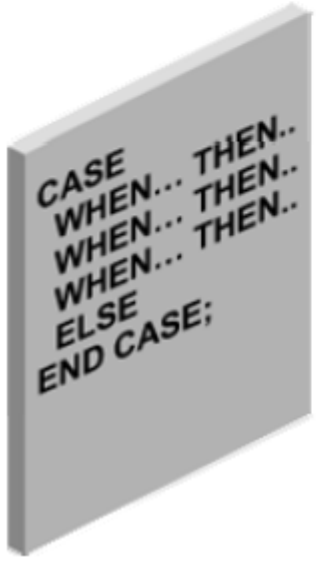
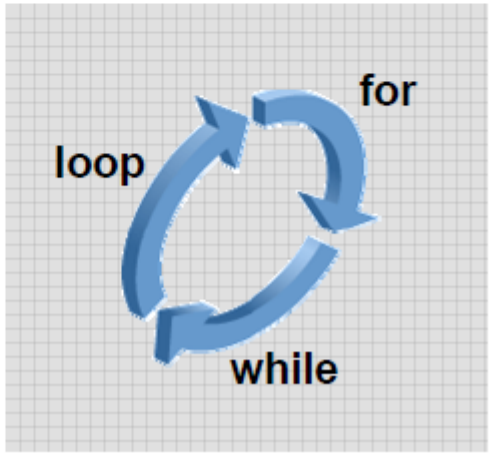
Writing Control structure

Controlling Flow of Execution





IF statement

1

```
IF x>10  Then
.....
......
End if;
```

2

```
IF x>10  Then
.....
......
ELSE
.......
End if;
```

3

```
IF x=10  Then
.....
ELSIF X=9
.......
ELSIF X=8
......
End if;
```

4

```
IF x=10  Then
.....
ELSIF X=9
.......
ELSIF X=8
......
ELSE
......
End if;
```

CASE Expressions

```
CASE selector
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
/
```

```
 CASE
    WHEN search_condition1 THEN result1
    WHEN search_condition2 THEN result2
    ...
    WHEN search_conditionN THEN resultN
   [ELSE resultN+1]
 END;
```

CASE Statement

A CASE expression evaluates the condition and returns a value, whereas a CASE statement evaluates the condition and performs an action. A CASE statement can be a complete PL/SQL block.

- CASE statements end with END CASE;
- CASE expressions end with END;

Handling Nulls

Consider the following example:

```
x := 5;
y := NULL;
...
IF x != y THEN  -- yields NULL, not TRUE
  --  sequence_of_statements that are not executed
END IF;
```

```
a := NULL;
b := NULL;
...
IF a = b THEN  -- yields NULL, not TRUE
  --  sequence_of_statements that are not executed
END IF;
```

Logic Tables

Build a simple Boolean condition with a comparison operator.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

- FALSE takes precedence in an AND condition, and TRUE takes precedence in an OR condition
- AND returns TRUE only if both of its operands are TRUE
- OR returns FALSE only if both of its operands are FALSE
- NULL AND TRUE always evaluates to NULL because it is not known whether the second operand evaluates to TRUE

Note: The negation of NULL (NOT NULL) results in a null value because null values are indeterminate.

Iterative Control: LOOP Statements

- Loops repeat a statement (or sequence of statements) multiple times.
- There are three loop types:
  - Basic loop (Should have exit)
  - FOR loop (based on count)
  - WHILE loop (Based on condition)

The Loop should have exist condition
Otherwise the loop is infinite

Basic Loops

Syntax:

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

WHILE Loops

Syntax:

```
WHILE condition LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Use the WHILE loop to repeat statements while a condition is TRUE.

FOR Loops

- Use a FOR loop to shortcut the test for the number of iterations.
- Do not declare the counter; it is declared implicitly.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

In the syntax:

counter	Is an implicitly declared integer whose value automatically increases or decreases (decreases if the REVERSE keyword is used) by 1 on each iteration of the loop until the upper or lower bound is reached
REVERSE	Causes the counter to decrement with each iteration from the upper bound to the lower bound **Note:** The lower bound is still referenced first.
lower_bound	Specifies the lower bound for the range of counter values
upper_bound	Specifies the upper bound for the range of counter values

Do not declare the counter. It is declared implicitly as an integer.

Nested Loops and Labels

- You can nest loops to multiple levels.
- Use labels to distinguish between blocks and loops.
- Exit the outer loop with the `EXIT` statement that references the label.

Nested Loops and Labels

You can nest `FOR`, `WHILE`, and basic loops within one another. The termination of a nested loop does not terminate the enclosing loop unless an exception was raised. However, you can label loops and exit the outer loop with the `EXIT` statement.

Label names follow the same rules as other identifiers. A label is placed before a statement, either on the same line or on a separate line. White space is insignificant in all PL/SQL parsing except inside literals. Label basic loops by placing the label before the word `LOOP` within label delimiters (<<*label*>>). In `FOR` and `WHILE` loops, place the label before `FOR` or `WHILE`.

If the loop is labeled, the label name can be included (optionally) after the `END LOOP` statement for clarity.

Thank You