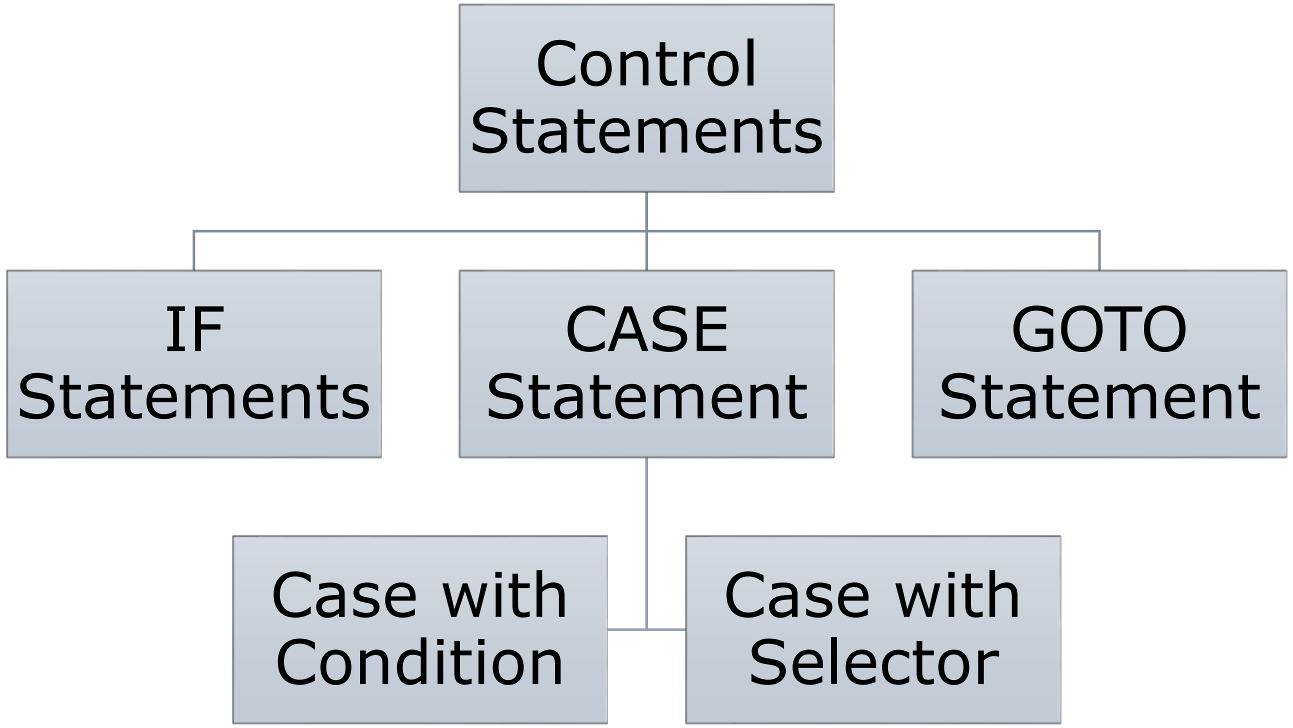
Datenbank und Informationssysteme

DI. Hilbe Klaus, MBA

Execution Control Statements



Conditions
IF Statement

Syntax:

IF *Condition* **THEN**
        Statements;
[**ELSIF** *Condition* **THEN**
        Statements;]
[**ELSE**
        Statements;]
END IF;

- There can be many ELSIF – sections
- The optional ELSE – section is the default if no prior condition is matching

Conditions
Example – IF Statement

```
…

IF v_color = 'Black' THEN
   dbms_output.Put_line('Color is black');
ELSIF v_color = 'White' THEN
   dbms_output.Put_line('Color is white');
ELSE
   dbms_output.Put_line('Color is not black nor
       white');
END IF;
…
```

Conditions
Example – Nested IF Statement

```
-- nested if statement
Set serveroutput ON;
DECLARE
    v_temperature      VARCHAR(30)  := 'High';
    v_rainprobability VARCHAR(30)  := 'Low';
BEGIN
    IF v_temperature = 'High' THEN
      IF v_rainprobability = 'Low' THEN
        dbms_output.Put_line('Make a Trip');
      ELSE
        dbms_output.Put_line('Learn PL/SQL');
      END IF;
    END IF;
END;
```

Conditions
CASE Statement with Selector

Syntax:

CASE *Selector*

  WHEN *Expression1*[*] **THEN** *Statements;*

  WHEN *Expression2* **THEN** *Statements*;

  …

  [**ELSE** *Statements;*]

END CASE;

ATTENTION: in PLSQL at least one expression must be true, if not you need the ELSE path – otherwise you get an error!

[*] Expression can be a literal (eg. 30, 'Sunday') or an expression like v_temperature*30

Conditions
Example – CASE Statement with Selector

```
DECLARE
    v_color VARCHAR(30) := 'Black';
BEGIN
    CASE v_color
      WHEN 'Black' THEN
        dbms_output.Put_line('Color is Black');
      WHEN 'White' THEN
        dbms_output.Put_line('Color is White');
      ELSE
        dbms_output.Put_line('Color is not
           Black nor White');
    END CASE;
END;
```

Conditions
CASE Statement with Condition

Syntax:

CASE
    WHEN *Condition1* **THEN** *Statements;*
    WHEN *Condition2* **THEN** *Statements*;
    …
    [**ELSE** *Statements;*]
END CASE;

ATTENTION: in PLSQL at least one expression must be true, if not you need the ELSE path – otherwise you get an error!

Conditions
Example – CASE Statement with Condition

```
DECLARE
    v_temperature INTEGER := -1;
BEGIN
    CASE
      WHEN v_temperature < 5 THEN
        dbms_output.Put_line('Freezing');
      WHEN v_temperature < 20 THEN
        dbms_output.Put_line('Cold');
      ELSE
        dbms_output.Put_line('Warm');
    END CASE;
END;
```

Tasks Control Statement

- Erstellen Sie eine Prozedur, welche das Einkommen eines übergebenen Mitarbeiters wie folgt erhöht:
  - ist die Firmenzugehörigkeit mehr als 35 Jahre -> 25%
  - ist die Firmenzugehörigkeit mehr als 26 Jahre -> 12%
  - in jedem anderen Fall: -> 9%
- Erstellen Sie eine Prozedur, welche den Provisionsprozentsatz für einen übergebenen Mitarbeiter wie folgt aktualisiert:
  - wenn das Gehalt mehr als 9000 ist -> 0.45
  - wenn das Gehalt < 9000, aber die Firmenzugehörigkeit > 7 Jahre ist -> 0.32
  - wenn das Gehalt <= 2500 -> 0.24
  - in jedem anderen Fall -> 0.09

Loops - Overview

- There are three possibilities to loop
  - FOR
  - WHILE
  - LOOP

FOR – Loop

Syntax:

FOR *Counter* **IN [REVERSE]** *Initial_value* **..** *Final_value*
LOOP
	Statement1;
	…
	StatementN;
END LOOP;

Example
FOR – Loop

```
-- for Loop from 1 to 10
Set serveroutput ON;
-- counter declaration not necessary
-- is declared IMPLICIT!!
-- DECLARE    v_counter INTEGER;
BEGIN
    FOR v_counter IN 1 .. 10
    LOOP
        dbms_output.Put_line('Value of counter is: '
           || v_counter);
    END LOOP;
END;
```

WHILE – Loop

Syntax:

WHILE *Condition*

LOOP

Statement1;

…

StatementN;

END LOOP;

$\left.\right\}$ Loop section is the same as for the „For“ loop

Example
WHILE– Loop

```
-- while loop from 1 to 10
Set serveroutput ON;
DECLARE
    v_counter INTEGER := 1;
BEGIN
    WHILE v_counter <= 10
    LOOP
        dbms_output.Put_line('Value of counter is: '
            || v_counter);
        v_counter := v_counter + 1;
    END LOOP;
END;
```

Loop

Syntax:

```
LOOP
        Statement1;
        …
        StatementN;
END LOOP;*
```

$\left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\}$ Loop section is the same as for the „For“ loop

* In the Loop a EXIT – statement is needed, otherwise the loop will iterate endlessly

Loop Control Statement

- EXIT statement: ends immediately the loop and starts to execute the code after the 'END LOOP' statement

Syntax:

EXIT;

Or

EXIT WHEN *Condition*;

- CONTINUE statement: starts immediately the next iteration at the beginning of the loop (tests the entry condition of the loop)

Syntax:

CONTINUE;

Nesting and Labelling Loops

- Loops can be nested
- Loops can be labelled with '<<' loopname '>>' and later called by this label

Syntax:

<< *LoopName* **>>**

Loop… | **For** … | **While** …

LoopSection

END LOOP [*LoopName*];

Example
Nesting and Labelling Loops

```
-- nested for loop from 1 to 100
Set serveroutput ON;
BEGIN
    << outer_loop >>
    FOR v_firstdigit IN 0 .. 9 LOOP
        << inner_loop >>
        FOR v_seconddigit IN 1 .. 10 LOOP
            dbms_output.Put_line('Value is: ' ||
               To_char(v_firstdigit * 10 +
                   v_seconddigit));
        END LOOP inner_loop;
    END LOOP outer_loop;
END;
```

Example
Loop with CONTINUE Statement

```
-- for loop from 1 to 10
-- display only odd numbers
Set serveroutput ON;
BEGIN
    FOR v_counter IN 1 .. 10 LOOP
        IF MOD(v_counter, 2) = 0 THEN
          CONTINUE;
        END IF;
        dbms_output.Put_line('Value is: '
          || v_counter);
    END LOOP;
END;
```

Example
Exiting a nested Loop

```
-- nested for loop from 1 to 50
-- as outer loop exits at counter = 5
Set serveroutput ON;
BEGIN
    << outer_loop >>
    FOR v_firstdigit IN 0 .. 9 LOOP
        << inner_loop >>
        FOR v_seconddigit IN 1 .. 10 LOOP
          IF v_firstdigit = 5 THEN
            EXIT outer_loop;
          END IF;
          dbms_output.Put_line('Value of counter is: '
          || To_char(v_firstdigit * 10 + v_seconddigit));
        END LOOP inner_loop;
    END LOOP outer_loop;
END;
```

GOTO Statement

Example:

```
-- goto example
Set serveroutput ON;
DECLARE
    v_text VARCHAR2(30) := 'Initial Text';
BEGIN
    GOTO mygotolabel;
    v_text := 'Not Executed due GOTO Statement';
    << mygotolabel>>
    dbms_output.Put_line(v_text);
END;
```

Don't Use It

- Code execution starts immediately at the labelled section mentioned in the GOTO statement
- Because of maintainability reason it is not recommended to use GOTO statements (control flow is hard to track)

Guidelines for Using Loops

- Use FOR Loop if the number of iteration is known upfront
- Use the WHILE if the condition has to be evaluated before entering the loop section
- Try to avoid basic loop and use instead while or for loops
- Try to avoid continue or exit statements

Tasks Schleife

- Erstellen Sie eine Prozedur, welche das Jahr, in dem die meisten Mitarbeiter eingestellt worden sind ermittelt (sie können 1997 hart kodieren). Geben Sie - je Monat - die Anzahl der aufgenommenen Mitarbeiter dieses Jahres aus
- Geben Sie in einer Prozedur den Nachnamen des Vorgesetzten, den Vorvorgesetzten und den Vorvorvorgesetzen des Mitarbeiters 104 in einer Schleife (fix 3 Durchläufe) aus und geben Sie den Nachnamen des jeweiligen Vorgesetzten aus.
- Erstellen Sie eine Funktion, die N-Faktorielle berechnet.
  Bsp:
  3! = 1*2*3 = 6
  5! = 1*2*3*4*5 = 120