SENTENCIAS DE CONTROL.

- PL/SQL dispone de sentencias que permiten construir las tres estructuras de control de la programación estructurada, es decir, secuencial, alternativa y repetitiva, teniendo en cuenta que para las dos últimas estructuras están permitidas las anidaciones, es decir, dentro de in if podremos encontrar otro, o dentro de un while podremos anidar por ejemplo otro while.

SENTENCIAS SECUENCIALES

- Se consideran una secuencia , al conjunto de instrucciones que se ejecutan una detrás de otra.
- Cada sentencia en PL/SQL finaliza con ;

SENTENCIAS ALTERNATIVAS

- Son las sentencias que nos permiten cambiar el flujo del programa dependiendo de que se cumpla o no una condición.
- Oracle cuenta con las siguientes sentencias alternativas

IF-THEN

- Como se aprecia en el primer gráfico que se presenta a continuación, , cuando se cumple la condición, es decir cuando devuelve True, se ejecutan una serie de sentencias (<action_block>) que no se ejecutarán si no se cumple la condición, pero en cualquiera de los dos casos, continúa la ejecución del programa después del END IF;
- La condición por tanto, siempre debe ser evaluada como true o false
- La sintaxis de un If será.

```
IF <condition: returns Boolean>
THEN
 -executed only if the condition returns TRUE
 <action_block>
END if;
```

- Cualquier condición evaluada como 'NULL', será tratada como ‘FALSE'.

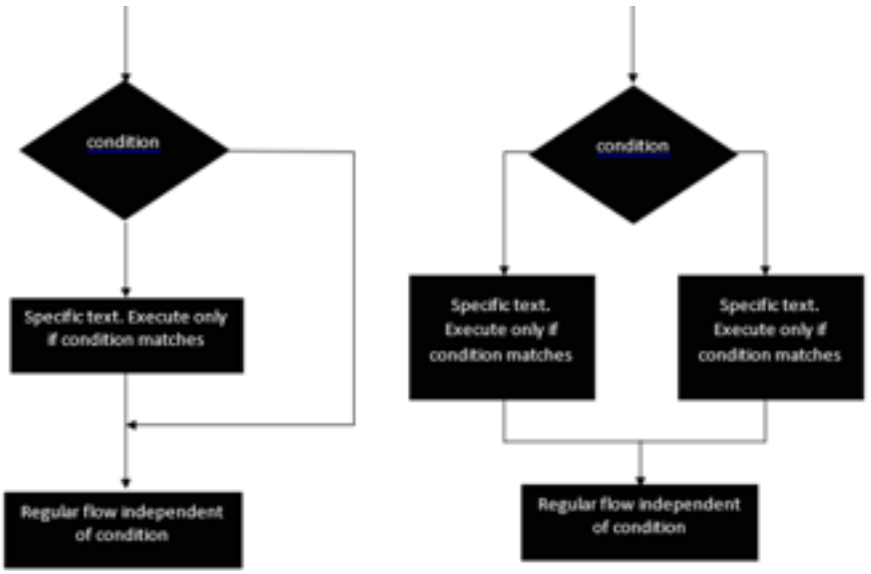
Ejemplo :

```
DECLARE
a CHAR(1) :=’u’;
BEGIN
IF UPPER(a) in ('A’,'E','I','0','U' ) THEN
dbms_output.put_line(‘El carácter es una vocal');
END IF;
END;
/
```

- Como vemos la función que coge el carácter y lo convierte a a mayusculas es UPPER(), exactamente igual que en SQL
- In , permite comprobar si el valor está entre los citados entre los ( )

IF-THEN-ELSE

- Como se aprecia en el segundo gráfico, si se cumple la condición se ejecuta un conjunto de sentencias y si no se cumple se ejecutan otra sentencia o conjunto de ellas, y después en cualquiera de los dos casos continúa con la ejecución del resto del programa



Decision Making Statement Diagram

```
IF <condition: returns Boolean>
THEN
        -executed only if the condition returns TRUE
        <action_blockl>
ELSE
        -execute if the condition failed (returns FALSE)
        <action_block2>
END if;
```

- Se ejecuta <action_blockl>cuando la condición devuelve true
- Se ejecuta <action_block2> cuando la condición devuelve False

IF-THEN-ELSIF

```
 IF <conditionl: returns Boolean>
THEN
-executed only if the condition returns TRUE <
action_blockl>
ELSIF <condition2 returns Boolean> <
action_block2>
ELSIF <condition3:returns Boolean> <
action_block3>
ELSE —optional
<action_block_else>
END if;
```

- Esta sentencia alternativa se utiliza cuando hay que seleccionar una alternativa entre un conjunto de ellas.
- La primera condición que devuelva verdadero, será la que se ejecute , y el resto no se ejecutará.
- En caso de que no se cumpla ninguna, se ejecutan las sentencias del bloque ELSE si es que existiese.

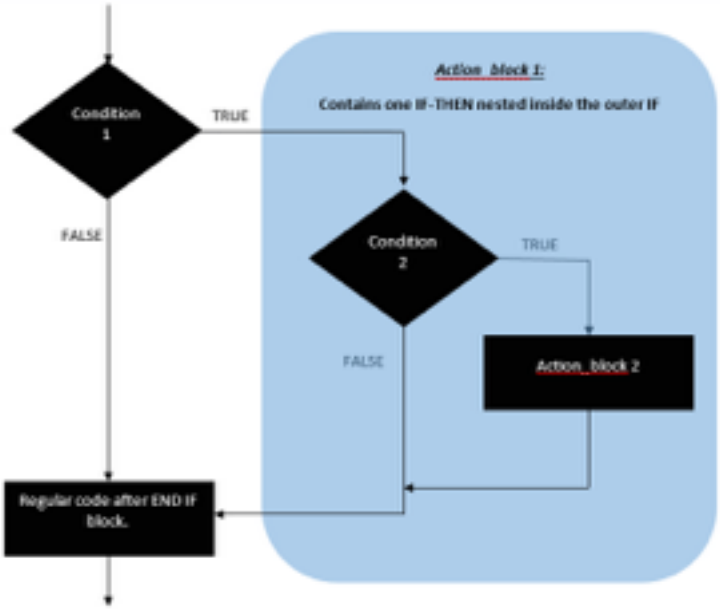
Ejemplo:

```
DECLARE
mark NUMBER :=55;
BEGIN
dbms_output.put_line('Program started.' );
IF( mark >= 70) THEN
dbms_output.put_line('Grade A');
ELSIF(mark >= 40 AND mark < 70) THEN
dbms_output.put_line('Grade B');
ELSIF(mark >=35 AND mark < 40) THEN
dbms_output.put_line('Grade C');
END IF;
dbms_output.put_line('Program completed.');
END;
/
```

Como vemos en este caso, la primera condición no la cumple IF (mark >= 70), por tanto , pasa a evaluar la siguiente . ELSIF(mark >= 40 AND mark < 70) esta si la cumple , por tanto ejecuta dbms_output.put_line('Grade B'); y dbms_output.put_line('Program completed.'); y finaliza el programa.

Bucles anidados

- En PL se pueden incluir unos if dentro de otro, anidando así los if que deseemos



IF <conditionl: returns Boolean>
THEN
—executed only if the condition returns TRUE
<action block1 starts>
IF <condition2: returns Boolean>
THEN
<action_block2>
END IF; —END IF corresponds to condition2
<action_blockl ends>
END IF; —END IF corresponds to condition1

CASE

Es simillar a a IF , pero selecciona un bloque de sentencias en función de la expresión, que ahora no tiene por qué ser un valor Booleano, puede ser un entero, cadena, etc.

El else se ejecuta cuando ninguna de las alternativas es seleccionada.

```
CASE (expression)
 WHEN <valuel> THEN action_blockl;
 WHEN <value2> THEN action_block2;
 WHEN <value3> THEN action_block3;
 ELSE action_block_default;
END CASE;
```

ejemplo:

```
DECLARE
a NUMBER :=55;
b NUMBER :=5;
arth_operation VARCHAR2(20) :='MULTIPLY’;
BEGIN
dbms_output.put_line(‘Program started.' );
CASE (arth_operation)
WHEN ‘ADD’ THEN dbms_output.put_line(‘Addition of the numbers are: '|| a+b );
WHEN ‘SUBTRACT' THEN dbms_output.put_line(‘Subtraction of the numbers are: '||a-b );
WHEN ‘MULTIPLY' THEN dbms_output.put_line(‘Multiplication of the numbers are: '|| a*b
);
WHEN ‘DIVIDE' THEN dbms_output.put_line(‘Division of the numbers are:'|| a/b);
ELSE dbms_output.put_line(‘No operation action defined. Invalid operation');
END CASE;
dbms_output.put_line(‘Program completed.' );
END;
/
```

SEARCHED CASE

Es un caso especial del CASE, pero no ponemos expresión en el CASE y las vamos poniendo en el WHEN. Cuando se cumpla una expresión se ejecuta el código asociado y finaliza el CASE.

```
CASE
WHEN <expression1> THEN action_blockl;
WHEN <expression2> THEN action_block2;
WHEN <expression3> THEN action_block3;
ELSE action_block_default;
END CASE;
```

Ejemplo:

```
DECLARE a NUMBER :=55;
b NUMBER :=5;
arth_operation VARCHAR2(20) :='DIVIDE';
BEGIN
dbms_output.put_line(‘Program started.' );
CASE
WHEN arth_operation = 'ADD'
THEN dbms_output.put_line(‘Addition of the numbers are: '||a+b );
WHEN arth_operation = ‘SUBTRACT'
THEN dbms_output.put_line(‘Subtraction of the numbers are: '|| a-b);
WHEN arth_operation = ‘MULTIPLY’
THEN dbms_output.put_line(‘Multiplication of the numbers are: '|| a*b );
WHEN arth_operation = ’DIVIDE'
THEN dbms_output.put_line(‘Division of the numbers are: '|| a/b ):
ELSE dbms_output.put_line(‘No operation action defined. Invalid operation');
END CASE;
dbms_output.put_line(‘Program completed.' );
END;
/
```