FOR Loop

The following example uses a simple `FOR` loop to insert ten rows into a database table. The values of a loop index, counter variable, and either of two character strings are inserted. Which string is inserted depends on the value of the loop index.

PL/SQL Block

```
DECLARE
   x NUMBER := 100;
BEGIN
   FOR i IN 1..10 LOOP
      IF MOD(i,2) = 0 THEN     -- i is even
         INSERT INTO temp VALUES (i, x, 'i is even');
      ELSE
         INSERT INTO temp VALUES (i, x, 'i is odd');
      END IF;
      x := x + 100;
   END LOOP;
   COMMIT;
END;
```

Output Table

```
SQL> SELECT * FROM temp ORDER BY col1;

NUM_COL1 NUM_COL2  CHAR_COL
-------- --------  ---------
       1      100  i is odd
       2      200  i is even
       3      300  i is odd
       4      400  i is even
       5      500  i is odd
       6      600  i is even
       7      700  i is odd
       8      800  i is even
       9      900  i is odd
      10     1000  i is even
```

Sample 2. Cursors

The following example uses a cursor to select the five highest paid employees from the `emp` table.

Input Table

```
SQL> SELECT ename, empno, sal FROM emp ORDER BY sal DESC;

ENAME          EMPNO      SAL
---------- --------- --------
KING            7839     5000
SCOTT           7788     3000
FORD            7902     3000
JONES           7566     2975
```

```
BLAKE           7698     2850
CLARK           7782     2450
ALLEN           7499     1600
TURNER          7844     1500
MILLER          7934     1300
WARD            7521     1250
MARTIN          7654     1250
ADAMS           7876     1100
JAMES           7900      950
SMITH           7369      800
```

PL/SQL Block

```
DECLARE
   CURSOR c1 is
      SELECT ename, empno, sal FROM emp
         ORDER BY sal DESC;   -- start with highest paid employee
   my_ename VARCHAR2(10);
   my_empno NUMBER(4);
   my_sal   NUMBER(7,2);
BEGIN
   OPEN c1;
   FOR i IN 1..5 LOOP
      FETCH c1 INTO my_ename, my_empno, my_sal;
      EXIT WHEN c1%NOTFOUND;
      INSERT INTO temp VALUES (my_sal, my_empno, my_ename);
      COMMIT;
   END LOOP;
   CLOSE c1;
END;
```

Output Table

```
SQL> SELECT * FROM temp ORDER BY col1 DESC;

NUM_COL1 NUM_COL2  CHAR_COL
-------- --------  --------
    5000      7839  KING
    3000      7902  FORD
    3000      7788  SCOTT
    2975      7566  JONES
    2850      7698  BLAKE
```

Sample 3. Scoping

The following example illustrates block structure and scope rules. An outer block declares two variables named `x` and `counter` and loops four times. Inside this loop is a sub-block that also declares a variable named `x`. The values inserted into the `temp` table show that the two `x`'s are indeed different.

PL/SQL Block

```
DECLARE
   x NUMBER := 0;
   counter NUMBER := 0;
BEGIN
```

```
   FOR i IN 1..4 LOOP
      x := x + 1000;
      counter := counter + 1;
      INSERT INTO temp VALUES (x, counter, 'in OUTER loop');
      DECLARE
         x NUMBER := 0;  -- this is a local version of x
      BEGIN
         FOR i IN 1..4 LOOP
            x := x + 1;  -- this increments the local x
            counter := counter + 1;
            INSERT INTO temp VALUES (x, counter, 'inner loop');
         END LOOP;
      END;
   END LOOP;
   COMMIT;
END;
```

Output Table

```
SQL> SELECT * FROM temp ORDER BY col2;

NUM_COL1 NUM_COL2  CHAR_COL
-------- --------  -------------
    1000         1  in OUTER loop
       1         2  inner loop
       2         3  inner loop
       3         4  inner loop
       4         5  inner loop
    2000         6  in OUTER loop
```