Control Structures: Iterative Control

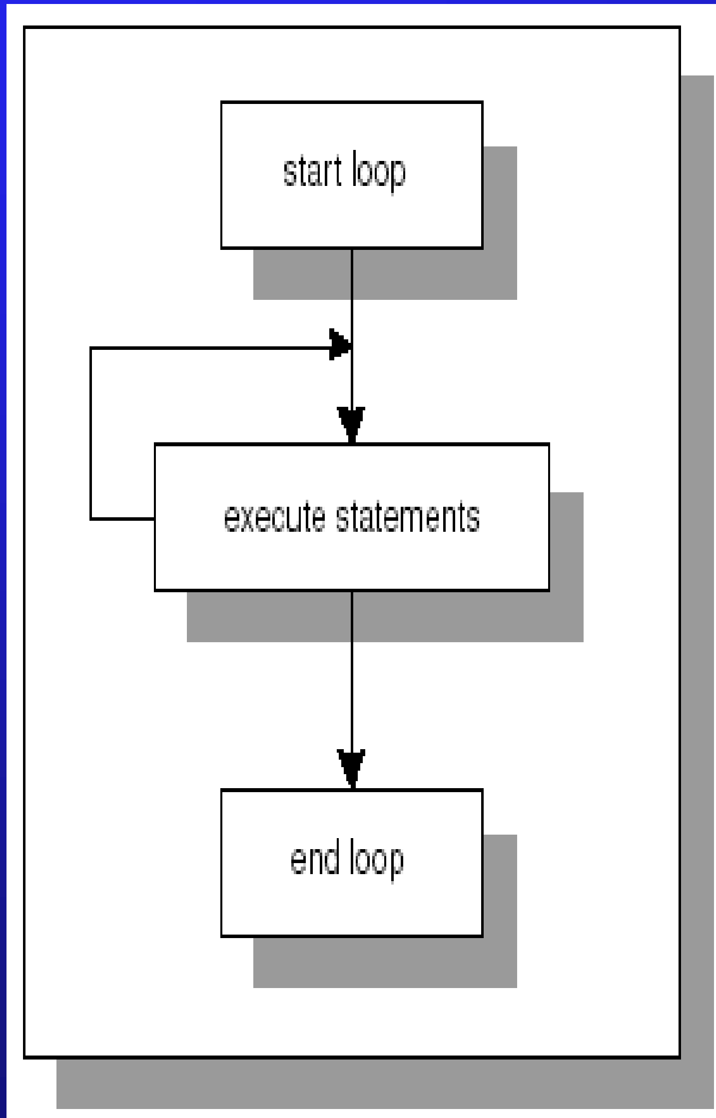
SIMPLE LOOPS

- A simple loop, as you can see from its name, is the most basic kind of loop and has the following structure:

```
LOOP
STATEMENT 1;
STATEMENT 2;
....
STATEMENT N;
END LOOP;
```

- The reserved word LOOP marks the beginning of the simple loop.
- Statements 1 through N are a sequence of statements that is executed repeatedly.
- These statements consist of one or more of the standard programming structures.
- END LOOP is a reserved phrase that indicates the end of the loop construct.

SIMPLE LOOPS



Bordoloi and Bock

- Every time the loop is iterated, a sequence of statements is executed, then control is passed back to the top of the loop.
- The sequence of statements will be executed an infinite number of times because there is no statement specifying when the loop must terminate.
- Hence, a simple loop is called an *infinite loop* because there is no means to exit the loop.

Exit

- The EXIT statement causes a loop to terminate when the *EXIT condition* evaluates to TRUE.
- The EXIT condition is evaluated with the help of an IF statement.
- When the EXIT condition is evaluated to TRUE, control is passed to the first executable statement after the END LOOP statement.

Exit

```
LOOP
STATEMENT 1;
STATEMENT 2;
IF CONDITION
THEN
EXIT;
END IF;
END LOOP;
STATEMENT 3;
```

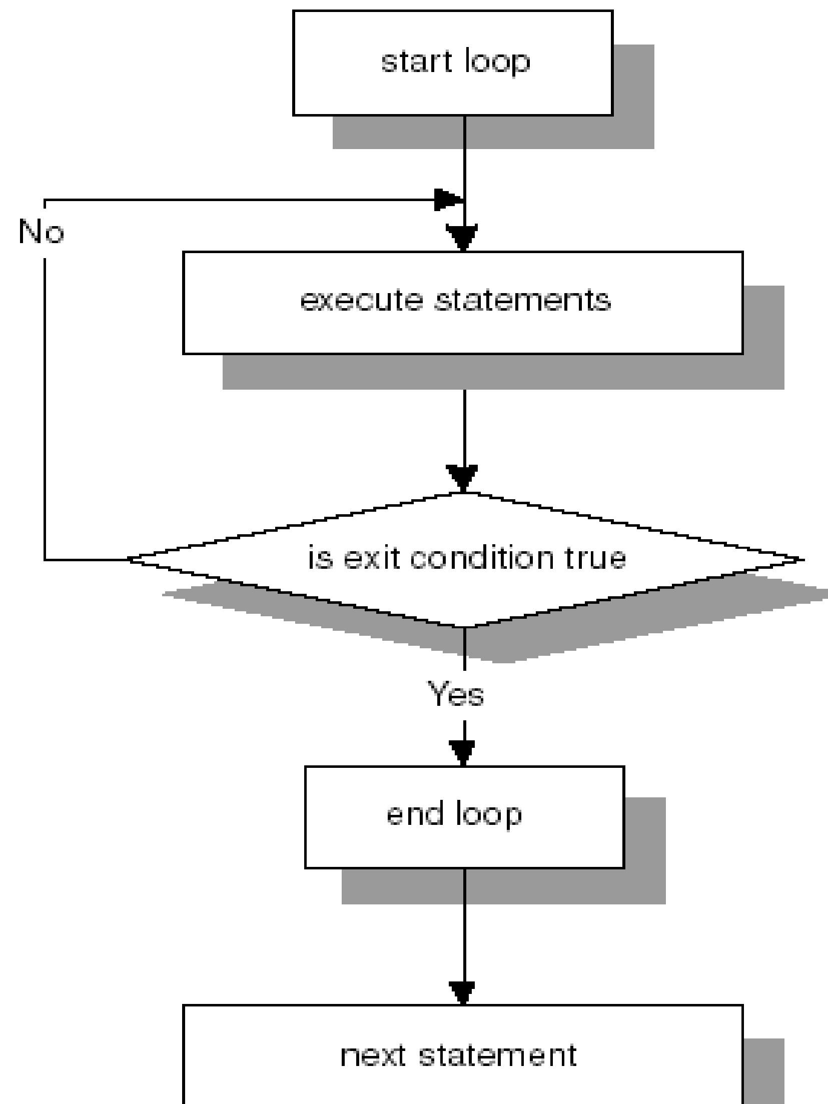
- In this example, you can see that after the EXIT condition evaluates to TRUE, control is passed to STATEMENT 3, which is the first executable statement after the END LOOP statement.

EXIT WHEN

- The EXIT WHEN statement causes a loop to terminate only if the *EXIT WHEN condition* evaluates to TRUE.
- Control is then passed to the first executable statement after the END LOOP statement.
- The structure of a loop using an EXIT WHEN clause is as follows:

```
LOOP
STATEMENT 1;
STATEMENT 2;
EXIT WHEN CONDITION;
END LOOP;
STATEMENT 3;
```

EXIT WHEN



EXIT

- When the EXIT statement is used without an EXIT condition, the simple loop will execute only once.
- Example.

```
DECLARE
    v_counter NUMBER := 0;
BEGIN
    LOOP
      DBMS_OUTPUT.PUT_LINE('v_counter ' = 
      ||v_counter);
      EXIT;
    END LOOP;
END;
```

EXIT

- This example produces the output:

  v_counter = 0

  PL/SQL procedure successfully completed.

- Because the EXIT statement is used without an EXIT CONDITION, the loop is terminated as soon as the EXIT statement is executed.

WHILE LOOPS

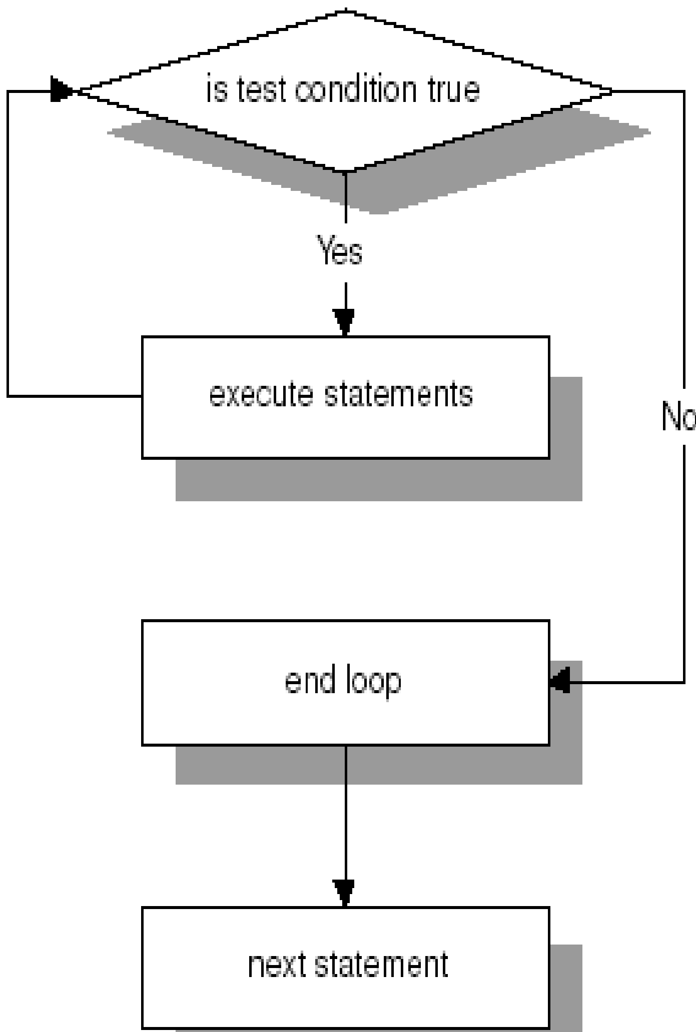
- A WHILE loop has the following structure:

```
WHILE CONDITION
LOOP
STATEMENT 1;
STATEMENT 2;
…
STATEMENT N;
END LOOP;
```

WHILE LOOPS

- The reserved word WHILE marks the beginning of a loop construct.
- The word *CONDITION* is the *test condition* of the loop that evaluates to TRUE or FALSE.
- The result of this evaluation determines whether the loop is executed.
- Statements 1 through N are a sequence of statements that is executed repeatedly.
- The END LOOP is a reserved phrase that indicates the end of the loop construct.

WHILE LOOP



- The test condition is evaluated prior to each iteration of the loop.
- If the test condition evaluates to TRUE, the sequence of statements is executed, and the control is passed to the top of the loop for the next evaluation of the test condition.
- If the test condition evaluates to FALSE, the loop is terminated, and the control is passed to

Example

```
DECLARE
  v_counter NUMBER := 5;
BEGIN
  WHILE v_counter < 5
  LOOP
DBMS_OUTPUT.PUT_LINE
   ('v_counter = '||v_counter);
   -- decrement the value of
   -- v_counter by one
v_counter := v_counter - 1;
END LOOP;
END;
```

- In this example the body of the loop is not executed at all because the test condition of the loop evaluates to FALSE.
- While the test condition of the loop must evaluate to TRUE at least once for the statements in the loop to execute, it is important to insure that the test condition will eventually evaluate to FALSE as well.
- Otherwise, the WHILE loop will execute continually.

Example

```
DECLARE
v_counter NUMBER := 1;
BEGIN
WHILE v_counter < 5
LOOP
DBMS_OUTPUT.PUT_LINE
      ('v_counter = '||v_counter);
-- decrement the value of v_counter
      by one
v_counter := v_counter - 1;
END LOOP;
END;
```

- This is an example of the infinite WHILE loop.
- The test condition always evaluates to TRUE because the value of v_counter is decremented by 1 and is always less than 5.

Use of Boolean Values to exit Loop

- Boolean expressions can be used to determine when the loop should terminate.

```
DECLARE
v_test BOOLEAN := TRUE;
BEGIN
WHILE v_test
LOOP
STATEMENTS;
IF TEST_CONDITION
THEN
v_test := FALSE;
END IF;
END LOOP;
END;
```

Premature Termination of Loop

- The EXIT and EXIT WHEN statements can be used inside the body of a WHILE loop.
- If the EXIT condition evaluates to TRUE before the test condition evaluates to FALSE, the loop is terminated prematurely.
- If the test condition yields FALSE before the EXIT condition yields TRUE, there is no premature termination of the loop.

- Example

```
DECLARE
 v_counter NUMBER := 1;
BEGIN
 WHILE v_counter <= 2
 LOOP
   DBMS_OUTPUT.PUT_LINE
   ('v_counter = '||v_counter);
   v_counter := v_counter + 1;
  IF v_counter = 5
  THEN
   EXIT;
  END IF;
 END LOOP;
END;
```


- In this example, the test condition is

  v_counter <= 2 and the EXIT condition is **v_counter = 5**
- So, in this case, the loop is executed twice as well.
- However, it does not terminate prematurely because the EXIT condition never evaluates to TRUE.
- As soon as the value of v_counter reaches 3, the test condition evaluates to FALSE, and the loop is terminated.

NUMERIC FOR LOOPS

- A numeric FOR loop is called numeric because it requires an integer as its terminating value.
- Its structure is as follows:

```
FOR loop_counter IN[REVERSE] lower_limit..upper_limit
LOOP
STATEMENT 1;
STATEMENT 2;
…
STATEMENT N;
END LOOP;
```

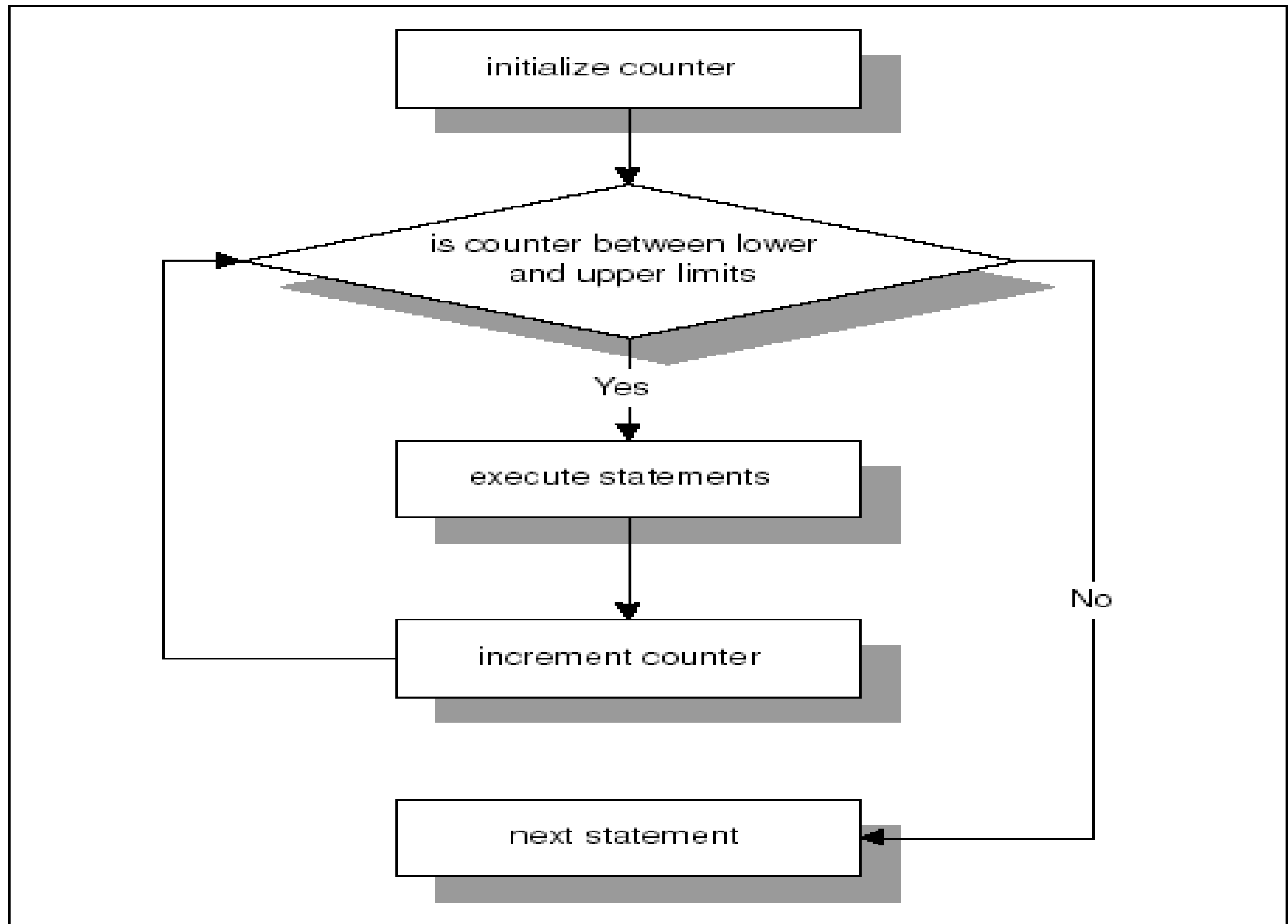
FOR LOOP

- The reserved word FOR marks the beginning of a loop construct.
- The variable, loop_counter, is an implicitly defined index variable. So, there is no need to define loop counter in the declaration section of the PL/SQL block.
- This variable is defined by the loop construct.
- *Lower_limit* and *upper_limit* are two integer numbers that define the number of iterations for the loop.
- The values of the lower_limit and upper_limit are evaluated once, for the first iteration of the loop. At this point, it is determined how many times the loop will iterate.
- Statements 1 through N are a sequence of statements that is executed repeatedly.
- END LOOP is a reserved phrase that marks the end of the loop construct.

FOR LOOP

- The reserved word IN or IN REVERSE must be present when defining the loop. If the REVERSE keyword is used, the loop counter will iterate from upper limit to lower limit.
- However, the syntax for the limit specification does not change.
- The lower limit is always referenced first.

FOR LOOP



- Example

```
BEGIN
 FOR v_counter IN 1..5
 LOOP
  DBMS_OUTPUT.PUT_LINE
   ('v_counter = '||v_counter);
 END LOOP;
END;
```

OUTPUT

v_counter = 1
v_counter = 2
v_counter = 3
v_counter = 4
v_counter = 5
PL/SQL procedure successfully completed.

- In this example, there is no declaration section for this PL/SQL block because the only variable used, v_counter, is the loop counter.
- Numbers 1..5 specify the range of the integer numbers for which this loop is executed.
- Notice that there is no statement **v_counter := v_counter + 1** anywhere, inside or outside the body of the loop.
- The value of v_counter is incremented implicitly by the FOR loop itself.

EXAMPLE

- As a matter of fact, if you include the statement **v_counter := v_counter + 1** in the body of the loop, PL/SQL script will compile with errors.

```
BEGIN
  FOR v_counter IN 1..5
  LOOP
      v_counter := v_counter + 1;
      DBMS_OUTPUT.PUT_LINE('v_counter = '|| v_counter);
  END LOOP;
END;
```

Output

```
BEGIN
*
ERROR at line 1:
ORA-06550: line 3, column 7:
PLS-00363: expression 'V_COUNTER' cannot be used as
an assignment target
ORA-06550: line 3, column 7:
PL/SQL: Statement ignored
```

Using the REVERSE option in the loop

```
BEGIN
    FOR v_counter IN REVERSE 1..5
 LOOP
    DBMS_OUTPUT.PUT_LINE
    ('v_counter = '||v_counter);
 END LOOP;
END;
```

Output

v_counter = 5

v_counter = 4

v_counter = 3

v_counter = 2

v_counter = 1

PL/SQL procedure successfully completed.

Premature Termination Of The Loop

- The EXIT and EXIT WHEN statements can be used inside the body of a numeric FOR loop.
- If the EXIT condition evaluates to TRUE before the loop counter reaches its terminal value, the FOR loop is terminated prematurely.
- If the loop counter reaches its terminal value before the EXIT condition yields TRUE, there is no premature termination of the FOR loop.

Example

```
BEGIN
FOR v_counter IN 1..5
LOOP
DBMS_OUTPUT.PUT_LINE('v_counter = '||v_counter);
EXIT WHEN v_counter = 3;
END LOOP;
END;
```

NESTED LOOPS

- Simple loops, WHILE loops, and numeric FOR loop can be nested inside one another.
- For example, a simple loop can be nested inside a WHILE loop and vice versa.

Example

```
DECLARE
   v_counter1 INTEGER := 0;
   v_counter2 INTEGER;
BEGIN
   WHILE v_counter1 < 3
   LOOP
            DBMS_OUTPUT.PUT_LINE('v_counter1: '||v_counter1);
            v_counter2 := 0;
            LOOP
                 DBMS_OUTPUT.PUT_LINE('v_counter2: '||v_counter2);
                 v_counter2 := v_counter2 + 1;
                 EXIT WHEN v_counter2 >= 2;
            END LOOP;
            v_counter1 := v_counter1 + 1;
   END LOOP;
END;
```

Example explained

- In this example, the WHILE loop is called an *outer loop* because it encompasses the simple loop.
- The simple loop is called an *inner loop* because it is enclosed by the body of the WHILE loop.
- The outer loop is controlled by the loop counter, v_counter1, and it will execute providing the value of v_counter1 is less than 3.
- With each iteration of the loop, the value of v_counter1 is displayed on the screen.
- Next, the value of v_counter2 is initialized to 0. It is important to note that v_counter2 is not initialized at the time of the declaration.
- The simple loop is placed inside the body of the WHILE loop, and the value of v_counter2 must be initialized every time before control is passed to the simple loop.

Example explained

- Once control is passed to the inner loop, the value of v_counter2 is displayed on the screen, and incremented by 1.
- Next, the EXIT WHEN condition is evaluated. If the EXIT WHEN condition evaluates to FALSE, control is passed back to the top of the simple loop.
- If the EXIT WHEN condition evaluates to TRUE, the control is passed to the first executable statement outside the loop.
- In our case, control is passed back to the outer loop, and the value of v_counter1 is incremented by 1, and the TEST condition of the WHILE loop is evaluated again.

Output

v_counter1: 0

v_counter2: 0

v_counter2: 1

v_counter1: 1

v_counter2: 0

v_counter2: 1

v_counter1: 2

v_counter2: 0

v_counter2: 1

PL/SQL procedure successfully completed.

LOOP LABELS

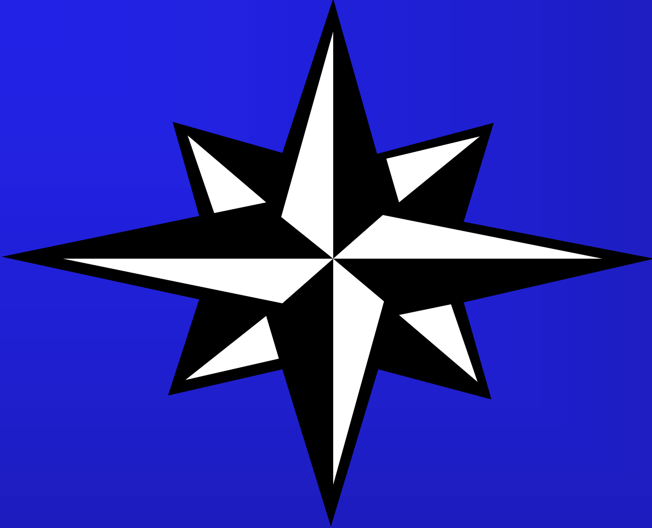
- Loops can be labeled in the similar manner as PL/SQL blocks.

```
<<label_name>>
FOR LOOP_COUNTER IN
   LOWER_LIMIT..UPPER_LIMIT
LOOP
STATEMENT 1;
…
STATEMENT N;
END LOOP label_name;
```

LOOP LABELS

- The label must appear right before the beginning of the loop.
- This syntax example shows that the label can be optionally used at the end of the loop statement.
- It is very helpful to label nested loops because labels improve readability.

```
BEGIN
   <<outer_loop>>
   FOR i IN 1..3
   LOOP
         DBMS_OUTPUT.PUT_LINE('i = '||i);
         <<inner_loop>>
         FOR j IN 1..2
         LOOP
               DBMS_OUTPUT.PUT_LINE('j = '||j);
         END LOOP inner_loop;
   END LOOP outer_loop;
END;
```



END