

# PL/SQL

**DECLARE**

**DECLARATION SECTION**

Optional

**BEGIN**

**BODY STATEMENTS**

.... Required

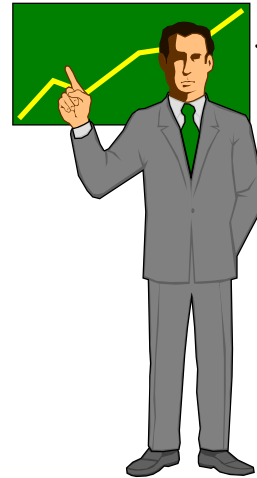
**EXCEPTION**

**STATEMENTS** Optional

**END;**

The above Construction is called PL/SQL BLOCK

## DATATYPES




- Binary Integer (-2 \*\*31-1,2\*\*31+1) signed integer fastest
  - CHAR(n) n up to 32767 note : 255 for RDBMS fields.
  - varchar2(n) n up to 32767 note : 2000 for RDBMS fields.
  - varchar(n) same as varchar2 note : not recommended.
  - long up to 32767 note: 2 GB for RDBMS fields.
  - raw up to 32767 note: 255 for RDBMS fields
  - long raw up to 32767 note: 2 GB for RDBMS fields
  - Boolean True, False or Null
  - Date Jan 1 4712 BC up to DEC 31, 4612 with time
  - Number(n,d) precision up to n=38 and d -84,127
  - Rowid stores Oracle rowids in readable format
- 
- float
  - Decimal
  - Numeric
  - Real
  - Integer

# DECLARATIONS

- Birthday Date;
- count integer := 6; -- Note initialized variable
- acc\_id varchar2(10) not null := 'SHC'

Note:- if a variable is declared not null it must be immediately initialized in declaration section

 acc\_id varchar2(10) not null --not valid

- Gravity constant Real := 9.8;
- Velocity Real := 2.0;
- Time Real := 1.0;
- F\_vel velocity - gravity x Time ;

Notes : A constant must be initialized in its declaration  
Which will be the constants final value

- my\_var number := 22 is identical to
- my\_var number **default** 22;
- %TYPE Tablename.Field%Type
- %ROWTYPE Tablename% Rowtype

The last two types are demonstrated in the following examples

## Examples :

```
1) Declare
    emp_hiredate          emp.hiredate%TYPE;
Begin
    select hiredate into emp_hiredate from emp
    where empno = 7499;
    if emp_hiredate .. etc
        .....
End;
```

```
2) Declare
    emp_rec    emp%rowtype;
begin
    select * into emp_rec
    from emp;
    if emp_rec.sal = 1200 Then
        .....
    end if;
end;
```

Same Dec. As

empno	emp.empno%type
ename	emp.ename%type
job	emp.job%type
mgr	emp.mgr%type
etc	
....	

```
3) Declare
    emp_rec    emp%rowtype;
    emp_rec2   emp_rec%rowtype
    cursor CC1 is select deptno from dept
    dept_rec   CC1%Rowtype;
```

Note:- *Cursors* Concepts will be introduced later

Note :-

Declare

```
x1 , X2 INTEGER ; -- Illegal
x1     INTEGER ; -- Legal
x2     INTEGER ; -- Legal
```

## SCOPE OF VARIABLES

Try the following from SQL Prompt  
SQL> SET SERVEROUTPUT ON

Declare

```
    A REAL;
```

Begin

```
    A := 5.2;
```

```
    DECLARE
```

```
        A Real;
```

```
    Begin
```

```
        A := 2.0;
```

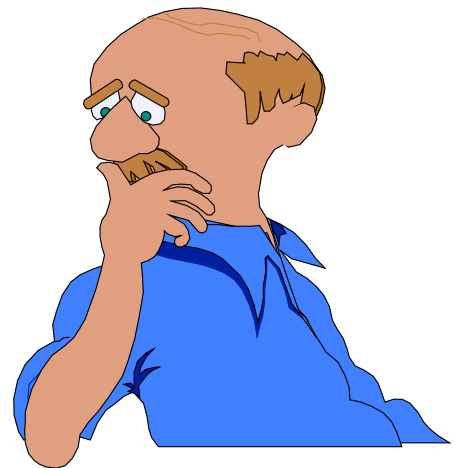
```
    END;
```

```
    DBMS_OUTPUT.PUT_LINE ('A = ' || A);
```

```
END;
```

```
A = 5.2.
```

⇒ The variable A declared in the inner block is different than the A declared in the outer block and is local for the inner block



```
DECLARE
    Iterate INTEGER;
BEGIN
    Iterate := Iterate +1 ;
    ....
END:
```

*Iterate will always be null because iterate is not initialized*

### Precedence when Evaluating Expressions

** , Not	↓ Decreasing
* , /	
+ , - ,	
= , != , < , > , etc. ..	
And	
Or	

Comparison with NULL yields NULL;  
NOT (NULL) = NULL

```
example  a:= Null;
          b := Null;
          IF a = b then
              dbms_output.put_line ('OK');
          End IF;
```

The message OK will not appear because the conditional statement will evaluate to Null

Note also that Zero Length strings are treated like null

Remember that if you wish to test for NULL use IS NULL

i.e. Select \* from emp where comm *IS NULL*;

## EXCEPTIONS:

In PL/SQL, a warning or error condition is called an *exception*. Exception can be internally defined (by runtime system) or user\_defined.

Examples of internally defined exceptions include *divide by zero* and *no data found*. Common internal exceptions have predefined name such as ZERO\_DIVIDE and NO\_DATA\_FOUND

When an error occurs, an exception is *raised*. That is, Normal execution stops and control transfers to the exception handling part of your PL/SQL Block.

To handle raised exceptions, you write separate routines *exception handlers*. After an exception handler runs, the current block stops executing.

Without exception handling, every time you issue a command, you must check for execution errors

## Example Using Exceptions

Declare

```
val number ;
```

Begin

```
select sal into val from emp where empno=9989;
```

```
if val < 800 then
```

```
    Update emp set sal=1000 where empno=9989;
```

```
end if;
```

Exception

When NO\_DATA\_FOUND Then

```
declare
```

```
    val1 number;
```

```
    val2 varchar2(50);
```

```
begin
```

```
    val1 := sqlcode;
```

```
    val2 := sqlerrm;
```

```
    insert into log values (val1,val2);
```

```
end;
```

End;





## Exercise

1-Write one SQL statement (NOT PL/SQL) that will update the EMP table so that all employees whose salary is more than 2000 will get 5% increase in salary and employees whose salary is less or equal to 2000 will get 10% increase.

2-Try the same thing using PL/SQL?

3- Write a PL/SQL block that will convert temperature from degrees C to degrees F. The range of temperature values are 0-35.

SOLUTION FOR the last problem

DECLARE

    y number;

BEGIN

    FOR i in 1 .. 30 loop

        y := (9/5)\*i + 32;

        DBMS\_OUTPUT.PUT\_LINE (i||' C= '|| y||' F');

    End loop;

END;

*Ammar Sajdi, OCP*

## PREDEFINED EXCEPTIONS

Exception Name	ORACLE error	SQLCODE
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_NUMBER	ORA-01722	-1722
NO_DATA_FOUND	ORA-01403	+100
PROGRAM_ERROR	ORA-06501	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

INVALID\_NUMBER can, for example, be raised when you try to insert a string into a number field in the database.

NO\_DATA\_FOUND is raised when SELECT INTO returns no rows or when you reference an uninitialized row in PL/SQL table (Introduced later).

TIME\_OUT\_RESOURCE is raised when you are waiting for a locked table or any other resource

**Note:** To comment a line use `--`, and to comment a group of lines use `/* xxxx */` Like C language

```
x := 10 ; -- This is a comment
```

## PL/SQL TABLES

PL/SQL TABLES are very similar to arrays know in common programming language. To use PL/SQL tables, you must first Declare a new Type call TABLE. Then you define a variable of that type

EXAMPLES:-

```
DECLARE
```

```
    TYPE my_table IS TABLE OF CHAR(10)
    INDEX BY BINARY_INTEGER;
```

```
    TYPE new_type IS TABLE OF dept.loc%Type
    INDEX BY BINARY_INTEGER;
```

```
scottab      my_table;
arr1        new_type;
```

Please note that you cannot initialize the table at declaration. You can reference the PL/SQL table in the traditional way an array is referenced in common programming languages. i.e

```
scottab(1) := sal*12;
```

Or by a loop

```
For i in 1 ..20 loop
    scottab(i) := i*10;
End Loop;
```

PALCO

If you try to reference an uninitialized row in a PL/SQL table, then NO\_DATA\_FOUND exception is raised.

More examples:-

```
For i in 1..50 Loop
    Insert into dept (deptno)
    values (scottab ( i ) );
End loop;
```

Declare

```
Type tabtype IS TABLE of number;
INDEX BY BINARY_INTEGER;
```

```
TYPE tabtype 1 IS TABLE of CHAR(20)
INDEX BY BINARY_INTEGER;
```

```
Empno_tab      tabtype;
ename_tab      tabtype1;
```

To Delete PL/SQL table or to deinitialize one element of the Table, simply assign it to NULL. i.e

```
scottab(i) := NULL;
```

## Example

DECLARE

```
TYPE my_array IS TABLE OF DATE  
INDEX BY BINARY_INTEGER;
```

```
arr_null my_array;  
arr1 my_array;
```

BEGIN

```
For i in 244669 .. 244699 Loop  
    arr1(i) := to_date(i,'j'); -- j joulean date  
    insert into emp (empno,hiredate,deptno)  
    values (1,arr1(i),5);  
END LOOP;  
arr1 := arr_null; -- delete arr1
```

END;

## NEXT, RECORD TYPE is INTRODUCED



## PLSQL RECORDS

You can define a record that holds the same structure of a record in the database as follows:-

```
DEF_REC      DEPT%ROWTYPE;
```

What if you want to define a record to your own

DECLARE

```
TYPE user_rec IS RECORD  
(deptno Number(2) := 20,  
  dname varchar2(20),  
  loc      dept.loc%type);
```

```
my_rec User_rec;  
my_rec2 User_rec;
```

BEGIN

```
my_rec.dname := 'ACCT';
```

END;

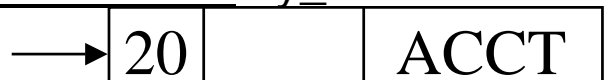
my\_rec



ASSIGNMENT

```
my_rec2 := my_rec;
```

my\_rec2



This is legal only if my\_rec and my\_rec2 belong to the same RECORD Type

Record can be nested as in the following example:-

```
DECLARE
    TYPE emp_rec IS RECORD
        (EMPNO      NUMBER(4),
         SAL        NUMBER(7,4),
         DEPTNO     NUMBER(3));

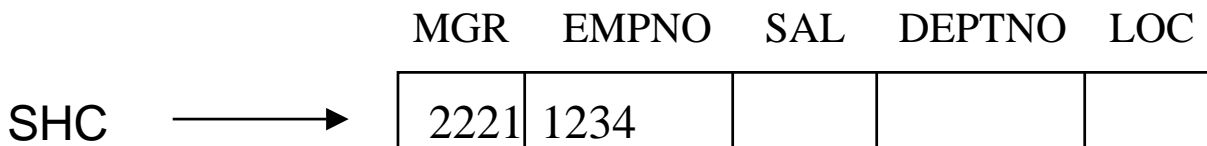
    TYPE comp_rec IS RECORD
        (MGR        NUMBER(4),
         EMP        EMP_REC,      -- nested record
         LOC        VARCHAR2(10));

    SHC    comp_rec; -- Record SHC of type comp_rec

BEGIN

    SHC.MGR          := '2221';
    SHC.EMP.EMPNO    := 1234;

END;
```



One of the advantages of RECORDS is that you collect information about the attributes of an object in one name. Later you can refer the object as a whole. For example, you can pass the whole RECORD to a function instead of passing the individual fields.

# CONTROL STRUCTURES

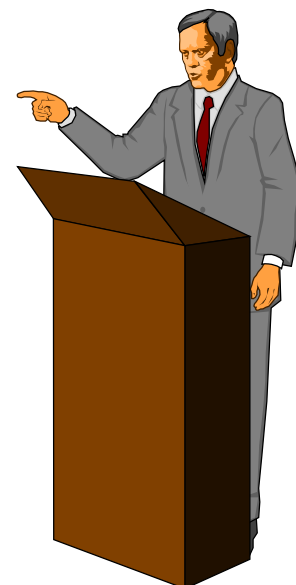
---

## IF-THEN-ELSIF

```
If sales > 50000 Then
    bonus := 1500;
Elsif sales > 35000 Then
    bonus := 500;
Else
    bonus := 100;
End If;
Insert into jpayroll Values (emp_id,bonus ..);
```

make note of ***Elsif*** , it does not contain the letter ‘e’ after Els, nor there is space before if i.e. (**ELSIF**)

DO NOT MIS-USE IF STATEMET  
AND DO NOT MAKE A SPELINLG  
MISTAKE WITH  
**IF - ELSE - ELSIF and END IF**





The following is not efficiently written :-

```
DECLARE
    overdrawn BOOLEAN;
    .....
BEGIN

    IF new_balance < minimum_balance THEN
        overdrawn := TRUE;
    ELSE
        overdrawn := FALSE;
    END IF;

    IF overdrawn = TRUE THEN
        .....
    END IF;
END;
```

YOU can replace the first IF statement with

```
overdrawn := new_balance < minimum_balance;
```

AND the second IF with

```
IF overdrawn THEN
```

Substitution Variables EXAMPLE ON NEXT PAGE

```
DECLARE
    deptno number(2);
BEGIN
SELECT COUNT(*) INTO deptno FROM EMP
WHERE deptno = &&dept_no;
IF deptno <3 THEN
    UPDATE emp SET deptno = 50
    WHERE deptno = &&dept_no;
ELSE
    DBMS_OUTPUT.PUT_LINE ('No of Emp. = '|| deptno);
END IF;
END;
```

How Does **&&dept\_no** differ from **&dept\_no**?

Remember: the DBMS\_OUTPUT.PUTLINE requires

SQL> SET SERVEROUTPUT ON

Try the above example, It should work!

Note the Semi-colon after END IF and after END

## LOOP and EXIT

```
LOOP
  x := cos (y+sqrt(z));
  IF x=1 THEN
    .....
    EXIT;                -- Exit loop here
  END IF;
END LOOP;
```

NOTE: **EXIT** is only meaningful within a **LOOP**

Equivalently one can user **EXIT-WHEN**

```
LOOP
  x := cos (y+sqrt(z));
  EXIT WHEN x=1;
END LOOP;
```

Can we nest loop ? The answer is YES

### **WHILE-LOOP**

```
WHILE condition LOOP
  Sequence of statements
END LOOP;
```

PALCO**FOR - LOOP**

```
FOR i in 1 .. 3 LOOP
    Sequence of Statement;
END LOOP;
```

```
FOR x in REVERSE 1.. 3 LOOP
    Sequence of Statements;
END LOOP;
```

IF you write

For i in 3 .. 3 LOOP etc.

Then your loop will execute once

NOTES:- The loop counter is Implicitly declared and is local to the loop. If you declare a variable of the same name it will be dealt with as a different variable global to the program.  
-The loop counter cannot be assigned values within the loop.

BASIC Language provides the following:

```
FOR I = 1 TO 100 STEP 5;
```

How can we implement this in PL/SQL?

PALCO

There is no direct way to do it in PL/SQL but if you use your mind, you can do the following:-

```
FOR i in 1..100 LOOP
  IF MOD(i,5) = 0 THEN
    sequence of statements;
  END IF;
END LOOP
```

HOW about **GOTO** Statement?

PL/SQL provides **GOTO** statement but I will not mention it because all programming teachers say

DO NOT USE IT!



BEGIN

```
...
GOTO FIRST; -- The label is FIRST...
```

```
....
<<FIRST>> -- Label
INSERT INTO emp VALUES ...
```

END;

*Ammar Sajdi, OCP*

## The World of Cursors

The set of rows returned by a query can consist of zero, one, or multiple rows, depending on your search condition. When query return multiple rows, you can explicitly define a cursor to process the rows.

### Explicit Declaration

```
- CURSOR C1 is Select id,name From mytable  
where id > 2000;
```

This is a declaration statement of a cursor that contains a simple SELECT and must be declared in the declaration section of a PL/SQL Block. C1 is the name of the cursor and can be any name.

One can use the cursor within the body of PL/SQL by using **OPEN, FETCH .. INTO** and **CLOSE**

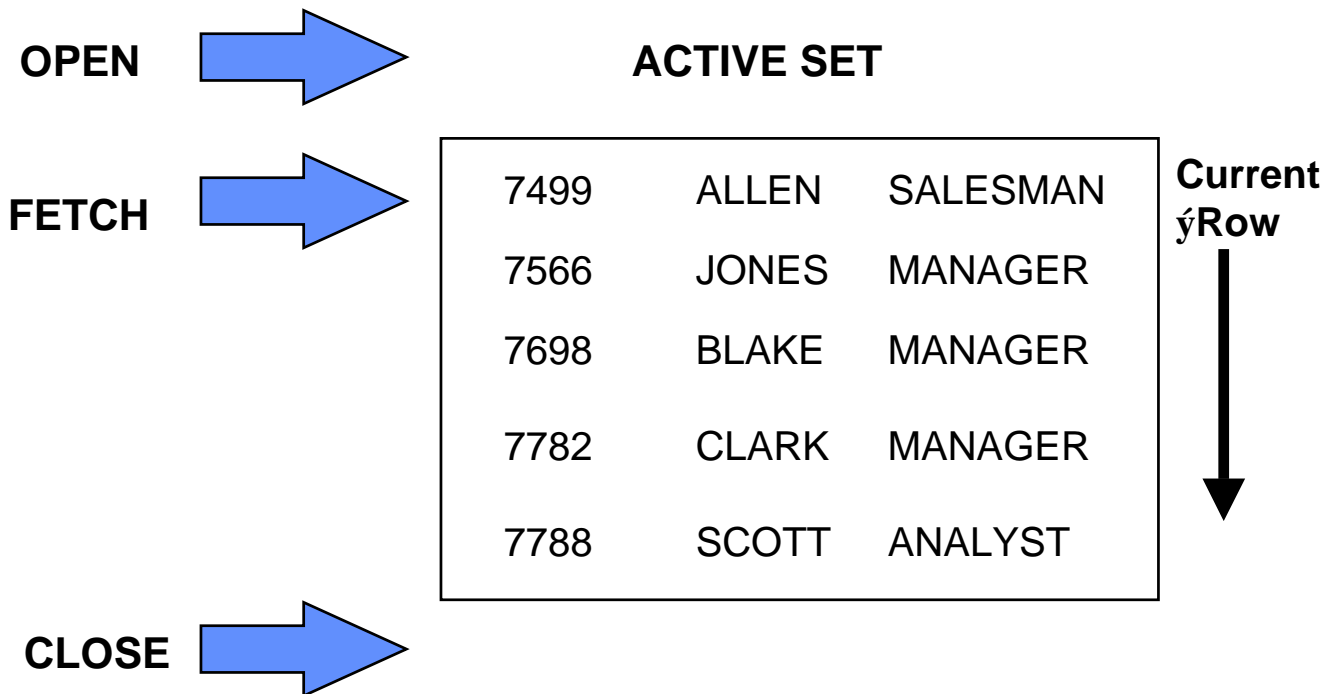
THE **OPEN** statement executes the query associated with the cursor, and identifies the active set, and positions the cursor at the first row. The **FETCH .. INTO** statement retrieves the current row AND stores it into local variables and advances the cursor to the next one. The **CLOSE** statement disables the cursor. The following diagram explains the concept.

# CURSOR CONTROL

**emp table**

7369	SMITH	CLERK	800		20
7499	ALLEN	SALESMAN	1600	300	30
7521	WARD	SALESMAN	1250	500	30
7566	JONES	MANAGER	2975		20
7654	MARTIN	SALESMAN	1250	1400	30
7698	BLAKE	MANAGER	2850		30
7782	CLARK	MANAGER	2450		10
7788	SCOTT	NANALYST	3000		20
	etc	....			

CURSOR C1 IS SELECT EMPNO,ENAME,JOB FROM EMP WHERE SAL > 1250;



## EXAMPLE

### DECLARE

```
CURSOR my_cursor IS  
SELECT sal+NVL(comm,0) wages , ename  
FROM EMP;  
X1 number;  
x2 varchar2 (20);
```

### BEGIN

```
OPEN my_cursor;  
LOOP  
    FETCH my_cursor INTO x1,x2;  
    EXIT WHEN my_cursor%NOTFOUND;  
    IF x1 > 2000 THEN  
        INSERT INTO anytable VALUES (x1,...);  
    END IF;  
END LOOP;  
CLOSE my_cursor;
```

END;

The same thing can also be done as follows

```
CURSOR my_cursor IS
```

```
SELECT sal+NVL(comm,0) wages , ename  
FROM EMP;  
my_rec my_cursor%ROWTYPE --
```

```
BEGIN
```

*CONT .Next Page*



PALCO

```
OPEN my_cursor;
  LOOP
      FETCH my_cursor INTO my_rec;
      EXIT WHEN my_cursor%NOTFOUND;
      IF my_rec.wages > 2000 THEN
          INSERT INTO anytable
              VALUES (my_rec.sal, .);
      END IF;
  END LOOP;
CLOSE my_cursor;
END;
```

## Passing Parameters to Cursors

When you open a cursor you can pass parameters to it.  
For example

```
DECLARE
    CURSOR MY_C (VAR 1 NUMBER) IS
    SELECT ENAME,SAL, DEPTNO FROM EMP
    WHERE EMPNO=VAR1;
BEGIN
    OPEN MY_C (7499);
        etc.
END;
```

PALCO  
MORE EXAMPLE ON FETCHING CURSORS

```
DECLARE
    CURSOR C1 IS SELECT
        EMPNO,ENAME,SAL FROM EMP;
    m_empno      NUMBER(10);
    m_ename      VARCHAR2(10);
    m_sal        NUMBER(2);
BEGIN
    OPEN C1;    -- ACTIVE SET IS IDENTIFIED.
    LOOP
        FETCH C1 INTO m_empno,m_ename,m_sal;
        EXIT WHEN C1%NOTFOUND;    -- when EOF
        IF m_sal < 2000 THEN
            UPDATE EMP SET SAL=SAL*1.1 where
                empno = m_empno;
        END IF;
    END LOOP;
    CLOSE C1
END;
```

Note: What happens if during the fetch of a certain record whose m\_sal is < 2000, the same record was changed by another user? How to correct the problem

```
DECLARE
CURSOR C1 IS SELECT JOB FROM EMP;
JOB1    EMP.JOB%TYPE;
BEGIN
    OPEN C1;
    LOOP
```

```
FETCH C1 INTO JOB;  
EXIT WHEN C1%NOTFOUND;  
  
...  
END LOOP;
```

## CURSOR ATTRIBUTES

%NOTFOUND  
%FOUND  
%ROWCOUNT  
%ISOPEN

Before the first fetch, the %NOTFOUND evaluates to NULL.  
If the last fetch fails to return a row %NOTFOUND evaluates to TRUE.

%FOUND is the opposite of %NOTFOUND.

When you open the cursor, %ROWCOUNT is zeroed.  
%ROWCOUNT is incremented every time a row is fetched.

%ISOPEN evaluates to TRUE if its cursor is open

```
IF c1%ISOPEN THEN -- cursor is open  
  
...  
ELSE OPEN c1
```

## EXAMPLE

```
DECLARE
    num1    data_table.n1%TYPE;
    num2    data_table.n2%TYPE;
    num3    data_table.n3%TYPE;
    result  temp.col1%TYPE;
    CURSOR c1 IS
    SELECT n1,n2,n3 from data_table
        WHERE experiment=1;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO num1,num2,num3;
        EXIT WHEN c1%NOTFOUND;  -- UNTIL EOF
        result := num2/(num1+num3);
        INSERT INTO temp VALUES (result);
    END LOOP;
    CLOSE c1;
    COMMIT;
END;
```

Note the Use of COMMIT to make sure that the inserted records are saved, the end of the block does not automatically commit a trans.

## UPDATING THE RECORD OF THE CURRENT CURSOR

We have seen the following example before. Is this the best way to implement the Update?

```
DECLARE
    CURSOR C1 IS SELECT
        EMPNO,ENAME,SAL FROM EMP;
    m_empno      NUMBER(10);
    m_ename      VARCHAR2(10);
    m_sal        NUMBER(2);
BEGIN
    OPEN C1;    -- ACTIVE SET IS IDENTIFIED.
    LOOP
        FETCH C1 INTO m_empno,m_ename,m_sal;
        EXIT WHEN C1%NOTFOUND;  -- when EOF
        IF m_sal < 2000 THEN
            UPDATE EMP SET SAL=SAL*1.1 where
                empno = m_empno;
        END IF;
    END LOOP;
    CLOSE C1
END;
```

Note that the Update statement is intended to update the record currently fetched by cursor C1. Instead of using Where empno=m\_empno , we can use FOR UPDATE and CURRENT OF Construct as the following example shows

```
DECLARE
    CURSOR C1 IS SELECT
        ENAME,SAL FROM EMP FOR
        UPDATE OF SAL;
    m_ename      VARCHAR2(10);
    m_sal        NUMBER(2);
BEGIN
    OPEN C1;    -- ACTIVE SET IS IDENTIFIED.
    LOOP
        FETCH C1 INTO m_ename,m_sal;
        EXIT WHEN C1%NOTFOUND;    -- when EOF
        IF m_sal < 2000 THEN
            UPDATE EMP SET SAL=SAL*1.1 where
                CURRENT OF C1;
        END IF;
    END LOOP;
    CLOSE C1
END;
```

Notes:

1- FOR UPDATE OF makes sure that the active set is locked and cannot be changed by other users.

2- CURRENT OF C1 means that the update will affect the currently fetched record.

3- Please remember that CURRENT OF is valid only if the cursor was declared using the FOR UPDATE

One problem with CURRENT OF usage is that it cannot be used if COMMIT is issued within the loop. Doing so will result in a “FETCH OUT OF SEQUENCE” error. i.e the following will give an error

LOOP

```
    FETCH C1 INTO m_ename,m_sal;  
    EXIT WHEN C1%NOTFOUND;  -- when EOF  
    IF m_sal < 2000 THEN  
        UPDATE EMP SET SAL=SAL*1.1 where  
            CURRENT OF C1;  
    END IF;  
    COMMIT  -- This will cause an error
```

END LOOP;

Before discussing who to fix this problem. Is it good to Commit within a loop?

Consider the following factors before you answer: ROLLBACK, PERFORMANCE, MEDIA FAILURE, POWER FAILURE.

FIX:

The best fix to the above mentioned problem is not to use the CURRENT OF construct. Instead, fetch the ROWID in the cursor and use it in the where clause of the Update statement to help identify the record being updated. The usage of ROWID will guarantee the fastest access path to your record. The following example illustrates:

*Ammar Sajdi, OCP*

```
DECLARE
    CURSOR C1 IS SELECT
        ENAME,SAL rowid FROM EMP FOR
        UPDATE OF SAL;
    m_ename          VARCHAR2(10);
    m_sal            NUMBER(2);
    m_rowid          rowid;
BEGIN
    OPEN C1;    -- ACTIVE SET IS IDENTIFIED.
    LOOP
        FETCH C1 INTO m_ename,m_sal,m_rowid;
        EXIT WHEN C1%NOTFOUND;    -- when EOF
        IF m_sal < 2000 THEN
            UPDATE EMP SET SAL=SAL*1.1 where
                rowid=m_rowid;
        END IF;
        COMMIT;
    END LOOP;
    CLOSE C1
END
```



## IMPLICIT CURSORS

ORACLE implicitly opens a cursor to process each SQL statement not associated with an explicitly declared cursor.

You can refer to most recent implicit cursor as the “SQL” cursor.

### Example

```
UPDATE daily_journal SET qty = qty + 1
WHERE part_id = 100;
IF SQL%NOTFOUND THEN -- No Rows updated
    INSERT INTO Purchase_order values (100);
END IF;
```

Note that SQL%NOTFOUND evaluates the LAST SQL statement only

```
DECLARE
```

```
    my_count;
```

```
BEGIN
```

```
SELECT MAX(sal) INTO my_count FROM emp
```

```
WHERE deptno = 8898;    -- CONTINUE NEXT PAGE
```

```
IF SQL%NOTFOUND THEN
    DELETE FROM EMP;
    /* THIS action is never taken because
    the function MAX evaluates to a value
    or a NULL and %NOTFOUND evaluates
    to FALSE */
END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN NULL;
-- never reaches this part.
END;
```

## CURSOR FOR LOOP

A cursor FOR LOOP implicitly declares its Loop index as a record of type %ROWTYPE, OPENS a cursor, FETCHES records and CLOSES the cursor

```
DECLARE
    result    temp.col1%TYPE;
    CURSOR C1 IS SELECT n1,n2,n3 from m_tab;
BEGIN
    FOR my_record IN C1 LOOP;
```

Let us explain this FOR Loop

The counter *my\_record* can be any name. When the FOR statement is executed, the *my\_record* is implicitly declared as a RECORD with fields and datatypes similar to those existing in the CURSOR *c1*. The FOR LOOP OPENS the CURSOR *c1* and FETCHES all column values of the current row and stores into the record fields which are

```
my_record.n1  
my_record.n2  
my_record.n3
```

This is equivalent to saying

```
DECLARE  
CURSOR C1 IS SELECT n1,n2,n3 from m_tab;  
my_record      c1%ROWTYPE;  
BEGIN  
    OPEN c1;  
    LOOP;  
        FETCH c1 into my_record.n1,  
        my_record.n2,my_record.n3;  
        EXIT WHEN C1%NOTFOUND;  
    END LOOP;  
    CLOSE c1  
END;
```

**MORE ABOUT ERROR HANDLING:-**

Define Your own EXCEPTION as Follows :-

```
DECLARE
    angry    EXCEPTION
    ...
BEGIN
    ...
    IF x > 100 then
        RAISE angry;
    END IF;
EXCEPTION
    WHEN angry THEN
        .....
    WHEN OTHERS THEN
        ....
END ;
```

It is also possible to handle Internal error that do not have PRE-DEFINED EXCEPTIONS. This is only possible if one knows the ORACLE's Error code which is associated with that internal error. SEE EXAMPLE ON NEXT PAGE.

PALCO

```
DECLARE
    no_privileges    EXCEPTION;
    PRAGMA EXCEPTION_INIT(no_privileges,-1031);
-- -1031 is the error code associated with trying to update
-- a table for which you only have SELECT privileges
BEGIN
    UPDATE team4.emp SET sal=sal+100;
EXCEPTION
    WHEN no_privileges THEN
        -- handle the newly defined exception
    WHEN others THEN
        -- Handle other exceptions
END;
```

***RAISE\_APPLICATION\_ERROR***

This is a procedure which lets you issue user\_Defined error messages from your PL/SQL Code

```
DECLARE
    x    number;
BEGIN
SELECT comm INTO x FROM emp WHERE empno= 7902;
IF x IS NULL THEN
    raise_application_error(-20001,'Comm is Null');
ELSE
    UPDATE emp set comm = comm+200 where
    empno = 7902;
END IF;
END;
```

*Ammar Sajdi, OCP*

RETRYING A TRANSATION :-

**DECLARE**

**empno number(4) := 7755;**

**BEGIN**

**FOR i in 1 .. 3 LOOP -- Retry Three Times**

**BEGIN**

SAVEPOINT start1; -- Mark a savepoint

INSERT INTO EMP (empno,sal)

VALUES(empno,1000);

-- might raise DUP\_VAL\_ON\_INDEX

-- if empno already exists;

COMMIT;

exit;

**EXCEPTION**

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK to start1; -- UNDO

empno := empno +1; -- try to fix

**END;**

**END LOOP;**

**END;**

CAN YOU EXPLAIN IT?

NOTE about *raise\_application\_error*. The code number you are allowed to use is restricted to the range -20000 .. -20999

This document was created with Win2PDF available at <http://www.daneprairie.com>.  
The unregistered version of Win2PDF is for evaluation or non-commercial use only.