Estructuras de control

Como cualquier lenguaje procedural, PL/SQL dispone de una serie de órdenes que nos permiten establecer controles y condiciones a los programas, pudiéndose por tanto representarlos mediante operaciones secuenciales, selectivas y repetitivas. Las dos últimas se apoyan en condiciones para establecer su comportamiento.

Sentencia IF

A menudo es necesario tomar alternativas de acción dependiendo de las circunstancias. La sentencia IF permite ejecutar una secuencia de acciones condicionalmente. Esto es, si la secuencia es ejecutada o no depende del valor de la condición a evaluar. Existen tres modos para esta instrucción: IF – THEN, IF – THEN – ELSE y IF – THEN – ELSIF.

IF – THEN

Este es el modo más simple y consiste en asociar una condición con una secuencia de sentencias encerradas entre las palabras reservadas THEN y END IF (no ENDIF).

```
IF condición THEN
        secuencia_de_sentencias
END IF;
```

La secuencia de sentencias es ejecutada sólo si la condición es verdadera. Si la condición es falsa o nula no realiza nada. Un ejemplo real de su utilización es la siguiente:

```
IF condición THEN
        calcular_bonus (emp_id);
        UPDATE sueldos SET pago = pago + bonus WHERE emp_no = emp_id;
END IF;
```

IF – THEN – ELSE

Esta segunda modalidad de la sentencia IF adiciona una nueva palabra clave: ELSE, seguida por una secuencia alternativa de acciones:

```
IF condición THEN
        secuencia_de_sentencias_1
ELSE
        secuencia_de_sentencias_2
END IF;
```

La secuencia de sentencias en la cláusula ELSE es ejecutada solamente si la condición es falsa o nula. Esto implica que la presencia de la cláusula ELSE asegura la ejecución de alguna de las dos secuencias de estamentos. En el ejemplo siguiente el primer UPDATE es ejecutado cuando la condición es verdadera, en el caso que sea falsa o nula se ejecutará el segundo UPDATE:

```
IF tipo_trans = 'CR' THEN
        UPDATE cuentas SET balance = balance + credito WHERE ...
ELSE
        UPDATE cuentas SET balance = balance – debito WHERE ...
END IF;
```

Las cláusulas THEN y ELSE pueden incluir estamentos IF, tal como lo indica el siguiente ejemplo:

```
IF tipo_trans = 'CR' THEN
        UPDATE cuentas SET balance = balance + credito WHERE ...
ELSE
        IF  nuevo_balance >= minimo_balance THEN
                UPDATE cuentas SET balance = balance – debito WHERE ...
        ELSE
                UPDATE cuentas SET balance = balance – minimo_balance WHERE ...
        END IF;
END IF;
```

IF – THEN – ELSIF

Algunas veces se requiere seleccionar una acción de una serie de alternativas mutuamente exclusivas. El tercer modo de la sentencia IF utiliza la clave ELSIF (no ELSEIF) para introducir condiciones adicionales, como se observa en el ejemplo siguiente:

```
IF condición_1 THEN
        secuencia_de_sentencias_1
ELSIF condición_2 THEN
        secuencia_de_sentencias_2
ELSE
        secuencia_de_sentencias_3
END IF;
```

Si la primera condición es falsa o nula, la cláusula ELSIF verifica una nueva condición. Cada sentencia IF puede poseer un número indeterminado de cláusulas ELSIF; la palabra clave ELSE que se encuentra al final es opcional.

Las condiciones son evaluadas una a una desde arriba hacia abajo. Si alguna es verdadera, la secuencia de sentencias que corresponda será ejecutada. Si cada una de las condiciones analizadas resultan ser falsas, la secuencia correspondiente al ELSE será ejecutada:

```
BEGIN
        IF  sueldo > 50000 THEN
                bonus : = 1500;
        ELSIF sueldo > 35000 THEN
                bonus : = 500;
        ELSE
                bonus : = 100;
        END IF;

        INSERT INTO sueldos VALUES (emp_id, bonus, );
END;
```

Si el valor de sueldo es mayor que 50.000, la primera y segunda condición son verdaderas, sin embargo a bonus se le asigna 1500, ya que la segunda condición jamás es verificada. En este caso sólo se verifica la primera condición para luego pasar el control a la sentencia INSERT.

```
DECLARE
    v_salary employees.salary%type;
    v_dept employees.department_id%type;
    v_avg_salary_dept employees.salary%type;
    v_diferencia employees.salary%type;
    v_id employees.employee_id%type := 107;
BEGIN
    Select e.salary, e.department_id
        into v_salary, v_dept
    from employees e where employee_id = v_id;

    DBMS_OUTPUT.put_line('Departamento:'||v_dept||' Salario: '||v_salary);

    Select avg(e.salary) into v_avg_salary_dept
    from employees e where e.department_id = v_dept;

    DBMS_OUTPUT.put_line('Media:'||v_avg_salary_dept);
    v_diferencia := v_salary - v_avg_salary_dept;

    if v_diferencia > 0 then
        DBMS_OUTPUT.put_line('El empleado gana por encima de la media');
    elsif v_diferencia < 0 then
        DBMS_OUTPUT.put_line('El empleado gana por debajo de la media');
    Else
        DBMS_OUTPUT.put_line('El empleado gana justo en la media');
    End if;

END;
```

Sentencia CASE

Se utiliza de forma similar al "Switch case de Java" que al final es otra forma de expresar una estructura IF – THEN – ELSIF.

```
CASE [ expression ]

  WHEN condition_1 THEN result_1
  WHEN condition_2 THEN result_2
  ...
  WHEN condition_n THEN result_n

  ELSE result

END
```

```
DECLARE
  n number := 2;
BEGIN
  CASE n
    WHEN 1 THEN dbms_output.put_line('n = 1');
    WHEN 2 THEN
      dbms_output.put_line('n = 2');
      dbms_output.put_line('That implies n > 1');
    WHEN 2+2 THEN
      dbms_output.put_line('n = 4');
    ELSE dbms_output.put_line('n is some other value.');
  END CASE;
END;
```

```
DECLARE
  quantity NUMBER := 100;
  projected NUMBER := 30;
  needed NUMBER := 999;
BEGIN
<<here>>
  CASE
    WHEN quantity is null THEN
      dbms_output.put_line('Quantity not available');
    WHEN quantity + projected >= needed THEN
      dbms_output.put_line('Quantity ' || quantity ||
        ' should be enough if projections are met.');
    WHEN quantity >= 0 THEN
      dbms_output.put_line('Quantity ' || quantity || ' is probably not enough.');
  END CASE here;
  EXCEPTION
    WHEN CASE_NOT_FOUND THEN
      dbms_output.put_line('Somehow quantity is less than 0.');
END;
```

```
DECLARE
    v_salary employees.salary%type;
    v_dept employees.department_id%type;
    v_avg_salary_dept employees.salary%type;
    v_diferencia employees.salary%type;
    v_id employees.employee_id%type := 101;
BEGIN
    Select e.salary, e.department_id
        into v_salary, v_dept
    from employees e where employee_id = v_id;

    DBMS_OUTPUT.put_line('Departamento:'||v_dept||' Salario: '||v_salary);
```

```
    Select avg(e.salary) into v_avg_salary_dept
    from employees e where e.department_id = v_dept;

    DBMS_OUTPUT.put_line('Media:'||v_avg_salary_dept);
    v_diferencia := v_salary - v_avg_salary_dept;

    CASE when v_diferencia < 0 then
            DBMS_OUTPUT.put_line('El empleado gana menos la media');
        when v_diferencia > 0 then
            DBMS_OUTPUT.put_line('El empleado gana mas la media');
        Else
            DBMS_OUTPUT.put_line('El empleado gana distinto a la media');
    End case;

END;
```

Ejercicio

(HR) Hacer un bloque anónimo de PL/SQL que me pida por pantalla dos id de empleados y que me diga si ambos empleados trabajan en el mismo país o no.

Estructuras repetitivas

PL/SQL proporciona prestaciones para estructurar bucles y así repetir varias veces una sentencia o un conjunto de éstas.

Los bucles pueden ser de 3 tipos:

- LOOP BÁSICO.
- WHILE.
- FOR.

Bucle Loop

Proporciona acciones repetitivas sin condiciones globales y requiere la sentencia EXIT para finalizar el bucle. Si se omite dicha cláusula daría lugar a un bucle infinito.

Sintaxis:

```
LOOP
        órdenes;
        EXIT [WHEN condición];
END LOOP;
```

Ejemplo 1:

Insertar en la tabla ITEM, diez registros, se saldrá del bucle cuando se hayan insertado esos registros.

```
...
LOOP
        INSERT INTO ITEM(ORDER, ITEMID) VALUES (V_ORDER, V_ITEMID);
        V_COUNTER:=V_COUNTER+1;
        EXIT WHEN V_COUNTER>10;
END LOOP;
…
```

Con cada iteración del ciclo las sentencias son ejecutadas. Para terminar estos ciclos de ejecución se utiliza la palabra clave EXIT. Es posible ubicar innumerables EXIT dentro del loop, obviamente ninguno fuera de él. Existen dos modalidades para utilizar esta sentencia: EXIT y EXIT – WHEN.

- EXIT: La cláusula EXIT obliga al loop a concluir incondicionalmente. Cuando se encuentra un EXIT en el código, el loop es completado inmediatamente y pasa el control a la próxima sentencia.
- EXIT – WHEN: Esta sentencia permite terminar el loop de manera condicional. Cuando se encuentra un EXIT la condición de la cláusula WHEN es evaluada. Si la condición es verdadera el loop es terminado y el control es pasado a la próxima sentencia.
    - Hasta que la condición no sea verdadera el loop no puede completarse, esto implica que necesariamente dentro de las sentencias el valor de la condición debe ir variando.

La sentencia EXIT – WHEN reemplaza la utilización de un IF. A modo de ejemplo se pueden comparar los siguientes códigos:

```
IF count > 100 THEN          |          EXIT WHEN count > 100;
        EXIT;                |
END IF;                      |
```

Ambos códigos son equivalentes, pero el EXIT – WHEN es más fácil de leer y de entender.

Bucle While

Nos permite asociar una determinada condición a la hora de ejecutar una serie de comandos. Se utiliza WHILE mientras la condición sea cierta.

Sintaxis:

```
WHILE condición LOOP
        órdenes;
END LOOP;
```

Ejemplo 2:

Diseñar un bucle WHILE que sume uno a una variable, mientras que esta sea menor o igual que 100.

```
...
WHILE V_X<=100 LOOP
        V_X:=V_X+1;
END LOOP;
...
```

La condición se evalúa antes de cara iteración. Si es verdadera se ejecuta la secuencia de órdenes y, si es falsa, el bucle termina. Si la condición del bucle no toma un valor 'true', el bucle no llega a ejecutarse.

Bucle For

Ejecuta las órdenes un número predeterminado de veces según la condición que va declarada. Las directrices para la construcción de FOR son:

- Hacer referencia al índice o variable dentro del bucle ya que no se define en DECLARE.
- Utilizar una expresión para hacer referencia al valor actual del índice.
- No hacer referencia al índice o variable como objetivo de una asignación.

Sintaxis:

```
FOR     indice IN [REVERSE]
                limite_inferior . . limite_superior LOOP
        órdenes;
END LOOP;
```

Ejemplo 3:

Insertar las 10 primeras líneas del pedido número 101.

```
...
V_ORDID ITEM.ORDID%TYPE := 101;
BEGIN
…
FOR I IN 1..10 LOOP
        INSERT INTO ITEM(ORDID, ITEMID)
        VALUES (V_ORDID,I);
END LOOP;
 …
```

- Contador_bucle: es el índice del bucle que por defecto es binary_integer. No es necesario declararlo.
- Los límites del bucle, solo se evalúan una vez, y determinan el nº total de iteraciones, en las que el índice varía, entre el límite superior e interior, incrementándose en una unidad.
- [REVERSE] sirve para inicializar al límite superior e ir decrementando en una unidad.