Oracle básico (IV): Programación en *PL/SQL*

El lenguaje de programación de *Oracle*, llamado *PL/SQL*, es un lenguaje portable, procedural y de transacción muy potente y de fácil manejo, con las siguientes características fundamentales:

1. Incluye todos los comandos de *SQL* estudiados en el artículo *Oracle Básico I y II* (ver revista ***Algoritmo*** números 8 y 9, respectivamente):

   - SELECT
   - INSERT
   - UPDATE
   - DELETE.

2. Es una extensión de SQL, ya que este es un lenguaje no completo dado que no incluye las herramientas clásicas de programación. Por eso, PL/SQL amplia sus posibilidades al incorporar las siguientes sentencias:

   - Control condicional

```
IF ... THEN ... ELSE ... ENDIF
```

   - Ciclos

```
FOR ... LOOP
WHILE ... LOOP
```

3. Incorpora opciones avanzadas en:

- Control y tratamiento de errores llamado excepciones.
- Manejo de cursores.
- Variedad de procedimientos y funciones empaquetadas incorporadas en el módulo *SQL*Forms* para la programación de disparadores (*Trigger*) y procedimientos del usuario (*Procedure*).

Estructura del bloque de código

Veamos a continuación la organización del bloque de código de *PL/SQL*, compuesto por cuatro secciones *DECLARE*, *BEGIN*, *EXCEPTION y END* como se detalla en el fuente 1:

```
/* --- Fuente 1 ------------------------------------------------------------

[<< nombre del bloque >>]
    Etiqueta que identifica al Bloque.

[DECLARE]
    Declaración de
        Variable
        Constante       Se inicializa con un valor que no se puede modificar.
        Cursor          Area de trabajo que contiene los datos de la fila de
                        la tabla en uso. El cursor es el resultado de una
                        sentencia SELECT.
```

```
    ExcepciónVariables   para control de errores.
BEGIN
    Código.
[EXCEPTION]
    Control y tratamiento de errores.
    Es el punto al que se transfiere el control del programa siempre que
    exista un problema. Los indicadores de excepción pueden ser definidos por
    el usuario o por el sistema, como es por ejemplo la excepción ZERO_DIVIDE.
    Las excepciones se activan automáticamente al ocurrir un error, existiendo
    la definición de la excepción OTHERS que considera aquellos errores no
    definidos y que siempre se ubica al final de todas las excepciones.

END [nombre del bloque];
    Fin del Bloque.
```

Con el ejemplo del fuente 2 ilustraremos las distintas secciones que componen un bloque de código en *PL/SQL*. En este caso deseamos calcular la venta promedio del día y, en caso que la misma sea menor a lo esperado, se debe registrar en la tabla *VENTABAJA*.

```
/* --- Fuente 2 ------------------------------------------------------------

DECLARE
    esperada          CONSTANT NUMBER(5) := 500;
    xtotal            NUMBER;
    xcant             NUMBER;
    xprom             NUMBER;
BEGIN
    /*Asigna a la variable xtotal el TOTAL de las ventas
    y a la variable xcant la cantidad de ventas del día.
    */
    SELECT SUM(valor),COUNT(valor) INTO xtotal,xcant
        FROM ventas WHERE fecha=sysdate;
    xprom:=xtotal/xcant;
    IF xprom >= esperada THEN
        message('Ventas por encima de la esperada');
        pause;
    ELSE
        /*Se registra en la tabla ventabaja las ventas por debajo
        del promedio esperado */
        INSERT INTO ventabaja VALUES (sysdate,xprom);
    END IF;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        message('No se realizaron ventas en el día');
    pause;
    WHEN OTHERS THEN
        message('Error Indefinido');
    pause;
END;
```

Asignación de valores

Las dos formas que existen para asignar valores a variables de memoria, vistas en el ejemplo anterior, son:

- Con el operador de asignación :=, como cuando calculamos el promedio de las ventas asignándole valor a la variable xprom con la siguiente sentencia:

```
    xprom:=xtotal/xcant;
```

- Con la sentencia SELECT que contiene la orden INTO, como se muestra, es la asignación de valores a las variables xtotal y xcant con el siguiente código:

```
    SELECT SUM(valor),
        COUNT(valor)
        INTO xtotal,xcant
        FROM ventas
        WHERE fecha=sysdate;
```

Veamos a continuación, con la creación del procedimiento *FECHAALTA*, la asignación de valores a una variable de registro llamada *Client_Rec*, que va a contener la estructura de una fila de la tabla *CLIENTES* y que estará formada por todos los campos correspondientes a la tabla. Para esto usaremos el atributo de variable *%ROWTYPE* que declara una variable de registro que contiene la estructura de la tabla, y después, con el uso de la sentencia *SELECT * INTO,* se asigna a la variable de registro los valores de la fila. La referencia a un dato contenido en la variable de registro se hace de la forma *variable_registro.campo*, como por ejemplo *cliente_rec.fecha* hace referencia a la fecha del alta del cliente.

Pasemos a mostrar lo anteriormente expuesto a través del código del fuente 3.

```
/* --- Fuente 3 ------------------------------------------------------------

PROCEDURE FECHAALTA IS
    BEGIN
       DECLARE
           cliente_rec  clientes%ROWTYPE;
       BEGIN
           SELECT * INTO cliente_rec
               FROM clientes
               WHERE codigo = 5;
           IF cliente_rec.fecha>sysdate-10
           THEN
               message( cliente_rec.nombre||
                        ' Dado de alta en los últimos 10 días');
               pause;
           ELSE
               message( cliente_rec.nombre||
                        ' Dado de alta hace más de 10 días');
               pause;
           END IF;
    END;
END;
```

SELECT con control de excepciones

La sentencia *SELECT* en *PL/SQL* no muestra en pantalla las filas resultantes de la consulta, como ocurre en *SQL* (el cual trabaja en forma interactiva) sino que, según sea la acción a realizar, así será la cantidad de filas devueltas por la consulta, existiendo en este caso una de las tres posibles situaciones recogidas en la tabla 1:

Cantidad de filas	Acción
Una	Se realiza la siguiente sentencia

Más de una	Ocurre la excepción TOO_MANY_ROWS
Ninguna	Ocurre la excepción NO_DATA_FOUND

Tabla 1: Situaciones posibles según la búsqueda realizada

Por esta razón, veremos a continuación, a través de un ejemplo, el uso de la sentencia *SELECT,* con control de excepciones para definir la acción a realizar en dependencia de la cantidad de filas devueltas por la consulta.

Veamos con el código del fuente 4 en el que se define el procedimiento *VentasDe* para consultar las ventas realizadas en el día de un determinado artículo:

```
/* --- Fuente 4 ------------------------------------------------------------

PROCEDURE ventasde(xarticulo ventas.articulo%TYPE) is
    BEGIN
       DECLARE xnombre  clientes.nombre%TYPE;
           xventas      NUMBER;
    BEGIN
       SELECT  nombre into xnombre
       FROM    clientes,ventas
       WHERE   clientes.codigo=ventas.codigo
           AND
               ventas.fecha=sysdate
           AND
               articulo=xarticulo;
               message( 'Solo una venta de '
                        xarticulo||' a: '||xnombre);
    EXCEPTION
       WHEN NO_DATA_FOUND THEN
           message('No hay ventas de '||xarticulo);
           pause;
       WHEN  TOO_MANY_ROWS THEN
           SELECT   COUNT(*) INTO XVENTAS
           FROM     ventas
           WHERE    ventas.fecha=sysdate
               AND
               articulo=xarticulo;
               message( TO_CHAR(xventas)||
                        ' Ventas de '||xarticulo);
               pause;
       WHEN OTHERS THEN
           message('Error Indefinido');
           pause;
    END;
END;
```

Este procedimiento *ventasde* recibe un parámetro que es el nombre del artículo a consultar, por lo cual, para ejecutarlo, se debe escribir *ventasde('PAPEL')*.

Obsérvese también que la sentencia *SELECT COUNT(*) INTO*, usada en este ejemplo, siempre devuelve una fila, ya que no existe formación de grupos al no estar presente la orden *GROUP BY*. Por lo tanto, en este caso la única acción posible a realizar es pasar a la siguiente sentencia, o sea, no se requiere control de excepciones.

Manejo de cursores

El conjunto de filas resultantes de una consulta con la sentencia *SELECT*, como vimos

anteriormente, puede estar compuesto por ninguna, una o varias filas, dependiendo de la condición que define la consulta. Para poder procesar individualmente cada fila de la consulta debemos definir un cursor (que es un área de trabajo de memoria) que contiene los datos de las filas de la tabla consultada por la sentencia *SELECT*.

Los pasos para el manejo de cursores, tema novedoso en la programación de *Oracle* con *PL/SQL*, son:

- Definir el cursor, especificando la lista de parámetros con sus correspondientes tipos de datos y estableciendo la consulta a realizar con la sentencia *SELECT*.
- Abrir el cursor para inicializarlo, siendo éste el momento en que se realiza la consulta.
- Leer una fila del cursor, pasando sus datos a las variables locales definidas a tal efecto.
- Repetir el proceso fila a fila hasta llegar a la última.
- Cerrar el cursor una vez que se terminó de procesar su última fila.

A continuación veremos un ejemplo de cursor con las siguientes características:

Objetivo: Consultar las ventas de una fecha dada ordenadas de mayor a menor.

Nombre: *CVENTAS*.

Parámetros: cfecha, variable que contiene la fecha a consultar.

Código de definición del cursor: Ver figura 1

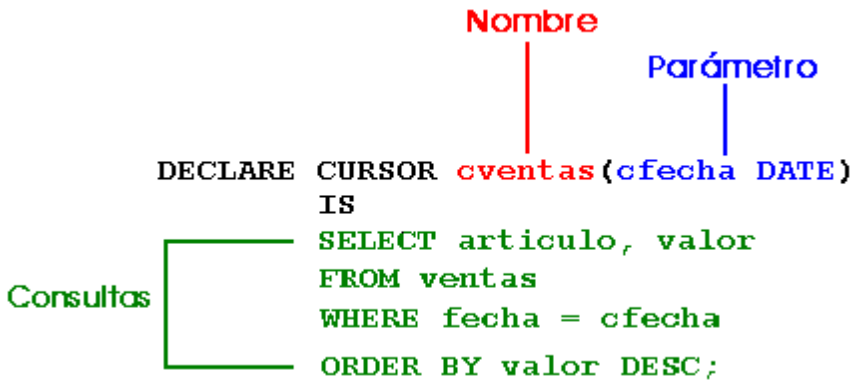

Figura 1: Código de definición de cursor

Con el procedimiento *VENTAS5* del fuente 5, mostraremos cómo usar el cursor *cventa* anteriormente definido, con el fin de registrar en la tabla *VENTAMAYOR* las 5 mayores ventas en una fecha dada.

```
/* --- Fuente 5 ------------------------------------------------------

PROCEDURE VENTAS5 (xfecha DATE) is
```

```
BEGIN

    DECLARE CURSOR cventas (cfecha DATE)
       IS SELECT articulo,valor
           FROM ventas
           WHERE fecha=cfecha
           ORDER BY valor DESC;

       xarticulo   ventas.articulo%TYPE;

       xvalor      ventas.valor%TYPE;

    BEGIN

       OPEN cventas(xfecha);

       FOR i IN 1..5 LOOP
           FETCH cventas INTO xarticulo,xvalor;
           EXIT WHEN cventas%NOTFOUND;
           INSERT INTO  ventamayor VALUES
                        (xfecha,xarticulo,xvalor);
           COMMIT;
       END LOOP;

       CLOSE cventas;
    END;
END;
```

Para llamar al procedimiento *ventas5* en una fecha dada, se puede escribir, por ejemplo:

```
ventas5(to_date('15/11/95','DD/MM/YY')
```

o

```
ventas5(sysdate).
```

A continuación detallaremos las sentencias usadas en este procedimiento:

```
DECLARE cursor
```

Define el cursor, su consulta y la lista de parámetros que se pasan a la orden *WHERE*, es solo la declaración del cursor y no la realización de la consulta.

```
xarticulo ventas.articulo%TYPE;
```

Define la variable *xarticulo* igual a la columna *articulo* de la tabla ventas, que con el uso del atributo de variable *%TYPE* permite declarar una variable del mismo tipo que una columna de la tabla. No es necesario conocer cómo está definida esa columna en la tabla y, en caso que la definición de la columna sea modificada, automáticamente se cambia la variable *xarticulo*.

```
OPEN cventas(xfecha);
```

Realiza la consulta asociada al cursor, pasando el valor del parámetro y guardando sus resultados en un área de la memoria, desde la cual, posteriormente, se pueden leer estas filas.

```
FOR i IN 1..5 LOOP
```

Ciclo numérico de repetición para poder consultar las 5 primeras ventas devueltas por el cursor.

```
FETCH cventas INTO xarticulo,xvalor;
```

Lee la siguiente fila de datos del cursor *cventas* y pasa los datos de la consulta a las variables *xarticulo* y *xvalor*.

```
EXIT WHEN cventas%NOTFOUND;
```

Garantiza la salida del ciclo antes de las última repetición, en caso que para una fecha dada se hayan efectuado menos de 5 ventas, ya que en esta situación la consulta del cursor devuelve menos de 5 filas.

%NOTFOUND es un atributo de cursor que es verdadero cuando la última sentencia *FETCH* no devuelve ninguna fila.

```
INSERT INTO ventamayor
   VALUES(xfecha,xarticulo,xvalor);
```

Insertar en la tabla *ventamayor* los valores leídos desde el cursor.

```
COMMIT;
```

Actualización de la tabla *ventamayor*.

```
END LOOP;
```

Fin del ciclo.

```
CLOSE cventas;
```

Cierra el cursor, eliminado sus datos del área de memoria.

Disparadores

El módulo *SQL*Forms* tiene incorporado una colección de procedimientos y funciones llamados "empaquetados" que se pueden incluir en el código de procedimientos o disparadores (*TRIGGER*) definidos por el usuario.

El disparador es un bloque de código que se activa cuando se pulsa una determinada tecla u ocurre cierto evento, como puede ser:

- Mover el cursor hacia o desde un campo, registro, bloque o forma.
- Realizar una consulta.
- Validar un dato.
- Hacer una transacción al insertar, modificar o eliminar registros de la base de datos.

Oracle asocia a cada tecla de función un procedimiento empaquetado, pudiendo el usuario redefinir esta asignación o capturar el disparador para ampliarlo o modificarlo con su propio código.

Usaremos como ejemplo la tecla [F9], que tiene asociado el disparador *KEY-LISTVAL*, que al activarse llama al procedimiento empaquetado *LIST_VALUES*, con el fin de consultar la lista de todos los valores del campo previamente codificado.

Ahora veamos cómo redefinimos el disparador *KEY-LISTVAL* con el objetivo de capturarlo para consultar, no sólo toda la lista de valores del campo, sino para incluir la potencia del operador *LIKE*, estudiados en el artículo *Oracle Básico (II): Consulta con SQL* (ver revista ***Algoritmo*** número 9), que nos permitirá consultar parte de la lista de valores (a partir de un patrón de consulta), brindando la posibilidad de incluir la consulta del tipo "comienzan con la palabra..." o de la consulta "contiene la palabra...".

En este caso el código sería el de la figura 2:

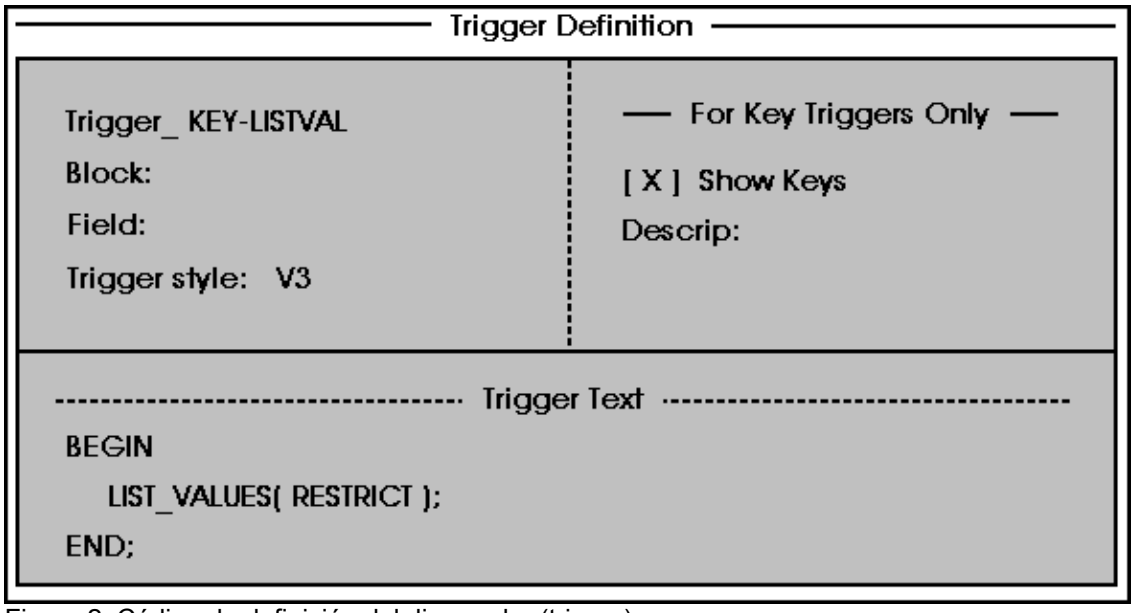

Figura 2: Código de definición del disparador (trigger)

Donde

KEY-LISTVAL Disparador que se activa con la tecla [F9] y que, por defecto, ejecuta el procedimiento *LIST_VALUES* sin parámetros, que da la lista de todos los valores diferentes del campo.

LIST_VALUES (RESTRICT) Es el procedimiento empaquetado para obtener la lista de valores del campo, que con el parámetro *RESTRICT,* con la lógica del operador *LIKE,* permite ampliar las posibilidades de esta consulta; como explicamos anteriormente.

A partir de la versión 7 de *Oracle* el usuario puede almacenar, en forma independiente, sus funciones y procedimientos sin tener que escribirlos repetidamente para cada forma, y pudiendo compilarlos independientemente de las formas que lo usen. Pero, además, las funciones y procedimientos se pueden agrupar en un paquete para compartir definiciones, variables globales, constantes, cursores y excepciones, así como garantizar y revocar los permisos a nivel de paquete.

En el caso que sea necesario modificar el contenido del paquete, como el mismo se encuentra almacenado separadamente, no es necesario recompilar nada que use ese paquete, lo que facilita la gestión y mantenimiento de todos los procedimientos almacenados como una sola entidad para una determinada aplicación.

Además, en la versión 7, existe un nuevo tipo de disparador llamado *de base de datos*, que es un procedimiento asociado a una tabla que se activa cuando se produce un suceso que afecta a esa tabla. Su uso más común consiste en la definición de restricciones complejas de integridad.

Hasta aquí, he mostrado los conceptos básicos de la programación en *PL/SQL*, a partir de los cuales el lector estará en condiciones de seguir profundizando en el tema. Recomiendo usar el Manual de *Oracle: PL/SQL: Guía del Usuario*, que no sólo es detallado y claro, sino que también contiene una gran variedad de ejemplos.

En nuestro próximo artículo pasaremos a la creación de reportes (informes) con el módulo *SQL*Report* y al diseño de menús con *SQL*Menu*.

Bibliografía

Oracle 7 Manual de Referencia
Koch, George.
Osborne/McGraw-Hill
1999.

Oracle Manual de Referencia.
Koch, George.
Osborne/McGraw-Hill.
1997.

Mastering Oracle.
Cronin, Daniel.
Hayden Books.
1999.