1. PL/SQL CODING FOR ADDITION OF TWO NUMBERS

```
SQL> declare
a number;
b number;
c number;
begin
a:=&a;
b:=&b;
c:=a+b;
dbms_output.put_line('sum of'||a||'and'||b||'is'||c);
end;
/
```

INPUT:

```
Enter value for a: 23
old 6: a:=&a;
new 6: a:=23;
Enter value for b: 12
old 7: b:=&b;
new 7: b:=12;
```

OUTPUT:

```
sum of23and12is35

PL/SQL procedure successfully completed.
```

2. PL/ SQL GENERAL SYNTAX FOR IF CONDITION:

```
SQL> DECLARE
 <VARIABLE DECLARATION>;
 BEGIN
 IF(CONDITION)THEN
 <EXECUTABLE STATEMENT >;
 END;
```

Coding for If Statement:

```
DECLARE
b number;
c number;
BEGIN
B:=10;
C:=20;
if(C>B) THEN
dbms_output.put_line('C is maximum');
end if;
```

```
end;
/
```

OUTPUT:

```
C is maximum

PL/SQL procedure successfully completed.
```

3. PL/ SQL GENERAL SYNTAX FOR IF AND ELSECONDITION:

```
SQL> DECLARE
 <VARIABLE DECLARATION>;
 BEGIN
 IF (TEST CONDITION) THEN
 <STATEMENTS>;
 ELSE
 <STATEMENTS>;
 ENDIF;
 END;
******************Less then or Greater Using IF ELSE **********************
SQL> declare
 n number;
 begin
 dbms_output. put_line('enter a number');
 n:=&number;
 if n<5 then
 dbms_output.put_line('entered number is less than 5');
 else
 dbms_output.put_line('entered number is greater than 5');

end if;
 end;
 /
```

Input

```
Enter value for number: 2
old 5: n:=&number;
new 5: n:=2;
```

Output:

```
entered number is less than 5

PL/SQL procedure successfully completed.
```

4.PL/SQL GENERAL SYNTAX FOR NESTED IF:

```
SQL> DECLARE
<VARIABLE DECLARATION>;
BEGIN
IF (TEST CONDITION) THEN
<STATEMENTS>;
ELSEIF (TEST CONDITION) THEN
<STATEMENTS>;
ELSE
<STATEMENTS>;
ENDIF;
END;
```

********** GREATEST OF THREE NUMBERS USING IF ELSEIF************

```
SQL> declare
 a number;
b number;
c number;
d number;
begin
 a:=&a;
 b:=&b;
  c:=&b;
 if(a>b)and(a>c) then
 dbms_output.put_line('A is maximum');
  elsif(b>a)and(b>c)then
 dbms_output.put_line('B is maximum');
 else
 dbms_output.put_line('C is maximum');
 end if;
 end;
  /
```

INPUT:

```
Enter value for a: 21
old 7: a:=&a;
new 7: a:=21;
Enter value for b: 12
old 8: b:=&b;
new 8: b:=12;
Enter value for b: 45
old 9: c:=&b;
new 9: c:=45;
```

OUTPUT:

```
C is maximum
```

PL/SQL procedure successfully completed.

5.PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

```
SQL> DECLARE
 <VARIABLE DECLARATION>;
 BEGIN
 LOOP
 <STATEMENT>;
 END LOOP;
 <EXECUTAVLE STATEMENT>;
 END;
***********SUMMATION OF ODD NUMBERS USING FOR LOOP***********
SQL> declare
n number;
sum1 number default 0;
endvalue number;
begin
endvalue:=&endvalue;
 n:=1;
for n in 1..endvalue
loop
 if mod(n,2)=1
then
sum1:=sum1+n;
end if;
 end loop;
dbms_output.put_line('sum ='||sum1);
end;
 /
```

```
INPUT:
Enter value for endvalue: 4
old 6: endvalue:=&endvalue;
new 6: endvalue:=4;

OUTPUT:
 sum =4
```

PL/SQL procedure successfully completed.

6.PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:

```
SQL> DECLARE
 <VARIABLE DECLARATION>;
```

```
BEGIN
WHILE <condition>
LOOP
<STATEMENT>;
END LOOP;
<EXECUTAVLE STATEMENT>;
END;
```

*********SUMMATION OF ODD NUMBERS USING WHILE LOOP**********

```
SQL> declare
n number;
sum1 number default 0;
endvalue number;
begin
endvalue:=&endvalue;
n:=1;
while(n<endvalue)
loop
sum1:=sum1+n;
n:=n+2;
end loop;

dbms_output.put_line('sum of odd no. bt 1 and' ||endvalue||'is'||sum1);
end;
/
```

INPUT:

```
Enter value for endvalue: 4
old 6: endvalue:=&endvalue;
new 6: endvalue:=4;
```

OUTPUT:

```
sum of odd no. bt 1 and4is4
PL/SQL procedure successfully completed.
```

7. TRIGGER

TYPE 1- TRIGGER AFTER UPDATE

```
SQL> CREATE OR REPLACE TRIGGER VIJAY
 AFTER UPDATE OR INSERT OR DELETE ON EMP
 FOR EACH ROW
 BEGIN
IF UPDATING THEN
 DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
ELSIF INSERTING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
ELSIF DELETING THEN
```

```
DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
END IF;
END;
/


Trigger created.
SQL> update emp set income =900 where empname='kumar';
TABLE IS UPDATED
1 row updated.
SQL> insert into emp values ( 4,'Chandru',700,250,80);
TABLE IS INSERTED
1 row created.
SQL> DELETE FROM EMP WHERE EMPID = 4;
TABLE IS DELETED
1 row deleted.
```

TYPE 2 - TRIGGER BEFORE UPDATE

```
-----------------------------------------------------
SQL> CREATE OR REPLACE TRIGGER VASANTH
BEFORE UPDATE OR INSERT OR DELETE ON EMPLOYEE
FOR EACH ROW
BEGIN
IF UPDATING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS UPDATED');
ELSIF INSERTING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS INSERTED');
ELSIF DELETING THEN
DBMS_OUTPUT.PUT_LINE('TABLE IS DELETED');
END IF;
END;
/
Trigger created.
SQL> INSERT INTO EMP VALUES (4,'SANKAR',700,98,564);
TABLE IS INSERTED
1 row created.
SQL> UPDATE EMP SET EMPID = 5 WHERE EMPNAME = 'SANKAR';
TABLE IS UPDATED
1 row updated.
SQL> DELETE EMP WHERE EMPNAME='SANKAR';
TABLE IS DELETED
1 row deleted
```

Create a Trigger to check the age valid or not Using Message Alert:

PROGRAM:

```
SQL> SET SERVEROUTPUT ON;
SQL> CREATE TRIGGER TRIGNEW
AFTER INSERT OR UPDATE OF AGE ON TRIG
FOR EACH ROW
BEGIN
IF(:NEW.AGE<0) THEN
DBMS_OUTPUT.PUT_LINE('INVALID AGE');
ELSE
DBMS_OUTPUT.PUT_LINE('VALID AGE');
END IF;
END;
/
Trigger created.
SQL> insert into trig values('abc',15);
Valid age
1 row created.
SQL> insert into trig values('xyz',-12);
Invalid age
1 row created.
NAME AGE
---------- ----------
abc 15
xyz -12
```

3. Create a Trigger to check the age valid and Raise appropriate error code and error message.

```
SQL> create table data(name char(10),age number(3));
Table created.
SQL> desc data;
Name Null? Type
----------------------------------------- -------- ------------------------

NAME CHAR(10)
AGE NUMBER(3)
SQL> CREATE TRIGGER DATACHECK
AFTER INSERT OR UPDATE OF AGE ON DATA
FOR EACH ROW
BEGIN
IF(:NEW.AGE<0) THEN
RAISE_APPLICATION_ERROR(-20000,'NO NEGATIVE AGE ALLOWED');
END IF;
END;
/
Trigger created.
SQL> INSERT INTO DATA VALUES('ABC',10);
1 ROW CREATED.
```

```
SQL> INSERT INTO DATA VALUES ('DEF',-15)
 *
ERROR at line 1:
ORA-20000: No negative age allowed
ORA-06512: at "4039.DATACHECK", line 3
ORA-04088: error during execution of trigger '4039.DATACHECK'
NAME AGE
---------- ----------
abc 10
```

4. Create a Trigger for EMP table it will update another table SALARY while inserting values.

```
SQL> CREATE TABLE SRM_EMP2(INAME VARCHAR2(10),
IID NUMBER(5),
SALARY NUMBER(10));
Table created.
SQL> CREATE TABLE SRM_SAL2(INAME VARCHAR2(10),
TOTALEMP NUMBER(5),
TOTALSAL NUMBER(10));
Table created.
```

8. IMPLEMENTATION OF FACTORIAL USING FUNCTION

I) PROGRAM:

```
 SQL>create function fnfact(n number)
return number is
b number;
begin
b:=1;
for i in 1..n
loop
b:=b*i;
end loop;
return b;
end;
/
 SQL>Declare
n number:=&n;
y number;
begin
y:=fnfact(n);
dbms_output.put_line(y);
end;
/
Function created.
Enter value for n: 5
```

```
old 2: n number:=&n;
new 2: n number:=5;
120
PL/SQL procedure successfully completed.
```

9. PROCEDURE USING POSITIONAL PARAMETERS:

```
PROCEDURE USING POSITIONAL PARAMETERS:
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PROCEDURE PROC1 AS
 2 BEGIN
 3 DBMS_OUTPUT.PUT_LINE('Hello from procedure...');
 4 END;
 5 /
 Output:
 Procedure created.
 SQL> EXECUTE PROC1
 Hello from procedure...

 PL/SQL procedure successfully completed.
 II) PROGRAM:
 PROCEDURE USING NOTATIONAL PARAMETERS:
 SQL> CREATE OR REPLACE PROCEDURE PROC2
  2 (N1 IN NUMBER,N2 IN NUMBER,TOT OUT NUMBER) IS
  3 BEGIN
  4 TOT := N1 + N2;
  5 END;
  6 /
 Output:
 Procedure created.
 SQL> VARIABLE T NUMBER
 SQL> EXEC PROC2(33,66,:T)
 PL/SQL procedure successfully completed.
 SQL> PRINT T
  T
 ----------
  99
```

RESULT:
Thus the pl/sql have been executed successfully.

EX.NO:11 DESIGN AND DEVELOP APPLICATIONS