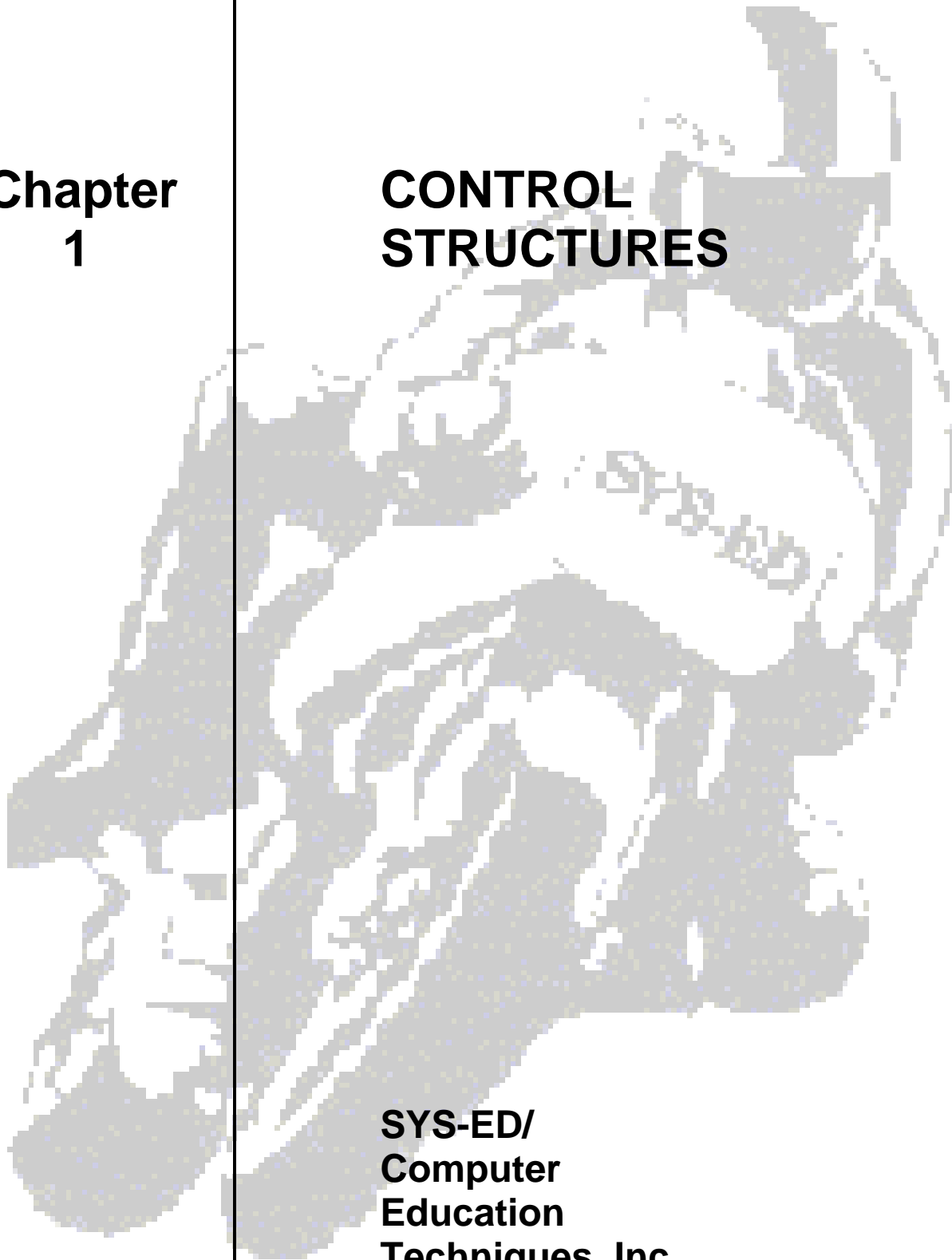


**Chapter
1**

**CONTROL
STRUCTURES**



**SYS-ED/
Computer
Education
Techniques, Inc.**

Objectives

You will learn:

- Uses and types of control structures.
- Constructing an IF statement.
- CASE expressions.
- Constructing and identifying different loop statements.
- Logic tables.
- Controlling block flow using nested loops and labels.

1 IF Statements

The structure of the PL/SQL IF statement is similar to the structure of IF statements in other procedural languages. It allows PL/SQL to perform actions selectively based on conditions.

There are three forms of IF statements:

- IF-THEN-END IF
- IF-THEN-ELSE-END IF
- IF-THEN-ELSIF-END IF

Syntax:

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

Keyword	Description
condition	Boolean variable or expression (TRUE, FALSE, or NULL). It is associated with a sequence of statements, which is executed only if the expression yields TRUE.
THEN	Clause that associates the Boolean expression that precedes it with the sequence of statements that follows it.
statements	Can be one or more PL/SQL or SQL statements. They may include further IF statements containing several nested IF, ELSE, and ELS IF statements.
ELSIF	Keyword that introduces a Boolean expression. If the first condition yields FALSE or NULL then the ELSI F keyword introduces additional conditions.
ELSE	Keyword that executes the sequence of statements that follows it if the control reaches it.

1.1 Simple IF Statements

Example:

PL/SQL assigns values to the following variables, only if the condition is TRUE:

If the last name is Smith:

- Set job ID to SAREP.
- Set department number to 50.

```
IF v_last_name = 'Smith' THEN
    v_job_id := 'SAREP';
    v_dept_no := 50;
END IF;
```

If the condition is FALSE or NULL, PL/SQL ignores the statements in the IF block. In either case, control resumes at the next statement in the program following the END IF.

1.2 Compound IF Statements

Compound IF statements use logical operators like AND and NOT.

Example:

The IF statement has two conditions to evaluate:

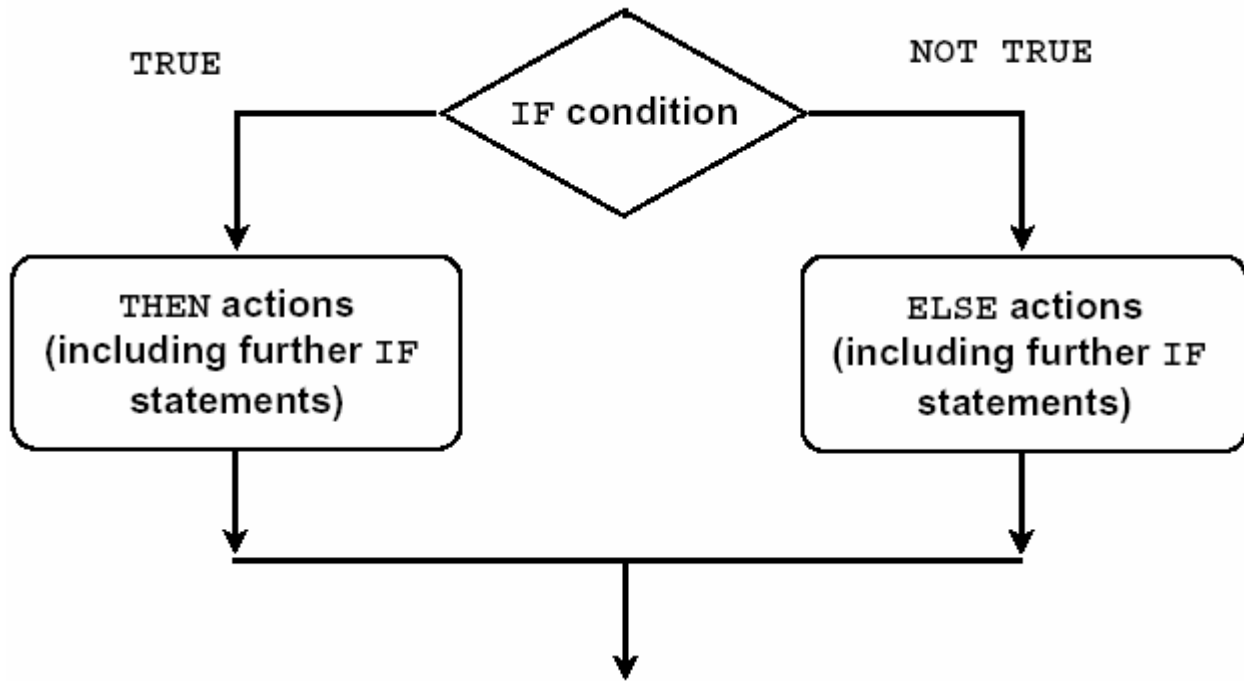
- Last name should be Smith.
- Salary should be greater than 5800.
- Only if both the above conditions are evaluated as TRUE, v_dept_no is set to 75.

```
IF v_last_name = 'Smith' AND v_salary > 5800 THEN
    v_dept_no := 75;
END IF;
```

1.3 IF-THEN-ELSE Statement Execution Flow

In an IF construct, if the condition is FALSE or NULL, the ELSE clause can be used to carry out other actions. As with the simple IF statement, control resumes in the program from the END IF clause.

```
IF conditional THEN
    statement;
ELSE
    statement2;
END IF;
```



1.4 Nested IF Statements

Either set of actions of the result of the first IF statement can include further IF statements before specific actions are performed. The THEN and ELSE clauses can include IF statements.

Each nested IF statement must be terminated with a corresponding END IF clause.

```
IF conditional THEN
    statement;
ELSE
    IF condition2 THEN
        statement2;
    END IF;
END IF;
```

2 CASE Expressions

A CASE expression selects a result and returns it. To select the result, the CASE expression uses a selector, an expression whose value is used to select one of several alternatives.

The selector is followed by one or more WHEN clauses, which are checked sequentially. The value of the selector determines which clause is executed. If the value of the selector equals the value of a WHEN-clause expression, that WHEN clause is executed.

PL/SQL also provides a searched CASE expression, which has the form:

```
CASE
    WHEN search conditional THEN result1
    WHEN search conditiona2 THEN result2
    .
    .
    .
    WHEN search conditionaN THEN resultN
[ELSE resultN+1;]
END;
/
```

A searched CASE expression has no selector. Also, its WHEN clauses contain search conditions that yield a boolean value, not expressions that can yield a value of any type.

Example:

```
SET SERVEROUTPUT ON

DEFINE p_grade = a
DECLARE
    v_grade CHAR(1) := UPPER('&p_grade');
    v_appraisal VARCHAR2 (20);
BEGIN
    v_appraisal :=
    CASE v_grade
    WHEN 'A' THEN 'Excellent'
    WHEN 'B' THEN 'Very Good'
    WHEN 'C' THEN 'Good'
    ELSE 'No such grade'
    END;

    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || 'Appraisal ' ||
    v_appraisal);

END;
/
```

The CASE expression uses the value in the v_grade variable as the expression. This value is accepted from the user using a substitution variable. Based on the value entered by the user, the CASE expression evaluates the value of the v_appraisal variable based on the value of the v_grade value.

Example:

```
DEFINE p_grade = a
DECLARE

    v_grade CHAR(1) := UPPER('&p grade');
    v_appraisal VARCHAR2 (20);
BEGIN
    v_appraisal :=
    CASE
    WHEN v_grade = 'A' THEN 'Excellent'
    WHEN v_grade = 'B' THEN 'Very Good'
    WHEN v_grade = 'C' THEN 'Good'
    ELSE 'No such grade'
    END;

    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || 'Appraisal ' ||
v_appraisal);
```

This code utilizes a searched CASE expression.

3 Handling Nulls

When working with nulls, following rules will minimize errors:

- Simple comparisons involving nulls always yield NULL.
- Applying the logical operator NOT to a null yields NULL.
- In conditional control statements, if the condition yields NULL, its associated sequence of statements is not executed.

Example 1:

```
x := 5;
y := NULL;
IF x != y THEN -- yields NULL, not TRUE,
    sequenceof statements; -- not executed
END IF;
```

Whether or not x is equal to y is unknown. Therefore, the IF condition yields NULL and the sequence of statements is bypassed.

Example 2:

Since there is no certainty with a and b being equal, the IF condition will yield NULL and the sequence of statements is bypassed.

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

4 Boolean Conditions with Logical Operators

A Boolean condition can be created by combining number, character, or date expressions with comparison operators.

A complex Boolean condition can be created by combining simple Boolean conditions with the logical operators AND, OR, and NOT. In the logic tables shown in the slide:

- 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 evaluate to NULL because it is not known whether the second operand evaluates to TRUE or not.
- The negation of NULL (NOT NULL) results in a null value because null values are indeterminate.

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

5 Iterative Control: LOOP Statements

PL/SQL provides a number of facilities to structure loops to repeat a statement or sequence of statements multiple times.

Looping constructs are the second type of control structure. PL/SQL provides the following types of loops:

- Basic loop that perform repetitive actions without overall conditions.
- FOR loops that perform iterative control of actions based on a count.
- WHILE loops that perform iterative control of actions based on a condition.

The EXIT statement is used to terminate loops.

5.1 Basic Loops

The simplest form of LOOP statement is the basic or infinite loop, which encloses a sequence of statements between the keywords LOOP and END LOOP. Each time the flow of execution reaches the END LOOP statement, control is returned to the corresponding LOOP statement above it.

A basic loop allows execution of its statement at least once, even if the condition is already met upon entering the loop. Without the EXIT statement, the loop would be infinite.

Syntax:

```

LOOP                               -- delimiter
    Statement1;                     -- statements
    ...
EXIT [WHEN condition];             -- EXIT statement
END LOOP;                           -- delimiter

```

Keyword	Description
Condition	A Boolean variable or expression (TRUE, FALSE, or NULL).

Example:

```
DECLARE
    v_country_id    locations.country_id%TYPE := 'FR';
    v_location_id   locations.location_id%TYPE;
    v_counter       NUMBER(2) := 1;
    v_city          locations.city%TYPE := 'Paris';
BEGIN
    SELECT MAX(location_id) INTO v_location_id FROM locations
    WHERE country_id = v_country_id;
    LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_location_id + v_counter) ,v_city, v_country_id );
        v_counter := v_counter + 1;
        EXIT WHEN v_counter > 3;
    END LOOP;
END;
/
```

This basic loop example slide is defined as follows: Insert three new locations IDs for the country code of FR and the city of Paris.

A basic loop allows execution of its statements at least once, even if the condition has been met upon entering the loop, provided the condition is placed in the loop so that it is not checked until after these statements. However, if the exit condition is placed at the top of the loop, before any of the other executable statements, and that condition is true, the loop will exit and the statements will never execute.

5.2 WHILE Loops

The WHILE loop is used to repeat a sequence of statements until the controlling condition is no longer TRUE. The condition is evaluated at the start of each iteration. The loop terminates when the condition is FALSE. If the condition is FALSE at the start of the loop, then no further iterations are performed.

Syntax:

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

Keyword	Description
Condition	A Boolean variable or expression (TRUE, FALSE, or NULL).
Statement	One or more PL/SQL or SQL statements.

If the variables involved in the conditions do not change during the body of the loop, then the condition remains TRUE and the loop does not terminate.

If the condition yields NULL, the loop is bypassed and control passes to the next statement.

Example:

```
DECLARE
    v_country_id    locations.country_id%TYPE := 'FR';
    v_location_id   locations.location_id%TYPE;
    v_city          locations.city%TYPE := 'Paris';
    v_counter       NUMBER := 1;
BEGIN
    SELECT MAX(location_id) INTO v_location_id FROM locations
    WHERE country_id = v_country_id;
    WHILE v_counter <= 3 LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_location_id + v_counter), v_city, v_country_id );
        v_counter := v_counter + 1;
    END LOOP;
END;
/
```

In this example, three new locations IDs for the country code of FR and the city of Paris are being added.

With each iteration through the WHILE loop, a counter (v_counter) is incremented. If the number of iterations is less than or equal to the number 3, the code within the loop is executed and a row is inserted into the LOCATIONS table. After the counter exceeds the number of items for this location, the condition that controls the loop evaluates to FALSE and the loop is terminated.

5.3 FOR Loops

FOR loops have the same general structure as the basic loop. In addition, they have a control statement before the LOOP keyword to determine the number of iterations that PL/SQL performs.

Syntax:

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

Keyword	Description
Counter	An explicitly declared integer whose value automatically increases 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.
lower_bound	The lower bound for the range of counter values.
upper_bound	The upper bound for the range of counter values.

Example:

```
DECLARE
    v_country_id      locations.country_id%TYPE := 'FR';
    v_location_id     locations.location_id%TYPE;
    v_city            locations.city%TYPE : 'Paris';

BEGIN
    SELECT MAX(location_id) INTO v_location_id
    FROM locations
    WHERE country_id = v_country_id;
    FOR i IN 1..3 LOOP
        INSERT INTO locations(v_location_id, v_city, v_country_id)
        VALUES ((v_location_id + i), v_city, v_country_id);
    END LOOP;
END;
```

This code inserts three new locations for the country code of FR and the city of Paris using a FOR loop.

5.4 Guidelines for Using Loops

A basic loop allows execution of its statement at least once, even if the condition is already met upon entering the loop. Without the EXIT statement, the loop would be infinite.

- WHILE loop is used to repeat a sequence of statements until the controlling condition is no longer TRUE. The condition is evaluated at the start of each iteration.

The loop terminates when the condition is FALSE. If the condition is FALSE at the start of the loop, then no further iterations are performed.

- FOR loops have a control statement before the LOOP keyword to determine the number of iterations that PL/SQL performs.

Use a FOR loop if the number of iterations is predetermined.

6 Nested Loops and Labels

Loops can be nested to multiple levels. The termination of a nested loop does not terminate the enclosing loop unless an exception was raised. However, the EXIT statement can be used for labeling loops and exiting the outer loop.

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. Label loops by placing the label before the word LOOP within label delimiters (<<label>>).

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

```
BEGIN
    <<Outer_loop>>
    LOOP
        v_counter := v_counter+1;
    EXIT WHEN v_counter>10;
    <<inner_loop>>
    LOOP
        ...
        EXIT Outer_loop WHEN total_done = 'YES';
        -- Leave both loops
        EXIT WHEN inner done = 'YES';
        -- Leave inner loop only
        ...
    END LOOP Inner_loop;
    ...
END LOOP Outer_loop;
END;
```

This code contains two loops.

- The outer loop is identified by the label, <<Outer_Loop>> and the inner loop is identified by the label <<Inner_Loop>>. The identifiers are placed before the word LOOP within label delimiters (<<label>>).
- The inner loop is nested within the outer loop. The label names are included after the END LOOP statement for clarity.

7 EXIT Statement

The EXIT statement can be used to terminate a loop. Control passes to the next statement after the END LOOP statement.

EXIT can be issued as either as an action within an IF statement or as a stand-alone statement within the loop. The EXIT statement must be placed inside a loop. In the latter case, a WHEN clause can be added to allow conditional termination of the loop.

When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition yields TRUE, the loop ends and control passes to the next statement after the loop. A basic loop can contain multiple EXIT statements.