PL / SQL Assignment

1.FUNCTIONS:

1. Write a Function to generate Fibonacci series
2. Write a function to generate n factorial number
3. Write a function to generate sum of first n number
4. Write a function to check the given no. is prime or not
5. Write a function which print the sum of all even number between 1 to 100
6. Write a function which print the sum of all odd numbers between 1 to 100
7. Write a function which accept i/o as number & print Whether it is Even or Odd
8. Write function to calculate income tax , pass basic (per month) as input
   DA = 12% of Basic, HRA = 10% of Basic, TA = 15 % of Basic, PF = 8% of Basic.
   Income tax on Annual Income slabs are as follows upto 1 Lack – Nil , 100000 to 150000 – 10%, 150000 to 250000 – 15%, < 250000 – 20%

2.PROCEDURE :

Write a procedure in oracle for

1. For an employee database raise the salary by 5 %

For all Manager assume { emp(emp_no , name , designation , salary)

2. To list down all manager
3. Write a procedure to set returndate as sysdate. Calculate fine if return date is more than 3 days. Fine will be Rs. 10 par extra day. Use rollno as input to procedure. Create table as (rollno, name, branch, bookno, issuedate, returndate, fine) During data insert returndate and fine should be null.

3 .PL/SQL BLOCK

3. Employee (emp_no , ename , job, mgr , date, sal)

1. Write a PL/SQL block to print the no. of employee joined in month of December
2. Write a PL/SQL block with curser to print the information of first five highest salary earner
3. Pass emp_no as an argument to procedure and modify salary of that employee.
4. Write cursor program which display name, bookno, issuedate of the rows where returndate is null.
5. Write a PL/SQL block to find grade of minimum 10 students.
6. Write a PL/SQL block to find area of circle,trangle,rectangle,circlemsquare take input from user for choice.
7. Write a PL/SQL block to find even and odd nos from 1 to 50.
8. Write a PL/SQL block to find prime nos from 1 to 50.
9. Write a PL/SQL block to Find name of employees having salary grater than 5000.
10. Write PL/SQL block to give 20 % comm. to only.

4 Emp

Use given database (empno , ename , job, manager, h-date , sal, deptno)

Dept(deptno, name , designation)

1. Pass a year to procedure and print the information of employee who were joined before

this year

2. Pass the name to a function and function will return the month in which that employee was hired
3. Write a delete Trigger for dept such that when dept is deleted , then the respective information from other tables is also deleted

5 EMBEDDED – SQL

Emp(emp_no , emp_name , address , salary)

1. Write program in embedded SQL to insert the data into the table and searching data from database
2. Write a PL/SQL block to delete a record from a table
3. write a PL/SQL block to update a record from a table

6. TRIGGER

1. Write an insert , update & delete triggers for student for student database
2. Write a trigger which does not allow inserting on Sunday.
3. Write trigger which will display total number of rows when new data is inserted in the table
4. Audit trigger.

5. Write a trigger which will performs

   If insert then display total number of rows in database before insert

   If updating then should not allow sal > 9000

   If deleting then display message that row is deleted.

6. Write a trigger with raise_exception error when insert operation is done then salary should not be less than 3000.

7 .Dynamic sql.

TRY IT……………………………………

1. select ename, initcap(ename) from emp;

2. select ename, length(ename) from emp;

3. select ename, lower(ename) from emp;

4. select ename, upper(ename) from emp;

5. select to_number('12')+ to_number('10') from dual;

6. select to_date('19 –dec-2006')+1 from dual;

7. select ename, hiredate, add_months(hiredate,12) from emp;

8. select ename,hiredate,months_between(sysdate,hiredate) from
emp;

9. select ename,hiredate,months_between(sysdate,hiredate)/12 from emp;

10.select name, hiredate, round(months_between(sysdate,hiredate)/12) from emp;

11. select ename, sal, nvl(comm, 0) from emp;

VIEWS

Syntax

```
        CREATE VIEW viewname AS
        SELECT columnname, columnname
        FROM tablename
        WHERE columnname=expression list;
```

Example :

1. CREATE VIEW vwemp AS SELECT empno, ename, deptno, sal FROM emp;

2. CREATE or replace VIEW vwemp AS SELECT a.empno, a.ename, a.deptno,

a.sal, b.dname FROM emp a, dept b where a.deptno=b.deptno;

insert into table / select from view

insert or update view /select from table

delete view <view name>;

Create table using any existing table

CREATE TABLE .. AS SELECT command

Example : create table remp as select * from emp where ename = 'WARD';

```
    DELETE FROM REMP;
    TRUNCATE TABLE REMP;
```

Will the outputs of the above two commands differ?

Both will result in deleting all the rows in the table EMP.

What does the following query do?

SELECT SAL+COMM FROM EMP;

Why output is not proper ? Sol……..

SELECT SAL + NVL(COMM,0) FROM EMP;

Concatenation

SELECT empid, lastname||', '||firstname full_name, NVL(spouse,'unmarried') spouse,
FROM HRAPP.EMP;

How to select required columns online?

```
    SQL> select &columns from emp;
    Enter value for columns: deptno,ename,mgr
    old   1: select &columns from emp
    new   1: select deptno,ename,mgr from emp
```

How to change the *where clause* online ?

```
    SQL> select ename, deptno, mgr from emp where &where_clouse;
    Enter value for where_clouse: deptno =10
    old   1: select ename, deptno, mgr from emp where &where_clouse
    new   1: select ename, deptno, mgr from emp where deptno =10
```

PL/SQL------------

```
CREATE TABLE T1( e INTEGER, f INTEGER );

INSERT INTO T1 VALUES(1, 3);

INSERT INTO T1 VALUES(2, 4);
```

PL/SQL block

```
DECLARE
   a NUMBER;
   b NUMBER;
BEGIN
   SELECT e,f INTO a,b FROM T1 WHERE e>1;
   INSERT INTO T1 VALUES(b,a);
END;
```

Control Flow in PL/SQL

PL/SQL allows you to branch and create loops in a fairly familiar way.
An IF statement looks like:

```
IF <condition>
THEN <statement_list>
ELSE <statement_list>
END IF;
```

The ELSE part is optional. If you want a multiway branch, use:

```
IF <condition_1> THEN ...
ELSIF <condition_2> THEN ...
... ...
ELSIF <condition_n> THEN ...
ELSE ...
END IF;
```

Example: This is slightly modified from the previous one, where now we only do the insertion if the second component is 1. If not, we first add 10 to each component and then insert:

```
DECLARE

   a NUMBER;

   b NUMBER;

BEGIN

   SELECT e, f INTO a, b FROM T1 WHERE e>1;

   IF b=1 THEN

      INSERT INTO T1 VALUES (b, a);

   ELSE

      INSERT INTO T1 VALUES (b+10,a+10);

   END IF;

END;
```

1. Update table emp set sal of employee sandeep to 777 if his record is not there then insert his details.

PL/SQL

```
declare
ename1 varchar2(13) := 'sandeep';
sal1 EMP.SAL% TYPE :=777;
begin
update emp set sal = sal1 where ename ='sandeep';
if sql %notfound then
insert into emp (empno,ename,sal) values(3343,'sandeep',888);
end if;
end;
```

2. Cerate new table and insert values into it with loop

```
create table test (rollno number(4), sname varchar2(10));
```

PL/SQL

```
begin
for v_loopcounter in 1..50 loop
insert into try(no)
values (v_loopcounter);
end loop;
end;
```

3. Get empno from user as input the get sal of said employee if less than 3000 then add 4000 to salary

PL/SQL

```
 declare
 mempno number(4);
 add number(8,2) := 4000.00;
 msal number(7,2);
 begin
 mempno :=&mempno;
 select sal into msal from emp where empno = mempno;
 if msal < 3000 then
 update emp set sal = sal+add where empno = mempno;
 end if;
 end;
```

----

get the salary of Mahesh if record not found the give message
(Exception Handle)
Set serveroutput on

```
Declare
ename1 emp.ename % type;
sal1 EMP.SAL% TYPE :=777;
begin
ename1 :=’&ename1’;
select sal into sal1 from emp where ename =ename1;
dbms_output.put_line (to_char(sal1));
exception
when no_data_found then
dbms_output.put_line(‘does not exit’);
end;
```

```
declare
empno1 emp.empno%type :=&empno;
csal emp.sal%type ;
ccomm emp.comm%type;
begin
select sal into csal from emp where empno = empno1;
if csal > 4000 then
update emp set comm = csal*10/100;
else
update emp set comm = csal*20/100;
end if;
end;
```

```
          Set serveroutput on
          FOR LOOP

Declare
I integer;
Begin
For I in 1..10
Loop
          Dbms_output.put_line('value of I : '|| to_char(i));
End loop;
End;
```

Date 14 Sept

```
 Declare
 inc emp.comm %type;
 asal  emp.sal %type;
  name emp.ename%type;
 Begin
 name := '&name';
 Select sal into asal from emp where ename = name;
 If  asal > 5000 then
    inc := asal * 10 / 100;
 elsif
  asal > 3000 then
   inc := asal * 20 / 100;
 elsif
  asal >2000 then
  inc := asal * 30 / 100;
 else
   inc := asal * 40 / 100;
 end if;
 update emp set comm = inc where ename = name;
 End;
```

WHILE LOOP

SYNTAX

```
          WHILE <CONDITION>
          LOOP

                    ……..
                    ……..
          END LOOP;
```

Example :

```
declare
        ctr number(2) := 1;
        value number(2);
        terms number(2);
        pdt number;
begin
 value := &value;
terms := &terms;
dbms_output.put_line(‘multi’);
while ctr <= terms
loop
        pdt := value*ctr;
        dbms_output.put_line(to_char(value) || ‘*’|| to_char(ctr) || ‘=’||to_char(pdt));
ctr :=ctr +1;
end loop;
end;
```

Exception Handle

get the salary of Mahesh if record not found the give message

Set serveroutput on

Example :

```
Declare
ename1 emp.ename % type;
sal1 EMP.SAL% TYPE :=777;
begin
ename1 :=’&ename1’;
select sal into sal1 from emp where ename =ename1;
dbms_output.put_line (to_char(sal1));
exception
when no_data_found then
dbms_output.put_line(‘does not exit’);
end;
```

Cursors

A *cursor* is a variable that runs through the tuples of some relation. This relation can be a stored table, or it can be the answer to some query. By fetching into the cursor each tuple of the relation, we can write a program to read and process the value of each such tuple. If the relation is stored, we can also update or delete the tuple at the current cursor position.

COMMANDS TO CONTROL ---------- OPEN , FETCH , CLOSE

OPEN ---- initialize the cursor with open
FETCH ---- to retrieve the first row, you can execute fetch repeatedly until all rows have been retrieved.
CLOSE ---- release the curor

Syntax -----declaration of cursor

CURSOR <CRSOR NAME> IS <SELECT STATEMENT> ;

Cursor *c1* is select empno, ename from emp;

Open *c1*

Fetch *c1* into <list of variable>

Fetch *c1* into <rowtype var>

Example : <list of variable>

```
Declare
       Cursor empcur is
       Select empno, ename, sal from emp;
       a emp.empno %type;
       b emp.ename %type;
       c emp.sal %type;
begin
       open empcur;
loop
        fetch empcur into a,b,c;
        exit when empcur % notfound;
        if c >= 2000 then
  dbms_output.put_line (' empno : ' || a || '  ename :   '|| b ||'
sal : '||c);
       end if ;
end loop;
close empcur;
end;
```

Example : Cursor with row type <row type>

```
Declare
       Cursor empcur is
       Select * from emp;
       a emp % rowtype;
```

```
begin
       open empcur;
loop
        fetch empcur into a;
        exit when empcur % notfound;
        if a.deptno = 20 then
  dbms_output.put_line (' empno : ' || a.empno || '  dept :
'|| a.deptno ||'  mgr : '||a.mgr);
       end if ;
end loop;
close empcur;
end;
```

<u>Example No. Cursor and For Loop ……</u>

```
Declare
 cnt number;
 cursor c2 is select deptno from emp;
 begin
open c2;   cnt :=0;
 for z in c2 loop
       cnt := cnt+1;
 end loop;
 dbms_output.put_line(to_char(cnt));
close c2;   end;
```

Cursor Attributes

%FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT

CURSORNAME%attribute

%FOUND

```
declare
```

```
cursor c2 is   select dname from dept1 where deptno = 10;
  flag number;
        z c2%rowtype;
begin
 open c2;
fetch c2 into z;
if c2%found then
 flag := 00000;
else
 flag :=11111;
end if;
dbms_output.put_line (to_char(flag));
close c2;
end;
```

% FOUND WITH WHILE …….LOOP

```
declare
totrow number := 0;
cursor c2 is   select dname from dept ;
  flag number;
        z c2%rowtype;
begin
 open c2;
fetch c2 into z;
while c2%found loop
   totrow := totrow+1;
   fetch c2 into z;
end loop;
   close c2;
dbms_output.put_line ('total no of rows are '||to_char(totrow));
end;
```

%NOTFOUND

```
declare
cnt number := 0;
cursor c2 is
  select empno from emp ;
          z c2%rowtype;
begin
 open c2;
loop
            fetch c2 into z;
            exit when c2%notfound;
            cnt := cnt+1;
 end loop;
   close c2;
dbms_output.put_line ('total no of rows are '||to_char(cnt));
end;
```

%ROWCOUNT

```
declare
 cursor c_rowtype is
   select empno from emp ;
            z c_rowtype %rowtype;
 begin
  open c_rowtype;
 dbms_output.put_line (to_char(c_rowtype%rowcount));
 fetch c_rowtype into z;
 dbms_output.put_line (to_char(c_rowtype%rowcount));
 fetch c_rowtype into z;
 dbms_output.put_line (to_char(c_rowtype%rowcount));
 fetch c_rowtype into z;
 dbms_output.put_line (to_char(c_rowtype%rowcount));
 fetch c_rowtype into z;
  close c_rowtype;
 end;
```

<u>List names of persons from dept 10 or 20;</u>

```
declare
cursor c1 is
select ename from emp where deptno = 10 or deptno =20;
begin
for z in c1 loop
            dbms_output.put_line (z.ename);
end loop;
end;
```

<u>list names of staff working more than 25 years</u>

```
declare
cursor c1 is
select ename, hiredate from emp where round(months_between(sysdate,hiredate)/12) > 25;
begin
for z in c1 loop
dbms_output.put_line (z.ename || '  '||z.hiredate);
end loop;
end;
```

Transaction Control Statements
Transaction processing ---- 2 concepts –session and transaction

A session s created when any user connects to oracle. Session is discontinued when the user disconnects from oracle.

Cursors

A *cursor* is a variable that runs through the tuples of some relation. This relation can be a stored table, or it can be the answer to some query. By fetching into the cursor each tuple of the relation, we can write a program to read and process the value of each such tuple. If the relation is stored, we can also update or delete the tuple at the current cursor position.

The example below illustrates a cursor loop. It uses our example relation T1(e,f) whose tuples are pairs of integers. The program will delete every tuple whose first component is less than the second, and insert the reverse tuple into T1.

```
 DECLARE

    a T1.e%TYPE;

     b T1.f%TYPE;



     CURSOR T1Cursor IS

         SELECT e, f

         FROM T1

         WHERE e < f

         FOR UPDATE;

BEGIN
    OPEN T1Cursor;

   LOOP


     FETCH T1Cursor INTO a, b;



        EXIT WHEN T1Cursor%NOTFOUND;


        DELETE FROM T1 WHERE CURRENT OF T1Cursor;


        INSERT INTO T1 VALUES(b, a);

    END LOOP;

    CLOSE T1Cursor;
 END;
```

TO CREATE PROCEDURE AND RUN

```
CREATE TABLE T2 (a INTEGER, b char(10))


create or replace procedure addtuple1(i in number)as
begin
insert into t2 values(i,'xxx');
end addtuple1;
```

TO RUN PROCEDURE

```
begin
addtuple1(34);
end;
/
```

TO CREATE PROCEDURE AND RUN

```
create procedure addt2( x t2.a%type,y t2.b%type) as
begin
insert into t2 (a,b) values (x,y);
end addt2;
```

TO RUN

```
begin
addt2(55,'hhh');
end;


show errors procedure <procedure_name>;
sho err
```

TRIGGER S

1. Write a trigger to check the deptno during every insert to table dept.

```
create or replace trigger deptin
before insert on dept
for each row
declare
code number(4);
begin
if (:new.deptno) < 40
then
raise_application_error(-20110,'dept number must be greater than 100');
end if;
end;
```

2. Write a trigger , which prohibit delete and update operations to the user but allow insertion on Sunday.

```
Create or replace trigger Sunday
Before insert or update or delete on dept
Declare
Uname varchar2(20);
Today varchar2(20);
Begin
Select user into uname from dual;
If inserting then
Select to_char(sysdate,’day’) into today from dual;
Dbms_output.put_line (dsj ||today);
If today =  sunday then
Raise_application_error(-20111,’insert is not allowed on other days’);
End if;
End if;
If updating or deleting then
Raise_application_error(-20111,’Update and delete is not allowed’);
End if;
End;
```

Write a trigger,which checks empno before inserting any row in the table emp;
Length(Empno) > 3 digits and empno not less than 6000

```
Create or replace trigger empin
Before insert on emp1
For each row
Declare
empd number(4);
Begin
empd := :new.empno;
if inserting then
If (length (empd)) < 2
then
Raise_application_error(-20110,'length of empno is less than 4 digits');
End if;
End if;
End;
```

Write a trigger which will not accept the hiredate less than sysdate.

```
Create or replace trigger hiredt
Before insert on emp2
For each row
Declare
today date;
Begin
```

```
select sysdate into today from dual;
If :new.hiredate < today
then
raise_application_error(-20012,'joining date less than sysdate');
end if;
end;

Create or replace trigger emp_in
Before insert on emp1
For each row
Declare
in number(6);
Begin
If (length (:new.empno)<3 )
Then
Raise_application_error(-20110,’length of empno is less than 4 digits’);
End if;
If  in = :new.empno < 6000 then
Raise_application_error(-20110,’input less than 6000’)
End if;
End;
```

Write a trigger which will show the total number of entries present in the table before insert is done. Raise error if entries are more than 10

```
Create table title (title varchar2(20) not null,
Title_id varchar2(10) constraint tit_key primary key, rel_date date, rent
number(5));

insert into title values ('shan','s01','03-march-06',70);
insert into title values ('ddlj','s03','06-march-03',88);
insert into title values ('MPK','s07','06-march-06',100);
insert into title values ('kaal','s02','23-sep-05',100);
insert into title values ('dus','s055','23-sep-05',100);

Create or replace trigger stop_ins
before insert on title
For each row
Declare
 Row_ct number;
Begin
 Select count(*) into row_ct from title;
 Dbms_output.put_line('Number of Titles available :' || row_ct);
If row_ct = 10 then
Raise_application_error(-20140,’more than 10 entries are not allowed’)

End;
```

Dynamic SQL

```
DECLARE
  sql_stmt    VARCHAR2(200);
  plsql_block VARCHAR2(500);
  emp_id      NUMBER(4) := 7566;
  salary      NUMBER(7,2);
  dept_id     NUMBER(2) := 70;
  dept_name   VARCHAR2(14) := 'PERSONNEL';
  location    VARCHAR2(13) := 'DALLAS';
  emp_rec     emp%ROWTYPE;
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE bonus1 (id NUMBER, amt NUMBER)';

  sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';
  EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, location;

  sql_stmt := 'SELECT * FROM emp WHERE empno = :emp_id';
  EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING emp_id;


  sql_stmt := 'UPDATE emp SET sal = 4000 WHERE empno = :1
    RETURNING sal INTO :2';
  EXECUTE IMMEDIATE sql_stmt USING emp_id RETURNING INTO salary;

  EXECUTE IMMEDIATE 'DELETE FROM dept WHERE deptno = :num'
    USING dept_id;

  EXECUTE IMMEDIATE 'ALTER SESSION SET SQL_TRACE TRUE';
END;
```

```
CREATE PROCEDURE delete_rows (
  table_name IN VARCHAR2,
  condition IN VARCHAR2 DEFAULT NULL) AS
  where_clause VARCHAR2(100) := ' WHERE ' || condition;
BEGIN
  IF condition IS NULL THEN where_clause := NULL; END IF;
  EXECUTE IMMEDIATE 'DELETE FROM ' || table_name || where_clause;
END;

begin
delete_rows('video', 'video_id = 66');
end;
```

```
DECLARE
lv_sql VARCHAR2(500);
lv_emp_name VARCHAR2(50):
ln_emp_no NUMBER;
ln_salary NUMBER;
ln_manager NUMBER;
BEGIN
ly_sql:=;SELECT emp_name,emp_no,salary,manager FROM emp WHERE
emp_no=:empmo:;
EXECUTE IMMEDIATE lv_sql INTO lv_emp_name,ln_emp_no:ln_salary,ln_manager
USING 1001;
Dbms_output.put_line('Employee Name:‘||lv_emp_name);
Dbms_output.put_line('Employee Number:‘||ln_emp_no);
Dbms_output.put_line(‘Salary:'||ln_salaiy);
Dbms_output.put_line('Manager ID:‘||ln_manager);
END;
/
```