**Control Statements**

(Notes #3)

Topics:

1. **Conditional** Selection statements

IF statement, IF-THEN, IF-ELSE, IF-ELSIF; nested IF.

1. CASE statements (also see Note1b\_Advanced\_SQL, #1) (page 4)
2. Iterative Processing with **Loops** (page 6)

Basic Loop Statement

Exit [when], Continue [When]

For Loop

While loop,

1. **Sequential** control statements, GOTO statement; Null statement (page 10)

PL/SQL has three categories of control statements: conditional selection statements, loop statements and sequential control statements.

1. **Conditional Selection Statements**

- IF … THEN … END IF;

- IF … THEN … ELSE … END IF;

- IF … THEN … ELSIF … THEN … ELSE … END IF;

The sequence of statements in the IF clause is executed only if the condition is TRUE.

Condition is a Boolean variable, its possible value true, false, null.

In coding, “IF Boolean\_variable THEN” is same as

“IF Boolean\_variable = TRUE THEN”. The former is preferred.

Logic structure:

if condition then

statement1; -- if the condition is true, execute these statements

statement2;

. . . -- if the condition is false or null, skip all these statements

end if;

if-then-else statement

if condition then

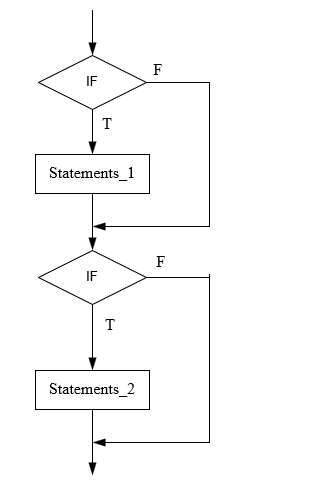
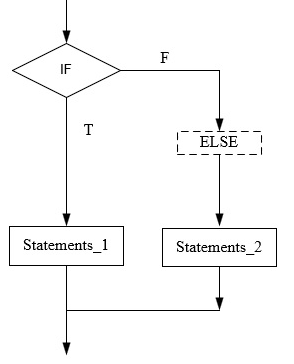
statements\_1 -- if the condition is true, execute these statements

-- also ignore the “else” below.

else -- Notice that there is No “THEN” after ELSE.

statements\_2 -- if the IF- condition is false or null, execute these

end if ;



**Note:** the difference between IF - ELSE and IF - IF.

**Example 1**. (IF-THEN, END IF;)

DECLARE

GRADE NUMBER;

BEGIN

GRADE := 80;

-- or GRADE := &g; that will prompt user for input, G is dummy place holder

IF GRADE >= 60 THEN

DBMS\_OUTPUT.PUT\_LINE ('Your grade is PASS.');

END IF;

END;

RESULT

Your grade is PASS.

**Example 2.** (IF-THEN, ELSE, END IF;)

DECLARE

GRADE NUMBER;

BEGIN

GRADE := 50; -- change the 50 to 80, run again

IF GRADE >= 60 THEN

DBMS\_OUTPUT.PUT\_LINE ('Your grade is PASS.');

ELSE

DBMS\_OUTPUT.PUT\_LINE ('Your grade is FAIL.');

END IF;

END;

RESULT

Your grade is FAIL.

**if-then-elsif statement:** -- Pay attention that “ElSIF”, no“e”

if condition1 then

statements\_1 -- if the condition1 is true, execute these statements

-- also ignore all the other conditions below.

elsif condition2 then -- any number of elsif is allowed

statements\_2 -- if the condition2 is true, execute these statements

-- also ignore all the other conditions below.

elsif condition3 then

statements\_x -- if the condition3 is true, execute these statements

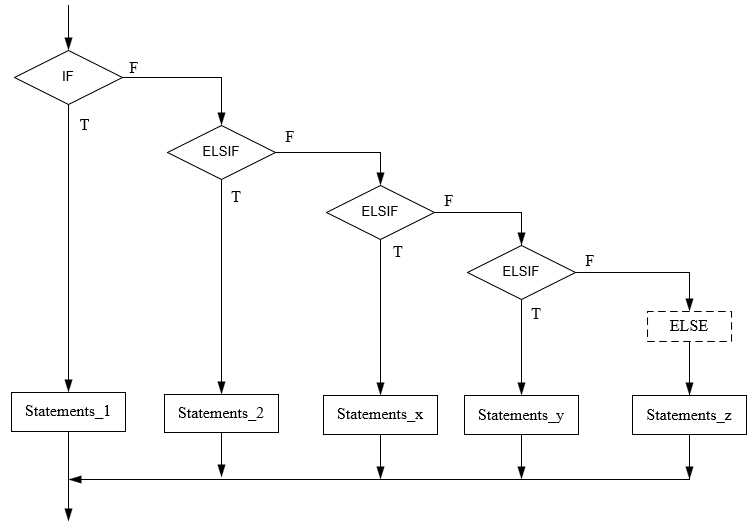
. . .

else -- this final else is optional; no “THEN” for “else”

statements\_Z -- if all the above if-elsif are false, execute these

end if;

Below is if-then, elsif-then, else, end if flow chart.



**Example 3.**

DECLARE

grade NUMBER;

BEGIN

grade := 80; /\* or grade := &grade, input from user \*/

IF grade >= 90 THEN

DBMS\_OUTPUT.PUT\_LINE ('your grade is A.');

ELSIF grade >= 80 THEN

DBMS\_OUTPUT.PUT\_LINE ('your grade is B.');

ELSIF grade >= 70 THEN

DBMS\_OUTPUT.PUT\_LINE ('your grade is C.');

ELSIF grade >= 60 THEN

DBMS\_OUTPUT.PUT\_LINE ('your grade is D.');

Else

DBMS\_OUTPUT.PUT\_LINE ('your grade is F.');

END IF;

END;

result

your grade is B.

You may replace the “ ELSE ...” with

ELSIF grade < 60 THEN

DBMS\_OUTPUT.PUT\_LINE ('your grade is F.');

**Example 4**. Nested IF THEN statement,

CREATE TABLE test\_emps AS SELECT \* FROM employees;

SET SERVEROUTPUT ON;

DECLARE

sales NUMBER := 10100; -- you may ask user to input these values

quota NUMBER := 10000;

emp\_id NUMBER := 120;

bonus NUMBER := 0;

BEGIN

IF sales > (quota + 200) THEN

bonus := (sales - quota)/4;

ELSE

**IF** sales > quota **THEN --** Nested IF THEN

bonus := 50;

**ELSE**

bonus := 0;

**END IF**;

END IF;

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

UPDATE test\_emps

SET salary = salary + bonus

WHERE employee\_id = emp\_id;

END;

-rollback;

You may find the results by running query as below before and after the program

select salary from test\_emps where employee\_id = 120;

1. **CASE statements**

The CASE statement chooses from a sequence of conditions and runs a corresponding

statement, it is appropriate when a different action is to be taken for each alternative.

Refer to section #1, CASE statement in Note1b.Advanced\_SQL section #1, where the samples are SQL select statements.

**2.1) Simple case statement**

The Simple CASE statement evaluates a single expression and compares it to several potential values.

Simple CASE Statements

CASE selector

WHEN selector\_value\_1 THEN statement\_1;

WHEN selector\_value\_2 THEN statement\_2;

...

WHEN selector\_value\_n THEN statement\_n;

[ELSE ELSE\_statements;]

END CASE;

**Example 5A**.

DECLARE

grade CHAR (1);

BEGIN

grade := 'B'; -- you may change the value and run again.

-- or := &grade to ask user to input.

CASE grade

WHEN 'A' THEN DBMS\_OUTPUT.PUT\_LINE ('Excellent');

WHEN 'B' THEN DBMS\_OUTPUT.PUT\_LINE ('Very Good');

WHEN 'C' THEN DBMS\_OUTPUT.PUT\_LINE ('Good');

WHEN 'D' THEN DBMS\_OUTPUT.PUT\_LINE ('Fair');

WHEN 'F' THEN DBMS\_OUTPUT.PUT\_LINE ('Poor');

ELSE DBMS\_OUTPUT.PUT\_LINE ('No such grade');

END CASE;

END;

**2.2) Searched CASE** Statements

The searched CASE statement evaluates multiple Boolean expressions and chooses

the first one whose value is TRUE.

It has the structure:

CASE

WHEN condition\_1 THEN statements\_1;

WHEN condition\_2 THEN statements\_2;

...

WHEN condition\_n THEN statements\_n;

[ELSE ELSE\_statements;]

END CASE;

**Example 5B**.

DECLARE

grade CHAR (1);

BEGIN

grade := 'B';

CASE

WHEN **grade =** 'A' THEN DBMS\_OUTPUT.PUT\_LINE ('Excellent');

WHEN grade = 'B' THEN DBMS\_OUTPUT.PUT\_LINE ('Very Good');

WHEN grade = 'C' THEN DBMS\_OUTPUT.PUT\_LINE ('Good');

WHEN grade = 'D' THEN DBMS\_OUTPUT.PUT\_LINE ('Fair');

WHEN grade = 'F' THEN DBMS\_OUTPUT.PUT\_LINE ('Poor');

ELSE DBMS\_OUTPUT.PUT\_LINE ('No such grade');

END CASE;

END;

Compare: in simple case, the condition is just the selector value; the latter is a condition statement. The ELSE clause can be replaced by EXCEPTION clause as below.

**Example 5C.**

DECLARE

grade CHAR (1);

BEGIN

grade := 'B';

CASE

WHEN grade = 'A' THEN DBMS\_OUTPUT.PUT\_LINE ('Excellent');

WHEN grade = 'B' THEN DBMS\_OUTPUT.PUT\_LINE ('Very Good');

WHEN grade = 'C' THEN DBMS\_OUTPUT.PUT\_LINE ('Good');

WHEN grade = 'D' THEN DBMS\_OUTPUT.PUT\_LINE ('Fair');

WHEN grade = 'F' THEN DBMS\_OUTPUT.PUT\_LINE ('Poor');

/\* ELSE DBMS\_OUTPUT.PUT\_LINE ('No such grade'); \*/

END CASE;

EXCEPTION

WHEN CASE\_NOT\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE ('No such grade');

END;

1. **Iterative Processing with Loops.**

Loop statements run the same statements with a series of different values.

The basic LOOP could be infinitive, we need to control/end the iteration (exit when); the counter, or (dummy) index may be used.

Looping structure,

Loop … End Loop

For in Loop ... End Loop -- indexed loop

While Loop ... End Loop

Cursor For loop ... End loop -- will be discuss next class.

* 1. **Basic Loop Statement**

[ <<LABEL>> ] LOOP --[ <<label>> ] optional

statements;

EXIT [WHEN] -- "exit" forces a break out of the loop.

END LOOP [LABEL]; -- the label at end of loop must match the claimed name

LOOP

statements....

IF condition THEN

EXIT; -- if condition is true, then exit.

END IF;

END LOOP;

**Exit and Exit When Statement**

The EXIT statement exits the current iteration of a loop unconditionally and transfers control to the *end* of either the *current loop* or an enclosing labeled loop.

EXIT WHEN condition -- same as: if condition then exit;

EXIT loop\_label\_name -- break out of the Labeled loop

**Example 6A.**

DECLARE

x NUMBER := 0; -- the counter needed to run the loop

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE ('Inside loop: x = ' || TO\_CHAR(x));

x := x + 1;

IF x > 3 THEN

EXIT;

END IF;

END LOOP;

**-- After EXIT, control resumes here**

DBMS\_OUTPUT.PUT\_LINE (' After loop: x = ' || TO\_CHAR(x));

END;

The statement of

Exit when x > 3;

is same as:

IF x > 3 THEN

EXIT;

END IF;

**Continue and Continue When Statement**

The CONTINUE statement exits the current iteration of a loop unconditionally and

transfers control to the *next iteration* of either the current loop or an enclosing labeled loop.

The CONTINUE WHEN statement exits the current iteration of a loop when the condition in its WHEN clause is true, and transfers control to the next iteration of either the current loop or an enclosing labeled loop.

**Example 6B.**

DECLARE

x NUMBER := 0;

BEGIN

LOOP **-- After CONTINUE statement, control resumes here**

DBMS\_OUTPUT.PUT\_LINE ('Inside loop: x = ' || TO\_CHAR(x));

x := x + 1;

IF x < 3 THEN

**CONTINUE;**

END IF;

-- when x = 0, 1, 2, “continue” executed, control goes back to “LOOP”

-- statements below are skipped.

**-- CONTINUE when x < 3; will be the same**

DBMS\_OUTPUT.PUT\_LINE('Inside loop, after CONTINUE: x = ' ||

TO\_CHAR(x));

EXIT WHEN x = 5;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE (' After loop: x = ' || TO\_CHAR(x));

END;

Result

Inside loop: x = 0

Inside loop: x = 1

Inside loop: x = 2

Inside loop, after CONTINUE: x = 3 -- no “continue” after x =3, or x>3.

Inside loop: x = 3

Inside loop, after CONTINUE: x = 4

Inside loop: x = 4

Inside loop, after CONTINUE: x = 5

After loop: x = 5

* 1. **For Loops**

The FOR LOOP statement runs one or more statements while the loop *index* is in a

specified range. The statement has this structure:

for *index*  in [reverse] *lower\_bound* ***..*** *upper\_bound* loop

statements; -- pay attention to double dot " **..** "

end loop;

* No need to declare the *index*; implicitly it should be an integer.
* Without reverse, the value of index starts at lower\_bound, and increases by one with each iteration of the loop until it reaches upper\_bound.
* With reverse, the value of index starts at upper\_bound, and decreases by one with each iteration of the loop until it reaches lower\_bound.

**Example 8a.**

BEGIN

FOR i IN 1 .. 5 LOOP -- i is dummy index, no need to declare

DBMS\_OUTPUT.PUT\_LINE ('The value of i is: ' || TO\_CHAR(i));

END LOOP;

END;

Output:

The value of i is: 1

. . .

**Warning:**

BEGIN

FOR i IN 1 .. 5 LOOP

DBMS\_OUTPUT.PUT\_LINE ('The value of i is: ' || TO\_CHAR(i));

-- ***i := i + 1*** -- this is wrong, you cannot change the index

END LOOP;

END;

**Example 8b.**

BEGIN

FOR i IN REVERSE **1 .. 5** LOOP -- order is still **1 .. 5**

DBMS\_OUTPUT.PUT\_LINE ('The value of i is: ' || TO\_CHAR(i));

END LOOP;

END;

/

Output:

The value of i is: 5

. . .

* 1. **While Loops Statement**

The WHILE LOOP statement runs one or more statements while a condition is true.

It has the structure:

#### while ***condition*** loop

#### statements

#### end loop;

-- The *condition* is evaluated, when true the statements in block is executed, when the condition is false, the loop ends.

To prevent an infinite loop, program statements should include condition test.

#### while loop

-- this is an infinite loop; program statement here

IF *CONDITION* THEN

EXIT; -- break out of the loop

END IF; -- or if ... then Raise ... , control passes to exception

END LOOP;

**Example 9.**

DECLARE

x NUMBER := 0;

BEGIN

WHILE x < 4 LOOP

DBMS\_OUTPUT.PUT\_LINE ('Inside loop: x = ' || TO\_CHAR(x));

x := x + 1;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE (' After loop: x = ' || TO\_CHAR(x));

END;

**Guidelines for Loops**

The basic loop could run infinitely; different loop statements have its own way to control when to exit, and manager the counter.

* Use the basic loop when the statements inside the loop must execute at least once.

The program inside the loop needs to manage the “exit” and counter.

* Use the WHILE loop if the condition must be evaluated at the start of each iteration.

The exit *condition* usually is defined in WHILE *condition* loop; and judge before the starting of the loop. Program defines/controls the counter

* Use a FOR loop if the number of iterations is known.

The counter is defined as dummy index in the “for” clause, it automatically increases one for each loop.

1. **Sequential control Statements**

**4.1) GOTO Label\_Name**

It is not encouraged to use this command.

Unconditionally branch.

goto label\_name;

<<label\_name >> --- somewhere in the program

executable statement;

Restriction rules:

* Branch into an *IF* statement, a loop statement, or a sub-block is not allowed
* Branch from one *IF* statement clause to another is not allowed
* Branch from an exception handler into the current block is not allowed.
* Branch out of a sub-program is not allowed

**4.2) Null statement: Do nothing**

The NULL statement does nothing other than pass control to the next statement.

It helps readability.

**Example 10.** NULL Statement Showing No Action

DECLARE

v\_job\_id VARCHAR2(10);

v\_emp\_id NUMBER(6) := 110;

BEGIN

SELECT job\_id INTO v\_job\_id

FROM employees

WHERE employee\_id = v\_emp\_id;

IF v\_job\_id = 'SA\_REP' THEN

UPDATE employees

SET commission\_pct = commission\_pct \* 1.2;

ELSE

**NULL; -- Employee is not a sales rep**

END IF;

END;

/