TEMA

Tema 3. Introducción a PL / SQL

Administración de sistemas informáticos en red

Administración de sistemas gestores de bases de datos

Autor: Joan Pou

Linkia FP

Tema 3: Introducción a PL/SQL

¿Qué aprenderás?

- Aplicar un lenguaje de programación de bases de datos.
- Crear funciones y procedimientos almacenados.
- Crear triggers.
- Crear y gestionar cursores.

¿Sabías que…?

- PL/SQL: Es un lenguaje de programación.
- Los triggers son procedimientos que se ejecutan automáticamente.
- Los cursores nos permiten manipular los datos de las tablas.

3. Introducción a PL/SQL

El PL/SQL es un lenguaje de programación de bases de datos. Este lenguaje permite incorporar todas las características propias de los lenguajes de tercera generación: variables, estructura modular (procedimientos y funciones), estructuras de control (condicionales, bucles y demás estructuras), control de errores, y una total integración en el entorno Oracle. PL/SQL soporta todos los comandos de consulta i manipulación de datos.

Los programas realizados en PL/SQL se almacenan como cualquier otro objeto en la base de datos. Como si fueran cualquier otro objeto. Estos programas se ejecutan en el servidor, suponiendo una disminución del tráfico de red y un ahorro de recursos.

Con PL/SQL vamos a poder programar:

- Procedimientos almacenados
- Funciones
- Triggers
- Scripts

3.1. Primeros pasos con PL/SQL

Para programar en PL/SQL es necesario conocer sus fundamentos:

- El 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.
- IDENTIFICADORES: Se utilizan para nombrar los objetos en los programas PL/SQL entre los que podemos destacar: constantes, cursores, variables, excepciones, etc. Deben cumplir las siguientes características:
  - Pueden tener entre 1 y 30 caracteres de longitud.
  - El primer carácter debe ser una letra.
  - Los restantes caracteres deben ser alfanuméricos o signos admitidos (letras, dígitos, almohadilla, subguión o signo del dólar).
- Las instrucciones PL/SQL terminan con un punto y coma (;), excepto algunas instrucciones como BEGIN, DECLARE, etc.
- Los comentarios:
  - de una línea se inician con --
  - y los de varias líneas inician con /* y terminan con */

3.1.1. Bloques PL/SQL

El bloque es la estructura básica que sigue un programa PL/SQL. En los bloques podemos diferenciar tres partes:

```
[DECLARE
  Declaración de variables] /*Parte declarativa*/
BEGIN
  Sentencias SQL y PL/SQL /*Parte de ejecucion*/
[EXCEPTION
  Manejadores de excepciones] /*Parte de excepciones*/
END;
```

- **DECLARE**: En esta parte se declaran las variables locales, las globales, las constantes y también los procedimientos y funciones. (opcional)
- **BEGIN**: Donde empiezan las instrucciones de PL/SQL.
- **EXCEPTION**: En esta zona se tratan los errores y las excepciones de PL/SQL, también es opcional.

En PL/SQL se pueden escribir cuatro tipos de bloques de código:

- **Bloques anónimos:** No se almacenan en la BD. Se ejecutan tras escribirlos.
- **Procedimientos:** Se almacenan en la BD. Pueden tener parámetros de entrada y de salida.
- **Funciones:** Se almacenan en la BD. Pueden tener parámetros y devuelven un único valor.
- **Triggers:** Se almacenan en la BD y se ejecutan automáticamente cuando ocurre algún evento.

3.2. Tipos de datos PL/SQL

Las variables y constantes almacenan diferentes tipos de datos, son similares a los soportados por SQL, los más importantes son:

Carácter: Almacena cadenas de caracteres con una longitud máxima de 32KBytes.

CHAR(n)	Cadena de caracteres de longitud fija, tiene un tamaño de n bytes. Si no se especifica n Oracle le da un tamaño de 255 bytes. Las posiciones no ocupadas se rellenan con blancos.
VARCHAR2(n)	Cadena de caracteres de longitud variable con un límite máximo especificado por el valor n. Es obligatorio especificar el tamaño. A diferencia del anterior las posiciones ocupadas no se rellenan.
LONG(n)	Es una versión más grande de VARCHAR2. Se recomienda utilizar CLOB que puede almacenar hasta 4GB.

Numéricos:

NUMBER(P,E)	Número de P dígitos de los cuales E son decimales. P y E son opcionales pero si especificamos E es obligatorio especificar P. Ejemplo Precio NUMBER(6,2); permite almacenar hasta 9999.99
PLS_INTEGER	Almacena internamente los valores en formato binario. Se utiliza para contadores, índices, etc. Facilita los cálculos. Puede contener valores entre –2147483647 y +2147483647
FLOAT(n)	Almacena un número en coma flotante sin restricción de dígitos decimales. La diferencia con el tipo NUMBER puede contener decimales con cualquier escala.

Fecha / Hora:

DATE	Almacena datos de tipo fecha, internamente se almacena como un dato numérico por lo que es posible realizar cálculos aritméticos con ellas. Por defecto sólo se muestra la fecha, pero también almacena la hora: día/mes/año horas:minutos:segundos
TIMESTAMP	Almacena también la fecha y la hora pero incluye las fracciones de segundo, por defecto 6. Podemos usar WITH [LOCAL] TIME ZONE para grabar con el desplazamiento de la hora local.
INTERVAL YEAR (y) TO MONTH	Representan el intervalo de tiempo entre dos fechas en años y meses (y) es el número de dígitos del año entre 0 y 9, por defecto 2.
INTERVAL DAY(d) TO SECOND(f)	Almacena el periodo de tiempo en días, horas, minutos y segundos. (d) es el número máximo dígitos en el día y (f) en los segundos.

Booleano:

BOOLEAN	Almacena datos los valores TRUE, FALSE y NULL, para especificar estos valores no utilizaremos comillas.

Otros tipos:

RAW(n)	Almacena datos binarios en longitud fija. Es obligatorio especificar el tamaño. Oracle recomienda utilizar BLOB.
LONG RAW	Objeto binario de longitud variable, hasta 2 gigabytes.
BLOB	Permite almacenar datos binarios no estructurados hasta un valor máximo de 8 terabytes.
CLOB	Se utiliza para almacenar un bloque de texto. Diseñado para almacenar datos de texto ASCII, incluido el texto formateado como HTML o PostScript.
ROWID	Cadena que representa un identificador único de una fila en una tabla. Sirve para guardar punteros a filas concretas.
BFILE	Contiene un identificador a un archivo binario almacenado fuera de la base de datos.

Atributos de tipo: Se usan para obtener información de un objeto de la base de datos.

%TYPE	Permite conocer el tipo de una variable, constante o campo de la base de datos.
%ROWTYPE	Permite obtener los tipos de todos los campos de una tabla de la base de datos, de una vista o de un cursor.

3.3. Declaración de variables y constantes

Las variables deben declararse en la sección DECLARE antes de su uso. En las rutinas almacenadas se declaran después de la cláusula IS, antes del bloque BEGIN que contienen las instrucciones a ejecutar. La sintaxis de la declaración de las variables es la siguiente:

<nombre_de_variable> <tipo> [NOT NULL] [{:= | DEFAULT} <valor>];

- La opción DEFAULT o bien la asignación := sirven para asignar valores por defecto a la variable en el momento de su creación.
- NOT NULL fuerza a que la variable siempre tenga un valor.

Ejemplos:

```
DECLARE
        precio NUMBER (6,2):=0;
        salario NUMBER (10,2);
        ciudad VARCHAR2(20) NOT NULL DEFAULT “Barcelona”;
        departamento CHAR(15) DEFAULT “Ventas”;
```

Podemos utilizar los atributos de tipo para declarar las variables que sean del mismo tipo que otros objetos.

Ejemplos:

```
DECLARE
        pvp producto.precio%TYPE;
        persona trabajadores%ROWTYPE;
```

persona puede contener una fila (registro) de la tabla trabajadores.

Una constante sirve para almacenar un valor fijo. Para declarar una constante seguimos la siguiente sintaxis:

<nombre_de_constante> CONSTANT <tipo> := <valor>;

Ejemplo:

iva CONSTANT number(4,2) := 21;

3.4. Interacción con el usuario

En PL/SQL disponemos de un paquete para poder visualizar cadenas de texto por la pantalla con fines de depuración. Se trata del paquete DBMS_OUTPUT y el procedimiento PUT_LINE, su formato es el siguiente:

DBMS_OUTPUT.PUT_LINE(<expresión>);

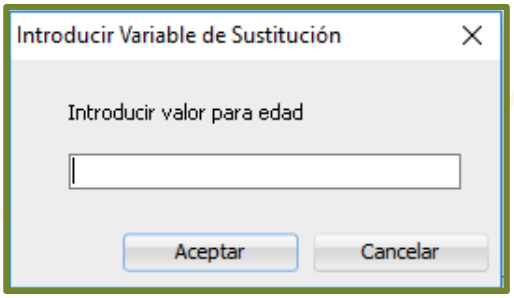
Para poder utilizar este procedimiento la variable SERVEROUTPUT debe estar en ON. Para cambiar el estado de esta variable utilizaremos la instrucción SET SERVEROUTPUT ON al principio de la sesión.

Ejemplo:

```
-- Visualización de mensajes por consola.
SET SERVEROUTPUT ON;
BEGIN
        dbms_output.put_line('Hola');
END;
/
```

Para introducir datos o valores por el teclado y asignarlos a las variables utilizaremos el símbolo & delante del nombre de la variable. Por ejemplo:

```
SET SERVEROUTPUT ON;
DECLARE
edad   BINARY_INTEGER := &edad;
nombre  VARCHAR2(20) := '&nombre';
BEGIN
dbms_output.put_line('Hola ' || nombre);
dbms_output.put_line('Tienes ' || edad || ' años');
END;
/
```

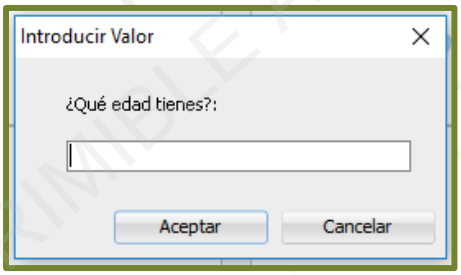


El operador || lo utilizamos para concatenar expresiones de texto.

Oracle nos mostrará en la consola el procedimiento con los valores que ha adquirido por el teclado. Si queremos que no aparezca esta información utilizaremos el comando SET VERIFY OFF al principio de la sesión.

Con el comando accept y prompt podemos cambiar el texto que aparece en el cuadro de diálogo de la variable. También podemos añadir una máscara con el parámetro FORMAT.

```
SET SERVEROUTPUT ON;
SET VERIFY OFF;
        accept nombre prompt 'Escribe tu nombre: ';
        accept edad   number format 999 prompt '¿Qué edad tienes?: ';
DECLARE
        edad   BINARY_INTEGER := &edad;
        nombre  VARCHAR2(20) := '&nombre';
BEGIN
        dbms_output.put_line('Hola ' || nombre);
        dbms_output.put_line('Tienes ' || edad || ' años');
END;
```



Vídeo PL/SQL.

3.5. Estructuras de control

En las estructuras de control se evalúa una condición que puede devolver tres resultados: TRUE, FALSE o NULL, el valor NULL es equivalente a FALSE.

- Alternativa simple.

Si la condición se cumple se ejecutan las instrucciones especificadas después del THEN. En caso contrario no ejecuta nada.

```
IF condición THEN
        Instrucciones a ejecutar cuando la condición es cierta;
END IF;
```

- Alternativa doble.

Si la condición se cumple se ejecutan las instrucciones especificadas después del THEN. En caso contrario se ejecutaran las especificadas después del ELSE.

```
IF condición THEN
        Instrucciones a ejecutar cuando la condición es cierta;
ELSE
        Instrucciones a ejecutar cuando la condición es falsa;
END IF;
```

- Alternativa múltiple.

Evalúa cada condición empezando desde el principio, hasta encontrar una que se cumpla, ejecutando las instrucciones que siguen a la cláusula THEN. El ELSE es opcional y solo se ejecutará cuando no se ha cumplido ninguna de las condiciones anteriores. Debemos tener en cuenta que la instrucción condicional a utilizar es ELSIF y no ELSEIF.

```
IF condición1 THEN
        Instrucciones a ejecutar cuando la condición1 es cierta;
ELSIF condición2 THEN
        Instrucciones a ejecutar cuando la condición2 es cierta;
…..
ELSE
        Instrucciones a ejecutar cuando todas las condiciones son falsas;
END IF;
```

Ejemplo:

En el siguiente ejemplo preguntamos un número por la consola y nos dice si es positivo o negativo.

```
SET SERVEROUTPUT ON;
SET VERIFY OFF;
        accept numero prompt 'Escribe un número: ';
DECLARE
        numero  BINARY_INTEGER := &numero;
BEGIN
        IF (numero>=0) THEN
                dbms_output.put_line('Es positivo ' || numero);
        ELSE
                dbms_output.put_line('Es negativo ' || numero);
        END IF;
END;
 /
```

- Estructura CASE.

Permite realizar las mismas operaciones que la alternativa múltiple. Su sintaxis es la siguiente:

```
CASE [expresión]
        WHEN condición1 THEN resultado1;
        WHEN condición2 THEN resultado2;
        …..
        WHEN condiciónN THEN resultadoN;
        ELSE resultado;
END CASE;
```

Ejemplo:

En el siguiente ejemplo en función del departamento de un trabajador asigna un determinado salario.

```
DECLARE
        salario NUMBER(6,2);
        departamento  VARCHAR2(20) := '&departamento';
BEGIN
        CASE departamento
                WHEN 'ventas' THEN salario:=1300;
                WHEN 'comercial' THEN salario:=1200;
                ELSE salario:=1000;
        END CASE;
        dbms_output.put_line('Salario: ' || salario);
END;
/
```

3.6. Estructuras de control

Un bucle permite repetir una acción un determinado número de veces. Debemos tener cuidado en no crear bucles infinitos. En PL/SQL disponemos de 3 tipos de bucles: LOOP, WHILE y FOR

- Bucle LOOP

Se repite tantas veces como sea necesario hasta que forcemos la salida del mismo con la instrucción EXIT. Su sintaxis es la siguiente:

```
LOOP
        -- Instrucciones
        IF condición THEN
                -- Instrucciones
                 EXIT;
        END IF;
END LOOP;
```

Ejemplo:

En el siguiente ejemplo el bucle se repite 6 veces hasta que la variable contador tiene un valor superior a 5.

```
DECLARE
        contador PLS_INTEGER DEFAULT 0;
BEGIN
        LOOP
                contador:=contador+1;
                dbms_output.put_line('Contador: ' || contador);
                IF (contador>5) THEN exit;
                END IF;
        END LOOP;
END;
```

- Bucle While.

El bucle WHILE se repite mientras que se cumpla la expresión. Como la expresión se evalúa antes de entrar en el bucle puede suceder el caso de que no se ejecuten las instrucciones nunca. La sintaxis es la siguiente:

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

Ejemplo:

En este ejemplo se visualizan los números del 9 al 0 de forma decreciente, en este caso el bucle se repite 10 veces.

```
DECLARE
        contador PLS_INTEGER DEFAULT 10;
BEGIN
        WHILE (contador>0) LOOP
                contador:=contador-1;
                dbms_output.put_line('Contador: ' || contador);
        END LOOP;
END;
```

- Bucle FOR**.**

El bucle FOR se repite tantas veces como le indiquemos en los identificadores de inicio y final. En el caso de especificar la cláusula REVERSE el bucle se recorre en sentido inverso.

```
FOR contador IN [REVERESE] inicio..final LOOP
-- Instrucciones
END LOOP;
```

Respecto a la variable de control del bucle no hay que declararla y es local al bucle, por tanto no es accesible desde el exterior, no se le puede asignar ningún valor, pero si que la podemos usar en una expresión en el interior del bucle.

Ejemplo:

En el siguiente ejemplo preguntamos un número por el teclado y se visualiza su tabla de multiplicar.

```
accept numero prompt 'Escribe un número: ';
DECLARE
        numero  PLS_INTEGER := &numero;
BEGIN
     dbms_output.put_line('Tabla de múltiplicar del ' || numero);
    FOR i IN 1..10 LOOP
        dbms_output.put_line(numero || ' x ' || i || ' = ' || numero*i);
    END LOOP;
END;
 /
```

```
Tabla de múltiplicar del 3
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

Procedimiento PL/SQL terminado correctamente.
```

3.7. Procedimientos y funciones

Son bloques PL/SQL que tienen un nombre y pueden recibir y devolver valores. Se guardan en la base de datos y podemos ejecutarlos llamándolos desde otros procedimientos y funciones.

3.7.1. Procedimientos

Son códigos PL/SQL compilados y ejecutados en el servidor que pueden aceptar parámetros de entrada y/o de salida. No devuelven un valor de retorno. La sintaxis de un procedimiento es la siguiente:

```
CREATE [OR REPLACE] PROCEDURE <nombre_procedimiento>
                [(<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 [<nombre_procedimiento>];
```

Un procedimiento se indica mediante las palabras clave CREATE PROCEDURE seguido del nombre y sus parámetros. Opcionalmente se puede utilizar OR REPLACE. Esto permitirá crear o reemplazar el procedimiento, si existe previamente.

Puede haber un número ilimitado de parámetros seguidos de un modo y un tipo. Los modos posibles son:

- IN: Solo lectura.
- OUT: Solo escritura.
- INOUT: Lectura y escritura.

A diferencia de las declaraciones de variables en PL/SQL los tipos especificados en la declaración de los parámetros no pueden tener restricciones. Por ejemplo VARCHAR2(5) o NUMBER(4,2) no son válidos, debemos utilizar VARCHAR2 o NUMBER en su lugar.

Ejemplo:

En el siguiente ejemplo se crea un procedimiento para calcular la tabla de multiplicar de un número pasado como parámetro.

```
BEGIN
        tablamulti(5);
        tablamulti(2);
END;
/
CREATE OR REPLACE PROCEDURE tablamulti (numero NUMBER)
IS
        i BINARY_INTEGER := 0;
BEGIN
        dbms_output.put_line('Tabla de multiplicar del ' || numero);
 FOR i IN 1..10 LOOP
        dbms_output.put_line(numero || ' x ' || i || ' = ' || numero*i);
 END LOOP;
END tablamulti;
/
```

3.7.2. Funciones

Una función es un código compilado en el servidor que se ejecuta en local, puede aceptar parámetros de entrada y devuelve un único valor de retorno. Puede incluirse en otras funciones o procedimientos y en expresiones. También se puede utilizar o incluir en instrucciones SELECT, UPDATE, INSERT y DELETE.

PL/SQL incorpora un gran número de funciones predefinidas muy útiles entre las que podemos destacar:

SYSDATE	Devuelve la fecha del sistema.
TO_DATE	Convierte una expresión al tipo fecha.
TO_CHAR	Convierte una expresión al tipo char.
TO_NUMBER	Convierte una expresión al tipo numérico.
LENGTH	Devuelve la longitud de una cadena.
INSTR	Busca una cadena de caracteres dentro de otra. Devuelve la posición donde empieza la cadena buscada.
REPLACE	Reemplaza un texto por otro en una cadena.
SUBSTR	Obtiene una parte de una cadena, desde una posición de inicio hasta una determinada longitud.
UPPER	Convierte una cadena de texto a mayúsculas.
LOWER	Convierte una cadena de texto a minúsculas.
RPAD,LPAD	Añaden N veces una determinada cadena de caracteres a la derecha o izquierda de una expresión.
RTRIM, LTRIM, TRIM	Elimina los espacios en blanco a la derecha, a la izquierda o a la derecha e izquierda de una determinada cadena.
MOD	Devuelve el resto de la división entera.

Ejemplo 1:

Muestra la fecha y fecha convertida a cadena de caracteres en formato largo y una conversión de una cadena a número.

```
DECLARE
        cadena VARCHAR2(5):='200';
BEGIN
 dbms_output.put_line(SYSDATE);
 dbms_output.put_line(TO_CHAR(SYSDATE, 'Day, DD "de" Month "de" YYYY'));
 dbms_output.put_line(TO_NUMBER(cadena)+100);
END;
/
```

```
10/09/17
Domingo   , 10 de Septiembre de 2017
300

Procedimiento PL/SQL terminado correctamente.
```

Ejemplo 2:

Utilización de diversas funciones que trabajan con cadenas de caracteres.

```
DECLARE
cadena  VARCHAR2(20) := 'Lenguaje PL/SQL';
BEGIN
  dbms_output.put_line('Cadena: ' || cadena);
  dbms_output.put_line('Longitud: ' || LENGTH(cadena));
  dbms_output.put_line('Posición PL: ' || INSTR(cadena,'PL',1));
  dbms_output.put_line('substitución e por a: ' || REPLACE(cadena,'e','a'));
  dbms_output.put_line('2 caracteres a partir posición 4: ' ||SUBSTR(cadena,4,2));
  dbms_output.put_line('mayúsculas: ' || UPPER(cadena));
  dbms_output.put_line('minúsculas: ' || LOWER(cadena));
END;
/
```

```
Cadena: Lenguaje PL/SQL
Longitud: 15
Posición PL: 10
substitución e por a: Languaja PL/SQL
2 caracteres a partir posición 4: gu
mayúsculas: LENGUAJE PL/SQL
minúsculas: lenguaje pl/sql

Procedimiento PL/SQL terminado correctamente.
```

También podemos crear funciones en lugar de procedimientos, la principal diferencia es que la función siempre retorna un valor. En su declaración y tras los parámetros pondremos RETURN y el tipo de valor que retornará.

La sintaxis de las funciones definidas por el usuario es la siguiente:

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

Ejemplo:

En el siguiente ejemplo se crea una función a la que se pasa como parámetro el dni de un trabajador y la función retorna su salario.

```
SET SERVEROUTPUT ON;
SET VERIFY OFF;
DECLARE
 salario trabajadores.salario%TYPE;
BEGIN
 salario:=obtenersalario('111122A');
 dbms_output.put_line('El salario es: ' || salario);
END;
/

CREATE OR REPLACE FUNCTION obtenersalario (v_dni trabajadores.dni%TYPE)
RETURN number
```

```
IS
   v_salario  trabajadores.salario%TYPE:=0;
BEGIN
 select salario into v_salario from trabajadores where dni=v_dni;
 return v_salario;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   return 0;
END;
/
```

Vídeo Procedimientos y funciones.

3.8. Excepciones

Las excepciones sirven para controlar los errores en tiempo de ejecución. Cuando se produce un error PL/SQL crea una excepción y pasa el control del programa a la sección EXCEPTION.

Si existe un bloque de excepción definido para el tipo de error producido se ejecuta dicho bloque. Si no existe un bloque de control de excepciones adecuado al tipo de error se ejecutará el bloque de excepción WHEN OTHERS THEN si está definido. WHEN OTHERS debe ser el último definido en la sección.

La sintaxis es la siguiente:

```
DECLARE
        -- Declaraciones
BEGIN
        -- Ejecución de las instrucciones.
EXCEPTION
        WHEN NO_DATA_FOUND THEN
        -- Se ejecuta cuando ocurre una excepción de tipo NO_DATA_FOUND
        WHEN OTHERS THEN
        -- Se ejecuta cuando ocurre una excepción no tratada  en los bloques
      anteriores
END;
```

3.8.1. Excepciones predefinidas

Son aquellas que se disparan automáticamente al producirse determinados errores. Son las más comunes y entre ellas podemos destacar.

TOO_MANY_ROWS	Se produce cuando un select into devuelve más de una fila.
NO_DATA_FOUND	Se produce cuando un select into no devuelve ninguna fila.
LOGIN_DENIED	Error cuando intentamos conectar a Oracle con un login y clave no válidos.
NOT_LOGGED_ON	Se produce cuando intentamos acceder a la base de datos sin estar conectados.

PROGRAM_ERROR	Se produce cuando hay un problema interno en la ejecución del programa.
VALUE_ERROR	Se produce cuando hay un error aritmético o de conversión.
ZERO_DIVIDE	Cuando hay división entre 0.
DUPVAL_ON_INDEX	Se crea cuando se intenta almacenar un valor que crearía duplicados en la clave primaria o en una columna con restricción UNIQUE.
INVALID_NUMBER	Se produce cuando se intenta convertir una cadena a un valor numérico.
INVALID_CURSOR	Se intenta realizar una operación no permitida sobre un cursor.

3.8.2. Excepciones definidas por el usuario

PL/SQL permite a un usuario definir sus propias excepciones, las que deberán ser declaradas y lanzadas utilizando la sentencia RAISE. Estas excepciones se usan para controlar errores definidos por el programador.

Las excepciones deben ser declaradas como cualquier otra variable en el segmento DECLARE de un bloque, subprograma o paquete y se le asigna el tipo EXCEPTION.

```
DECLARE
    -- Declaraciones
    <nombre_excepción> EXCEPTION;
```

Con la instrucción RAISE se lanzan en la parte de instrucciones del programa con la siguiente orden:

```
RAISE <nombre_excepcion>;
```

Se tratan en la sección de EXCEPTION del mismo modo y formato que las vistas anteriormente.

```
WHEN <nombre_excepcion> THEN <tratamiento>;
```

Ejemplo:

En el siguiente ejemplo se pregunta un edad por la consola, si esta edad tiene un valor negativo se lanza una excepción con el comando RAISE llamada EDAD_ERRONEA que nos informa de que la edad no puede ser negativa. En caso contrario nos informa de la edad introducida por el teclado.

```
SET SERVEROUTPUT ON;
DECLARE
   EDAD_ERRONEA EXCEPTION;
   edad NUMBER :=&edad;
BEGIN
 IF edad < 0 THEN
    RAISE EDAD_ERRONEA;
 END IF;
 dbms_output.put_line('Tienes ' || edad || ' años');
EXCEPTION
 WHEN EDAD_ERRONEA THEN
   dbms_output.put_line('La edad no puede ser negativa');
END;
/
```

3.8.3. Uso de SQLCODE y SQLERRM

En oracle existen errores internos que no tienen asignados ninguna excepción. Con las funciones predefinidas SQLCode y SQLERRM se puede aclarar al usuario el error generado.

- SQLcode devuelve el número del error de Oracle y un 0 (cero) en caso de éxito al ejecutarse una sentencia SQL.
- SQLERRM devuelve el correspondiente mensaje de error.

Estas funciones son muy útiles cuando se utilizan en el bloque de excepciones OTHERS, para aclarar el significado de la excepción.

Ejemplo:

```
DECLARE
  num_error NUMBER;
  msg_error VARCHAR2(255);
  resultado  NUMBER;
BEGIN
  resultado:=1/0;
EXCEPTION
  WHEN OTHERS THEN
    num_error := SQLCODE;
    msg_error := SQLERRM;
    DBMS_OUTPUT.put_line('Error:'||TO_CHAR(num_error));
    DBMS_OUTPUT.put_line(msg_error);
END;
```

```
Error:-1476
ORA-01476: el divisor es igual a cero

Procedimiento PL/SQL terminado correctamente.
```

Recursos y enlaces

- Lenguaje PL/SQL

- Tipos de datos

Conceptos clave

- **PL/SQL**: Lenguaje de programación de base de datos.
- **Bloque anónimo**: Código que se ejecuta y no se almacena en la base de datos.
- **Excepciones:** Permiten controlar los errores en tiempo de ejecución.

Test de autoevaluación

Los bloques anónimos en PL/SQL sirven para:

a) Almacenar procedimientos y funciones en la BD.
b) Ejecutar triggers o disparadores.
c) Ejecutar código que no se almacena en la base de datos.
d) Controlar los errores.

Para declarar una variable del mismo tipo que el campo de una tabla utilizamos:

a) %TYPE
b) %FIELD
c) %ROWTYPE
d) El mismo tipo de dato

Para declarar una constante precio con el valor 200 utilizamos:

a) CONSTANT number(6,2) precio:=200;
b) precio CONSTANT number(6,2):=200;
c) DECLARE precio number(6,2):=200;
d) precio number(6,2) VALUE 200;

Para escribir una línea en la consola utilizamos la orden:

a) print
b) echo
c) dbms_output.put_line
d) print_line

¿Cuál de las siguientes no es un tipo de bucle en PL/SQL?

a) LOOP .. END LOOP
b) WHILE.. LOOP
c) FOR LOOP
d) DO .. WHILE

Para definir un usuario sus propias excepciones utiliza la orden:

a) SQLCODE
b) RAISE
c) EXCEPTION
d) SQLERRM

Los modos de un parámetro de un procedimiento pueden ser:

a) IN
b) READ
c) WRITE
d) OUT

Ponlo en práctica

Actividad 1

Utilizando la tabla de vehículos. Crea una función llamada obtenerprecio a la que le pasamos la matrícula de un vehículo y nos retorna su precio.

SOLUCIONARIOS

Test de autoevaluación

Los bloques anónimos en PL/SQL sirven para:

e) Almacenar procedimientos y funciones en la BD.

f) Ejecutar triggers o disparadores.

g) Ejecutar código que no se almacena en la base de datos.

h) Controlar los errores.

Para declarar una variable del mismo tipo que el campo de una tabla utilizamos:

e) %TYPE

f) %FIELD

g) %ROWTYPE

h) El mismo tipo de dato

Para declarar una constante precio con el valor 200 utilizamos:

e) CONSTANT number(6,2) precio:=200;

f) precio CONSTANT number(6,2):=200;

g) DECLARE precio number(6,2):=200;

h) precio number(6,2) VALUE 200;

Para escribir una línea en la consola utilizamos la orden:

e) print

f) echo

g) dbms_output.put_line

h) print_line

¿Cuál de las siguientes no es un tipo de bucle en PL/SQL?

e) LOOP .. END LOOP
f) WHILE.. LOOP
g) FOR LOOP
h) DO .. WHILE

Para definir un usuario sus propias excepciones utiliza la orden:

e) SQLCODE
f) RAISE
g) EXCEPTION
h) SQLERRM

Los modos de un parámetro de un procedimiento pueden ser:

e) IN
f) READ
g) WRITE
h) OUT

Ponlo en práctica

Actividad 1

Utilizando la tabla de vehículos. Crea una función llamada obtenerprecio a la que le pasamos la matrícula de un vehículo y nos retorna su precio.

Solución:

```
SET SERVEROUTPUT ON;
SET VERIFY OFF;
DECLARE
 precio tvehiculos.precio%TYPE;
BEGIN
 precio:=obtenerprecio('1234-EFG');
 dbms_output.put_line('El precio es: ' || precio);
END;
/

CREATE OR REPLACE FUNCTION obtenerprecio
(v_matricula tvehiculos.matricula%TYPE) RETURN number
IS
   v_precio  tvehiculos.precio%TYPE:=0;
BEGIN
 select precio into v_precio from tvehiculos where matricula=v_matricula;
 return v_precio;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
   return 0;
END;
/
```