Descripción PL/SQL

SQL es un lenguaje de consulta, para los sistemas de bases de datos relacionales, que no posee la potencia de los lenguajes de programación. No permite el uso de variables, estructuras de control de flujo, bucles y demás elementos característicos de la programación. No es de extrañar, SQL es un lenguaje de consulta, no un lenguaje de programación.

Sin embargo, SQL es la herramienta ideal para trabajar con bases de datos. Cuando se desea realizar una aplicación completa, para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. **PL/SQL** es el lenguaje de programación que proporciona Oracle para extender el SQL estándar con otro tipo de instrucciones y elementos propios de los lenguajes de programación.

Con PL/SQL vamos a poder programar las unidades de programa de la base de datos Oracle:

- Procedimientos almacenados
- Funciones
- Triggers
- Scripts

Pero además, PL/SQL nos permite realizar programas sobre las siguientes herramientas de Oracle:

- Oracle Forms / APEX
- Oracle Reports
- Oracle Graphics
- Oracle Aplication Server

Conceptos básicos

Como introducción vamos a ver algunos elementos y conceptos básicos del lenguaje. PL/SQL no es CASE-SENSITIVE, es decir, no diferencia mayúsculas de minúsculas como otros lenguajes de programación como C o Java. Sin embargo, debemos recordar que Oracle es CASE-SENSITIVE en las búsquedas de texto.

Una línea en PL/SQL contiene grupos de caracteres, conocidos como UNIDADES LÉXICAS, que pueden ser clasificadas como: DELIMITADORES, IDENTIFICADORES, LITERALES, COMENTARIOS o EXPRESIONES.

- **DELIMITADOR:** Es un símbolo, simple o compuesto, que tiene una función especial en PL/SQL:
    - Operadores Aritméticos
    - Operadores Lógicos
    - Operadores Relacionales
- **IDENTIFICADOR:** Son empleados para nombrar objetos de programas en PL/SQL, así como a unidades dentro del mismo:
    - Constantes
    - Cursores
    - Variables
    - Subprogramas
    - Excepciones
    - Paquetes
- **LITERAL:** Es un valor de tipo numérico, carácter, cadena o lógico no representado por un identificador (es un valor explícito).

- **COMENTARIO:** Es una aclaración que el programador incluye en el código. Son soportados dos estilos de comentarios, el de línea simple y de multilínea, para lo cual son empleados ciertos caracteres especiales.

Tipos de datos

Cada constante y variable tiene un tipo de dato en el que se especifica el formato de almacenamiento, restricciones y rango de valores válidos. PL/SQL proporciona una variedad predefinida de tipos de datos. Casi todos los tipos de datos manejados por PL/SQL son similares a los soportados por SQL. A continuación se muestran los tipos de datos más comunes:

- *NUMBER* (numérico): Almacena números enteros o de punto flotante, virtualmente de cualquier longitud, aunque puede ser especificada la precisión (número de dígitos) y la escala, que es la que determina el número de decimales.
- *CHAR* (carácter): Almacena datos de tipo carácter con un tamaño máximo de 32.767 bytes y cuyo valor de longitud por defecto es 1.
- *VARCHAR2* (carácter de longitud variable): Almacena datos de tipo carácter empleando sólo la cantidad necesaria aún cuando la longitud máxima sea mayor.
- *BOOLEAN* (lógico): Se emplea para almacenar valores TRUE o FALSE.
- *DATE* (fecha): Almacena datos de tipo fecha. Las fechas se almacenan internamente como datos numéricos, por lo que es posible realizar operaciones aritméticas con ellas.
- *Atributos de tipo*. Un atributo de tipo PL/SQL es un modificador que puede ser usado para obtener información de un objeto de la base de datos. El atributo %TYPE permite conocer el tipo de una variable, constante o campo de la base de datos. El atributo %ROWTYPE permite obtener los tipos de todos los campos de una tabla de la base de datos, de una vista o de un cursor.

Tablas de PL/SQL

Las tablas de PL/SQL son tipos de datos que nos permiten almacenar varios valores del mismo tipo de dato. Una tabla PL/SQL, que es similar a un array, tiene dos componentes:

- Un índice de tipo BINARY_INTEGER que permite acceder a los elementos en la tabla PL/SQL.
- Una columna de escalares o registros que contiene los valores de la tabla PL/SQL.

Puede incrementar su tamaño dinámicamente. La sintaxis general para declarar una tabla de PL es la siguiente:

```
TYPE <nombre_tipo_tabla> IS TABLE OF

<tipo_datos>[NOT NULL]

INDEX BY BINARY_INTEGER ;
```

Una vez que hemos definido el tipo, podemos declarar variables y asignarle valores.

```
DECLARE   /* Definimos el tipo PAISES como tabla PL/SQL */

   TYPE PAISES IS TABLE OF NUMBER INDEX BY BINARY_INTEGER ;

      /* Declaramos una variable del tipo PAISES */

   tPAISES PAISES;

BEGIN

   tPAISES(1) := 1;

   tPAISES(2) := 2;

   tPAISES(3) := 3;

END;
```

No es posible inicializar las tablas en la declaración. El rango de binary integer es –2147483647.. 2147483647, por lo tanto el índice puede ser negativo, lo cual indica que el índice del primer valor no tiene que ser necesariamente el cero.

Tablas PL/SQL de registros

Es posible declarar elementos de una tabla PL/SQL como de tipo registro.

```
DECLARE
        TYPE PAIS IS RECORD   (
      CO_PAIS
      NUMBER NOT NULL ,
      DESCRIPCION VARCHAR2(50),
      CONTINENTE  VARCHAR2(20)  );
      TYPE PAISES IS TABLE OF PAIS INDEX BY BINARY_INTEGER ;
  tPAISES PAISES;
BEGIN
  tPAISES(1).CO_PAIS := 27;
  tPAISES(1).DESCRIPCION := 'ITALIA';
  tPAISES(1).CONTINENTE  := 'EUROPA';
END;
```

Cuando trabajamos con tablas de PL podemos utilizar las siguientes funciones:

- FIRST. Devuelve el menor índice de la tabla. NULL si está vacía.
- LAST. Devuelve el mayor índice de la tabla. NULL si está vacía.
- EXISTS(i). Utilizada para saber si en un cierto índice hay almacenado un valor. Devolverá TRUE si en el índice i hay un valor.
- COUNT. Devuelve el número de elementos de la tabla PL/SQL.
- PRIOR (n). Devuelve el número del índice anterior a n en la tabla.
- NEXT(n). Devuelve el número del índice posterior a n en la tabla.
- TRIM. Borra un elemento del final de la tabla PL/SQL.
- TRIM(n). Borra n elementos del final de la tabla PL/SQL.
- DELETE. Borra todos los elementos de la tabla PL/SQL.
- DELETE(n) Borra el correspondiente al índice n.
- DELETE(m,n) Borra los elementos entre m y n.

Estructuras

Estas son las diversas estructuras que se manejan en PL/SQL:

Estructuras de control de flujo

En PL/SQL sólo disponemos de la estructura condicional IF. Su sintaxis se muestra a continuación:

```
IF (expresión) THEN
            -- Instrucciones
      ELSIF (expresión) THEN
            -- Instrucciones
      ELSE
```

```
            -- Instrucciones
END IF;
```

Un aspecto a tener en cuenta es que la instrucción condicional anidada es ELSIF y no "ELSEIF".

Sentencia GOTO

PL/SQL dispone de la sentencia GOTO. La sentencia GOTO desvía el flujo de ejecución a una determinada etiqueta. En PL/SQL las etiquetas se indican del siguiente modo: **<< etiqueta >>**. El siguiente ejemplo ilustra el uso de GOTO:

```
DECLARE
  flag NUMBER;
BEGIN
  flag :=1 ;
  IF (flag = 1) THEN
      GOTO paso2;
  END IF;
  <<paso1>>
      dbms_output.put_line('Ejecucion de paso 1');
  <<paso2>>
      dbms_output.put_line('Ejecucion de paso 2');
END;
```

Bucles

En PL/SQL tenemos a nuestra disposición los siguientes iteradores o bucles: **LOOP, WHILE, FOR**. El bucle LOOP se repite tantas veces como sea necesario hasta que se fuerza su salida con la instrucción EXIT. Su sintaxis es la siguiente:

```
LOOP
            -- Instrucciones
            IF (expresión) THEN
            -- Instrucciones
            EXIT;
            END IF;
END LOOP;
```

El bucle WHILE, se repite mientras que se cumpla la expresión:

```
WHILE (expresión) LOOP
   -- Instrucciones
END LOOP;
```

El bucle FOR se repite tanta veces como le indiquemos en los identificadores inicio y final:

```
FOR contador IN [REVERSE] inicio..final LOOP
      -- Instrucciones
```

```
END LOOP;
```

En el caso de especificar REVERSE el bucle se recorre en sentido inverso.

Bloques PL/SQL

Un programa de PL/SQL está compuesto por bloques, como mínimo de uno. Los bloques de PL/SQL pueden ser de los siguientes tipos:

- Bloques anónimos.
- Subprogramas.

Estructura de un Bloque

Los bloques PL/SQL presentan una estructura específica compuesta de tres partes bien diferenciadas:

- La sección declarativa, donde se declaran todas las constantes y variables que se van a utilizar en la ejecución del bloque.
- La sección de ejecución, que incluye las instrucciones a ejecutar en el bloque PL/SQL.
- La sección de excepciones, en donde se definen los manejadores de errores que soportará el bloque PL/SQL.

Cada una de las partes anteriores se delimita por una palabra reservada, de modo que un bloque PL/SQL se puede representar como sigue:

```
[declare | is | as ]     /*Parte declarativa*/

begin /*Parte de ejecucion*/

[ exception ]      /*Parte de excepciones*/

end;
```

De las anteriores partes, únicamente la sección de ejecución es obligatoria, que quedaría delimitada entre las cláusulas BEGIN y END. Veamos un ejemplo de bloque PL/SQL muy genérico. Se trata de un bloque anónimo, es decir, no lo identifica ningún nombre. Los bloques anónimos identifican su parte declarativa con la palabra reservada DECLARE.

```
DECLARE

      /*Parte declarativa*/

      nombre_variable DATE;

BEGIN

      /*Parte de ejecución

      * Este código asigna el valor de la columna "nombre_columna"

            * a la variable identificada por "nombre_variable"

      */

      SELECT SYSDATE INTO nombre_variable FROM DUAL;

EXCEPTION

      /*Parte de excepciones*/

      WHEN OTHERS THEN

      dbms_output.put_line('Se ha producido un error');

END;
```

Cursores en PL/SQL

PL/SQL utiliza cursores para gestionar las instrucciones SELECT. Un cursor es un conjunto de registros devuelto por una instrucción SQL. Técnicamente, los cursores son fragmentos de memoria reservados para procesar los resultados de una consulta SELECT. Podemos distinguir dos tipos de cursores:

- Cursores implícitos. Este tipo de cursores se utiliza para operaciones SELECT INTO. Se usan cuando la consulta devuelve un único registro.
- Cursores explícitos. Son los cursores que son declarados y controlados por el programador. Se utilizan cuando la consulta devuelve un conjunto de registros. Ocasionalmente también se utilizan en consultas que devuelven un único registro por razones de eficiencia. Son más rápidos.

Un cursor se define como cualquier otra variable de PL/SQL y debe nombrarse de acuerdo a los mismos convenios que cualquier otra variable. Los cursores implícitos no necesitan declaración. El siguiente ejemplo declara un cursor explícito:

```
declare
   cursor c_paises is
   SELECT CO_PAIS, DESCRIPCION
   FROM PAISES;
begin
   /* Sentencias del bloque ...*/
end;
```

Para procesar instrucciones SELECT que devuelvan más de una fila, son necesarios cursores explícitos combinados con un estructura de bloque. Un cursor admite el uso de parámetros. Los parámetros deben declararse junto con el cursor. El siguiente ejemplo muestra la declaración de un cursor con un parámetro, identificado por p_continente.

```
declare
   cursor c_paises (p_continente IN VARCHAR2) is
   SELECT CO_PAIS, DESCRIPCION
   FROM PAISES
   WHERE CONTINENTE = p_continente;
begin
 /* Sentencias del bloque ...*/
end;
```

Cursores implícitos

Los cursores implícitos se utilizan para realizar consultas SELECT que devuelven un único registro. Deben tenerse en cuenta los siguientes puntos cuando se utilizan cursores implícitos:

- Con cada cursor implícito debe existir la palabra clave INTO.
- Las variables que reciben los datos devueltos por el cursor tienen que contener el mismo tipo de dato que las columnas de la tabla.
- Los cursores implícitos solo pueden devolver una única fila. En caso de que se devuelva más de una fila (o ninguna fila) se producirá una excepción.

Excepciones asociadas a los cursores implícitos

Los cursores implícitos sólo pueden devolver una fila, por lo que pueden producirse determinadas excepciones. Las más comunes que se pueden encontrar son no_data_found y too_many_rows. La siguiente tabla explica brevemente estas excepciones:

Excepción	**Explicación**
NO_DATA_FOUND	Se produce cuando una sentencia SELECT intenta recuperar datos pero ninguna fila satisface sus condiciones. Es decir, cuando "no hay datos".
TOO_MANY_ROWS	Dado que cada cursor implícito sólo es capaz de recuperar una fila , esta excepción detecta la existencia de más de una fila.

Cursores explícitos

Los cursores explícitos se emplean para realizar consultas SELECT que pueden devolver cero filas o más de una fila. Para trabajar con un cursor explícito necesitamos realizar las siguientes tareas:

- Declarar el cursor.
- Abrir el cursor con la instrucción OPEN.
- Leer los datos del cursor con la instrucción FETCH.
- Cerrar el cursor y liberar los recursos con la instrucción CLOSE

A continuación se muestra un ejemplo del empleo de cursores:

```
CURSOR nombre_cursor IS

  instrucción_SELECT // Se declara el cursor


CURSOR nombre_cursor(param1 tipo1, ..., paramN tipoN) IS

  instrucción_SELECT // Declaracion de cursor con parametro


OPEN nombre_cursor; // Abrir un cursor

      

OPEN nombre_cursor(valor1, valor2, ..., valorN); // abrir cursor con
parametros


FETCH nombre_cursor INTO lista_variables; // Recuperada datos en variables


FETCH nombre_cursor INTO registro_PL/SQL; // Recupera datos en registro


CLOSE nombre_cursor; // Cierra el cursor
```

El siguiente ejemplo ilustra el trabajo con un cursor explícito. Hay que tener en cuenta que, al leer los datos del cursor, debemos hacerlo sobre variables del mismo tipo de datos de la tabla (o tablas) que trata el cursor.

```
DECLARE

  CURSOR cpaises

  IS  SELECT CO_PAIS, DESCRIPCION, CONTINENTE
```

```
  FROM PAISES;
  co_pais VARCHAR2(3);
  descripcion VARCHAR2(50);
  continente  VARCHAR2(25);
BEGIN
  OPEN cpaises;
  FETCH cpaises INTO co_pais,descripcion,continente;
  CLOSE cpaises;
END;
```

Manejo de excepciones

En PL/SQL una advertencia o condición de error se llama excepción. Las excepciones se controlan dentro de su propio bloque, cuya estructura se muestra a continuación:

```
DECLARE
 -- Declaraciones
BEGIN
  -- Ejecución
EXCEPTION
  -- Excepción
END;
```

Cuando ocurre un error, se ejecuta la porción del programa marcada por el bloque EXCEPTION, transfiriéndose el control a ese bloque de sentencias. El siguiente ejemplo muestra un bloque de excepciones que captura las excepciones NO_DATA_FOUND y ZERO_DIVIDE. Cualquier otra excepción será capturada en el bloque WHEN OTHERS THEN.

```
DECLARE
 -- Declaraciones
BEGIN
  -- Ejecucion
EXCEPTION
WHEN NO_DATA_FOUND THEN
  -- Se ejecuta cuando ocurre una excepción de tipo NO_DATA_FOUND
WHEN ZERO_DIVIDE THEN
  -- Se ejecuta cuando ocurre una excepción de tipo ZERO_DIVIDE
WHEN OTHERS THEN
  -- Se ejecuta cuando ocurre una excepción de un tipo no tratado
  -- en los bloques anteriores
END;
```

Como se ha indicado, cuando ocurre un error se ejecuta el bloque EXCEPTION, transfiriéndose el control a las sentencias del bloque. Una vez finalizada la ejecución del bloque de EXCEPTION no se continúa ejecutando el bloque anterior. Si existe un bloque de excepción apropiado para el tipo de excepción se ejecuta dicho bloque. Si no existe un bloque de control de excepciones adecuado al tipo de excepción se ejecutará el bloque de excepción WHEN OTHERS THEN (¡si existe!). WHEN OTHERS debe ser el último manejador de excepciones.

Las excepciones pueden ser definidas de forma interna o explícitamente por el usuario. Ejemplos de excepciones definidas de forma interna son la división por cero y la falta de memoria en tiempo de ejecución. Estas mismas condiciones excepcionales tienen sus propio tipos y pueden ser referenciadas por ellos: ZERO_DIVIDE y STORAGE_ERROR. Las excepciones definidas por el usuario deben ser alcanzadas explícitamente utilizando la sentencia RAISE.

Con las excepciones se pueden manejar los errores cómodamente sin necesidad de mantener múltiples chequeos por cada sentencia escrita. También provee claridad en el código ya que permite mantener las rutinas correspondientes al tratamiento de los errores de forma separada de la lógica del negocio.

Excepciones predefinidas

PL/SQL proporciona un gran número de excepciones predefinidas que permiten controlar las condiciones de error más habituales. Las excepciones predefinidas no necesitan ser declaradas, simplemente se utilizan cuando éstas son lanzadas por algún error determinado. La siguiente lista muestra las excepciones predeterminadas por PL/SQL y una breve descripción de cuándo son accionadas:

Excepción	Descripción
ACCESS_INTO_NULL	El programa intentó asignar valores a los atributos de un objeto no inicializado -6530.
COLLECTION_IS_NULL	El programa intentó asignar valores a una tabla anidada aún no inicializada -6531.
CURSOR_ALREADY_OPEN	El programa intentó abrir un cursor que ya se encontraba abierto. Recuerde que un cursor de ciclo FOR automáticamente lo abre y ello no se debe especificar con la sentencia OPEN -6511.
DUP_VAL_ON_INDEX	El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index) -1.
INVALID_CURSOR	El programa intentó efectuar una operación no válida sobre un cursor -1001.
INVALID_NUMBER	En una sentencia SQL, la conversión de una cadena de caracteres hacia un número falla cuando esa cadena no representa un número válido -1722.
LOGIN_DENIED	El programa intentó conectarse a Oracle con un nombre de usuario o password inválido -1017.
NO_DATA_FOUND	Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada 100.
NOT_LOGGED_ON	El programa efectuó una llamada a Oracle sin estar conectado -1012.

PROGRAM_ERROR PL/SQL	Tiene un problema interno -6501.
ROWTYPE_MISMATCH	Los elementos de una asignación (el valor a asignar y la variable que lo contendrá) tienen tipos incompatibles. También se presenta este error cuando un parámetro pasado a un subprograma no es del tipo esperado -6504.
SELF_IS_NULL	El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo -30625.
STORAGE_ERROR	La memoria se terminó o está corrupta -6500.
SUBSCRIPT_BEYOND_COUNT	El programa está tratando de referenciar un elemento de una colección indexada que se encuentra en una posición más grande que el número real de elementos de la colección -6533.
SUBSCRIPT_OUTSIDE_LIMIT	El programa está referenciando un elemento de una tabla utilizando un número fuera del rango permitido (por ejemplo, el elemento "-1") -6532.
SYS_INVALID_ROWID	La conversión de una cadena de caracteres hacia un tipo ROWID falló porque la cadena no representa un número -1410.
TIMEOUT_ON_RESOURCE	Se excedió el tiempo máximo de espera por un recurso en Oracle -51.
TOO_MANY_ROWS	Una sentencia SELECT INTO devuelve más de una fila -1422.
VALUE_ERROR	Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo, sucede cuando se intenta introducir un valor muy grande dentro de una variable más pequeña -6502.
ZERO_DIVIDE	El programa intentó efectuar una división por cero -1476.

Excepciones definidas por el usuario

PL/SQL permite al usuario definir sus propias excepciones, que deberán ser declaradas y lanzadas explícitamente utilizando la sentencia RAISE. Las excepciones deben ser declaradas en el segmento DECLARE de un bloque, subprograma o paquete. Se declara una excepción como cualquier otra variable, asignándole el tipo EXCEPTION. Las mismas reglas de alcance aplican tanto sobre variables como sobre las excepciones.

```
DECLARE
 -- Declaraciones
      MyExcepcion EXCEPTION;
BEGIN  -- Ejecución
EXCEPTION  -- Excepción
END;
```

Reglas de alcance

Una excepción es válida dentro de su ambito de alcance, es decir, el bloque o programa donde ha sido declarada. Las excepciones predefinidas son siempre válidas. Como las variables, una excepción declarada en un bloque es local a ese bloque y global a todos los subbloques que comprende.

La sentencia RAISE

La sentencia RAISE permite lanzar una excepción en forma explícita. Es posible utilizar esta sentencia en cualquier lugar que se encuentre dentro del alcance de la excepción.

```
DECLARE
 -- Declaramos una excepción identificada por VALOR_NEGATIVO
      VALOR_NEGATIVO EXCEPTION;
      valor NUMBER;
BEGIN
  -- Ejecución valor := -1;
      IF valor < 0 THEN
            RAISE VALOR_NEGATIVO;
      END IF;
EXCEPTION  -- Excepción
WHEN VALOR_NEGATIVO THEN
      dbms_output.put_line('El valor no puede ser negativo');
END;
```

Con la sentencia RAISE podemos lanzar una excepción definida por el usuario o predefinida, siendo el comportamiento habitual lanzar excepciones definidas por el usuario.

Propagación de excepciones

Una de las características más interesantes de la excepciones es su propagación. Cuando se lanza una excepción, el control se transfiere hasta la sección EXCEPTION del bloque donde se ha producido la excepción. Entonces se busca un manejador válido de la excepción (WHEN THEN, WHEN OTHERS THEN) dentro del bloque actual. En el caso de que no se encuentre ningún manejador válido, el control del programa se desplaza hasta el bloque EXCEPTION del bloque que ha realizado la llamada PL/SQL.

Subprogramas en PL/SQL

Como hemos visto anteriormente, los bloques de PL/SQL pueden ser bloques anónimos (scripts) y subprogramas. Los subprogramas son bloques de PL/SQL a los que asignamos un nombre identificativo y que normalmente almacenamos en la propia base de datos para su posterior ejecución. Los subprogramas pueden recibir parámetros. Los subprogramas pueden ser de varios tipos:

- Procedimientos almacenados.
- Funciones.
- Triggers.
- Subprogramas en bloques anónimos.

Triggers

Un trigger es un bloque PL/SQL asociado a una tabla, que se ejecuta como consecuencia de una determinada instrucción SQL (una operación DML: INSERT, UPDATE o DELETE) sobre dicha tabla. La sintaxis para crear un trigger es la siguiente:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
{BEFORE|AFTER}
            {DELETE|INSERT|UPDATE [OF col1, col2, ..., colN]
            [OR {DELETE|INSERT|UPDATE [OF col1, col2, ..., colN]...]}
ON <nombre_tabla>
[FOR EACH ROW [WHEN (<condicion>)]]
DECLARE
  -- variables locales
BEGIN
  -- Sentencias
[EXCEPTION]
  -- Sentencias control de excepcion
END <nombre_trigger>;
```

El uso de OR REPLACE permite sobreescribir un trigger existente. Si se omite y el trigger existe se producirá un error. Los triggers pueden definirse para las operaciones INSERT, UPDATE o DELETE, y pueden ejecutarse antes o después de la operación. El modificador BEFORE AFTER indica que el trigger se ejecutará antes o después de ejecutarse la sentencia SQL definida por DELETE, INSERT, UPDATE. Si incluimos el modificador OF el trigger, solo se ejecutará cuando la sentencia SQL afecte a los campos incluidos en la lista.

El alcance de los disparadores puede ser la fila o de orden. El modificador FOR EACH ROW indica que el trigger se disparará cada vez que se realicen operaciones sobre una fila de la tabla. Si se acompaña del modificador WHEN, se establece una restricción; el trigger solo actuará, sobre las filas que satisfagan la restricción. La siguiente tabla resume los contenidos anteriores.

Valor	Descripción
INSERT, DELETE, UPDATE	Define qué tipo de orden DML provoca la activación del disparador.
BEFORE, AFTER	Define si el disparador se activa antes o después de que se ejecute la orden.
FOR EACH ROW	Los disparadores con nivel de fila se activan una vez por cada fila afectada por la orden que provocó el disparo. Los disparadores con nivel de orden se activan sólo una vez, antes o después de la orden. Los disparadores con nivel de fila se identifican por la cláusula FOR EACH ROW en la definición del disparador

La cláusula WHEN sólo es válida para los disparadores con nivel de fila.

Procedimiento

Un procedimiento es un subprograma que ejecuta una acción especifica y que no devuelve ningún valor. Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código. La sintaxis de un procedimiento almacenado es la siguiente:

```
CREATE [OR REPLACE]
PROCEDURE <procedure_name> [(<param1> [IN|OUT|IN OUT] <type>,
                              <param2> [IN|OUT|IN OUT] <type>, ...)]
IS
  -- Declaración de variables locales
BEGIN
  -- Sentencias
[EXCEPTION]
  -- Sentencias control de excepción
END [<procedure_name>];
```

El uso de OR REPLACE permite sobreescribir un procedimiento existente. Si se omite, y el procedimiento existe, se producirá un error. La sintaxis es muy parecida a la de un bloque anónimo, salvo porque se reemplaza la sección DECLARE por la secuencia PROCEDURE … IS y en la especificación del procedimiento debemos especificar el tipo de datos de cada parámetro. Al especificar el tipo de dato del parámetro no debemos especificar la longitud del tipo. Los parámetros pueden ser de entrada (IN), de salida (OUT) o de entrada salida (IN OUT). El valor por defecto es IN, y se toma ese valor en caso de que no especifiquemos nada.

```
CREATE OR REPLACE
PROCEDURE Actualiza_Saldo(cuenta NUMBER,
                          new_saldo NUMBER)
IS
  -- Declaración de variables locales
BEGIN
  -- Sentencias
  UPDATE SALDOS_CUENTAS
      SET SALDO = new_saldo,
            FX_ACTUALIZACION = SYSDATE
  WHERE CO_CUENTA = cuenta;
END Actualiza_Saldo;
```

Función

Una función es un subprograma que devuelve un valor. La sintaxis para construir funciones es la siguiente:

```
CREATE [OR REPLACE]
FUNCTION <fn_name>[(<param1> IN <type>, <param2> IN <type>, ...)]
RETURN
 <return_type>
IS
  result <return_type>;
BEGIN   return(result);
[EXCEPTION]
  -- Sentencias control de excepción
END [<fn_name>];
```

El uso de OR REPLACE permite sobreescribir una función existente. Si se omite, y la función existe, se producirá un error. La sintaxis de los parámetros es la misma que en los procedimientos almacenados, exceptuando que sólo pueden ser de entrada.