PLSQL:- PROCEDURAL/PROG/EXTENSION OF SQL.

ADVANTAGES:-

1.REDUCE NETWORK TRAFFIC BETWEEN CLIENT AND SERVER.

2.CONTROL ST: IF CONDITION/LOOPS

3.CURSOR: WE CAN ACCESS DATA FROM A TABLE IN PLSQL PROG.

4.EXCEPTION HANDLING:HANDLE RUN TIME EXCEPTIONS.

5.CREATE SUB PROGRAMS,PROCEDURE,FUNCTIONS,PACKAGES...

6.INVOKE THEM OUTSIDE THE DB.

SOFTWARE??

NOTE:- IN THE OLDER VERSION ORCL DID NOT HAVE PLSQL. WE USE TO DEPEND ON PRO*C.

GUI: FORMS/REPORTS/D2K/ORACLE DEVELOPER=>PLSQL.1.0

ORCL DB(5.0)=>1.0

ORCLE DB(6.0): 1.1

ORCL DB(7.0): 2.0HOW

7.1 =>2.1

7.2 =>2.2

8.0=>8.0

HOW TO WRITE PLSQL PROGRAMS??

PLSQL BLOCK:

1.DECLARE:-

THIS SECTION IS USED FOR DECLARE VARIABLES OR U CAN INTIALIZED THEM.

EX:

DECL

DECLARE

V_EMPNO INTEGER:=778;

V_ENAME VARCHAR2(20);

V_SAL NUMBER(10,2);

2.BEGIN:- THIS SECTION IS USED FOR WRITE YOUR LOGIC. CONTROL ST,LOOPS,SQL STAEMENTS...

```
BEGIN
SELECT ENAME V_ENAME FROM EMP WHERE EMPNO=V_EMPNO;
 DBMS_OUTPUT.PUT_LINE('THE NAME IS ' ||V_ENAME);
```

3.EXCEPTION:- THIS SECTION IS USED TO HANDLE RUN TIME EXCEPTIONS.

EX:

```
EXCECPTION
  WHEN NO_DATA_FOUND THEN
   DBMS....('EMPNO DOES NOT EXISTS');
```

4.END:-IT ENDS THE PLSQL PROG.

EXAMPLE:-

```
DECLARE
 V_EMPNO INTEGER:=7788;
  V_ENAME VARCHAR2(20);
BEGIN
 SELECT ENAME INTO V_ENAME FROM EMP WHERE    EMPNO=V_EMPNO;
 DBMS_OUTPUT.PUT_LINE('THE VAL IS   '||V_ENAME);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('EMPNO DOES NOT  EXISTS');
END;
```

NOTE:- DECLARE AND EXCEPTION SECTIONS ARE OPTIONAL. BEGIN AND END IS MANDATORY.

EXAMPLE:-

```
BEGIN
DBMS_OUTPUT.PUT_LINE('HELLO WORLD');
END;
```

TYPES OF PLSQL BLOCKS:-

1.NAMED PLSQL BLOCK:- PLSQL PROG THAT HAVE A UNIQUE NAME.

EX: PROCEDURE,FUNCTIONS,PCKAGE....

2.UN-NAMED PLSQL BLOCK:- PLSQL PROG THAT DO NOT HAVE ANY NAME.USED FOR TESTING OR LOGIC BUILDING PURPOSE.

```
SELECT EMPNO FROM EMP;
```

```
CREATE OR REPLACE VIEW V1 AS SELECT EMPNO FROM EMP;
```

OPERATORS:-

WE CAN USE ALL THE SQL OPERATIONS IN A PLSQL PROGRAM.

PLSQL OPERATORS:

1. -- SINGLE LINE COMMENT (--)

2. /*

MULTI LINE

COMMENT

***/**

3. := THIS IS CALLED ASSIGNEMENT OPR.

USE TO STORE A USER DEFINED VALUE INTO A VAR.

EX:

DECLARE

V_EMPNO INTEGER:=7788;

4. INTO:- THIS OPER IS USED TO STORE A DB VALUE INTO A VARIABLE.

EX:

SELECT SAL INTO V_SAL FROM EMP WHERE EMPNO=7788;

EMPNOI SAL N(10,2) ENAME

7788 3000.40 SHANTHI

5.ATTRIBUTE OPR:-

1 %TYPE: YOU CAN ASSIGN THE DEFAULT DATA TYPE OF A COLUMN IN A TABLE TO A VARIABLE.

2 %ROWTYPE: USED TO STORE THE ENTIRE ROW DATA TYPES OF A TABLE.

EX:

DECLARE

V_EMPNO EMP.EMPNO%TYPE:=7788;

V_SAL EMP.SAL%TYPE;

BEGIN

SELECT SAL INTO V_SAL FROM EMP WHERE EMPNO=V_EMPNO;

DECLARE

V_REC EMP%ROWTYPE;

BEGIN

V_REC.EMPNO:=7788;

SELECT * INTO V_REC FROM EMP WHERE EMPNO=V_REC.EMPNO;

DATA TYPES:-

SUPPORTS ALL THE DATA TYPES OF SQL.

PLSQL DATA TYPES:

1.BINARY_INTEGER:

2.BOOLEAN:

3.EXCEPTION:

WHERE TO WRITE THE PLSQL CODE??

1.WRITE DIRECTLY IN SQL PROMPT.

2.ED

/

NO OUTPUT IS COMING??

```
BEGIN
 DBMS_OUTPUT.PUT_LINE('TESTING');
END;
```

```
DECLARE
 V_REC EMP%ROWTYPE;
BEGIN
 V_REC.EMPNO:=&EMPNO;
  SELECT * INTO V_REC FROM EMP WHERE    EMPNO=V_REC.EMPNO;
   DBMS_OUTPUT.PUT_LINE(V_REC.ENAME||'  '||V_REC.JOB);
END;
```

CONTROL STATEMENTS:-

1.SIMPLE IF:-

```
IF <COND> THEN
 <STMTNS>;
[ELSE
  <STMTNS>;]
END IF;
```

EX: WRITE A PSLQL PROGRAM THAT PROMPTS FOR EMPNO,ENAME,SAL FROM EMP TABLE.

IF EMPNO PASSED DOES NOT EXISTS IN THE EMP TABLE THEN INSERT THE VALUES INTO EMP TABLE.

7788,'ROCK',1000. ELSE UPDATE THE ENAME AND SAL

HOW TO CHECK THE EMPNO EXISTS OR NOT???

```
DECLARE
V_EMPNO EMP.EMPNO%TYPE;
  V_ENAME EMP.ENAME%TYPE;
   V_SAL EMP.SAL%TYPE;
    V_COUNT BINARY_INTEGER;
BEGIN
 V_EMPNO:=&EMPNO;
  V_ENAME:='&ENAME';
   V_SAL:=&SAL;

   SELECT COUNT(*) INTO V_COUNT FROM EMP
    WHERE EMPNO=V_EMPNO;
   IF V_COUNT=0 THEN
    INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES
     (V_EMPNO,V_ENAME,V_SAL);
    DBMS_OUTPUT.PUT_LINE('REC INSERTED');
```

```
 ELSE
  UPDATE EMP SET ENAME=V_ENAME,SAL=V_SAL
   WHERE EMPNO=V_EMPNO;
   DBMS_OUTPUT.PUT_LINE('REC UPDATED');
 END IF;
 COMMIT;
END;
```

2.IF ELSIF:-

```
IF <COND1> THEN
 <STMTNS>;
 ELSIF <COND2> THEN
 <STMTNS>;
 [ELSE
 <STMTNS>;]
END IF;
```

EX:- WRITE A PLSQL PROGRAM THAT

PROMPT FOR EMPNO, IF THE EMPLOYEE

WORKS IN DEPTNO 10 THEN UPDATE SALARY BY 10% ELSIF DEPTNO 20 THEN UPDATE BY 20%, ELSE UPDATE BY 5%.

```
DECLARE
V_EMPNO EMP.EMPNO%TYPE;
 V_DEPTNO EMP.DEPTNO%TYPE;
  V_SAL EMP.SAL%TYPE;
BEGIN
V_EMPNO:=&EMPNO;

SELECT DEPTNO,SAL INTO V_DEPTNO,V_SAL FROM EMP
```

```
 WHERE EMPNO=V_EMPNO;

IF V_DEPTNO=10 THEN
 V_SAL:=V_SAL+V_SAL*10/100;
 ELSIF V_DEPTNO=20 THEN
  V_SAL:=V_SAL+V_SAL*20/100;
ELSE
 V_SAL:=V_SAL+V_SAL*5/100;
END IF;
DBMS_OUTPUT.PUT_LINE('SALARY INCREASED');
UPDATE EMP SET SAL=V_SAL WHERE EMPNO=V_EMPNO;
COMMIT;
END;
```

3.NESTED IF:-

```
IF <COND1> THEN
 IF <COND2> THEN
  <STMTNS>;
 ELSIF <COND3> THEN
  <STMTN>;
 END IF;
ELSE
<STMTN>;
END IF;

DECLARE
V_EMPNO EMP.EMPNO%TYPE;
 V_DEPTNO EMP.DEPTNO%TYPE;
 V_SAL EMP.SAL%TYPE;
  V_COUNT BINARY_INTEGER;
```

```
BEGIN
V_EMPNO:=&EMPNO;
 SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE
   EMPNO=V_EMPNO;
 IF V_COUNT>0 THEN
SELECT DEPTNO,SAL INTO V_DEPTNO,V_SAL FROM EMP
  WHERE EMPNO=V_EMPNO;

IF V_DEPTNO=10 THEN
  V_SAL:=V_SAL+V_SAL*10/100;
 ELSIF V_DEPTNO=20 THEN
   V_SAL:=V_SAL+V_SAL*20/100;
ELSE
 V_SAL:=V_SAL+V_SAL*5/100;
END IF;
DBMS_OUTPUT.PUT_LINE('SALARY INCREASED');
UPDATE EMP SET SAL=V_SAL WHERE EMPNO=V_EMPNO;
COMMIT;
ELSE
 DBMS_OUTPUT.PUT_LINE('EMPNO DOES NOT EXISTS');
END IF;
END;

9IF V_EMPNO=7788 THEN
  8 IF V_ENAME='SCOTT' THEN
  <ST>;
  ELSIF V_ENAME='RAJ' THEN
   <ST..>;
  8 END IF;
 ELSE
  <ST2>;
```

9 END IF;

LOOPS:-

SET OF STATEMENTS THAT CAN EXECUTED REPEATEDLY.

3 TYPES:

1.SIMPLE LOOP:-

A.IT WILL EXECUTES THE SET OF STATEMENTS PROIDED THE CONDITION IS FALSE. OTHER WORDS IT WILL EXIT/TERMINATE THE LOOP WHEN THE CONDITION IS TRUE.

*.IT CAN RUN THE LOOP ATLEAST ONCE.

HOW TO CREATE A SIMPLE LOOP:-

BEGIN

LOOP

<STMTS>;

...

EXIT WHEN <CONDITION IS TRUE>;

END LOOP;

EX: WRITE A PLSQL PROG THAT WILL INSERT THE FOLLOWING RECORDS INTO THE MYTEST1 TABLE: STOP THE INSERT WHEN THE RESULT IS MORE THAN 10.

SRNO RES

----

0 2

1 3

2 4

.

..

```
DECLARE
 S BINARY_INTEGER:=0;
  R BINARY_INTEGER:=0;
BEGIN
LOOP
R:=S+2;
S:=S+1;
INSERT INTO MYTEST1 VALUES(S,R);
 S:=S+1;
EXIT WHEN R>10;
END LOOP;
END;
```

2.WHILE LOOP:-

A.IT WILL EXECUTES SET OF STATEMENTS WHEN CONDITION IS TRUE.IF CONDITION IS FALSE IT WILL TERMINATE THE LOOP.

2.IT CANNOT RUN THE LOOP ATLEAST ONCE.

```
BEGIN
 WHILE <CONDTRUE> LOOP
  <STMTNS>;
 END LOOP;
```

SAME EX DO USING WHILE LOOP:

```
DECLARE
 S BINARY_INTEGER:=0;
  R BINARY_INTEGER:=0;
BEGIN
```

```
WHILE R<10 LOOP
R:=S+2;
INSERT INTO MYTEST1 VALUES(S,R);
 S:=S+1;
END LOOP;
END;
```

3.FOR LOOP:-

USE THIS LOOP WHEN WE WANT TO RUN STATEMENTS SPECIFIC NUMBER OF TIMES.

```
 FOR I IN [REVERSE] <LL>..<UL> LOOP
  ..
   ...
 END LOOP;
```

I:- IS UN DEFINED VAR THAT CONTAINS THE NUMBER OF TIMES THE LOOP IS RUN.

<LL>: THE STARTING RANG.

<UL>: THE ENDING RANGE.

.. : A REFRENCE OPERATOR

REVERSE: THE LOOP CAN RUN IN A REVERSE ORDER FROM UL TO LL.

EX:

```
FOR I IN 1..10 LOOP
END LOOP;
```

EX:- INSERT THE FOLLOWING RECORDS INTO THE MYTEST1 TABLE

SRNO	RES
1	2
2	4
3	6 , 4 8 , 5 10

```
DECLARE
S BINARY_INTEGER:=1;
 R BINARY_INTEGER:=0;
BEGIN
 FOR I IN 1..10 LOOP
  R:=S*2;
   INSERT INTO MYTEST1 VALUES(S,R);
 S:=S+1;
 END LOOP;
END;

BEGIN
 FOR I IN  1..5 LOOP
   INSERT INTO MYTEST1 VALUES(I,I*2);
  END LOOP;
END;
```

2 COMMAND THAT CAN BE USED IN LOOPS:

1.EXIT:- COMMAND USED TO TERMINATE/BREAK THE LOOP.

NOTE: THIS IS MANDATORY IN A SIMPLE LOOP.

EX:

```
BEGIN
 FOR I IN  1..5 LOOP
  IF I=3 THEN
   EXIT;
  ELSE
   INSERT INTO MYTEST1 VALUES(I,I*2);
 END IF;
```

```
 END LOOP;
END;
```

2. CONTINUE:- U CAN SKIP THE ITERATION.

```
BEGIN
 FOR I IN  1..5 LOOP
  IF I IN(1,2) THEN
   CONTINUE;
  ELSE
   INSERT INTO MYTEST1 VALUES(I,I*2);
 END IF;
  END LOOP;
END;
```

NOTE: CAN WE USE THE EXIT/CONTINUE STATEMENTS COMMANDS OUTSIDE A LOOP??NO

CURSOR:-

```
DECLARE
 V_EMPNO EMP.EMPNO%TYPE;
  V_ENAME EMP.ENAME%TYPE;
BEGIN
  V_EMPNO:=&EMPNO;
   SELECT ENAME INTO V_ENAME FROM EMP WHERE
     EMPNO=V_EMPNO;
    DBMS_OUTPUT.PUT_LINE(V_ENAME);
END;
```

1.WHENEVER WE HAVE A PROG WITH ANY SQL STATEMENT,ORCL WILL CREATE PRIVATE SQL WORK AREA??TO STORE OR PROCESS THE SQL STATEMENT IN THE PROG. THAT STEP IS CALLED **”DECLARE"**

2.ORCL WILL USE THIS AREA TO POPULATE THE DATA REQUIRED FOR YOUR PROG.

EX: WE WANT THE ENAME FROM EMP TABLE. ORACLE WILL PULL THE DATA FROM THE TABLE IN THE HD INTO THIS AREA. THIS AREA WILL POINT THE DATA. THIS STEP IS CALLED AS **"OPEN"**

IN OTHER WORDS WE CAN SAY THAT THE PROG SQL STATEMENT IS EXECUTED.

3.THE DATA FROM THE MEMORY OF THE DB SERVER SHOULD GOT INTO THE PROG IN FORM VARIABLES. THAT STEP IS CALLED **"FETCH"**

4.AFTER THE FETCH IS COMPLETED WE HAVE TO CLOSE/RELASE THE SPACE IN THE MEMORY OF THE DB SERVER,THIS STEP IS CALLED **“CLOSE”**

1.DECLARE:

2.OPEN:

3.FETCH:

4.CLOSE:

WHAT IS A CURSOR:-

IT IS PRIVATE SQL WORK AREA ON THE DB SERVER USED FOR PROCESS SQL STATEMENTS GIVEN IN A PLSQL PROGRAM.

2 TYPES:

1.IMPLICIT CURSOR:- THE ABOVE 4 STPES ARE DONE IMPLICITLY BY ORCL/SYSTEM.

USED FOR SINGLE ROW PROCESSING.

WHAT IS THE NAME GIVEN FOR THE IMPLICIT CURSOR???SQL

2.EXPLICIT CURSOR:- THE ABOVE 4 STEPS WILL BE DONE BY THE PROGRAMER/DEVELOPER/USER EXPLICITLY.

4 IN-BUIT FUNCTTIONS;

1. %ROWCOUNT:-RETURNS A INTEGER VALUE INDICATING THE NUMBER OF ROWS ALREADY FETCHED FROM THE CURSOR INTO YOUR PROG/NUBER OF ROWS PROCESSED.

EX:

```
DECLARE
 V_EMPNO EMP.EMPNO%TYPE;
  V_ENAME EMP.ENAME%TYPE;
BEGIN
  V_EMPNO:=&EMPNO;
   SELECT ENAME INTO V_ENAME FROM EMP WHERE
     EMPNO=V_EMPNO;
       IF SQL%ROWCOUNT>0 THEN
    DBMS_OUTPUT.PUT_LINE('GOOD');
END IF;
END;
```

2.%FOUND:- RETURNS A BOOLEAN VALUE TRUE/FALSE. INCATING IF THE VALUE WAS SUCCESSFULLY FETCH FROM THE CURSOR INTO THE PROG RETURNS TRUE ELSE RETURNS FALSE.

```
DECLARE
 V_EMPNO EMP.EMPNO%TYPE;
  V_ENAME EMP.ENAME%TYPE;
BEGIN
  V_EMPNO:=&EMPNO;
   SELECT ENAME INTO V_ENAME FROM EMP WHERE
     EMPNO=V_EMPNO;
       IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE('CURSOR CREATED');
END IF;
```

END;

3.%NOTFOUND:- RETURNS A BOOLEAN VALUE INDICATING THAT LAST FETCH IS NOT SUCCESSFUL RETURNS TRUE IF SUCCESSFUL RETURNS FALSE.

```
FETCH..
WHILE %FOUND LOOP
  DBMS...();
FETCH..
END LOOP;
```

```
LOOP
 FETCH..
EXIT WHEN %NOTFOUND;
DBMS..();

END LOOP;
```

4.%ISOPEN:- THIS ALSO RETURNS A BOOLEAN VALUE INDICATING THE CUSOR IS OPEN RETURNS TRUE ELSE FALSE.

NOTE:- FOR IMPLICIT CURSOR THIS FUNCTION WILL ALWAYS RETURNS FALSE TO THE DEVELOPER.

IT IS CONTROLLED BY THE ORCL.

```
BEGIN
IF %ISOPEN THEN
  FETCH..
ELSE
OPEN...
END IF;
```

NOTE:- USE THE ABOVE 4 FUNCTIONS WHEN WE HAVE CURSORS IN A PLSQL PROG.

NOTE:- THESE FUNCTION CAN BE USED WITH IMPLICIT OR EXPLICIT CURSORS.

EXPLICIT CURSOR: THE DEVELOPER/PROG WILL DO THE 4 STEPS (DECLARE,OPEN,FETCH,CLOSE) EXPLICITLY.

ADVANTAGES:-

1.USER CONTROL?? THE DEV CAN CONTROL THE CURSOR IF WANT REOPEN THE SAME CURSOR MORE THAN ONCE IN THE SAME PROG.

2.MULTI ROW PROCESSING: WE RETURN MULTI ROWS/PROCESS MULTIPLE ROWS.

3.WE CAN PASS ARGUMENTS/PARAMETERS TO THE CURSORS.

SELECT MOD(3,2) FROM DUAL;

4 STEPS:

1.DECLARE:

DECLARE

CURSOR <CNAME> IS <QUERY>;

EX:-

```
DECLARE
 CURSOR V_C1 IS SELECT EMPNO FROM EMP;
  V_EMPNO EMP.EMPNO%TYPE;
```

2.OPEN:-

OPEN <CURNAME>;

EX:

BEGIN

OPEN V_C1;

NOTE: ORCL WILL RUN THE SQL STATEMENT.

EX: SELECT EMPNO FROM EMP;

101

=102

103

104

CALLED AS ACTIVE DATA SET.

3.FETCH:- THE DATA FROM CURSOR WILL BE FETCHED INTO THE PROG INFORM OF VAR.

FETCH <CNAME> INTO <VAR>;

EX:

BEGIN

FETCH V_C1 INTO V_EMPNO;101

FETCH V_C1 INTO V_EMPNO;102

FETCH V_C1 INTO V_EMPNO;

NOTE: THE FETCH WILL BE IN A LOOP.

BEGIN

OPEN V_C1;

```
 LOOP
   FETCH V_C1 INTO V_EMPNO;
    EXIT WHEN V_C1%NOTFOUND;
    DBMS...();
END LOOP;



BEGIN
OPEN V_C1;
 FETCH V_C1 INTO V_EMPNO;
  WHILE V_C1%FOUND LOOP
   DBMS...();
    FETCH V_C1 INTO V_EMPNO;
  END LOOP;
```

4.CLOSE:-

```
BEGIN
 CLOSE <CNAME>;
EX:

BEGIN
 CLOSE V_C1;
END;
----
EX:

DECLARE
  CURSOR V_C1 IS SELECT EMPNO FROM EMP;
```

```
  V_EMPNO EMP.EMPNO%TYPE;
BEGIN
 OPEN V_C1;
 LOOP
  FETCH V_C1 INTO V_EMPNO;
   EXIT WHEN V_C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(V_EMPNO);
  END LOOP;
 CLOSE V_C1;
END;
```

NOTE:- IT IS NOT MANDATORY TO CLOSE THE CUSOR IF U ARE NOT REOPENING IT ONCE AGAIN.

EX:-

```
DECLARE
 CURSOR V_C1 IS SELECT EMPNO FROM EMP;
  V_EMPNO EMP.EMPNO%TYPE;
BEGIN
 OPEN V_C1;
  FETCH V_C1 INTO V_EMPNO;
   WHILE V_C1%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE(V_EMPNO);
    FETCH V_C1 INTO V_EMPNO;
 END LOOP;
 CLOSE V_C1;
END;
```

MODIFY THE ABOVE PROG TO ALSO PRINT ENAME??

```
 DECLARE
 CURSOR V_C1 IS SELECT EMPNO,ENAME,SAL FROM EMP;
 V_REC V_C1%ROWTYPE;
BEGIN
 OPEN V_C1;
 LOOP
  FETCH V_C1 INTO V_REC;
   EXIT WHEN V_C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(V_REC.EMPNO||' '||V_REC.ENAME||' '||V_REC.SAL);
  END LOOP;
 CLOSE V_C1;
END;
```

EX:

DISPLAY TOP N SALARIES??

5

4

3

2

1

```
DECLARE
CURSOR V_C2 IS SELECT DISTINCT SAL FROM EMP ORDER BY SAL DESC;
V_SAL EMP.SAL%TYPE;
BEGIN
 OPEN V_C2;
LOOP
 FETCH V_C2 INTO V_SAL;
DBMS_OUTPUT.PUT_LINE(V_SAL);
```

```
 EXIT WHEN V_C2%ROWCOUNT=&NO;
  END LOOP;
CLOSE V_C2;
END;
```

```
DECLARE
 CURSOR V_C1 IS SELECT EMPNO FROM EMP;
  V_EMPNO EMP.EMPNO%TYPE;
BEGIN
OPEN V_C1;
LOOP
  FETCH V_C1 INTO V_EMPNO;
   EXIT WHEN V_C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(V_EMPNO);
  END LOOP;
CLOSE V_C1;
OPEN V_C1;
LOOP
  FETCH V_C1 INTO V_EMPNO;
   EXIT WHEN V_C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(V_EMPNO);
  END LOOP;
CLOSE V_C1;
```

END;

EXPLICIT CURSOR:

EX: WRITE A PLSQL PROGRAM THAT WILL DO 2 LOGICS. FIRST TO DISPLAY ALL THE EMPLOYEES FROM DEPTNO IN 10 EMP TABLE. INSERT THE EMPLOYEES INTO EMPT TABLE FOR THAT DEPTNO 20.

```
DECLARE
CURSOR V_C1 IS SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP WHERE DEPTNO=10;
BEGIN
 OPEN V_C1;
   LOOP
    FETCH...
     EXIT.
 IF DENO=10 THEN
      DBMS..();
ELSIF DENO=20 THEN
  INSERT..

  END LOOP;
```

CURSOR WITH PARAMTERS:

THE DEV/USERS CAN PASS RUN TIME VALUE TO THE CURSOR WHEN WE OPEN OPEN THE CURSOR.

HOW TO DECLARE PARAMETERS??

1.DECLARE:

```
 DECLARE
   CURSOR <CNAME>  IS <QUERY PASS PARAMETER CONDITION>;
```

DECLARE

CURSOR V_C1(V_DEPTNO EMP.DEPETNO%TYPE) IS SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP WHERE DEPTNO=V_DEPTNO;

2.OPEN:

BEGIN

OPEN <CNAME>(<VAL>);

EX:

OPEN V_C1(10);

OPEN V_C1(20);

3.FETCH

4.CLOSE

NOTE: FETCH AND CLOSE WILL BE SAME.

=>

103 RAJANI 3000 20

104 RAKESH 4000 20

DECLARE

CURSOR V_C1(V_DEPTNO EMP.DEPTNO%TYPE) IS

SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP WHERE

DEPTNO=V_DEPTNO;

V_REC V_C1%ROWTYPE;

```
BEGIN
 OPEN V_C1(10);
  LOOP
   FETCH V_C1 INTO V_REC;
    EXIT WHEN V_C1%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(V_REC.ENAME||' '||V_REC.DEPTNO);
 END LOOP;
 CLOSE V_C1;
 OPEN V_C1(20);
  LOOP
   FETCH V_C1 INTO V_REC;
    EXIT WHEN V_C1%NOTFOUND;
     INSERT INTO EMPT VALUES V_REC;
    END LOOP;
 CLOSE V_C1;
END;
```

ACCOUNTING

101 RAJ

102 JAY

NOTE:- IF WE REOPEN THE CURSOR MORE THAN ONCE BY PASSING DIFFERENT PARAMETER VALUES THE LENGHT OF THE PROG WILL INCREASE.

3.CURSOR WITH FOR LOOP:- IT IS A COMBINATION OF IMPLICIT AND EXPLICIT CURSOR.

EXPLICITY DEV WILL DECLARE THE CURSOR. OPEN,FETCH AND CLOSE IS DONE BY ORCL USING SPECIAL FOR LOOP.

```
FOR V_REC IN V_C1(10) LOOP
DBMS....();
```

END LOOP;

V_C1(10)=> ORCL IS IMPLICTILY OPENING THE CURSOR.

THEN ORCL WILL FETCH THE FIRST ROW FROM THE CURSOR INTO V_REC.

ORCL WILL ALSO CLOSE THE CURSOR IMPLICITLY.

EX: WRITE A PLSQL PROG THAT WILL PROMPT FOR A DEPTNO FROM EMP TABLE. BASED ON THE DEPTNO U PASS DO AN INSERT INOT THE EMPT TABLE AND ALSO UPDATE THOSE EMPLOYEE SALARY BY 5%.

=>101 JAY 2000 10

102 RAJ 3000 10

```
DECLARE
 CURSOR V_C2(V_DEPTNO EMP.DEPTNO%TYPE) IS SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP WHERE
  DEPTNO=V_DEPTNO FOR UPDATE NOWAIT;
BEGIN
  FOR V_REC IN V_C2(&DNO) LOOP
   INSERT INTO EMPT VALUES V_REC;
     UPDATE EMP SET SAL=V_REC.SAL+V_REC.SAL*5/100 WHERE EMPNO=V_REC.EMPNO;
  END LOOP;
 COMMIT;
END;
```

FOR UPDATE:- THIS COMMAND IS USED WITH A QUERY?? IN ORDER TO LOCK A SPECIFIC TABLE.

EX: SELECT *FROM EMP FOR UPDATE;

NOWAIT:- THIS OPION IS USED WITH THE FOR UPDATE SO THAT IF THE TABLE IS BUSY/LOCKED BY OTHER USER THEN SO ORCL WILL NOT WAIT FOR THE TABLE TO BE UNLOCK,IT WILL JUST GIVE MSG "RESOURCE BUSY"

WAIT <SEC>.

EX:

SELECT * FROM EMP FOR UPDATE NOWAIT/WAIT 10;

BEFORE GIVING ANY UPDATE/DELETE ON A TABLE IT ADVISABLE TO LOCK THE TABLE USING FOR UPDATE NOWAIT OPTION IN A QUERY.

NOTE: WE CAN LOCK SPECIFIC ROWS OF THE SAME BY DIFFERENT DB USERS.

```
DECLARE
 CURSOR V_C2(V_DEPTNO EMP.DEPTNO%TYPE) IS SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP WHERE
  DEPTNO=V_DEPTNO FOR UPDATE NOWAIT;
BEGIN
  FOR V_REC IN V_C2(&DNO) LOOP
   INSERT INTO EMPT VALUES V_REC;
    UPDATE EMP SET SAL=V_REC.SAL+V_REC.SAL*5/100 WHERE CURRENT OF V_C2;
  END LOOP;
 COMMIT;
END;
```

WHAT IS WHERE CURRENT OF OPTION??

BY DEFAULT IT WILL USE INDEX TO UPDATE EVERY ROW IN THE PROG??EMPNO IS AUTO INDEX AS IS A PK. PERFORMANCE IS SLOW.

USE THE CURRENT OF OPTION SO THAT INDEX CAN BE AVOIDED AND USE THE CURRENT CURSOR POINTER AS A CONDITION.

CURSOR WITH UPDATE CLAUSE/COMMAND:

1. FOR UPDATE NOWAIT>AVOID WAIT EVENTS.

2.WHERE CURRENT OF CNAME.IMPROVE PERFORMANCE.

CAN WE USE CURRENT OF OPTION WITH FOR UPDATE??

=======

```
DECLARE
 CURSOR V_C1 IS SELECT * FROM DEPT;
  CURSOR V_C2 IS SELECT * FROM EMP;
V_REC EMP%ROWTYPE;
 V_REC1 DEPT%ROWTYPE;
CHOICE BINARY_INTEGER;
BEGIN
 CHOICE:=&CHOICE;

IF CHOICE=1 THEN
 OPEN V_C1;
  LOOP
   FETCH V_C1 INTO V_REC1;
    EXIT WHEN V_C1%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(V_REC1.DNAME);
 END LOOP;
 CLOSE V_C1;
ELSIF CHOICE=2 THEN
  OPEN V_C2;
   LOOP
     FETCH V_C2 INTO V_REC;
      EXIT WHEN V_C2%NOTFOUND;
```

```
        DBMS_OUTPUT.PUT_LINE(V_REC.ENAME);
    END LOOP;
 CLOSE V_C2;
 ELSE
   DBMS_OUTPUT.PUT_LINE('INVALID CHOICE');
 END IF;
END;
======================


DECLARE
 CURSOR V_C1(V_DEPTNO EMP.DEPTNO%TYPE) IS
SELECT EMPNO,ENAME,SAL,DEPTNO FROM EMP WHERE
  DEPTNO=V_DEPTNO;
  BEGIN
 FOR V_REC IN V_C1(10) LOOP
      DBMS_OUTPUT.PUT_LINE(V_REC.ENAME||'  '||V_REC.DEPTNO);
 END LOOP;
 FOR V_REC IN V_C1(20) LOOP
    INSERT INTO EMPT VALUES V_REC;
    END LOOP;
    COMMIT;
END;
```

REF CURSORS: IS CONCEPT IN PLSQL WHICH CAN POINT TO MUTIPLE QUERIES BUT ANY ONE OF THEM AT ONE TIME.

ADV: GIVES US OPTIMIZATION OF MEMORY.

HOW TO DECLARE THE REF CURSOR??

SYSDEFINED DATA TYPE:

1. NUMBER

EX:

X NUMBER;

X:=10;

X:=20;

X:=30;

USER DEFINED DATA TYPE:- DATA TYPE DEFINED BY USER.

DECLARE

TYPE <DATANAME> IS REF CURSOR;

EX:

DECLARE

TYPE T1 IS REF CURSOR; => T1 IS A USER DEFINED DATA TYPE

X T1;

OPEN X FOR SELECT * FROM DEPT;

OPEN X FOR SELECT * FROM EMP;

OPEN X FOR SELECT * FROM SALGRADE;

FETCH AND CLOSE IS SAME OF OTHER CURSORS.

DECLARE

V_DEPTNO NUMBER;

X NUMBER;

Y NUMBER;

DECLERE

```
====================
DECLARE
 CURSOR V_C1 IS SELECT * FROM DEPT;
  CURSOR V_C2 IS SELECT * FROM EMP;
V_REC EMP%ROWTYPE;
 V_REC1 DEPT%ROWTYPE;
CHOICE BINARY_INTEGER;
BEGIN
 CHOICE:=&CHOICE;

IF CHOICE=1 THEN
 OPEN V_C1;
  LOOP
   FETCH V_C1 INTO V_REC1;
    EXIT WHEN V_C1%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(V_REC1.DNAME);
 END LOOP;
 CLOSE V_C1;
ELSIF CHOICE=2 THEN
  OPEN V_C2;
   LOOP
     FETCH V_C2 INTO V_REC;
      EXIT WHEN V_C2%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(V_REC.ENAME);
   END LOOP;
 CLOSE V_C2;
 ELSE
  DBMS_OUTPUT.PUT_LINE('INVALID CHOICE');
 END IF;
END;
================================================
```

```
DECLARE
 TYPE T1 IS REF CURSOR;
  V_C1 T1;
V_REC EMP%ROWTYPE;
 V_REC1 DEPT%ROWTYPE;
CHOICE BINARY_INTEGER;
BEGIN
 CHOICE:=&CHOICE;

IF CHOICE=1 THEN
 OPEN V_C1 FOR SELECT * fROM DEPT;
  LOOP
   FETCH V_C1 INTO V_REC1;
    EXIT WHEN V_C1%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(V_REC1.DNAME);
 END LOOP;
 CLOSE V_C1;
ELSIF CHOICE=2 THEN
  OPEN V_C1 FOR SELECT * FROM EMP;
   LOOP
    FETCH V_C1 INTO V_REC;
     EXIT WHEN V_C1%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(V_REC.ENAME);
   END LOOP;
 CLOSE V_C1;
 ELSE
  DBMS_OUTPUT.PUT_LINE('INVALID CHOICE');
 END IF;
END;
```

WRITE A PLSQL PROG THAT WILL DISPLAY THE JOB BY PASSING THE EMPNO?

```
DECLARE
  V_EMPNO EMP.EMPNO%TYPE;
   V_JOB EMP.JOB%TYPE;
BEGIN
V_EMPNO:=&EMPNO;77
  ==>SELECT JOB INTO V_JOB  FROM EMP WHERE
    EMPNO=V_EMPNO;
    DBMS_OUTPUT.PUT_LINE(V_JOB);
END;
```

EXCEPTION HANDLING:-

WE HANDLE RUN TIME EXCEPTIONS IN GIVEN PLSQL PROG.

ADVANTAGES:-

1. WE CAN GIVE OUR OWN USER DEFINED MESSAGES.

2.AUDITING:

3.GRACEFULLY EXIT THE PROG.

NOTE: IS EXCEPTION MANDATORY???NO

HOW TO HANDLE EXCEPTION:

EXCEPTION HANDLERS: HANDLERS USED TO HANDLE SPECIFIC TYPES OF EXCEPTIONS.

2 TYPES:

1.SYSTEM DEFINED EXPCEPTIONS HANDLERS:-HANDLERS GIVEN BY ORCL.

EX:

1.NO_DATA_FOUND:- THIS HANDLER WILL BE USED IN PLSQL PROG WHEN A QUERIES WHERE CONDITION FAILS.

WHERE TO HANDLE THE EXCEPTIONS??EXCEPTION BLOCK.

```
EXCEPTION
 WHEN NO_DATA_FOUND THEN
  ....
DECLARE
 V_EMPNO EMP.EMPNO%TYPE;
  V_JOB EMP.JOB%TYPE;
BEGIN
V_EMPNO:=&EMPNO;
 SELECT JOB INTO V_JOB  FROM EMP WHERE
   EMPNO=V_EMPNO;
   DBMS_OUTPUT.PUT_LINE(V_JOB);

EXCEPTION
 WHEN NO_DATA_FOUND THEN
INSERT INTO ERROR_TAB1 VALUES(V_EMPNO);
COMMIT;
RAISE_APPLICATION_ERROR(-20001,'Empno Does not Exists');
END;
```

NOTE:- IT BETTER U HAVE A UNIQUE EXCEPTION CODE FOR EVERY MESSAGE GIVEN BY THE PROG/DEV.

FOR FURTHER DEBUGGING. LIKE WHAT IS CAUSE AND WHAT IS ACTION??

```
RAISE_APPLICATION_ERROR(<ERRORCODE>,'MSG')
```

ERROCODE: -20000 TO -20999

NOTE: THE RAISE_APPLICATION_ERROR METHOD WILL ROLLBACK ALL THE TRANSACTIONS AND GRACEFULLY EXIT.

2.TOO_MANY_ROWS:-

```
DECLARE
  V_DEPTNO NUMBER(1);
   V_JOB EMP.JOB%TYPE;
BEGIN
V_DEPTNO:=&DEPTNO;
  SELECT JOB INTO V_JOB  FROM EMP WHERE
    DEPTNO=V_DEPTNO;
    DBMS_OUTPUT.PUT_LINE(V_JOB);

EXCEPTION
  WHEN NO_DATA_FOUND THEN
   RAISE_APPLICATION_ERROR(-20001,'Deptno does not Exists');
 WHEN TOO_MANY_ROWS THEN
   RAISE_APPLICATION_ERROR(-20002,'returns more than one row use explicit cursors');
END;
```

3.OTHERS:-IT WILL HANDLE THOSE EXCEPTION WHICH THE DEV HAS NOT HANDLED.

```
DECLARE
  V_DEPTNO NUMBER(1);
   V_JOB EMP.JOB%TYPE;
   X NUMBER;
    Y VARCHAR2(1000);
BEGIN
V_DEPTNO:=&DEPTNO;
  SELECT JOB INTO V_JOB  FROM EMP WHERE
```

```
   DEPTNO=V_DEPTNO;
   DBMS_OUTPUT.PUT_LINE(V_JOB);

EXCEPTION
  WHEN NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR(-20001,'Deptno does not Exists');
 WHEN TOO_MANY_ROWS THEN
  RAISE_APPLICATION_ERROR(-20002,'returns more than one row use explicit cursors');
WHEN VALUE_ERROR THEN
  RAISE_APPLICATION_ERROR(-20004,'size too small contact devel');
 WHEN OTHERS THEN
 X:=SQLCODE;
 Y:=SQLERRM;
 INSERT INTO ERROR_TAB2 VALUES(X,Y);
 COMMIT;
 RAISE_APPLICATION_ERROR(-20003,'Sorry Please contact the Developer');
END;
```

SQLCODE:- RETURN THE ORCL ERROR CODE.

SQLERRM:- RETURN ORCL ERROR CODE AND MSG.

```
CREATE TABLE ERROR_TAB2(ERRCODE NUMBER,ERRMSG VARCHAR2(1000));
```

4.VALUE_ERROR:- HANDLE THOSE EXCEPTIONS WHEN A VAR LEN IS TOO SMALL OR DATA TYPE MISMATCHED.

5.CURSOR_ALREADY_OPEN:- IT WILL HANDLE THOSE EXCEPTION FOR CURSORS WHEN A CURSOR IS NOT CLOSED BUT REOPENING ONCE AGAIN.

6.INVALID_CURSOR:- USE FETCH WITHOUT OPENING. OR OPEN IN FOR FOR LOOP CURSOR, OPEN A CURSOR NOT DECLARED.

EX:

```
DECLARE
V_DEPTNO DEPT.DEPTNO%TYPE;
BEGIN
  V_DEPTNO:=&DEPTNO;
   DELETE FROM DEPT WHERE DEPTNO=V_DEPTNO;
    DBMS_OUTPUT.PUT_LINE('REC DELETED');
     COMMIT;
END;
```

```
ALTER TABLE DEPT ADD PRIMARY KEY(DEPTNO);
```

```
ALTER TABLE EMP ADD FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO);
```

ORACLE DEFINED ERROR CODE METHOD:

1.DECLARE A VARIABLE OF EXCEPTION. TO STORE THE ORCL ERROR CODE.

EX:

```
DEL_MAS EXCEPTION;
```

2.INITIALIZE THE VAR WITH THE ORCL ERROR CODE.

```
 PRAGMA EXCEPTION_INIT(DEL_MAS,-02292);
```

EXCEPTION_INIT:- IT A INBUILT FUNCTION GIVEN BY ORCL TO INTIALIAZE THE VAR WITH THE ERROR CODE.

PRAGMA: IT IS PLSQL COMPILIER/DIRECTIVE THAT WILL INVOKE THE FUNCTION EXCEPTION_INIT

WHEN DEL_MAS THEN

.....

```
DECLARE
V_DEPTNO DEPT.DEPTNO%TYPE;
DEL_MAS EXCEPTION;
 PRAGMA EXCEPTION_INIT(DEL_MAS,-02292);
BEGIN
  V_DEPTNO:=&DEPTNO;
   DELETE FROM DEPT WHERE DEPTNO=V_DEPTNO;
    DBMS_OUTPUT.PUT_LINE('REC DELETED');
     COMMIT;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   RAISE_APPLICATION_ERROR(-20002,'Deptno does not exists');
  WHEN DEL_MAS THEN
    RAISE_APPLICATION_ERROR(-20001,'CANNOT DELETE THE MASTER AS CHILD FOUND');
END;
```

A.NAMED:- HANDLERS THAT HAVE NAME. EX: NO_DATA_FOUND...

B.UN-NAMED:- USE THE ORCL ERROR CODE.

NOTE:- FOR UN-NAMED HANDLER U MUST KNOW THE ERROR CODE.

OR DEPENDENT ON ERROR_TAB2 METHOD.

2.USER DEFINED EXPCEPTIONS HANDLERS:-

WHEN ORCL IS NOT ABLE TO THROW THE EXCEPTION WHEN EX: DELETE/UPDATE WHERE CONDTION FAILS OR LOGICAL ERROR..

HOW TO USE THIS METHOD??

1.DECLARE A VAR OF TYPE EXCEPTION.

EX:

```
DECLARE
 INV_DEPTNO EXCEPTION;
```

2.THROW THE EXCEPTION:

```
BEGIN
 RAISE INV_DEPTNO;
```

ORCL WILL STOP THE PROG IN BEGIN BLOCK AND GO THE EXCEPTION.

3.HANDLE THE EXCEPTION IN THE EXCEPTION BLOCK.

```
EXCEPTION
 WHEN INV_DEPTNO THEN
  RAISE_APPLICATION_ERROR(-20004,'Deptno does not exists');
```

```
DECLARE
V_DEPTNO DEPT.DEPTNO%TYPE;
DEL_MAS EXCEPTION;
 PRAGMA EXCEPTION_INIT(DEL_MAS,-02292);
```

```
 INV_DEPTNO EXCEPTION;
BEGIN
 V_DEPTNO:=&DEPTNO;
  DELETE FROM DEPT WHERE DEPTNO=V_DEPTNO;
   IF SQL%ROWCOUNT=0 THEN
    RAISE INV_DEPTNO;
ELSE

   DBMS_OUTPUT.PUT_LINE('REC DELETED');
END IF;
    COMMIT;
EXCEPTION
 WHEN INV_DEPTNO THEN
  RAISE_APPLICATION_ERROR(-20002,'Deptno does not exists');
 WHEN DEL_MAS THEN
   RAISE_APPLICATION_ERROR(-20001,'CANNOT DELETE THE MASTER AS CHILD FOUND');
END;
```

WHEN TO USE USER DEFINE EXCEPTION HANDLER: WHEN ORCL WILL NOT THROW THE EXCEPTION.

NOTE: WHEN ORCLE WILL NOT THROW EXCEPTION THEN USE USER DEFINED HANDLE.

NOTE: WHEN ORCL WILL THROW EXCEPTION THEN USE FIRST USE NAMED IF THERE ELSE USE UN-NAMED HANDLER.

NOTE:- IT IS ADVISABLE TO HANDLE AT LEAST ONCE EXCEPTION FOR ANY PL SQL PROGRAM.

ADVANTAGES OF NAMED BLOCK :-

1.STORE DATABASE.

2.COMPILE ONCE AND EXECUTE REPEATEDLY.

3.INVOKE THEM IN OTHER PLSQL PROGS.

1.PLSQL BLOCK.

2.OPERATORS

3.DATA TYPES

4.CONTROL STMTS

5.LOOPS

6.CURSORS

7.EXCEPTION HANDLING.

1.PROCEDURE:-

IS SET OF SQL OR PLSQL STATEMENTS THAT PERFORMS A TASK WHICH MAY OR MAY NOT RETURN A VALUE.

WHEN TO USE PROCEDURES??WHEN YOUR LOGIC ARE HAVING ANY DML STATMTS.

HOW TO CREATE??

CREATE [OR REPLACE] PROCEDURE <PRONAME>

[(<ARGMENTS>,..)]

IS

[<VAR>;]

BEGIN

....

[EXCEPTION

....]

END [<PRONAME>];

/=>COMPILE

PROCEDURE CREATED/PLSQL PROC CRE WITH COMPILITATION ERROR.

HOW TO SEE THE SYNTAX ERRORS??

SHO[W] ERR[OR]

ED

/

HOW TO EXECUTE??

SQL:

EXEC[UTE] <PROANME>(<ARGMTNSVAL>)

PLSQL:

BEGIN

<PRONAME>(<ARGVAL>);

END;

EX:

CREATE A PROCEDURE THAT WILL UDPATE AN EMPLOYEE SALARY BY 10%.

EXEC REC_UPDATE(7788)

CREATE OR REPLACE PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE)

IS

INV_EMPNO EXCEPTION;

```
BEGIN
 UPDATE EMP SET SAL=SAL+SAL*10/100 WHERE EMPNO=V_EMPNO;
 IF SQL%ROWCOUNT=0 THEN
RAISE INV_EMPNO;
ELSE
 DBMS_OUTPUT.PUT_LINE('REC UPDATED');
END IF;
 COMMIT;
EXCEPTION
 WHEN INV_EMPNO THEN
  RAISE_APPLICATION_ERROR(-20001,'Empno Does not exists');
END;
```

3 TYPES ARGUMENTS:-

1.IN:- VAL SENT TO THE PROG. WRITE ONLY.

2.OUT:- VAL SENT FROM THE PROG. READ ONLY.

3.IN OUT:- THE VAL SENT AND ALSO RETURN FROM THE PROG. READ AND WRITE.

NOTE:- DEFAULT IS IN.

CURSOR C1(V_DETPNO IN OUT EMP.DEPTNO%TYPE) IS SELECT..

EX:- CREATE A PROCEDURE THAT WILL INSERT THE EMPNO,ENAME,SAL INTO EMP TABLE.

```
CREATE OR REPLACE PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME
EMP.ENAME%TYPE,V_SAL EMP.SAL%TYPE)
IS
INV_EMPNO EXCEPTION;
V_COUNT BINARY_INTEGER;
```

```
BEGIN
SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE EMPNO=V_EMPNO;
 IF V_COUNT=0 THEN
 INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(V_EMPNO,V_ENAME,V_SAL);
DBMS_OUTPUT.PUT_LINE('REC INSERTED');
  ELSE
 RAISE INV_EMPNO;
END IF;
COMMIT;
EXCEPTION
WHEN INV_EMPNO THEN
  RAISE_APPLICATION_ERROR(-20001,'Empno already exists cannot insert');
END;

EXEC REC_INSERT(88,'ROCKY',2000)

BEGIN
REC_INSERT(101,'ROCKY',2000);
END;



----------
CREATE OR REPLACE PROCEDURE REC_DELETE(V_EMPNO EMP.EMPNO%TYPE)
IS
INV_EMPNO EXCEPTION;
BEGIN
DELETE FROM EMP WHERE EMPNO=V_EMPNO;
 IF SQL%ROWCOUNT=0 THEN
 RAISE INV_EMPNO;
 ELSE
 DBMS_OUTPUT.PUT_LINE('REC DELETED');
```

```
 END IF;
  COMMIT;
EXCEPTION
 WHEN INV_EMPNO THEN
  RAISE_APPLICATION_ERROR(-20002,'Empno Does not Exists for Delete');
END;
END;
```

==========

```
CREATE OR REPLACE PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE)
IS
INV_EMPNO EXCEPTION;
BEGIN
 UPDATE EMP SET SAL=SAL+SAL*10/100 WHERE EMPNO=V_EMPNO;
 IF SQL%ROWCOUNT=0 THEN
RAISE INV_EMPNO;
ELSE
 DBMS_OUTPUT.PUT_LINE('REC UPDATED');
END IF;
 COMMIT;
EXCEPTION
 WHEN INV_EMPNO THEN
  RAISE_APPLICATION_ERROR(-20001,'Empno Does not exists');
END;
```

```
SCOTT>>GRANT EXECUTE ON REC_UPDATE TO MYSAI;
```

```
MYSAI>>EXEC SCOTT.REC_UPDATE(7788)
```

NOTE: WITH PROCEDURE WE CAN ALSO HAVE SECURITY.

DEFINER: THE USER WHO OWNES THE PROCEDURE.

CURRENT_USER: THE USER WHO INVOKES THE PROCEDURE

MYSAI>>EXEC SCOTT.REC_UPDATE(7788)=>UPDATING THE SCOTT TABLE. BUT WE WANT TO UPDATE MYSAI EMP TABLE??YOU CHANGE THE RIGHTS OF THE DEFINER TO CURRENT_USER

```
CREATE OR REPLACE PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE)
 AUTHID CURRENT_USER
IS
INV_EMPNO EXCEPTION;
BEGIN
 UPDATE EMP SET SAL=SAL+SAL*10/100 WHERE EMPNO=V_EMPNO;
 IF SQL%ROWCOUNT=0 THEN
RAISE INV_EMPNO;
ELSE
 DBMS_OUTPUT.PUT_LINE('REC UPDATED');
END IF;
 COMMIT;
EXCEPTION
 WHEN INV_EMPNO THEN
  RAISE_APPLICATION_ERROR(-20001,'Empno Does not exists');
END;
```

MYSAI>EXEC SCOTT.REC_UPDATE(7788).

SO THE UPDATION WILL HAPPEN ON THE MYSAI TABLE NOT ON SCOTT TABLE BECAUSE RIGHTS HAVE CHANGED.

HOW TO SEE THE SOURCE CODE OF A PROCEDURE??

SCOTT>>SELECT TEXT FROM USER_SOURCE WHERE NAME='REC_UPDATE';

HOW TO DROP THE PROCEDURE??

DROP PROCEDURE REC_INSERT;

EX OF OUT ARGUMENT:

CREATE A PROCEDURE THAT RETURNS THE ENAME OF AN EMPLOYEE?

```
CREATE OR REPLACE PROCEDURE RET_ENAME(V_EMPNO EMP.EMPNO%TYPE,V_ENAME OUT
EMP.ENAME%TYPE)
IS
BEGIN
 SELECT ENAME INTO V_ENAME FROM EMP WHERE
  EMPNO=V_EMPNO;

EXCEPTION
 WHEN NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR(-20001,'Empno Does not Exists');
END;
```

SESSION VARIABLE:- VARIABLES THAT ARE DELCARED ON THE CLIENT SYSTEM AND CAN BE USED ONLY FOR THAT SESSION/WINDOW.

```
VAIABLE NAME VARCHAR2(40);
VARIABLE NO NUMBER

EXEC REC_ENAME(7788,:NAME)
PRINT :NAME
SELECT :NAME FROM DUAL;
```

EX: WITH IN OUT:

CREATE A PROCEDURE THAT WILL TAKE A MOB NUMBER EX: 9885140127 THEN RETURN (988)514-0127

```
CREATE OR REPLACE PROCEDURE RET_MOB(MOB IN OUT VARCHAR2)
IS
BEGIN
MOB:='('||SUBSTR(MOB,1,3)||')'||SUBSTR(MOB,4,3)||'-'||SUBSTR(MOB,7);
END;

VARIABLE V_MOB VARCHAR2(20)

EXEC :V_MOB:='1234567890'

EXEC RET_MOB(:V_MOB)

PRINT :V_MOB
==================

DECLARE
 V_EMPNO EMP.EMPNO%TYPE;
   V_ENAME EMP.ENAME%TYPE;
BEGIN
 V_EMPNO:=&EMPNO;
   RET_ENAME(V_EMPNO,V_ENAME);
DBMS_OUTPUT.PUT_LINE(V_ENAME);
END;
```

BEGIN

DBMNS...(V_ENAME);

END;

LOCAL VARIABLES:- CAN BE USED ONLY WITH THE GIVEN PROG.

NOTE:- WHEN U WANT TO INVOKE PROCEDURES THAT HAVE OUT ARGUMENTS FROM A SQL PROMPT??

DECLARE SESSION VAR.

==========

CAN WE INVOKE A PROCEDURE FROM ANOTHER PROCEDURE??? YES

EX: YOU HAVE THE REC_UPDATE PROCEDURE THAT DOES INCRASES AN EMPLOYEE SALARY BY 10%.

CREATE ANOTHER PROCEDURE CALL_RECUPDATE THAT WILL INVOKE THE REC_UPDATE PROCEDURE BY PASSING EVERY EMPNO TO IT?

WHAT IS WRAP?? IS A BATCH FILE USED TO WRAP THE SOURCE OF A GIVEN PROCEDURE.

1.COMPILE THE SOURCE CODE AND SAVE.

CREATE OR REPLACE PROCEDURE CALL_RECUPDATE

IS

CURSOR C1 IS SELECT EMPNO FROM EMP;

BEGIN

FOR REC IN C1 LOOP

REC_UPDATE(REC.EMPNO);

END LOOP;

END;

==========

WRAP:-

```
CREATE OR REPLACE PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE)
 AUTHID CURRENT_USER
IS
INV_EMPNO EXCEPTION;
BEGIN
 UPDATE EMP SET SAL=SAL+SAL*10/100 WHERE EMPNO=V_EMPNO;
 IF SQL%ROWCOUNT=0 THEN
RAISE INV_EMPNO;
ELSE
 DBMS_OUTPUT.PUT_LINE('REC UPDATED');
END IF;
 COMMIT;
EXCEPTION
 WHEN INV_EMPNO THEN
  RAISE_APPLICATION_ERROR(-20001,'Empno Does not exists');
END;
/
```

SAVE THE SOURCE CODE??

```
SQL>SAVE C:\MX.SQL
```

3.RUN WRAP FROM COMMAND PROMPT.

```
C:\>  WRAP INAME=MX.SQL ONAME=MM
```

4.RPLB FILE FROM SCOTT USER.

```
SCOTT>>START C:\MM.PLB
```

2.FUNCTIONS:-

SET OF STATEMENTS THAT PERFORMS A TASK AND WILL ALWAYS RETURN A VALUE.

1.SYSTEM DEFINED :-

FUNCTIONS THAT ARE OWNED BY SYSTEM/ORACLE

2.USER DEFINED :-

FUNCTIONS CREATED BY DEV/PROG.

WHEN TO USE FUNCTION?? FOR ANY CALCULATIONS

ADVANTAGE:-

USED FOR CAL.

HOW TO CRAETE A FUNCTION??

CREATE [OR REPLACE] FUNCTION <FNAME>[(<ARGMTNS>)] RETURN <DATA TYPE>

IS

[<LVAR>;]

BEGIN

..

..

RETURN <VAL>;

[EXCEPTION]

...

END [<FNAME>];

/

SHOW ERROR

1.SQL:

A.SQL STATEMENT:

SELECT <FNAME>(<ARVAL>) FROM DUAL/<TNAME>;

B.EXECUTE:

EXECUTE :<SVARNAME>:=<FNAME>(ARVAL)

PLSQL:

BEGIN

<LVAR>:=<FNAME>(<ARVAL>);

END;

SOURCE CODE:

SELECT TEXT FROM USER_SOURCE WHERE NAME='FNAME';

NOTE:- WE CAN WRAP A FUNCTION,WE HAVEA AUDITID CURRENT_USER ALSO FOR FUNCTIONS.

DROP FUNCTION <FNAME>;

EX: CREATE A FUNCTION THAT RETURNS THE A EMPLOYEE NETSAL(SAL+COMM)?

CREATE OR REPLACE FUNCTION CAL_NETSAL(V_EMPNO EMP.EMPNO%TYPE)

RETURN EMP.SAL%TYPE

IS

V_NETSAL EMP.SAL%TYPE;

BEGIN

SELECT SAL+NVL(COMM,0) INTO V_NETSAL FROM EMP WHERE EMPNO=V_EMPNO;

```
RETURN V_NETSAL;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR(-20002,'Empno does not exists');
END;
```

SQL:-

```
SELECT CAL_NETSAL(7654) FROM DUAL;
SELECT EMPNO,ENAME,SAL,COMM,CAL_NETSAL(EMPNO) FROM EMP;
```

EXECUTE:-

```
VARIABLE NETSAL NUMBER
EXEC :NETSAL:=CAL_NETSAL(7654)
PRINT :NETSAL
```

PLSQL:

```
DECLARE
 V_EMPNO EMP.EMPNO%TYPE;
  V_NETSAL EMP.SAL%TYPE;
BEGIN
 V_EMPNO:=&EMPNO;
   V_NETSAL:=CAL_NETSAL(V_EMPNO);
    DBMS_OUTPUT.PUT_LINE(V_NETSAL);
END;
```

```
SELECT TEXT FROM USER_SOURCE WHERE NAME='CAL_NETSAL';
```

DROP FUNCTION CAL_NETSAL;

EX: CREATE A FUNCTION THAT CALCULATES THE PCT OF SAL OF AN EMPLOYEE?

```
CREATE OR REPLACE FUNCTION CAL_PCT(V_EMPNO EMP.EMPNO%TYPE,PCT EMP.SAL%TYPE)
RETURN EMP.SAL%TYPE
IS
V_P  EMP.SAL%TYPE;
BEGIN
  SELECT SAL*PCT/100 INTO V_P FROM EMP WHERE EMPNO=V_EMPNO;
 RETURN V_P;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   RAISE_APPLICATION_ERROR(-20002,'Empno does not exists');
END;
/
SELECT CAL_PCT(7788,10) FROM DUAL;
```

HOW DECLARE SUB PROG WITH DEFAULT VALUE??

DEFAULT 'VAL'

```
CREATE OR REPLACE FUNCTION CAL_PCT(V_EMPNO EMP.EMPNO%TYPE,PCT EMP.SAL%TYPE
DEFAULT 5)
RETURN EMP.SAL%TYPE
IS
V_P  EMP.SAL%TYPE;
BEGIN
  SELECT SAL*PCT/100 INTO V_P FROM EMP WHERE EMPNO=V_EMPNO;
```

```
 RETURN V_P;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   RAISE_APPLICATION_ERROR(-20002,'Empno does not exists');
END;
```

CAN INVOKE A FUNCTION IN A PROCEDURE??

EX: CREATE A PROCEDURE THAT TAKES 3 ARUGMENTS FOR EMPNO,ENAME,SAL AND INSERT THEM IN THE EMP TABLE.

BEFORE U INSERT CALL THE FUNCTION THAT RETURN TRUE IF MAXSAL>SAL ELSE RETURN FALSE.

```
CREATE OR REPLACE FUNCTION CHK_MAXSAL
(NSAL EMP.SAL%TYPE) RETURN BOOLEAN
IS
 V_MAXSAL EMP.SAL%TYPE;
BEGIN
 SELECT MAX(SAL) INTO V_MAXSAL FROM EMP;
   IF V_MAXSAL>=NSAL THEN
    RETURN TRUE;
   ELSE
    RETURN FALSE;
  END IF;
END;
/
```

```
CREATE OR REPLACE PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME
EMP.ENAME%TYPE,V_SAL EMP.SAL%TYPE)
IS
```

```
 INV_SAL EXCEPTION;
BEGIN
 IF CHK_MAXSAL(V_SAL) THEN
  INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(V_EMPNO,V_ENAME,V_SAL);
   DBMS_OUTPUT.PUT_LINE('REC INSERTED');
 ELSE
 RAISE INV_SAL;
END IF;
COMMIT;
EXCEPTION
 WHEN INV_SAL THEN
  RAISE_APPLICATION_ERROR(-20002,'Nsal cannot be more than Maxsal');
END;
```

DIFFERENCE BETWEEN PROCEDURE AND FUNCTION??

1.PROCE FOR DML OPR/FUNCTION FOR CAL.

2.PROC MAY OR MAY RETURN A VALUE/FUNCTION MUST ALWAYS RETURNS VALUE.

3.PROCE WE CANNOT INVOKE EXECU NOT USING SQL ST/FUNCTION CAN INVOKE USING SQL/EXECU

4.IS PROC HAVING RETURN? FUNCTION RETURN??

```
CREATE OR REPLACE FUNCTION F1(V_EMPNO EMP.EMPNO%TYPE) RETURN NUMBER
IS
BEGIN
  INSERT INTO EMP(EMPNO) VALUES(V_EMPNO);
  RETURN 0;
END;
/

SELECT F1(102) FROM DUAL;
```

NOTE:- SOME FUNCTION CANNOT BE CALLED USING SQL STATEMENTS??THEY VIOLATE SQL STANDARDS. THEN HOW TO CALL??USE EXECUTE.

```
EXEC REC_INSERT(101,'ROCKY',9000)


CREATE OR REPLACE PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME
EMP.ENAME%TYPE,V_SAL EMP.SAL%TYPE)
IS
 INV_SAL EXCEPTION;
BEGIN
RETURN;
 IF TO_CHAR(SYSDATE,'DY')='TUE' THEN
RETURN;
ELSE
 IF CHK_MAXSAL(V_SAL) THEN
  INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(V_EMPNO,V_ENAME,V_SAL);
   DBMS_OUTPUT.PUT_LINE('REC INSERTED');
 ELSE
 RAISE INV_SAL;
END IF;
COMMIT;
END IF;
EXCEPTION
 WHEN INV_SAL THEN
  RAISE_APPLICATION_ERROR(-20002,'Nsal cannot be more than Maxsal');
END;
```

NOTE:- IN PROCEDURE WE CAN USE RETURN STATEMENT??IT WILL GRACEFULL EXIT THE PROG.

```
CREATE OR REPLACE PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE)
IS
INV_EMPNO EXCEPTION;
BEGIN
 UPDATE EMP SET SAL=SAL+SAL*10/100 WHERE EMPNO=V_EMPNO;
 IF SQL%ROWCOUNT=0 THEN
RAISE INV_EMPNO;
ELSE
 DBMS_OUTPUT.PUT_LINE('REC UPDATED');
END IF;
 COMMIT;
EXCEPTION
 WHEN INV_EMPNO THEN
  RAISE_APPLICATION_ERROR(-20001,'Empno Does not exists');
END;

/
```

THE ABOVE PROG HAS 2 OBEJCTS=>PROCEDURE AND TABLE.

1.REFERENCED OBJECT:- AN OBJECT THAT IS REFERRENCED BY OTHER OBJECT. EX: EMP IS REFERRED BY THE PROCE REC_UPDATE.

2.DEPENDENT OBJECT:- AN OBJECT THAT IS DEPENDENT ON THE OTHER OBJECT. EX: REC_UPDATE PROCEDURE DEPENDS ON EMP OBJECT.

NOTE:- AN STRUCTURAL CHANGES MADE TO THE REFERENCED OBJECT THEN ALL THE DEPENDENT OBJECTS BECOME INVALID.(THEY MUST ME RECOMPILED ONCE AGAIN).

EX: ALTER TABLE EMP ADD(ADDRESS VARCHAR2(20));

HOW TO VARIFY??

SELECT STATUS FROM USER_OBJECTS WHERE OBJECT_NAME='REC_UPDATE';

HOW TO MAKE THE PROCEDURE VALID??

1.ALTER PROCEDURE REC_UPDATE COMPILE;

ONLY COMPILE.

2.EXEC REC_INSERT(7788)=>COMPILE=>EXEC

USER_DEPENDENCIES

SELECT NAME FROM USER_DEPENDENCIES WHERE REFERENCED_NAME='EMP';

==========

3.PACKAGE:-

COLLECTION OF VARIOUSE PROCEDURE,FUNCTIONS(METHODS),VAR,CURSORS THAT CAN BE DECLARED LOCALLY OR GLOBALLY.

ADV:-

1.EXECUTION ARE FASTER, WE CAN REDUCE THE NUMBER OF I/O ON THE HARD DISK.

LOCAL:- THE MEMBERS OF THE PACKAGE CAN USE THE LOCAL METHODS ONLY. OTHER PLSQL PROG WHO ARE NOT THE MEMBERS CANNOT USE IT.

ADV: SECURITY.

GLOBAL:- THE METHODS OR VAR OR CURSOR CAN BE USED OR CALLED IN OTHER PLSQL PROGS. ADV?? CAN BE USED IN OTHER PROG.

2 TYPES OF PACKAGES

1.SYSTEM DEFINED:-

DBMS_OUTPUT.PUT_LINE

2.USER DEFINED:- CREATED BY THE DEVELOPER.

2 SECTIONS:-

1.PACKAGE SPECIFICATION:-

IT CONTAINS ALL THE PROC,FUNC,CUR,VAR THAT CAN BE DECLARED GLOBALLY. THEY CAN BE CALLED BY OTHER PLSQL PROG.

2.PACKAGE BODY:-

WE DECLARE THE LOCAL METHODS AND THEIR LOGICS. THEY ALSO THE LOGICS FOR YOUR GLOBAL PROC/FUNCTION DECLARED IN THE PACKAGE SPECIFICATION.

PK1:

SPEC:

REC_INSERT()

REC_UPDATE()

REC_DELETE()

/

BODY:

CAL_MAXSAL

....

...

REC_INSERT.

INSERT....

REC_UPDATE

..UPDATE ...

DEL..

..

/

HOW TO CREATE THE PACKAGE SPEC??

CREATE OR REPLACE PACKAGE PK1

IS

PROCE....;

FUNCTION...;

CURSORS...;

VAR...;

END <PKNAME>;

/

SHOW ERROR

EX:

CREATE OR REPLACE PACKAGE PK1

IS

PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL EMP.SAL%TYPE);

END;

/

```
CREATE OR REPLACE PACKAGE BODY PK1
IS
 V_COUNT BINARY_INTEGER;
FUNCTION CAL_MAXSAL(NSAL EMP.SAL%TYPE) RETURN
BOOLEAN
IS
V_MAXSAL EMP.SAL%TYPE;
BEGIN
  SELECT MAX(SAL) INTO V_MAXSAL FROM EMP;
    IF V_MAXSA>=NSAL THEN
      RETURN TRUE;
    ELSE
   RETURN FALSE;
END IF;
END;
PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)
IS
BEGIN
  SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE
    EMPNO=V_EMPNO;
  IF V_COUNT=0 THEN
   IF CAL_MAXSAL(V_SAL) THEN
    INSERT....
    ELSE
    RAISE...
   END IF;
 ELSE
  RAISE...;
END IF;
```

3.PACKAGES:-

GROUP OF COLLECTION OF PROCEDURES,FUNCTIONS,CURSORS,VAR THAT CAN BE DECLARED LOCALLY OR GLOBALL.

LOCALLY:- THE METHODS(PROC/FUNCTION),CURSOR,VAR CAN BE ONLY USED WITH THE PACKAGE.

GLOBALLY:- WE CAN CALL THEM FROM OTHER PROG.

TYPES:

1.SYSTEM DEFINED:- GIVEN ORCL. EX: DBMS_OUTPUT.PUT_LINE

2.USER DEFINED:- CREATED BY THE DEVELOPER.

2 SECTIONS:-

1.PACKAGE SPECIFICATION:-

IN THIS SECTION WE CAN DEFINE PROC,FUNC,CURS,VAR AS GLOBAL. THEY ONLY CONTAIN PROTYPE/SPEC/NO CODE/NO LOGIC.

2.PACKAGE BODY:-

WE CAN DECLARE LOCAL METHODS,VAR,CURSOR AND ALSO LOGIC FOR THE METHODS. IT ALSO CONTAIN LOGICS FOR THE GLOBAL METHODS.

PACKAGE SPEC AND BODY MUST HAVE THE SAME NAME.

EX: CREATE A PACKAGE PK1 WITH 3 PROCEDURES REC_INSERT,REC_UPDATE,REC_DELETE.

ALSO A LOCAL FUNCTION CAL_MAXSAL

CREATE OR REPLACE PACKAGE PK1

```
IS
PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE);
END;
/

----------

CREATE OR REPLACE PACKAGE BODY PK1
IS
V_COUNT BINARY_INTEGER;
FUNCTION CAL_MAXSAL(NSAL EMP.SAL%TYPE) RETURN BOOLEAN
IS
V_MAXSAL EMP.SAL%TYPE;
BEGIN
 SELECT MAX(SAL) INTO V_MAXSAL FROM EMP;
  IF V_MAXSAL>=NSAL THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;

  END IF;
END;
PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)
IS

BEGIN
 SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE
  EMPNO=V_EMPNO;

   IF V_COUNT=0 THEN
```

```
    IF CAL_MAXSAL(V_SAL) THEN
      INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(V_EMPNO,V_ENAME,V_SAL);
DBMS_OUTPUT.PUT_LINE('REC INSERTED');
ELSE
 RAISE_APPLICATION_ERROR(-20001,'Newsal    cannot be more then Maxsal');
END IF;
ELSE
 RAISE_APPLICATION_ERROR(-20002,'Cannot   Insert Empno already exists');
END IF;
 COMMIT;
END;
END;
/

SHOW ERROR
------
```

MODIFY THE PROCEDURE TO ADD REC_UPDATE??

```
EXEC PK1.REC_INSERT(101,'SHAM',9000)

EXEC PK1.REC_INSERT(7788,'SHAM',2000)
```

```
CREATE OR REPLACE PACKAGE PK1

IS

PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE);

PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE);

END;

/



CREATE OR REPLACE PACKAGE BODY PK1

IS

V_COUNT BINARY_INTEGER;

FUNCTION CAL_MAXSAL(NSAL EMP.SAL%TYPE) RETURN BOOLEAN

IS

V_MAXSAL EMP.SAL%TYPE;

BEGIN

  SELECT MAX(SAL) INTO V_MAXSAL FROM EMP;

    IF V_MAXSAL>=NSAL THEN

      RETURN TRUE;

    ELSE

      RETURN FALSE;


    END IF;

 END;

PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)

IS


BEGIN

  SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE

    EMPNO=V_EMPNO;
```

```
    IF V_COUNT=0 THEN
     IF CAL_MAXSAL(V_SAL) THEN
       INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(V_EMPNO,V_ENAME,V_SAL);
 DBMS_OUTPUT.PUT_LINE('REC INSERTED');
 ELSE
  RAISE_APPLICATION_ERROR(-20001,'Newsal    cannot be more then Maxsal');
 END IF;
 ELSE
  RAISE_APPLICATION_ERROR(-20002,'Cannot   Insert Empno already exists');
 END IF;
  COMMIT;
 END;
PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)
IS
BEGIN
 SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE EMPNO=V_EMPNO;
 IF V_COUNT>0 THEN
   IF CAL_MAXSAL(V_SAL) THEN
     UPDATE EMP SET ENAME=V_ENAME,SAL=V_SAL        WHERE EMPNO=V_EMPNO;
      DBMS_OUTPUT.PUT_LINE('REC UPDATED');
   ELSE
    RAISE_APPLICATION_ERROR(-20003,'Cannot     update new sal as more than Maxsal');
   END IF;
 ELSE
  RAISE_APPLICATION_ERROR(-20004,'Cannot  update as empno not found');
 END IF;
 COMMIT;
END;
END;
```

```
EXEC PK1.REC_UPDATE(101,'RAVI',3000);


BEGIN
PK1.REC_UPDATE(&ENO,'&ENAME',&SAL);
END;


----------
```

MODIFY THE PACKAGE FOR DELETE??

```
CREATE OR REPLACE PACKAGE PK1
IS
PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE);
PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE);
PROCEDURE REC_DELETE(V_EMPNO EMP.EMPNO%TYPE);
END;



CREATE OR REPLACE PACKAGE BODY PK1
IS
V_COUNT BINARY_INTEGER;
FUNCTION CAL_MAXSAL(NSAL EMP.SAL%TYPE) RETURN BOOLEAN
IS
V_MAXSAL EMP.SAL%TYPE;
BEGIN
 SELECT MAX(SAL) INTO V_MAXSAL FROM EMP;
   IF V_MAXSAL>=NSAL THEN
    RETURN TRUE;
   ELSE
```

```
    RETURN FALSE;


  END IF;
END;
PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)
IS


BEGIN
 SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE
  EMPNO=V_EMPNO;


   IF V_COUNT=0 THEN
    IF CAL_MAXSAL(V_SAL) THEN
     INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(V_EMPNO,V_ENAME,V_SAL);
DBMS_OUTPUT.PUT_LINE('REC INSERTED');
ELSE
 RAISE_APPLICATION_ERROR(-20001,'Newsal   cannot be more then Maxsal');
END IF;
ELSE
 RAISE_APPLICATION_ERROR(-20002,'Cannot   Insert Empno already exists');
END IF;
 COMMIT;
END;
PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)
IS
BEGIN
SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE EMPNO=V_EMPNO;
IF V_COUNT>0 THEN
  IF CAL_MAXSAL(V_SAL) THEN
    UPDATE EMP SET ENAME=V_ENAME,SAL=V_SAL       WHERE EMPNO=V_EMPNO;
```

```
     DBMS_OUTPUT.PUT_LINE('REC UPDATED');
   ELSE
    RAISE_APPLICATION_ERROR(-20003,'Cannot     update new sal as more than Maxsal');
   END IF;
  ELSE
  RAISE_APPLICATION_ERROR(-20004,'Cannot  update as empno not found');
 END IF;
 COMMIT;
END;
PROCEDURE REC_DELETE(V_EMPNO EMP.EMPNO%TYPE)
IS
BEGIN
  SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE    EMPNO=V_EMPNO;
    IF V_COUNT>0 THEN
      DELETE FROM EMP WHERE EMPNO=V_EMPNO;
        DBMS_OUTPUT.PUT_LINE('REC DELETED');
      ELSE
       RAISE_APPLICATION_ERROR(-20005,'Cannot    Delete as EMpno not found');
  END IF;
  COMMIT;
 END;
END;
/
```

WHAT ERROR ERRORS WE WILL GET??U CAN JUST DECLARE THE METHODS ON THE TOP AND LATER WRITE CODE FOR THOSE METHODS. FORWARD DECLARATION.

==========

```
EXIT
SQLPLUS
EXEC PK1.REC_UPDATE....
```

WELCOME SCOTT

REC UPDATE.

EXEC PK1.REC_DELETE(101)

LOCAL:-

```
BEGIN
  IF PK1.CAL_MAXSAL(2000) THEN
    DBMS_OUTPUT.PUT_LINE('TRUE');
END IF;
END;
```

GLOBAL:-

```
CREATE OR REPLACE PACKAGE PK1
IS
PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE);
PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE);
PROCEDURE REC_DELETE(V_EMPNO EMP.EMPNO%TYPE);
CURSOR C1 IS SELECT EMPNO FROM EMP;
 V_GLOBAL BINARY_INTEGER;
END;
/
```

```
BEGIN
 FOR REC IN PK1.C1 LOOP
   DBMS_OUTPUT.PUT_LINE(REC.EMPNO);
END LOOP;
END;
/
----------
BEGIN
 SELECT COUNT(*) INTO PK1.V_GLOBAL FROM EMP;
  DBMS_OUTPUT.PUT_LINE(PK1.V_GLOBAL);
END;
----------
SCOTT>> SELECT TEXT FROM USER_SOURCE WHERE NAME='PK1';



SCOTT>>GRANT EXECUTE ON PK1 TO MYSAI;
MYSAI>>EXEC SCOTT.PK1.REC_INSERT(103,'R',100);


MYSAI>>SELECT TEXT FROM ALL_SOURCE WHERE NAME='PK1' AND OWNER='SCOTT';
```

NOTE:- BY DEFINING METHODS IN PACKAGES WE CAN GAIN SECURITY BY NOT ALLOWING OTHER DB USERS TO SEE OUR SOURCE CODE.

BODY:-

CAL_MAXSAL

REC_INSERT

REC_UPDATE

REC_INSERT

=======

ONE TIME PROCEDURE:-

IT IS AN OPTIONAL PIECE CODE IN A PACKAGE.

IT WILL EXECUTE ONLY ONCE FOR EVERY SESSION.

EXEC PK1.REC_INSERT(...)

WELECOME SCOTT

REC INSERT...

EXEC PK1.REC...().

REC INSERT.

EX:

```
CREATE OR REPLACE PACKAGE BODY PK1
IS
V_USER VARCHAR2(40);
V_COUNT BINARY_INTEGER;
FUNCTION CAL_MAXSAL(NSAL EMP.SAL%TYPE) RETURN BOOLEAN
IS
V_MAXSAL EMP.SAL%TYPE;
BEGIN
  SELECT MAX(SAL) INTO V_MAXSAL FROM EMP;
    IF V_MAXSAL>=NSAL THEN
      RETURN TRUE;
    ELSE
      RETURN FALSE;

    END IF;
 END;
```

```
PROCEDURE REC_INSERT(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)
IS

BEGIN
 SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE
   EMPNO=V_EMPNO;

    IF V_COUNT=0 THEN
     IF CAL_MAXSAL(V_SAL) THEN
       INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(V_EMPNO,V_ENAME,V_SAL);
 DBMS_OUTPUT.PUT_LINE('REC INSERTED');
 ELSE
  RAISE_APPLICATION_ERROR(-20001,'Newsal    cannot be more then Maxsal');
 END IF;
 ELSE
  RAISE_APPLICATION_ERROR(-20002,'Cannot   Insert Empno already exists');
 END IF;
  COMMIT;
 END;
PROCEDURE REC_UPDATE(V_EMPNO EMP.EMPNO%TYPE,V_ENAME EMP.ENAME%TYPE,V_SAL
EMP.SAL%TYPE)
IS
BEGIN
 SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE EMPNO=V_EMPNO;
 IF V_COUNT>0 THEN
  IF CAL_MAXSAL(V_SAL) THEN
    UPDATE EMP SET ENAME=V_ENAME,SAL=V_SAL        WHERE EMPNO=V_EMPNO;
     DBMS_OUTPUT.PUT_LINE('REC UPDATED');
  ELSE
   RAISE_APPLICATION_ERROR(-20003,'Cannot     update new sal as more than Maxsal');
  END IF;
```

```
 ELSE
 RAISE_APPLICATION_ERROR(-20004,'Cannot  update as empno not found');
END IF;
COMMIT;
END;
PROCEDURE REC_DELETE(V_EMPNO EMP.EMPNO%TYPE)
IS
BEGIN
 SELECT COUNT(*) INTO V_COUNT FROM EMP WHERE    EMPNO=V_EMPNO;
   IF V_COUNT>0 THEN
     DELETE FROM EMP WHERE EMPNO=V_EMPNO;
       DBMS_OUTPUT.PUT_LINE('REC DELETED');
     ELSE
      RAISE_APPLICATION_ERROR(-20005,'Cannot    Delete as EMpno not found');
 END IF;
 COMMIT;
END;
BEGIN
 SELECT USER INTO V_USER FROM DUAL;
 DBMS_OUTPUT.PUT_LINE('WELCOME '||V_USER);
END;
```

```
ALTER PACKAGE PK1 COMPILE SPECIFICATION/BODY/PACKAGE;
```

```
DROP PACKAGE PK1;

DROP PACKAGE [BODY] PK1;
```

NOTE: WE CAN SPEC WITHOUT BODY(EX: ONLY FOR GVAR OR GCURS).

==========

PK1

PK SPEC BOTH.

PK BODY:(ONLY)

==========

TRIGGERS:-

PROCEDURE/FUNCTION/PACKAGE??HOW ARE THE CALLED? EXPLICITLY

EXEC PK1.REC_INSERT...

EXEC PK1.REC_UPATE...

SELECT CAL_NETSAL(EMPNO) FROM EMP;

TRIGGER: THEY CALLED IMPLICITY.

INSERT INTO EMP....();

TRIGGER: IS SUB PROGRAM THAT WILL EXECUTE IMPLICITY WHEN DML/DDL ON TABLE/BY USER.

ADV:-

1.SECURITY.

2.RESTRICTION.

3.AUDIT.

4.INTIGRITY: WITHOUT PK/FK.

1.NAME: CHK_INSERT

2.TIMING: BEFORE: FIRST EXECUTES YOUR TRIG THEN TR.

AFTER: FIRST EXEC TR AND THEN TRIG.

3.TR: INSERT...

4.ON WHICH COLUMNS:

5.ON TABLE/VIEW: EMP,DEPT,...V1.

TYPES OF TRIGGER:-

1.DML TRIGGERS: INSERT OR UPDATE OR DELETE.

2.INSTEAD OF TRIGGER: FOR VIEWS

3.DDL/SYSTEM TRIGGER: CREATE OR ALTER OR DROP.

4.EVENT TRIGGER: LOGOFF/LOGON

LEVEL OF TRIGGER:-

1.STATEMENT/TABLE LEVEL: ONLY ONCE.

2.ROW LEVEL: FOR EACH ROW.

UPDATE EMP SET SAL=2000;

EX:

CREATE A DML TRIGGER THAT WILL RESTRICT THE USER FROM PERFORM DML OPER ON EMP TABLE.

```
CREATE OR REPLACE TRIGGER CHK_DML BEFORE
INSERT OR UPDATE OR DELETE  ON EMP
BEGIN
  RAISE_APPLICATION_ERROR(-20001,'Cannot perform dml oper');
END;

SHOW ERR
```

HOW TO INVOKE THE TRIGGER??

HOW TO GIVE DIFF MSG??

```
CREATE OR REPLACE TRIGGER CHK_DML BEFORE
INSERT OR UPDATE OR DELETE  ON EMP
BEGIN
 IF INSERTING THEN
   RAISE_APPLICATION_ERROR(-20001,'Cannot insert');
ELSIF UPDATING THEN
RAISE_APPLICATION_ERROR(-20002,'Cannot UPDATE');
ELSIF DELETING THEN
 RAISE_APPLICATION_ERROR(-20003,'Cannot DELETE');
END IF;
END;

DELETE FROM EMP;
```

EX: WRITE A TRIG THAT WILL RESTRICT THE USER FROM INSERT ON EMP ON SUNDAY?

```
CREATE OR REPLACE TRIGGER CHK_DAY BEFORE INSERT  ON EMP
BEGIN
  IF TO_CHAR(SYSDATE,'DY')='WED' THEN
   RAISE_APPLICATION_ERROR(-20001,'Holiday');
 END IF;
END;
```

EX:

CREATE TABLE EMP_AUDIT(EMPNO INTEGER,USERNAME VARCHAR2(20),TRDATE TIMESTAMP,TRTYPE VARCHAR2(20))

EX: CRAATE A TRIGGER FOR INSERT OR UPDATE OR DELETE ON EMP. CAP THE REQ INFO AND INSERT INTO EMP AUDIT.

```
CREATE OR REPLACE TRIGGER CHK_AUDIT
AFTER INSERT OR UPDATE OR DELETE ON EMP
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;

BEGIN
IF INSERTING THEN
INSERT INTO EMP_AUDIT VALUES(:NEW.EMPNO,USER,SYSTIMESTAMP,'INSERT');
ELSIF UPDATING THEN
INSERT INTO EMP_AUDIT VALUES(:OLD.EMPNO,USER,SYSTIMESTAMP,'UPDATE');
ELSE
INSERT INTO EMP_AUDIT VALUES(:OLD.EMPNO,USER,SYSTIMESTAMP,'DELETE');
END IF;
COMMIT;
END;
/
```

WHAT IS AUTONOMOUS TRANSACTION:- WE CAN MAKE THE TR IN THE TRIGGER AUTONMOUS BY USING COMMIT. THIS COMMIT IS NOT APPLICABLE FOR OTHER TR OUTSIDE THE TRIGGER.

```
INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(1111,'SHAM',3000)
```

SYSTEM VARIABLES:

INSERT:

:NEW.EMPNO

UPDATE:

:NEW.EMPNO

:OLD.EMPNO

DELETE:

:OLD.EMPNO

SQL: INSERT INTO EMP(EMPNO,ENAME,SAL) VALUES(101,'ROCK',2000);

==========

2.INSTEAD OF TRIGGER:- TRIGGER FOR VIEWS.

SCOTT>>DELETE FROM EMP WHERE JOB IS NULL;

SCOTT>>ALTER TABLE EMP MODIFY(JOB NOT NULL);

SCOTT>>CREATE OR REPLACE VIEW V1 AS SELECT EMPNO,ENAME,SAL FROM EMP;

SCOTT>>GRANT ALL ON V1 TO MYSAI;

MYSAI>>INSERT INTO V1 VALUES(101,'RAJ',1000);

NOT WORK??NOT NULL FOR JOB.

SCOTT>>

```
CREATE OR REPLACE TRIGGER CHK_VIEW INSTEAD OF INSERT ON VVX1 FOR EACH ROW
BEGIN
 INSERT INTO EMP(EMPNO,ENAME,SAL,JOB) VALUES(:NEW.EMPNO,:NEW.ENAME,:NEW.SAL,'UNK');
END;
```

INSERT INTO VVX1 VALUES(101,'RAJ',1000);

CREATE OR REPLACE VIEW V2 AS SELECT EMPNO,ENAME,SAL,EMP.DEPTNO,DNAME,LOC FROM EMP,DEPT WHERE EMP.DEPTNO=DEP.DEPTNO;

INSERT INTO V2 VALUES(110,'RAJA',2500,11,'COMPUTERS','HDC');

==========

3.DDL TRIGGERS:-

CREATE A TRIGGER FOR CREATE WHICH WILL CAPTURE THE UNAME,CRDATE,OBJTYPE,OBJNAME.

SCOTT>>CREATE TABLE SCOTT_OBJS(USERNAME VARCHAR2(20),CREDATE TIMESTAMP,OBJTYPE VARCHAR2(30),OBJNAME VARCHAR2(30));

NOTE: ALL DDL TRIG ARE OF STATEMENT/TABLE LEVEL.

```
CREATE OR REPLACE TRIGGER CHK_CREATE AFTER CREATE  ON SCOTT.SCHEMA
BEGIN
 INSERT INTO SCOTT_OBJS
VALUES(USER,SYSTIMESTAMP,ORA_DICT_OBJ_TYPE,ORA_DICT_OBJ_NAME);
END;
```

CREATE MX12(CID INTEGER);

==========

4.EVENT TRIGGERS:-

TRIGGERS FOR LOGON OR LOGOFF

AFTER LOGON

BEFORE LOGOFF

```
CREATE OR REPLACE TRIGGER CHK_LOGON AFTER LOGON ON SCOTT.SCHEMA
BEGIN
EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_LANGUAGE=''FRENCH''';
END;
/
```

NOTE: TO USE DDL COMMANDS IN PLSQL WE WILL USE EXECUTE IMMEDIATE '<<DDCOMMAN>>';

.===============

ALTER TRIGGER CHK_DML COMPILE;

ALTER TRIGGER CHK_DML DISABLE/ENABLE;

DROP TRIGGER CHK_DML;

USER_TRIGGERS

MUTATING TRIGGERS:-

```
CREATE OR REPLACE TRIGGER CHK_DML AFTER INSERT ON EMP FOR EACH ROW
BEGIN
 INSERT INTO EMP VALUES(:NEW.....);
END;
```

SQL>INSERT INTO EMP VALUES...;

==========

SQL LOADER:-

IT IS UTILITY/TOOL OWNED BY ORCL WHICH IS USED TO LOAD DATA FROM A EXTERNAL TEXT FILE INTO ORCL DB TABLE.

WHAT IS REQ TO WORK WITH SQL LOADER:

1.SOURCE FILE:- A FILE THAT HAS DATA TO LOADED INTO THE ORCL DB TABLE.

2.TABLE:- THE DESTINATION TABLE THAT WILL CONTAIN DATA FROM THE EXTERNAL TEXT FILE.

3.CONTROL FILE:- IS A READABLE FILE THAT CONTAINS THE INFORMATION OF THE SOURCE FILE THAT CONTAINS DATA,LOCATIONS.., ALSO THE DATABASE TABLE.

4.LOG FILE:- GIVES THE LOADING INFORMATION.ROWS LOAD,ROWS REJECT,ANY ERRORS...

==========

1.CREATE THE SOURCE FILE.

OPEN NOTEPAD AND ENTER THE SAMPLE DATA.

101,"ROCKY",2000

102,"RAJ",3000

103,"JACK",4000

SAVE IN YOUR LOCAL FOLDER. mysrc

2.IN DB

SCOTT>>CREATE TABLE E_EMP(EMPNO INTEGER,ENAME VARCHAR2(20),SAL NUMBER);

3.CREATE THE CONTROL FILE.

load data

infile 'c:\william1\mysrc.txt'

insert into table e_emp fields terminated by ',' optionally enclosed by '"' trailing nullcols(empno,ename,sal)

SAVE THE ABOVE CONTENTS A SEPARATE FILE(myctl.ctl). EITHER IN DOUBLE QUOTES OR UNDER ALL FILES.

4.RUN SQLLOADER.

CD william1

c:\william1>sqlldr control=myctl.ctl log=mylog.log

username: scott/tiger@orcl

5.

SCOTT>>SELECT * FROM E_EMP;

IF WE RERUN THE LOADER ONCE,WE WILL GET ERROR?? FOR INSERT OPTION TABLE MUST BE EMPTY.

WHAT ARE THE OTHER OPTIONS??

2.TRUNCATE:

IT WILL FIRST EMPTY THE TABLE IF REC ARE THERE AND THEN LOADS THE DATA FROM THE TEXT F ILE.

load data

infile 'c:\william1\mysrc.txt'

truncate into table e_emp fields terminated by ',' optionally enclosed by '"' trailing nullcols(empno,ename,sal)

3.APPEND:-THIS WILL RETAIN THE OLD REC IN THE TABLE AND APPEND NEW REC FROM THE SOURCE FILE.

load data

infile 'c:\william1\mysrc.txt'

append into table e_emp fields terminated by ',' optionally enclosed by '"' trailing nullcols(empno,ename,sal)

OPTIONAL FILES:

1.BAD FILE: THIS FILE WILL CAPTURE ALL THE RECORD FROM THE SOURCE FILE WHICH HAVE VIOLATED SOME RESTRICTIION, EX: CONSTRAINTS/DATA MISMATCHED.

EX:

101,"ROCKY1",4000

102,"JAY",

103,"RAJ",5000

SCOTT>>ALTER TABLE E_EMP MODIFY(SAL NOT NULL);

load data

infile 'c:\william1\mysrc.txt'

badfile 'c:\william1\mysrc.bad'

append into table e_emp fields terminated by ',' optionally enclosed by '"' trailing nullcols(empno,ename,sal)

WHEN CONDITION:

WE CAN GIVE CONDITIONS WHILE LOADING DATA FROM EXTERNAL TEXT FILE INTO ORCL DB TABLE.

load data

infile 'c:\william1\mysrc.txt'

badfile 'c:\william1\mysrc.bad'

discardfile 'c:\william1\mysrc.dsc'

truncate into table when empno<>"101" e_emp fields terminated by ',' optionally enclosed by '"' trailing nullcols(empno,ename,sal)

DISCARD FILE:- THIS FILE THAT CONTAINS THAT RECORDS THAT DID NOT SATISFY THE WHEN CONDITION.

CONCLUSION: TOOL USED TO LOAD DATA FROM EXTERNAL TEXT FILE INTO THE ORCL DB TABLE.

==========

UTL_FILE:-

IT IS PACKAGE GIVEN BY ORCL USED TO READ OR WRITE DATA FROM EXTERNAL TEXT FILES.

READ: WE CAN READ THE CONTENTS OF THE FILE USING PLSQL.

WRITE: WE CAN WRITE DATA FROM A DB TABLE INTO THE EXTERNAL TEXT FILE.

1.UTL_FILE.FOPEN:

THIS METHOD WILL PROVIDE USE TO LOCATE,NAME OUR FILE AND ALSO TO READ OR WRITE.

TAKES 3 ARGUMENTS??

1.LOCATIONS,2FILE NAME,3. MODE: READ/WRITE/APPEND

EX:

VAR1:=UTL_FILE.FOPEN('C:\WILL...','A1.TXT','r/w/a

);

r/a=> THE FILE MUST ALREADY EXISTS

w=> WILL CREATE A NEW FILE.

NOTE: THIS METHOD IS A FUNCTION.

2.UTL_FILE.FILE_TYPE:- IT IS METHOD FOR DATA TYPE FOR FILE TYPE.

EX:

VAR1 UTL_FILE.FILE_TYPE;

3.UTL_FILE.PUTF:- METHOD USED TO WRITE THE CONTENTS IN THE FILE. 2 ARGUMENTS.

1.VAR1,2 WHAT TO WRITE.

EX:

UTL_FILE.PUTF(VAR1,'WELCOME TO ORCL');

NOTE: IT IS A PROCEDURE.

4.UTL_FILE.GET_LINE:- THIS METHOD WILL READ THE CONTENTS FROM THE FILE.TAKES 2 ARGUMENTS:

1VAR1,VARFOROUTPUT

EX:

UTL_FILE.GET_LINE(VAR1,MYVAR1);

5.UTL_FILE.FCLOSE:- USED TO CLOSE THE FILE.ONLY ONE. VAR1

EX:

UTL_FILE.FCLOSE(VAR1);

==========

DECLARE

VAR1 UTL_FILE.FILE_TYPE;

BEGIN

VAR1:=UTL_FILE.FOPEN('C:\...','A1.TXT','w');

UTL_FILE.PUTF(VAR1,'welcome to orcl');

UTL_FILE.FCLOSE(VAR1);

END;

CREATE DIRECTORY:-

WHO WILL CREATE DIRECTORY IN DB??

NOTE: FIRST CREATE A FOLDER IN WINDOWS.(EX:WILLIAM1).

SYS>>CREATE DIRECTORY D1 AS 'C:\WILLIAM1';

SYS>>GRANT READ,WRITE ON DIRECTORY D1 TO SCOTT;

SCOTT>>

DECLARE

VAR1 UTL_FILE.FILE_TYPE;

BEGIN

VAR1:=UTL_FILE.FOPEN('D1','A1.TXT','w');

UTL_FILE.PUTF(VAR1,'welcome to orcl');

UTL_FILE.FCLOSE(VAR1);

END;

```
DECLARE
VAR1 UTL_FILE.FILE_TYPE;
MYVAR1 VARCHAR2(100);
BEGIN
VAR1:=UTL_FILE.FOPEN('D1','A1.TXT','r');
UTL_FILE.GET_LINE(VAR1,MYVAR1);
DBMS_OUTPUT.PUT_LINE(MYVAR1);
UTL_FILE.FCLOSE(VAR1);
END;
```

-EX: CREATE A PROCEDURE THAT WILL WRITE THE CONTENTS OF EMPNO INTO A EXTERNAL TEXT FILE A2.TXT

```
CREATE OR REPLACE PROCEDURE WRITE_EMPNO
IS
VAR1 UTL_FILE.FILE_TYPE;
CURSOR V_C1 IS SELECT EMPNO FROM EMP;
BEGIN
VAR1:=UTL_FILE.FOPEN('D1','A2.TXT','w');

FOR REC IN V_C1 LOOP
UTL_FILE.PUTF(VAR1,REC.EMPNO||'\n');
END LOOP;
UTL_FILE.FCLOSE(VAR1);
END;

------------------------------
CREATE OR REPLACE PROCEDURE READ_EMPNO
IS
MYVAR1 NUMBER;
VAR1 UTL_FILE.FILE_TYPE;
```

```
BEGIN
VAR1:=UTL_FILE.FOPEN('D1','A2.TXT','r');
FOR I IN 1..1000 LOOP
UTL_FILE.GET_LINE(VAR1,MYVAR1);
DBMS_OUTPUT.PUT_LINE(MYVAR1);
END LOOP;
UTL_FILE.FCLOSE(VAR1);
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('END OF LINE');
END;
```

==========