;

ISSUE\_DATE DATE;

EXPECTED\_RETURN\_DATE DATE;

BEGIN

SELECT COUNT(\*)

INTO COUNT\_STAFF

FROM STAFF\_MASTERS

WHERE STAFF\_CODE = S\_CODE;

SELECT COUNT(\*)

INTO COUNT\_STUDENT

FROM STUDENT\_MASTERS

WHERE STUDENT\_CODE = S\_CODE;

SELECT SYSDATE,

CASE

WHEN TO\_CHAR((SYSDATE+10),'Day') LIKE '%Saturday%'

OR TO\_CHAR((SYSDATE+10),'Day') like '%Sunday%' THEN next\_day((sysdate+10),'Monday')

ELSE (sysdate+10)

END return\_date

INTO ISSUE\_DATE,EXPECTED\_RETURN\_DATE

FROM DUAL;

IF COUNT\_STAFF = 0 AND COUNT\_STUDENT = 0 THEN

RAISE NO\_DATA\_FOUND;

ELSIF COUNT\_STAFF = 1 THEN

INSERT INTO BOOK\_TRANSACTIONS (BOOK\_CODE,STAFF\_CODE,BOOK\_ISSUE\_DATE,BOOK\_EXPECTED\_RETURN\_DATE) VALUES (B\_CODE,S\_CODE,ISSUE\_DATE,EXPECTED\_RETURN\_DATE);

ELSE

INSERT INTO BOOK\_TRANSACTIONS (BOOK\_CODE,STUDENT\_CODE,BOOK\_ISSUE\_DATE,BOOK\_EXPECTED\_RETURN\_DATE) VALUES (B\_CODE,S\_CODE,ISSUE\_DATE,EXPECTED\_RETURN\_DATE);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('NO SUCH STUDENT/STAFF CODE FOUND.');

END BOOK\_TRANSACTION\_PROCEDURE;

4.11: Write a function named ‘get\_total\_records’, to pass the table name as a parameter,

and get back the number of records that are contained in the table. Test your function

with multiple tables.

CREATE OR REPLACE FUNCTION GET\_TOTAL\_RECORDS(T\_NAME IN VARCHAR)

RETURN NUMBER

AS

V\_COUNT NUMBER;

BEGIN

SELECT NUM\_ROWS

INTO V\_COUNT

FROM USER\_TABLES

WHERE TABLE\_NAME = T\_NAME;

RETURN V\_COUNT;

END;

4.12.The Procedure should update the salary of an employee and at the same time

retrieve the employee's name and new salary into PL/SQL variables.

CREATE OR REPLACE PROCEDURE update\_salary(emp\_id NUMBER)

IS

v\_name emp\_copy.ename%TYPE;

v\_newsal emp\_copy.sal%TYPE;

BEGIN

UPDATE emp\_copy SET sal = sal \* 1.1

WHERE empno = emp\_id;

SELECT ename, sal

INTO v\_name, v\_newsal

FROM emp\_copy

WHERE empno = emp\_id;

DBMS\_OUTPUT.PUT\_LINE('Emp Name:' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Ename:' || v\_newsal);

END;

4.13.The following procedure attempts to delete data from table passed as parameter.This

procedure has compilation errors. Identify and correct the problem.

CREATE or REPLACE PROCEDURE gettable(table\_name in varchar2) AS

BEGIN

execute immediate 'DELETE FROM ' || table\_name;

END;

4.14

Write a procedure which prints the following report using procedure:

The procedure should take deptno as user input and appropriately print the emp details.

Also display : Number of Employees,Total Salary,Maximum Salary,Average Salary

**Note:** The block should achieve the same without using Aggregate Functions.

Sample output for deptno 10 is shown below:

CREATE OR REPLACE PROCEDURE DETAILS(V\_DEPT\_CODE IN STAFF\_MASTERS.DEPT\_CODE%TYPE)

AS

TOTAL\_EMPLOYEE NUMBER := 0;

TOTAL\_SALARY NUMBER := 0;

MINIMUM\_SALARY NUMBER := 0;

CURSOR STAFF IS

SELECT A.STAFF\_NAME AS S\_NAME ,A.STAFF\_SAL AS S\_SALARY,B.DESIGN\_NAME AS D\_NAME

FROM STAFF\_MASTERS A,DESIGNATION\_MASTERS B

WHERE DEPT\_CODE = V\_DEPT\_CODE

AND A.DESIGN\_CODE = B.DESIGN\_CODE;

BEGIN

SELECT MAX(STAFF\_SAL)

INTO MINIMUM\_SALARY

FROM STAFF\_MASTERS

WHERE DEPT\_CODE = V\_DEPT\_CODE;

FOR I IN STAFF

LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee Name : '||I.S\_NAME);

DBMS\_OUTPUT.PUT\_LINE('Employee Job : '||I.D\_NAME);

DBMS\_OUTPUT.PUT\_LINE('Employee Salary : '||I.S\_SALARY);

DBMS\_OUTPUT.PUT\_LINE('Employee Commission : ');

DBMS\_OUTPUT.PUT\_LINE('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');

TOTAL\_EMPLOYEE := TOTAL\_EMPLOYEE + 1;

TOTAL\_SALARY := TOTAL\_SALARY + I.S\_SALARY;

IF I.S\_SALARY < MINIMUM\_SALARY THEN

MINIMUM\_SALARY := I.S\_SALARY;

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Number of Employees : '||TOTAL\_EMPLOYEE);

DBMS\_OUTPUT.PUT\_LINE('Total Salary : '||TOTAL\_SALARY);

DBMS\_OUTPUT.PUT\_LINE('Minimum Salary : '||MINIMUM\_SALARY);

DBMS\_OUTPUT.PUT\_LINE('Average Salary : '||ROUND((TOTAL\_SALARY/TOTAL\_EMPLOYEE),2));

DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------');

END DETAILS;

4.15: Write a query to view the list of all procedures ,functions and packages from the

Data Dictionary.DECLARE

CURSOR PACKAGE1 IS

SELECT OBJECT\_NAME,OBJECT\_TYPE

FROM ALL\_OBJECTS

WHERE OWNER = 'SYS'

AND OBJECT\_TYPE IN ('PACKAGE','PACKAGE BODY');

CURSOR PROCEDURE1 IS

SELECT OBJECT\_NAME,OBJECT\_TYPE

FROM ALL\_OBJECTS

WHERE OWNER = 'SYS'

AND OBJECT\_TYPE IN ('PROCEDURE');

CURSOR FUCNTION1 IS

SELECT OBJECT\_NAME,OBJECT\_TYPE

FROM ALL\_OBJECTS

WHERE OWNER = 'SYS'

AND OBJECT\_TYPE IN ('FUNCTION');

BEGIN

FOR I IN PACKAGE1

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.OBJECT\_NAME||CHR(9)||I.OBJECT\_TYPE);

END LOOP;

FOR I IN PROCEDURE1

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.OBJECT\_NAME||CHR(9)||I.OBJECT\_TYPE);

END LOOP;

FOR I IN FUCNTION1

LOOP

DBMS\_OUTPUT.PUT\_LINE(I.OBJECT\_NAME||CHR(9)||I.OBJECT\_TYPE);

END LOOP;

END;

Lab5

5.1 Create appropriate Test Cases for the case study followed up by Self/Peer to Peer

Review and close any defects for the same.CREATE TABLE CUSTOMER\_MASTERS (

CUST\_ID NUMBER(6) NOT NULL,

CUST\_NAME VARCHAR2(20) NOT NULL,

ADDRESS VARCHAR2(50),

DATE\_OF\_ACC\_CREATION DATE,

CUSTOMER\_TYPE CHAR(3),

CONSTRAINT CUST\_TYPE CHECK (CUSTOMER\_TYPE='IND' OR CUSTOMER\_TYPE = 'NRI'));

CREATE TABLE ACCOUNT\_MASTERS (

ACCOUNT\_NUMBER NUMBER(6) NOT NULL,

CUST\_ID NUMBER(6),

ACCOUNT\_TYPE CHAR(3),

LEDGER\_BALANCE NUMBER(10),

CONSTRAINT ACC\_TYPE CHECK (ACCOUNT\_TYPE='SAV' OR ACCOUNT\_TYPE='SAL'),

CONSTRAINT MIN\_AMOUNT CHECK((ACCOUNT\_TYPE='SAL') OR (ACCOUNT\_TYPE='SAV' AND LEDGER\_BALANCE > 5000));

CREATE TABLE TRANSACTION\_MASTERS (

TRANSACTION\_ID NUMBER(6) NOT NULL,

ACCOUNT\_NUMBER NUMBER(6),

DATE\_OF\_TRANSACTION DATE,

FROM\_ACCOUNT\_NUMBER NUMBER(6) NOT NULL,

TO\_ACCOUNT\_NUMBER NUMBER(6) NOT NULL,

AMOUNT NUMBER(10) NOT NULL,

TRANSACTION\_TYPE CHAR(2) NOT NULL,

CONSTRAINT TRAN\_TYPE CHECK ( TRANSACTION\_TYPE = 'CR' OR TRANSACTION\_TYPE = 'DB'));

5.2Write a procedure to accept customer name, address, and customer type and account

type. Insert the details into the respective tables.

5.3. Write a procedure to accept customer id, amount and the account number to which

the customer requires to transfer money. Following validations need to be done

 Customer id should be valid

 From account number should belong to that customer

 To account number cannot be null but can be an account which need not exist in

account masters (some other account)

 Adequate balance needs to be available for debit

CREATE SEQUENCE CUST\_ID\_SEQ START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE ACC\_NUM\_SEQ START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE PACKAGE CUSTOMER\_PACKAGE

AS

PROCEDURE INSERT\_CUSTOMER(C\_NAME IN VARCHAR2,C\_ADDRESS IN VARCHAR2,C\_TYPE IN CHAR,A\_TYPE CHAR);

PROCEDURE TRANSFER\_MONEY(C\_ID IN NUMBER,C\_AMOUNT IN NUMBER, A\_NUMBER IN NUMBER);

END;

CREATE OR REPLACE PACKAGE BODY CUSTOMER\_PACKAGE

AS

CUST\_NO NUMBER;

PROCEDURE INSERT\_CUSTOMER(C\_NAME IN VARCHAR2,C\_ADDRESS IN VARCHAR2,C\_TYPE IN CHAR,A\_TYPE CHAR)

IS

CHECK\_CONSTRAINT\_VIOLATED EXCEPTION;

PRAGMA EXCEPTION\_INIT(CHECK\_CONSTRAINT\_VIOLATED, -2290);

BEGIN

SELECT CUST\_ID\_SEQ.NEXTVAL

INTO CUST\_NO

FROM DUAL;

INSERT INTO CUSTOMER\_MASTERS VALUES(CUST\_NO,C\_NAME,C\_ADDRESS,SYSDATE,C\_TYPE);

INSERT INTO ACCOUNT\_MASTERS VALUES(ACC\_NUM\_SEQ.NEXTVAL,CUST\_NO,A\_TYPE,5001);

EXCEPTION

WHEN CHECK\_CONSTRAINT\_VIOLATED THEN

DBMS\_OUTPUT.PUT\_LINE('Insert failed due to check constraint violation.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Some exception has occurred.' || SQLCODE || ' ' || SQLERRM);

END INSERT\_CUSTOMER;

PROCEDURE TRANSFER\_MONEY(C\_ID IN NUMBER,C\_AMOUNT IN NUMBER, A\_NUMBER IN NUMBER)

IS

INVALID\_CUSTOMER\_ID EXCEPTION;

FROM\_ACCOUNT EXCEPTION;

INADEQUATE\_BALANCE EXCEPTION;

COUNT\_CUST NUMBER;

BEGIN

SELECT COUNT(\*)

INTO COUNT\_CUST

FROM ACCOUNT\_MASTERS

WHERE CUST\_ID = C\_ID;

IF COUNT\_CUST != 1 THEN

RAISE INVALID\_CUSTOMER\_ID;

ELSE

SELECT ACCOUNT\_NUMBER

INTO COUNT\_CUST

FROM ACCOUNT\_MASTERS

WHERE CUST\_ID = C\_ID;

IF COUNT\_CUST != A\_NUMBER THEN

RAISE FROM\_ACCOUNT;

ELSE

SELECT LEDGER\_BALANCE

INTO COUNT\_CUST

FROM ACCOUNT\_MASTERS

WHERE CUST\_ID = C\_ID

AND ACCOUNT\_NUMBER = A\_NUMBER;

IF (COUNT\_CUST-C\_AMOUNT) < 5000 THEN

RAISE INADEQUATE\_BALANCE;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Trasnfer was successful');

END IF;

END IF;

END IF;

EXCEPTION

WHEN INVALID\_CUSTOMER\_ID THEN

DBMS\_OUTPUT.PUT\_LINE('Invalid Customer Id!!');

WHEN FROM\_ACCOUNT THEN

DBMS\_OUTPUT.PUT\_LINE('Your account is not present!!');

WHEN INADEQUATE\_BALANCE THEN

DBMS\_OUTPUT.PUT\_LINE('You do not have enough balance to transfer!!');

END TRANSFER\_MONEY;

END CUSTOMER\_PACKAGE;

Lab6

6.1.Create appropriate Test Cases for the case study followed up by Self/Peer to Peer

Review and close any defects for the same.

create table employee (

empno number(5) not null,

ename varchar(40) not null,

city varchar(20) default 'Mumbai',

designation varchar(15) not null,

salary number(6,2) not null,

primary key(empno));

6.2

Recreate the procedure (run\_task) which is more efficient in performing the same.

CREATE OR REPLACE PROCEDURE run\_task (task\_number\_in IN INTEGER)

AS

BEGIN

CASE task\_number\_in

WHEN 1 THEN

--add\_emp;

--should add new emps in myEmp.

--EmpNo should be inserted through Sequence.

--All other data to be taken as parameters.Default location is Mumbai.

DBMS\_OUTPUT.PUT\_LINE('IN 1');

WHEN 2 THEN

--raise\_sal;

--should modify salary of an existing emp.

--should take new salary and empno as input parameters

--Should handle exception in case empno not found

--upper limit of rasing salary is 30%. should raise exception appropriately

DBMS\_OUTPUT.PUT\_LINE('IN 2');

WHEN 3 THEN

--remove\_emp;

--should remove an existing emp

--should take empno as parameter

--Handle exception if empno not available

DBMS\_OUTPUT.PUT\_LINE('IN 3');

END CASE;

END run\_task;

6.3:Also, create relevant procedures (add\_emp , raise\_sal ,remove\_emp)

with relevant logic (read comments)to verify the same.

--SEQUQNCE FOR GENERATING EMPNO

CREATE SEQUENCE emp\_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE PROCEDURE run\_task (task\_number\_in IN INTEGER)

AS

PROCEDURE addemp(E\_NAME IN VARCHAR,E\_CITY IN VARCHAR,E\_DESIGNATION IN VARCHAR,E\_SALARY IN NUMBER)

IS

BEGIN

INSERT INTO EMPLOYEE (EMPNO,ENAME,CITY,DESIGNATION,SALARY) VALUES (EMP\_SEQ.NEXTVAL,E\_NAME,E\_CITY,E\_DESIGNATION,E\_SALARY);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Some constraint was violated!!Please check you insertion values.');

END;

PROCEDURE raise\_sal(e\_no IN NUMBER,newsal IN NUMBER)

IS

EMPLOYEE\_NOT\_FOUND EXCEPTION;

RAISE\_TOO\_HIGH EXCEPTION;

E\_COUNT NUMBER;

BEGIN

SELECT COUNT(\*)

INTO E\_COUNT

FROM EMPLOYEE

WHERE EMPNO = E\_NO;

IF E\_COUNT != 1 THEN

RAISE EMPLOYEE\_NOT\_FOUND;

ELSE

SELECT SALARY

INTO E\_COUNT

FROM EMPLOYEE

WHERE EMPNO = E\_NO;

E\_COUNT := E\_COUNT \* 1.3;

IF newsal > E\_COUNT THEN

RAISE RAISE\_TOO\_HIGH;

ELSE

UPDATE EMPLOYEE SET SALARY = newsal WHERE EMPNO = E\_NO;

END IF;

END IF;

EXCEPTION

WHEN EMPLOYEE\_NOT\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee was not found in the table.');

WHEN RAISE\_TOO\_HIGH THEN

DBMS\_OUTPUT.PUT\_LINE('Raise is too high, maximum raise allowed is 30% of the current salary');

END;

PROCEDURE remove\_emp(e\_no IN NUMBER)

IS

EMPLOYEE\_NOT\_FOUND EXCEPTION;

E\_COUNT NUMBER;

BEGIN

SELECT COUNT(\*)

INTO E\_COUNT

FROM EMPLOYEE

WHERE EMPNO = E\_NO;

IF E\_COUNT != 1 THEN

RAISE EMPLOYEE\_NOT\_FOUND;

ELSE

DELETE FROM EMPLOYEE WHERE EMPNO = e\_no;

END IF;

EXCEPTION

WHEN EMPLOYEE\_NOT\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee was not found in the table.');

END;

BEGIN

--Add the parameters to the procedures

CASE task\_number\_in

WHEN 1 THEN

addemp('Anwesha','Kolkata','Analyst',2200.0);

WHEN 2 THEN

raise\_sal(5,2300.0);

WHEN 3 THEN

remove\_emp(5);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Please enter the correct code');

END CASE;

END run\_task;

6.4 Extend the above implementation using Packages

CREATE OR REPLACE PACKAGE Employee\_Package

AS

PROCEDURE run\_task (task\_number\_in IN INTEGER);

PROCEDURE addemp(E\_NAME IN VARCHAR,E\_CITY IN VARCHAR,E\_DESIGNATION IN VARCHAR,E\_SALARY IN NUMBER);

PROCEDURE raise\_sal(e\_no IN NUMBER,newsal IN NUMBER);

PROCEDURE remove\_emp(e\_no IN NUMBER);

END Employee\_Package;

CREATE OR REPLACE PACKAGE BODY Employee\_Package AS

PROCEDURE run\_task(task\_number\_in IN INTEGER)

IS

BEGIN

--Add the parameters to the procedures

CASE task\_number\_in

WHEN 1 THEN

Employee\_Package.addemp('Anwesha','Kolkata','Analyst',2200.0);

WHEN 2 THEN

Employee\_Package.raise\_sal(5,2300.0);

WHEN 3 THEN

Employee\_Package.remove\_emp(5);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Please enter the correct code');

END CASE;

END run\_task;

PROCEDURE addemp(E\_NAME IN VARCHAR,E\_CITY IN VARCHAR,E\_DESIGNATION IN VARCHAR,E\_SALARY IN NUMBER)

IS

BEGIN

INSERT INTO EMPLOYEE (EMPNO,ENAME,CITY,DESIGNATION,SALARY) VALUES (EMP\_SEQ.NEXTVAL,E\_NAME,E\_CITY,E\_DESIGNATION,E\_SALARY);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Some constraint was violated!!Please check you insertion values.');

END addemp;

PROCEDURE raise\_sal(e\_no IN NUMBER,newsal IN NUMBER)

IS

EMPLOYEE\_NOT\_FOUND EXCEPTION;

RAISE\_TOO\_HIGH EXCEPTION;

E\_COUNT NUMBER;

BEGIN

SELECT COUNT(\*)

INTO E\_COUNT

FROM EMPLOYEE

WHERE EMPNO = E\_NO;

IF E\_COUNT != 1 THEN

RAISE EMPLOYEE\_NOT\_FOUND;

ELSE

SELECT SALARY

INTO E\_COUNT

FROM EMPLOYEE

WHERE EMPNO = E\_NO;

E\_COUNT := E\_COUNT \* 1.3;

IF newsal > E\_COUNT THEN

RAISE RAISE\_TOO\_HIGH;

ELSE

UPDATE EMPLOYEE SET SALARY = newsal WHERE EMPNO = E\_NO;

END IF;

END IF;

EXCEPTION

WHEN EMPLOYEE\_NOT\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee was not found in the table.');

WHEN RAISE\_TOO\_HIGH THEN

DBMS\_OUTPUT.PUT\_LINE('Raise is too high, maximum raise allowed is 30% of the current salary');

END raise\_sal;

PROCEDURE remove\_emp(e\_no IN NUMBER)

IS

EMPLOYEE\_NOT\_FOUND EXCEPTION;

E\_COUNT NUMBER;

BEGIN

SELECT COUNT(\*)

INTO E\_COUNT

FROM EMPLOYEE

WHERE EMPNO = E\_NO;

IF E\_COUNT != 1 THEN

RAISE EMPLOYEE\_NOT\_FOUND;

ELSE

DELETE FROM EMPLOYEE WHERE EMPNO = e\_no;

END IF;

EXCEPTION

WHEN EMPLOYEE\_NOT\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee was not found in the table.');

END remove\_emp;

END Employee\_Package;

.

/

lab7

7.1

Declare

TextHandler UTL\_File.File\_type;

WriteMessage Varchar2(400);

ReadMessage Varchar2(400);

line VARCHAR2(80);

Begin

TextHandler:=UTL\_File.Fopen('d:\Sample','TestFile.txt','W');

WriteMessage:='FOPEN is a Function, which returns the value of type

File\_Type \n UTL\_file.PUT\_LINE is a procedure in UTL\_FILE, which write a line

to a file,Specific line terminator will be appended \n';

UTL\_file.Putf(Texthandler,writemessage);

UTL\_File.Fflush(Texthandler);

UTL\_File.Fclose(Texthandler);

Texthandler := UTL\_FILE.FOPEN('d:\sample', 'TestFile.txt', 'R');

LOOP

UTL\_FILE.GET\_LINE(Texthandler, line);

dbms\_output.put\_line(line);

END LOOP;

UTL\_File.Fflush(Texthandler);

UTL\_File.Fclose(Texthandler);

End;

/

7.2

Declare

TextHandler UTL\_File.File\_type;

WriteMessage Varchar2(400);

ReadMessage Varchar2(400);

line VARCHAR2(80);

Begin

TextHandler:=UTL\_File.Fopen('d:\Sample','TestFile.txt','W');

WriteMessage:='FOPEN is a Function, which returns the value of type

File\_Type \n UTL\_file.PUT\_LINE is a procedure in UTL\_FILE, which write a line

to a file,Specific line terminator will be appended \n';

UTL\_file.Putf(Texthandler,writemessage);

UTL\_File.Fflush(Texthandler);

UTL\_File.Fclose(Texthandler);

Texthandler := UTL\_FILE.FOPEN('d:\sample', 'TestFile.txt', 'R');

LOOP

UTL\_FILE.GET\_LINE(Texthandler, line);

dbms\_output.put\_line(line);

END LOOP;

UTL\_File.Fflush(Texthandler);

UTL\_File.Fclose(Texthandler);

EXCEPTION

WHEN utl\_file.invalid\_path THEN

raise\_application\_error(-20000, 'ERROR: Invalid path for file.');

WHEN no\_data\_found THEN

dbms\_output.put\_line('No more data to read');

End;

/

7.3

Declare

TextHandler UTL\_File.File\_type;

WriteMessage Varchar2(400);

ReadMessage Varchar2(400);

line VARCHAR2(80);

Begin

TextHandler:=UTL\_File.Fopen('TMP','TestFile.txt','W');

WriteMessage:='FOPEN is a Function, which returns the value of type

File\_Type \n UTL\_file.PUT\_LINE is a procedure in UTL\_FILE, which write a line

to a file,Specific line terminator will be appended \n';

UTL\_file.Putf(Texthandler,writemessage);

UTL\_File.Fflush(Texthandler);

UTL\_File.Fclose(Texthandler);

Texthandler := UTL\_FILE.FOPEN('TMP', 'TestFile.txt', 'R');

LOOP

UTL\_FILE.GET\_LINE(Texthandler, line);

dbms\_output.put\_line(line);

END LOOP;

UTL\_File.Fflush(Texthandler);

UTL\_File.Fclose(Texthandler);

EXCEPTION

WHEN utl\_file.invalid\_path THEN

raise\_application\_error(-20000, 'ERROR: Invalid path for file.');

WHEN no\_data\_found THEN

dbms\_output.put\_line('No more data to read');

commit;

End;

/

Lab8

8.1: Using Multiple Spacing Techniques

Suppose you have more than one column in your ORDER BY clause and wish to insert

space when each column’s value changes. Each BREAK command you enter replaces

the previous one.

CLEAR BREAKS

CLEAR COLUMNS

CLEAR COMPUTES

COLUMN DEPT\_CODE HEADING DEPT\_ID

COLUMN DESIGN\_NAME HEADING JOB\_ID

COLUMN STAFF\_NAME HEADING ENAME

COLUMN STAFF\_SAL HEADING SALARY

BREAK ON DEPT\_CODE SKIP 1 ON DESIGN\_NAME SKIP 1 DUP

COMPUTE AVG OF STAFF\_SAL ON DESIGN\_NAME

COMPUTE SUM OF STAFF\_SAL ON DEPT\_CODE

SELECT A.DEPT\_CODE,B.DESIGN\_NAME,A.STAFF\_NAME,A.STAFF\_SAL

FROM STAFF\_MASTERS A

JOIN DESIGNATION\_MASTERS B

ON A.DESIGN\_CODE = B.DESIGN\_CODE

ORDER BY A.DEPT\_CODE,B.DESIGN\_NAME;

8.2.Computing and Printing Subtotals

CLEAR BREAKS

CLEAR COLUMNS

CLEAR COMPUTES

TTITLE OFF

BTITLE OFF

SET PAGESIZE 20

TTITLE LEFT 'SALES DEPARTMENT PERSONNEL REPORT' SKIP 2 - LEFT 'PERFECT WIDGETS' SKIP 1 RIGHT FORMAT DD-MON-YYYY \_DATE SKIP 1 RIGHT 'PAGE:' FORMAT 9 SQL.PNO SKIP 2

BTITLE LEFT 'COMPANY CONFIDENTIAL'

COLUMN DEPT\_CODE HEADING DEPARTMENT\_ID

COLUMN STAFF\_SAL HEADING SALARY

COLUMN STAFF\_NAME HEADING LAST\_NAME

BREAK ON REPORT SKIP 2

COMPUTE SUM LABEL '' OF STAFF\_SAL ON REPORT

SELECT DEPT\_CODE,STAFF\_NAME,STAFF\_SAL FROM STAFF\_MASTERS ORDER BY DEPT\_CODE,STAFF\_NAME;

CLEAR BREAKS

CLEAR COLUMNS

CLEAR COMPUTES

TTITLE OFF

BTITLE OFF

COLUMN DEPT\_CODE HEADING 'Dept|No.'

COLUMN DESIGN\_NAME HEADING 'Job|Name'

COLUMN COUNT1 HEADING 'No. of|Employees'

COLUMN AVERAGE1 HEADING 'Average|Salary/Job' FORMAT $999,999.99

BREAK ON DEPT\_CODE SKIP 1 ON REPORT

COMPUTE SUM LABEL 'TOTAL' OF AVERAGE1 ON REPORT

COMPUTE SUM LABEL 'No. of Employees' OF COUNT1 ON REPORT

SELECT A.DEPT\_CODE,B.DESIGN\_NAME,COUNT(STAFF\_NAME) AS COUNT1, AVG(STAFF\_SAL) AS AVERAGE1

FROM STAFF\_MASTERS A

JOIN DESIGNATION\_MASTERS B

ON A.DESIGN\_CODE = B.DESIGN\_CODE

GROUP BY A.DEPT\_CODE,B.DESIGN\_NAME

ORDER BY A.DEPT\_CODE,B.DESIGN\_NAME;