**NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES**

**CL 203-Database Systems**

**Lab Session 10**

# Loops

1. **Simple Loops** Run until you explicitly end the loop
2. **WHILE Loops** Run until a specified condition occurs
3. **FOR Loops** Run a predetermined number of times

# Simple Loops

A simple loop runs until you explicitly end the loop.

LOOP

*Statements*

END LOOP;

# WHILE Loops

A While loop runs until a specified condition occurs.

WHILE condition LOOP

*Statements*

END LOOP;

Ex:

Counter :=0;

WHILE counter <6 LOOP

Counter := counter +1;

END LOOP;

# FOR Loops

A FOR loop runs a predetermined number of times; you determine the number of times the loop runs by specifying the lower and upper bounds for a loop variable . The loop variable is then incremented (or decremented) each time around the loop. The syntax for a FOR loop is as follows:

FOR Loop\_variable IN***lower\_bound . . upper\_bound*** LOOP

*Statements*

END LOOP;

Ex:

FOR count2 IN 1 . . 5 LOOP

DBMS\_OUTPUT.PUT\_LINE (count2);

END LOOP;

# Cursors

You use cursor when you have a SELECT statement that returns more than one row from the database. A cursor is basically a set of rows that you can access one at a time. You may follow five steps when using a cursor:

1. **Declare variables to store column values from the SELECT statement.**
2. **Declare the cursor, specifying your SELECT statement.**
3. **Open the Cursor.**
4. **Fetch the rows from the cursor.**
5. **Close the cursor.**

**STEP 1 : Declare variables to store column values**

These variables must be compatible with the column types.

DECLARE

v\_Empno emp.empno%TYPE;

v\_Ename emp.ename%TYPE;

v\_Job emp.job%TYPE;

v\_Deptno emp.deptno%TYPE;

v\_Sal emp.sal%TYPE;

**STEP 2**: **Declare the cursor**

CURSOR cursor\_name IS

*SELECT\_STATEMENT;*

---

CURSOR cv\_emp\_cursor IS

SELECT empno,ename,sal

FROM products

Order by empno;

**STEP 3: Open the Cursor**

This step runs the SELECT statement. It must be placed in the executable section of the block (BEGIN).

OPEN cv\_emp\_cursor;

**STEP 4: Fetch the rows from the cursor**

To read each row from the cursor, you can use the FETCH statement. The FETCH statement reads the column values into the variables that you specify.

FETCH cursor\_name

INTO variable[, variable. . . . ];

FETCH cv\_emp\_cursor

INTO v\_Empno,v\_Ename , v\_Job , v\_Deptno ,v\_Sal;

A cursor may have many rows; therefore, a loop is required to read each row in turn.

LOOP

FETCH CV\_EMP\_CURSOR

INTO V\_EMPNO ,V\_ENAME , V\_JOB , V\_DEPTNO ,V\_SAL;

--*exit the loop when there are no more rows, as indicated by*

*--the Boolean variable cv\_emp\_cursor%NOTFOUND (=true when*

*--there are no more rows)*

EXIT WHEN cv\_emp\_cursor%NOTFOUND;

--*use DBMS\_OUTPUT.PUT\_LINE () to display the variable*

DBMS\_OUTPUT.PUT\_LINE(

‘v\_Empno= ‘ || v\_Empno ,’v\_Ename=’ || v\_ename ,’ v\_Job= ‘ || v\_job , ‘ v\_Deptno= ‘ || v\_Deptno ,’v\_Sal = ‘|| v\_sal;

**STEP 5: Close the Cursor**

Closing your cursors frees up system resources.

CLOSE cv\_emp\_cursor;

***Assemble the above code together so that you can run it on SQL\* PLUS:***

­­­­­­­­­­­­­­­­­­­­­­­­­­­­

­­­­­­­­­­­

# Cursors and for loops

You can use the power of FOR loop to access the rows in a cursor. When you use a FOR loop, you don’t have to explicitly open and close the cursor------ the FOR loop does this automatically.

DECLARE

CURSOR cv\_emp\_cursor2 IS

SELECT empno, ename,job

FROM emp WHERE deptno = 30;

BEGIN

FOR v\_emp IN cv\_emp\_cursor2 LOOP

DBMS\_OUTPUT.PUT\_LINE(

‘EMPNO =’ || v\_emp.empno || ‘,ename = ‘|| v\_emp.ename || ‘,job = ‘|| v\_emp.job );

END LOOP;

END;

/

# Exceptions

Exceptions are used to handle errors that occur in your PL/SQL code. Following are some exceptions occur in code

| **Exception** | **Raised when ...** |
| --- | --- |
| ACCESS\_INTO\_NULL | Your program attempts to assign values to the attributes of an uninitialized (atomically null) object. |
| CASE\_NOT\_FOUND | None of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause. |
| COLLECTION\_IS\_NULL | Your program attempts to apply collection methods other than EXISTS to an uninitialized (atomically null) nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray. |
| CURSOR\_ALREADY\_OPEN | Your program attempts to open an already open cursor. A cursor must be closed before it can be reopened. A cursor FOR loop automatically opens the cursor to which it refers. So, your program cannot open that cursor inside the loop. |
| DUP\_VAL\_ON\_INDEX | Your program attempts to store duplicate values in a database column that is constrained by a unique index. |
| INVALID\_CURSOR | Your program attempts an illegal cursor operation such as closing an unopened cursor. |
| INVALID\_NUMBER | In a SQL statement, the conversion of a character string into a number fails because the string does not represent a valid number. (In procedural statements, VALUE\_ERROR is raised.) This exception is also raised when the LIMIT-clause expression in a bulk FETCH statement does not evaluate to a positive number. |
| LOGIN\_DENIED | Your program attempts to log on to Oracle with an invalid username and/or password. |
| NO\_DATA\_FOUND | A SELECT INTO statement returns no rows, or your program references a deleted element in a nested table or an uninitialized element in an index-by table. SQL aggregate functions such as AVG and SUM always return a value or a null. So, a SELECT INTO statement that calls an aggregate function never raises NO\_DATA\_FOUND. The FETCH statement is expected to return no rows eventually, so when that happens, no exception is raised. |
| NOT\_LOGGED\_ON | Your program issues a database call without being connected to Oracle. |
| PROGRAM\_ERROR | PL/SQL has an internal problem. |
| ROWTYPE\_MISMATCH | The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types. For example, when an open host cursor variable is passed to a stored subprogram, the return types of the actual and formal parameters must be compatible. |
| SELF\_IS\_NULL | Your program attempts to call a MEMBER method on a null instance. That is, the built-in parameter SELF (which is always the first parameter passed to a MEMBER method) is null. |
| STORAGE\_ERROR | PL/SQL runs out of memory or memory has been corrupted. |
| SUBSCRIPT\_BEYOND\_COUNT | Your program references a nested table or varray element using an index number larger than the number of elements in the collection. |
| SUBSCRIPT\_OUTSIDE\_LIMIT | Your program references a nested table or varray element using an index number (-1 for example) that is outside the legal range. |
| SYS\_INVALID\_ROWID | The conversion of a character string into a universal rowid fails because the character string does not represent a valid rowid. |
| TIMEOUT\_ON\_RESOURCE | A time-out occurs while Oracle is waiting for a resource. |
| TOO\_MANY\_ROWS | A SELECT INTO statement returns more than one row. |
| VALUE\_ERROR | An arithmetic, conversion, truncation, or size-constraint error occurs. For example, when your program selects a column value into a character variable, if the value is longer than the declared length of the variable, PL/SQL aborts the assignment and raises VALUE\_ERROR. In procedural statements, VALUE\_ERROR is raised if the conversion of a character string into a number fails. (In SQL statements, INVALID\_NUMBER is raised.) |
| ZERO\_DIVIDE | Your program attempts to divide a number by zero. |