ESTRUCTURAS DE CONTROL

SENTENCIAS IF

PL/SQL dispone de estructuras para controlar el flujo de ejecución de los programas.

ESTRUCTURAS ALTERNATIVAS:

Alternativa simple

`IF <condicion> THEN instrucciones;END IF;`	Si la condición se cumple, se ejecutan las instrucciones que siguen a la cláusula THEN.

Alternativa doble

`IF <condicion> THEN instrucciones1;ELSE instrucciones2;END IF;`	Si la condición se cumple, se ejecutan las instrucciones que siguen a la cláusula THEN. En caso contrario, se ejecutarán las instrucciones que siguen a la cláusula ELSE.

Alternativa múltiple con ELSIF

`IF <condicion1> THEN instrucciones1; ELSIF <condicion2> THEN instrucciones2; ELSIF <condicion3> THEN instrucciones3; ... [ELSE instruccionesotras;] END IF;`	Evalúa, comenzando desde el principio, cada condición, hasta encontrar alguna condición que se cumpla, en cuyo caso ejecutará las instrucciones que siguen a la cláusula THEN correspondiente. La cláusula ELSE es opcional; en caso de que se utilice, se ejecutará cuando no se ha cumplido ninguna de las condiciones anteriores.

La mayoría de las estructuras de control requieren evaluar una condición que en PL/SQL puede dar tres resultados: **TRUE**, **FALSE** o **NULL.** Pues bien, a efectos de estas estructuras, el valor NULL es equivalente a FALSE, es decir, se considerará que se cumple la condición sólo si el resultado es TRUE. En caso contrario (FALSE o NULL), se considerará que no se cumple.

SENTENCIA CASE

La estructura CASE no está disponible en versiones anteriores a la 91. En estos casos deberemos utilizar la alternativa múltiple ELSIF.

Alternativa múltiple con CASE de comprobación

```
CASE expresión
WHEN <valorcomprobac1> THEN
  instrucciones1;
WHEN <valorcomprobac2> THEN
  instrucciones2;
WHEN <valorcomprobac3> THEN
  instrucciones3;
...
[ELSE
  instruccionesotras;]
  END CASE;
```

Calcula el resultado de la expresión que sigue a la cláusula CASE. A continuación, comprueba si el valor obtenido coincide con alguno de los valores especificados detrás de las cláusulas WHEN, en cuyo caso ejecutará la instrucción o instrucciones correspondientes. La cláusula ELSE es opcional; se ejecutará en caso de que no se encuentre un valor coincidente en las cláusulas WHEN precedentes.

Alternativa múltiple con CASE de búsqueda

```
CASE
WHEN <condicion1> THEN
  instrucciones1;
WHEN <condicion2> THEN
  instrucciones2;
WHEN <condicion3> THEN
  instrucciones3;
...
[ELSE
     instruccionesotras;]
  END CASE;
```

Evalúa, comenzando desde el principio, cada condición, hasta encontrar alguna condición que se cumpla, en cuyo caso ejecutará las instrucciones que siguen a la cláusula THEN correspondiente. La cláusula ELSE es opcional; en caso de que se utilice, se ejecuta cuando no se ha cumplido ninguna de las condiciones anteriores.

Veamos un ejemplo:

Caso práctico

1 Supongamos que pretendemos modificar el salario de un empleado especificado en función del número de empleados que tiene a su cargo:

- Si no tiene ningún empleado a su cargo la subida será 50 €.
- Si tiene 1 empleado la subida será 80 €.
- Si tiene 2 empleados la subida será 100 €.
- Si tiene más de tres empleados la subida será 110 €.

Además, si el empleado es PRESIDENTE se incrementará el salario en 30 €.

```
DECLARE
    v_empleado_no NUMBER(4,0);        -- emple al que subir salario
    v_c_empleados NUMBER(2);          -- cantidad  empl dependen de él
    v_aumento NUMBER(7) DEFAULT 0;    -- importe que vamos a aumentar.
    v_oficio VARCHAR2(10);
```

(Continúa)

(Continuación)

```
BEGIN
    v_empleado_no := &vt_empno;     -- var de sustitución lee nºemple

    SELECT oficio INTO v_oficio FROM emple
        WHERE emp_no = v_empleado_no;

    IF v_oficio = 'PRESIDENTE' THEN -- alternativa simple
        v_aumento := 30;
    END IF;

    SELECT COUNT(*) into v_c_empleados FROM emple
         WHERE dir = v_empleado_no;

    IF v_c_empleados = 0 THEN       -- alternativa múltiple
        v_aumento := v_aumento + 50;
    ELSIF v_c_empleados = 1  THEN
        v_aumento :=  v_aumento + 80;
    ELSIF v_c_empleados = 2 THEN
        v_aumento :=  v_aumento + 100;
    ELSE
        v_aumento :=  v_aumento + 110;
    END IF;

    UPDATE emple SET salario = salario + v_aumento WHERE emp_no = v_empleado_no;
    DBMS_OUTPUT.PUT_LINE(v_aumento);
END;
/
```

El resultado de la ejecución será:

```
Introduzca valor para vt_empno: 7839
140
```

En el programa anterior hemos utilizado una estructura ELSIF pero podíamos haber utilizado una estructura CASE en cualquiera de sus dos formatos:

CON CASE DE BÚSQUEDA

```
CASE
WHEN v_c_empleados = 0 THEN
    v_aumento := v_aumento + 50;
WHEN v_c_empleados = 1  THEN
    v_aumento :=  v_aumento + 80;
WHEN v_c_empleados = 2 THEN
   v_aumento :=  v_aumento + 100;
ELSE
   v_aumento :=  v_aumento + 110;
END CASE;
```

CON CASE DE COMPROBACIÓN

```
CASE v_c_empleados
WHEN 0 THEN
    v_aumento := v_aumento + 50;
WHEN 1  THEN
    v_aumento :=  v_aumento + 80;
WHEN 2 THEN
    v_aumento :=  v_aumento + 100;
ELSE
    v_aumento :=  v_aumento + 110;
END CASE;
```

SENTENCIAS REPETITIVAS: FOR . LOOP Y WHILE

En ocasiones, se puede usar NULL como una instrucción que no hace nada por motivos de claridad o facilidad de codificación:

```
CASE val
WHEN 1 THEN
   INSERT INTO TEMP VALUES 'UNO';
WHEN 2 THEN
   INSERT INTO TEMP VALUES 'DOS';
WHEN 0 THEN
   NULL;                          -- No hace nada.
ELSE THEN
  INSERT INTO TEMP VALUES 'ERROR'
END CASE;
```

ESTRUCTURAS REPETITIVAS	
Iterar	

`LOOP`
`  Instrucciones;`
`EXIT [WHEN <condición>];`
`  instrucciones;`
`END LOOP;`
	Se trata de un bucle que se repetirá indefinidamente hasta que encuentre una instrucción EXIT sin condición o hasta que se cumpla la condición asociada a la cláusula EXIT WHEN. Es una **condición de salida.**

Mientras	

`WHILE <condicion> LOOP`
`  instrucciones;`
`END LOOP;`
	Se evalúa la condición y, si se cumple, se ejecutarán las instrucciones del bucle. El bucle se seguirá ejecutando mientras se cumpla la condición. Es una **condición de continuación.**

En un **bucle WHILE**, si la condición no se cumple al comienzo, no se ejecutará ni una sola línea pues la comprobación se hace antes de entrar en el bucle y posteriormente, una vez finalizadas todas las instrucciones que incluye. **LOOP**, sin embargo, siempre entrará en el bucle, ejecutará las primeras instrucciones y saldrá del bucle cuando se cumpla la condición.

Caso práctico

2 Supongamos que deseamos analizar una cadena que contiene los dos apellidos para guardar el primer apellido en una variable a la que llamaremos v_1apel. Entendemos que el primer apellido termina cuando encontramos cualquier carácter distinto de los alfabéticos (en mayúsculas).

```
DECLARE
    v_apellidos VARCHAR2(25);
    v_1apel VARCHAR2(25);
    v_caracter CHAR;
    v_posicion INTEGER :=1;
```

(Continúa)

(Continuación)

```
BEGIN
    v_apellidos := '&vs_apellidos';

    v_caracter := SUBSTR(v_apellidos,v_posicion,1);
    WHILE  v_caracter BETWEEN 'A' AND 'Z' LOOP
        v_1apel := v_1apel || v_caracter;
        v_posicion := v_posicion + 1;
        v_caracter := SUBSTR(v_apellidos,v_posicion,1);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('1er Apellido:'||v_1apel||'*');
END;
/
```

El resultado de la ejecución será:

```
Introduzca valor para vs_apellidos: GIL MORENO
1er Apellido:GIL*
Procedimiento PL/SQL terminado con éxito.
```

El mismo ejemplo con un bucle LOOP... END LOOP será:

```
DECLARE
    v_apellidos VARCHAR2(25);
    v_1apel VARCHAR2(25);
    v_caracter CHAR;
    v_posicion INTEGER :=1;
BEGIN
    v_apellidos := '&vs_apellidos';

    -- desaparece la asignación de v_caracter antes del bucle
    -- se asignará dentro al comienzo del bucle.
    LOOP
        v_caracter := SUBSTR(v_apellidos,v_posicion,1);
      EXIT WHEN v_caracter NOT BETWEEN 'A' AND 'Z';
        v_1apel := v_1apel || v_caracter;
        v_posicion := v_posicion + 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('1er Apellido:'||v_1apel||'*');
END;
/
```

EXIT es una instrucción en sí misma (por eso lleva punto y coma al final) y puede ser utilizada con o sin la cláusula **WHEN**.

```
LOOP
     instrucciones;
     IF <condición> THEN
```

```
            EXIT;
        END IF;
        instrucciones;
    END LOOP;
```

PL/SQL permite la utilización de EXIT incluso en otros bucles y estructuras. Esta práctica es totalmente desaconsejable.

Para

```
FOR <variablecontrol> IN <valorInicio>..<valorFinal> LOOP
    instrucciones;
END LOOP;
```

Donde <variablecontrol> es la variable de control del bucle que se declara de manera implícita como variable local al bucle de tipo BINARY_INTEGER. Esta variable tomará, en primer lugar, el valor especificado en la expresión numérica <valorInicio>, incrementándose en uno para cada nueva iteración hasta alcanzar el valor especificado en la expresión numérica <valorFinal>.

Se utiliza esta estructura cuando se conoce o se puede conocer a priori el número de veces que se debe ejecutar un bucle.

El incremento siempre es una unidad, pero puede ser negativo utilizando la **opción REVERSE:**

```
FOR <variable> IN REVERSE <valorFinal>..<valorInicio>
  LOOP
    instrucciones;
    ...;
END LOOP;
```

En este caso, comenzará por el valor especificado en segundo lugar e irá restando una unidad en cada iteración:

```
SQL> BEGIN
  2    FOR i IN REVERSE 1..3 LOOP
  3        DBMS_OUTPUT.PUT_LINE(i);
  4    END LOOP;
  5  END;
  6  /

3
2
1
Procedimiento PL/SQL terminado con éxito.
```

Cuando empleamos la opción REVERSE, comenzará por el segundo valor hasta tomar el que sea igual o menor que el especificado en primer lugar. El bloque que se muestra a continuación tiene un error de diseño, ya que no llega siquiera a entrar en el bucle:

```
SQL> BEGIN
  2    FOR i IN REVERSE 5..1 LOOP   -- ERROR
  3        DBMS_OUTPUT.PUT_LINE(i);
  4    END LOOP;
  5  END;
  6  /

Procedimiento PL/SQL terminado con éxito.
```

Podemos indicar los valores mínimo y máximo mediante expresiones:

```
DECLARE
  Num1 INTEGER;
  Num2 INTEGER;
  ...
BEGIN
  ...
  FOR x IN Num1..Num2 LOOP
    ...
  END LOOP;
  ...
END;
```

PL/SQL no dispone de la **opción STEP** que tienen otros lenguajes, la cual permite especificar incrementos distintos de uno.

Respecto a la variable de control, hay que tener en cuenta que:

- No hay que declararla.
- Es local al bucle y no es accesible desde el exterior del bucle, ni siquiera en el mismo bloque.
- Se puede usar dentro del bucle en una expresión, pero no se le pueden asignar valores.

En el siguiente ejemplo, definimos una variable e intentamos usarla como variable de control. Aun en ese caso la estructura creará la suya propia como local, quedando la nuestra como global en el bucle:

```
<<ppal>>
DECLARE
    i INTEGER;
BEGIN
    ...
    FOR i IN 1..10 LOOP
        ...
/* cualquier referencia a i será entendida como a la
   variable local al bucle. Si quisiéramos referirnos  a
   la otra lo debemos hacer como ppal.i */
    END LOOP;
    ...
    /*La variable local del bucle ya no existe aquí */
END ppal;
```

Veamos algunos ejemplos de aplicación:

Caso práctico

3 Vamos a construir de dos maneras un bloque PL/SQL que escriba la cadena 'HOLA' al revés.

Utilizando un bucle FOR

```
SQL> DECLARE
  2    r_cadena VARCHAR2(10);
  3  BEGIN
  4    FOR i IN REVERSE 1..LENGTH('HOLA') LOOP
  5      r_cadena := r_cadena||SUBSTR('HOLA',i,1);
  6    END LOOP;
  7    DBMS_OUTPUT.PUT_LINE(r_cadena);
  8* END;
SQL> /
ALOH
Procedimiento PL/SQL terminado con éxito.
```

Utilizando un bucle WHILE

```
SQL> DECLARE
2    r_cadena VARCHAR2(10);
3    i BINARY_INTEGER;
4  BEGIN
5    i := LENGTH('HOLA');
6    WHILE i >= 1 LOOP
7      r_cadena=r_cadena||SUBSTR('HOLA',i,1);
8      i := i - 1;
9    END LOOP;
10    DBMS_OUTPUT.PUT_LINE(r_cadena);
11* END;
SQL> /
ALOH
Procedimiento PL/SQL terminado con éxito.
```

Actividades propuestas

2 Escribe un bloque PL/SQL que realice la misma función del ejemplo anterior pero usando un bucle ITERAR.

A. Utilización de etiquetas

Las **etiquetas** sirven para marcar o nombrar determinadas partes del código del programa como bloques, estructuras de control y otras.

Podemos situar etiquetas en nuestros programas utilizando el formato:

```
<<nombreetiqueta>>
```

Donde los delimitadores << >> forman parte de la sintaxis y `nombretiqueta` representa un identificador válido PL/SQL que utilizaremos después (en este caso, sin los delimitadores) para hacer referencia al elemento.

Podemos etiquetar bucles y otras estructuras para conseguir mayor legibilidad:

```
<<mibucle>>
LOOP
    instrucciones;
    ...
END LOOP mibucle;
```

También se pueden etiquetar las estructuras para eliminar ambigüedades, hacer visibles variables globales y conseguir otras funcionalidades:

```
<<bucleexterno>>
LOOP
    ...
    LOOP
        ...
        EXIT bucleexterno WHEN ... -- sale de ambos bucles
    END LOOP;
END LOOP bucleexterno;
```

En PL/SQL se puede usar la instrucción **GOTO etiqueta.** Para poder utilizar esta orden se deben cumplir las siguientes condiciones:

- No puede haber otra etiqueta en el entorno actual con el mismo nombre.
- La etiqueta debe preceder a un bloque o a un conjunto de órdenes ejecutables.
- La etiqueta no puede estar dentro de un IF, de un LOOP ni de un SUB-BLOQUE internos al bloque donde se produce la llamada.
- Desde una excepción no se puede pasar el control del programa a una etiqueta que está en otra sección del mismo bloque.

Ejemplos de usos correctos

```
BEGIN
   ...
   GOTO insertar_fila;
   ...
   <<insertar_fila>>
   INSERT INTO empleados VALUES ...
END;

También es posible:

BEGIN
   ...
   <<insertar_fila>>
   BEGIN
      INSERT INTO empleados VALUES ...
      ...
   END;
   ...
   GOTO insertar_fila;
   ...
END;
```

Ejemplos de usos ilegales

```
BEGIN
  ...
  FOR i IN 1..10 LOOP
      ...
      GOTO fin_loop;
      ...
      <<fin_loop>> --ilegal, no hay instrucciones ejecutables
   END LOOP;
   ...
   GOTO insertar_fila;
   IF ... THEN
      ...
      <<insertar_fila>>
      INSERT INTO empleados VALUES ...--ilegal, está en un if
      ...
   END IF;
   ...
   GOTO otro_sub;    -- ilegal, está en otro sub-bloque
   ...
   BEGIN
  ...
       <<otro_sub>>
      DELETE FROM ...
   END;
   ...
   <<mi_etiqueta>>
   ...
EXCEPTION
   WHEN ... THEN
      GOTO mi_etiqueta;   --ilegal está en el bloque actual.
END;
...
```