

# **CONTROL STRUCTURES: CONDITIONAL CONTROLS**

# Conditional Control

- Conditional control allows you to control the flow of the execution of the program based on a condition.
- In programming terms, it means that the statements in the program are not executed sequentially.
- Rather, one group of statements, or another will be executed depending on how the condition is evaluated.
- In PL/SQL, there are two types of conditional control:
  - IF statement and
  - ELSIF statement.

# IF STATEMENTS

- An IF statement has two forms:  
IF-THEN and IF-THEN-ELSE.
- An IF-THEN statement allows you to specify only one group of actions to take.
- In other words, this group of actions is taken only when a condition evaluates to TRUE.
- An IF-THEN-ELSE statement allows you to specify two groups of actions, and the second group of actions is taken when a condition evaluates to FALSE.

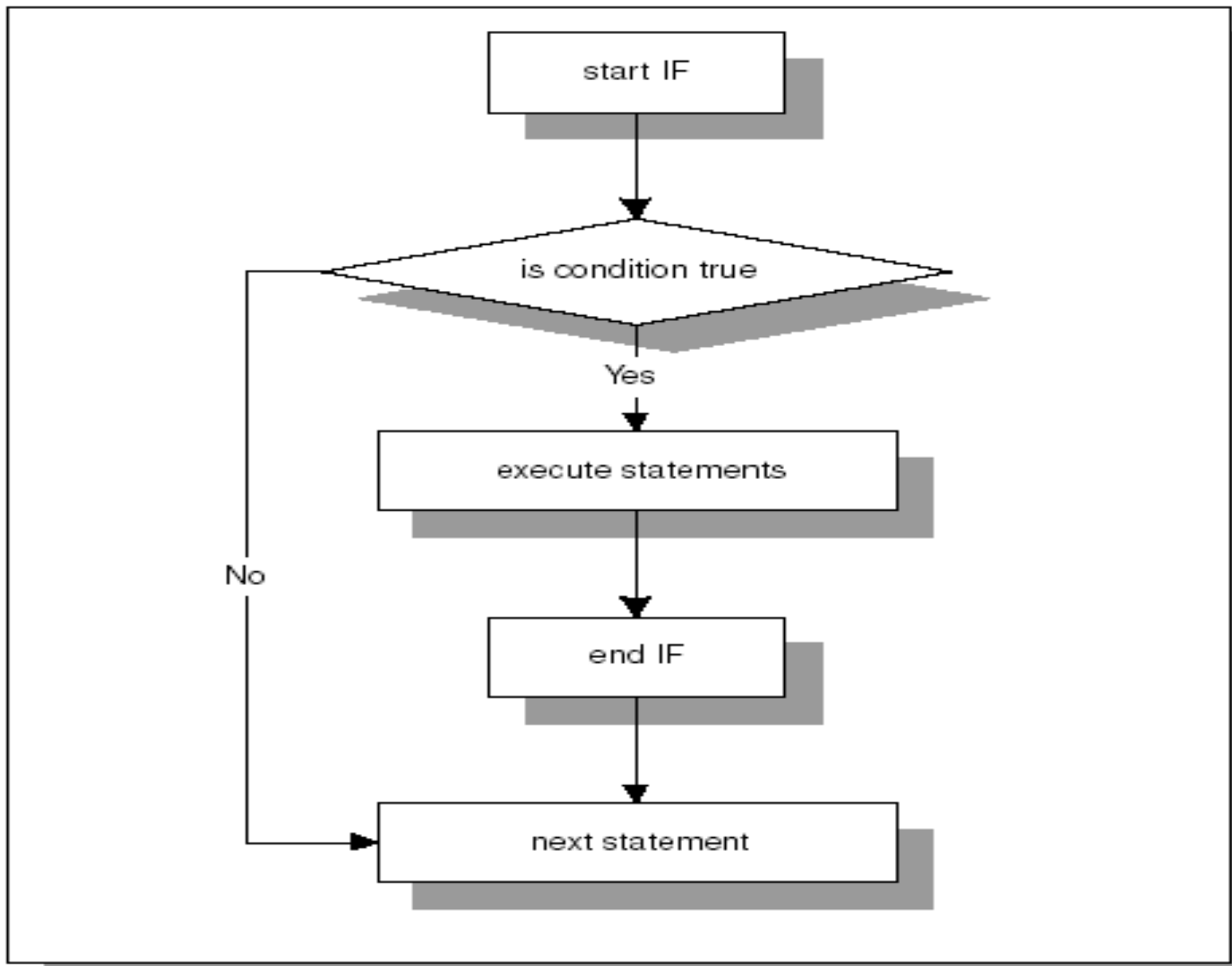
# IF-THEN STATEMENTS

- An IF-THEN statement is the most basic kind of a conditional control and has the following structure:  

```
IF CONDITION
THEN
    STATEMENT 1;
    ...
    STATEMENT N;
END IF;
```
- The reserved word IF marks the beginning of the IF statement.
- Statements 1 through N are a sequence of executable statements that consist of one or more of the standard programming structures.

# IF-THEN STATEMENTS

- The word *CONDITION* between keywords IF and THEN determines whether these statements are executed.
- END IF is a reserved phrase that indicates the end of the IF-THEN construct.



# IF-THEN STATEMENTS

- When an IF-THEN statement is executed, a condition is evaluated to either TRUE or FALSE.
- If the condition evaluates to TRUE, control is passed to the first executable statement of the IF-THEN construct.
- If the condition evaluates to FALSE, the control is passed to the first executable statement after the END IF statement.

# Example

DECLARE

v\_num1 NUMBER := 5;

v\_num2 NUMBER := 3;

v\_temp NUMBER;

BEGIN

-- if v\_num1 is greater than v\_num2 rearrange their

-- values

IF v\_num1 > v\_num2

THEN

v\_temp := v\_num1;

v\_num1 := v\_num2;

v\_num2 := v\_temp;

END IF;

-- display the values of v\_num1 and v\_num2

DBMS\_OUTPUT.PUT\_LINE('v\_num1 = '||v\_num1);

DBMS\_OUTPUT.PUT\_LINE('v\_num2 = '||v\_num2);

Bordoloi and Bock  
END;



## Example explained

- In this example, condition 'v\_num1 > v\_num2' evaluates to TRUE because 5 is greater than 3.
- Next, the values are rearranged so that 3 is assigned to v\_num1, and 5 is assigned to v\_num2.
- It is done with the help of the third variable, v\_temp, that is used as a temporary storage.
- This example produces the following output:

**v\_num1 = 3**

**v\_num2 = 5**

**PL/SQL procedure successfully completed.**

# IF-THEN-ELSE STATEMENT

- An IF-THEN statement specifies the sequence of statements to execute only if the condition evaluates to TRUE.
- When this condition evaluates to FALSE, there is no special action to take except to proceed with execution of the program.
- An IF-THEN-ELSE statement enables you to specify two groups of statements.
  - One group of statements is executed when the condition evaluates to TRUE.
  - Another group of statements is executed when the condition evaluates to FALSE.

# IF-THEN-ELSE STATEMENT

*IF CONDITION*

*THEN*

*STATEMENT 1;*

*ELSE*

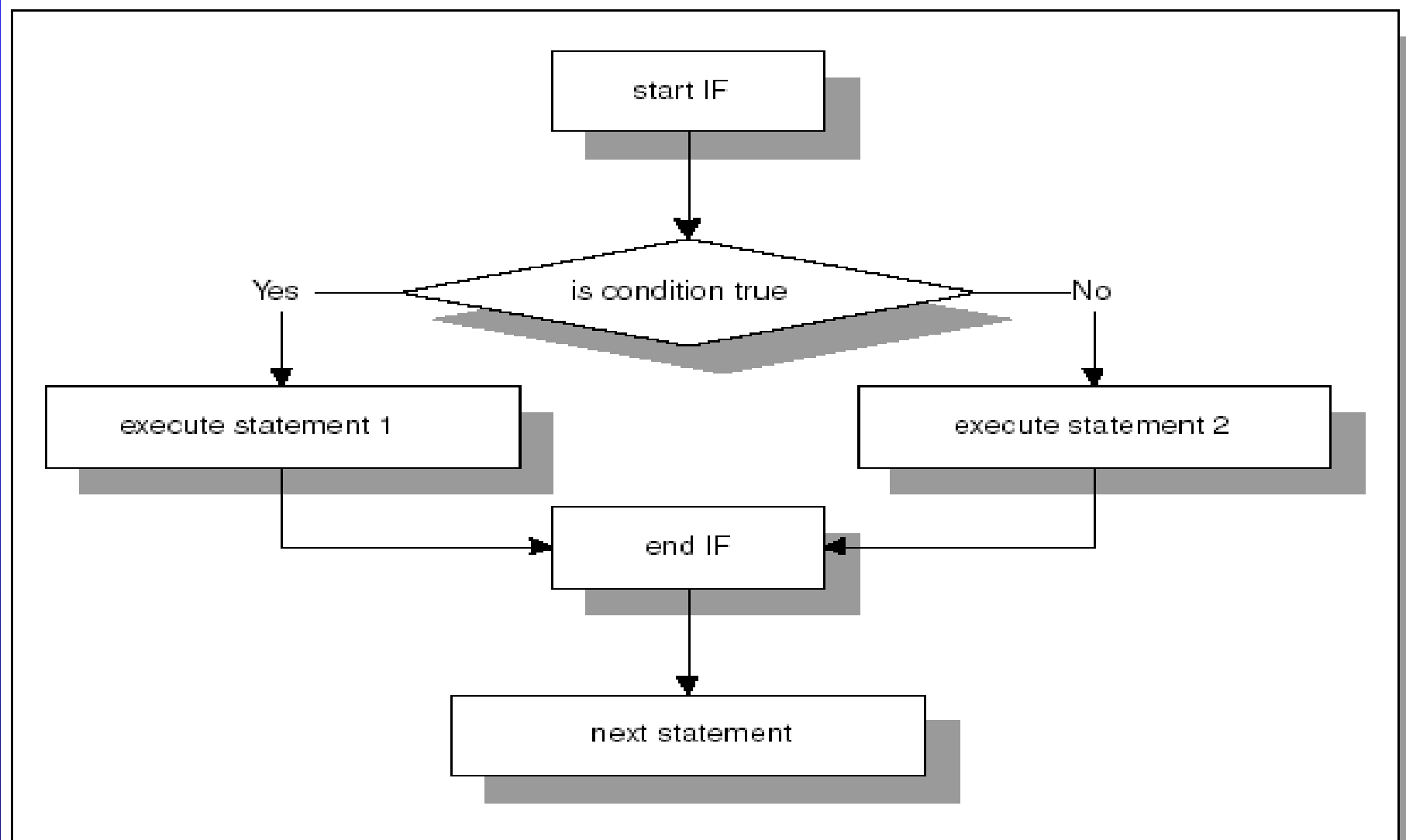
*STATEMENT 2;*

*END IF;*

*STATEMENT 3;*

- When *CONDITION* evaluates to TRUE, control is passed to *STATEMENT 1*;
- When *CONDITION* evaluates to FALSE, control is passed to *STATEMENT 2*.
- After the IF-THEN-ELSE construct has completed, *STATEMENT 3* is executed.

# IF-THEN-ELSE STATEMENT



# Example

```
DECLARE
    v_num NUMBER := &sv_user_num;
BEGIN
-- test if the number provided by the user is even
    IF MOD(v_num,2) = 0
    THEN
        DBMS_OUTPUT.PUT_LINE(v_num||' is even
        number');
    ELSE
        DBMS_OUTPUT.PUT_LINE(v_num||' is odd
        number');
    END IF;
    DBMS_OUTPUT.PUT_LINE('Done...');
END;
```

## Example explained

- It is important to realize that for any given number only one of the DBMS\_OUTPUT.PUT\_LINE statements is executed.
- Hence, the IFTHEN-ELSE construct enables you to specify two and only two mutually exclusive actions.
- When run, this example produces the following output:

Enter value for v\_user\_num: 24

old 2: v\_num NUMBER := &v\_user\_num;

new 2: v\_num NUMBER := 24;

24 is even number

Done...

PL/SQL procedure successfully completed.

# NULL CONDITION

- In some cases, a condition used in an IF statement can be evaluated to NULL instead of TRUE or FALSE.
- For the IF-THEN construct, the statements will not be executed if an associated condition evaluates to NULL.
- Next, control will be passed to the first executable statement after END IF.
- For the IF-THEN-ELSE construct, the statements specified after the keyword ELSE will be executed if an associated condition evaluates to NULL.

# Example

```
DECLARE
```

```
    v_num1 NUMBER := 0;
```

```
    v_num2 NUMBER;
```

```
BEGIN
```

```
    IF v_num1 = v_num2
```

```
    THEN
```

```
        DBMS_OUTPUT.PUT_LINE('v_num1 = v_num2');
```

```
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE('v_num1 != v_num2');
```

```
    END IF;
```

```
END;
```



## Example explained

- This example produces the following output:  
`v_num1 != v_num2`  
**PL/SQL procedure successfully completed.**
- The condition `v_num1 = v_num2` is evaluated to NULL because a value is not assigned to the variable `v_num2`.
- Therefore, variable `v_num2` is NULL.
- Notice that IF-THEN-ELSE construct is behaving as if the condition evaluated to FALSE, and second `DBMS_OUTPUT.PUT_LINE` statement is executed.

# ELSIF STATEMENTS

- An ELSIF statement has the following structure:

```
IF CONDITION 1
THEN
    STATEMENT 1;
ELSIF CONDITION 2
THEN
    STATEMENT 2;
ELSIF CONDITION 3
THEN
    STATEMENT 3;
    ...
ELSE
    STATEMENT N;
END IF;
```

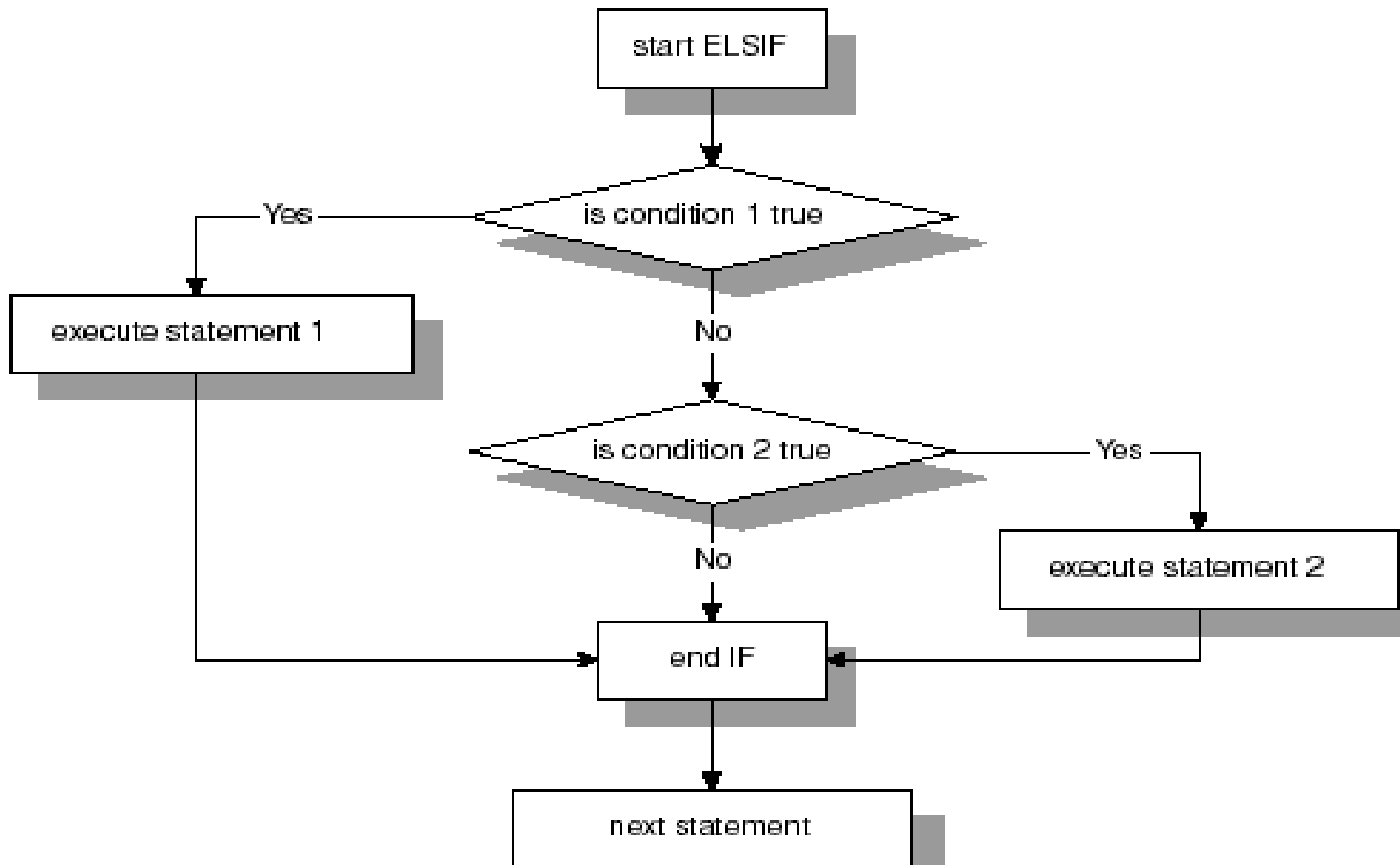
# ELSIF STATEMENTS

- The reserved word IF marks the beginning of an ELSIF construct.
- The words *CONDITION 1* through *CONDITION N* are a sequence of the conditions that evaluate to TRUE or FALSE.
- These conditions are mutually exclusive. In other words, if *CONDITION 1* evaluates to TRUE, STATEMENT1 is executed, and control is passed to the first executable statement after the reserved phrase END IF. The rest of the ELSIF construct is ignored.

# ELSIF STATEMENTS

- When *CONDITION 1* evaluates to FALSE, the control is passed to the ELSIF part and *CONDITION 2* is evaluated, and so forth.
- If none of the specified conditions yield TRUE, the control is passed to the ELSE part of the ELSIF construct.
- An ELSIF statement can contain any number of ELSIF clauses.

# ELSIF STATEMENTS



# Example

```
DECLARE
```

```
    v_num NUMBER := &sv_num;
```

```
BEGIN
```

```
    IF v_num < 0
```

```
    THEN
```

```
        DBMS_OUTPUT.PUT_LINE (v_num||' is a negative number');
```

```
    ELSIF v_num = 0
```

```
    THEN
```

```
        DBMS_OUTPUT.PUT_LINE (v_num ||' is equal to zero');
```

```
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE (v_num||' is a positive number');
```

```
    END IF;
```

```
END;
```

## Example explained

- The value of `v_num` is provided at runtime and evaluated with the help of the ELSIF statement.
- If the value of `v_num` is less than zero, the first `DBMS_OUTPUT.PUT_LINE` statement executes, and the ELSIF construct terminates.
- If the value of `v_num` is greater than zero, both conditions `v_num < 0` and `v_num = 0` evaluate to FALSE, and the ELSE part of the ELSIF construct executes.

## Example explained

- Assume the value of v\_num equals five at runtime.
- This example produces the output shown below:

**Enter value for sv\_num: 5**

**old 2: v\_num NUMBER := &sv\_num;**

**new 2: v\_num NUMBER := 5;**

**5 is a positive number**

**PL/SQL procedure successfully completed.**



# ELSIF STATEMENTS

- There is no second “E” in the “ELSIF”.
- Conditions of an ELSIF statement must be mutually exclusive.
- These conditions are evaluated in sequential order, from the first to the last.
- Once a condition evaluates to TRUE, the remaining conditions of the ELSIF statement are not evaluated at all.
- Consider this example of an ELSIF construct:

# Example

```
IF v_num >= 0
```

```
THEN
```

```
    DBMS_OUTPUT.PUT_LINE ('v_num is greater than 0');
```

```
ELSIF v_num =< 10
```

```
THEN
```

```
    DBMS_OUTPUT.PUT_LINE ('v_num is less than 10');
```

```
ELSE
```

```
    DBMS_OUTPUT.PUT_LINE ('v_num is less than ? or  
greater than ?');
```

```
END IF;
```

# Example explained

- Assume that the value of `v_num` is equal to 5.
- Both conditions of the ELSIF statement potentially can evaluate to TRUE because 5 is greater than 0, and 5 is less than 10.
- However, once the first condition, `v_num >= 0` evaluates to TRUE, the rest of the ELSIF construct is ignored.
- For any value of `v_num` that is greater or equal to 0 and less or equal to 10, these conditions are not mutually exclusive.

## Example explained

- Therefore, DBMS\_OUTPUT.PUT\_LINE statement associated with ELSIF clause will not execute for any such value of v\_num.
- In order for the second condition,  $v\_num \leq 10$ , to yield TRUE, the value of v\_num must be less than 0.

# ELSIF STATEMENTS

- When using an ELSIF construct, it is not necessary to specify what action should be taken if none of the conditions evaluate to TRUE.
- In other words, an ELSE clause is not required in the ELSIF construct.
- Consider the following example:

# Example

```
DECLARE
```

```
    v_num NUMBER := &sv_num;
```

```
BEGIN
```

```
    IF v_num < 0
```

```
    THEN
```

```
        DBMS_OUTPUT.PUT_LINE (v_num||' is a negative  
        number');
```

```
    ELSIF v_num > 0
```

```
    THEN
```

```
        DBMS_OUTPUT.PUT_LINE (v_num||' is a positive  
        number');
```

```
    END IF;
```

```
    DBMS_OUTPUT.PUT_LINE ('Done...');
```

```
END;
```

## Example explained

- As you can see, there is no action specified when v\_num is equal to zero.
- If the value of v\_num is equal to zero, both conditions will evaluate to FALSE, and the ELSIF statement will not execute at all.
- When value of zero is specified for v\_num, this example produces the following output.

**Enter value for sv\_num: 0**

**old 2: v\_num NUMBER := &sv\_num;**

**new 2: v\_num NUMBER := 0;**

**Done...**

**PL/SQL procedure successfully completed.**

# NESTED IF STATEMENTS

- You have encountered different types of conditional controls: IF-THEN statement, IF-THEN-ELSE statement, and ELSIF statement.
- These types of conditional controls can be nested inside of another—for example, an IF statement can be nested inside an ELSIF and vice versa.



# Example

```
DECLARE
```

```
    v_num1 NUMBER := &sv_num1;
```

```
    v_num2 NUMBER := &sv_num2;
```

```
    v_total NUMBER;
```

```
BEGIN
```

```
    IF v_num1 > v_num2
```

```
    THEN
```

```
        DBMS_OUTPUT.PUT_LINE('IF part of the outer IF');
```

```
        v_total := v_num1 - v_num2;
```

```
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE('ELSE part of the outer IF');
```

```
        v_total := v_num1 + v_num2;
```

## Example contd.

```
IF v_total < 0
```

```
THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Inner IF');
```

```
    v_total := v_total * (-1);
```

```
END IF;
```

```
END IF;
```

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

```
END;
```

## Example explained

- The IF-THEN-ELSE statement is called an *outer IF statement* because it encompasses the IF-THEN statement.
- The IF-THEN statement is called an *inner IF statement* because it is enclosed by the body of the IF-THEN-ELSE statement.
- Assume that the value for `v_num1` and `v_num2` are `-4` and `3` respectively.
- First, the condition `v_num1 > v_num2` of the outer IF statement is evaluated. Since `-4` is not greater than `3`, the ELSE part of the outer IF statement is executed.
- As a result, the message **ELSE part of the outer IF**

## Example explained

is displayed, and the value of  $v\_total$  is calculated.

- Next, the condition  $v\_total < 0$  of the inner IF statement is evaluated.
- Since that value of  $v\_total$  is equal  $-1$ , the condition yields TRUE, and message **Inner IF** is displayed.
- Next, the value of  $v\_total$  is calculated again.
- This logic is demonstrated by the output produced by the example:

# Output

**Enter value for sv\_num1: -4**

**old 2: v\_num1 NUMBER := &sv\_num1;**

**new 2: v\_num1 NUMBER := -4;**

**Enter value for sv\_num2: 3**

**old 3: v\_num2 NUMBER := &sv\_num2;**

**new 3: v\_num2 NUMBER := 3;**

**ELSE part of the outer IF**

**Inner IF**

**v\_total = 1**

**PL/SQL procedure successfully completed.**

# LOGICAL OPERATORS

- So far you have seen examples of different IF statements. All these examples used test operators such as  $>$ ,  $<$ , and  $=$ , to test a condition.
- Logical operators can be used to evaluate a condition as well.
- In addition, they allow a programmer to combine multiple conditions into a single condition if there is such a need.

# Example

```
DECLARE
```

```
    v_letter CHAR(1) := '&sv_letter';
```

```
BEGIN
```

```
    IF (v_letter >= 'A' AND v_letter <= 'Z')
```

```
        OR (v_letter >= 'a' AND v_letter <= 'z')
```

```
    THEN
```

```
        DBMS_OUTPUT.PUT_LINE('This is a letter');
```

```
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE('This is not a letter');
```

```
        IF v_letter BETWEEN '0' and '9'
```

```
        THEN
```

## Example contd.

```
        DBMS_OUTPUT.PUT_LINE('This is a number');
ELSE
        DBMS_OUTPUT.PUT_LINE('This is not a number');
END IF;
END IF;
END;
```



# Example explained

- In the example above, the condition  
(v\_letter >= 'A' AND v\_letter <= 'Z')  
OR (v\_letter >= 'a' AND v\_letter <= 'z')  
uses logical operators AND and OR.
- There are two conditions  
(v\_letter >= 'A' AND v\_letter <= 'Z')  
and  
(v\_letter >= 'a' AND v\_letter <= 'z')  
combined into one with the help of the OR operator.

## Example explained

- It is also important for you to realize the purpose of the parentheses.
- In this example, they are used to improve the readability only because the operator AND takes precedence over the operator OR.
- When the symbol “?” is entered at runtime, this example produces the following output

# Output

**Enter value for sv\_letter: ?**

**old 2: v\_letter CHAR(1) := '&sv\_letter';**

**new 2: v\_letter CHAR(1) := '?';**

**This is not a letter**

**This is not a number**

**PL/SQL procedure successfully completed.**



END