TEMA 7. PL/SQL

1. INTRODUCCIÓN

Una de las mayores limitaciones que nos encontramos con el lenguaje SQL es la imposibilidad de tratar de forma independiente las filas devueltas por una consulta. Esto se consigue con PL/SQL.

PL/SQL es un lenguaje de programación procedural estructurado en bloques que amplía el lenguaje estándar SQL, uniendo la potencia de éste, con la capacidad de los lenguajes de programación tradicionales. De esta forma, PL/SQL no sólo nos permite manipular los datos de la base de datos, sino que dispone de técnicas procedurales como los bucles o el control condicional.

También nos permite controlar los errores que se pudieran producir (excepciones), como por ejemplo que una consulta no devuelva filas. Existen errores propios (excepciones predefinidas) y errores que puede provocar y definir el usuario.

Aunque el lenguaje PL/SQL fue creado por Oracle, hoy día todos los gestores de bases de datos utilizan un lenguaje procedimental muy parecido al ideado por Oracle para poder programar las bases de datos.

2. VENTAJAS

Además de la capacidad procedural, el soporte a SQL, PL/SQL presenta las siguientes ventajas:

- **Mejora del rendimiento en entornos de red cliente/servidor**: ya que permite mandar bloques PL/SQL desde el cliente al servidor a través de la red, reduciendo de esta forma el tráfico y así no tener que mandar una a una las sentencias SQL correspondientes.

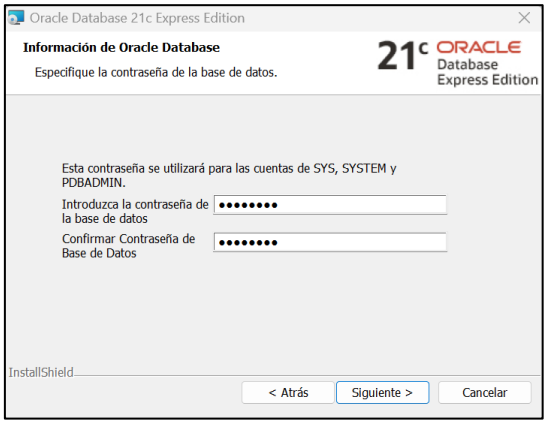
- **Portabilidad**: las aplicaciones escritas con PL/SQL son portables a cualquier sistema operativo y plataforma en la cual se encuentre corriendo el SGBD en la que se hayan creado. Esto significa que se pueden codificar librerías que podrán ser reutilizadas en otros entornos.

- **Seguridad:** los procedimientos almacenados habilitan la división lógica entre cliente y servidor, de forma que se previene que se manipulen los datos desde el cliente. Además permite a los usuarios ejecutar solo aquellos procedimientos para los cuales tengan privilegios.

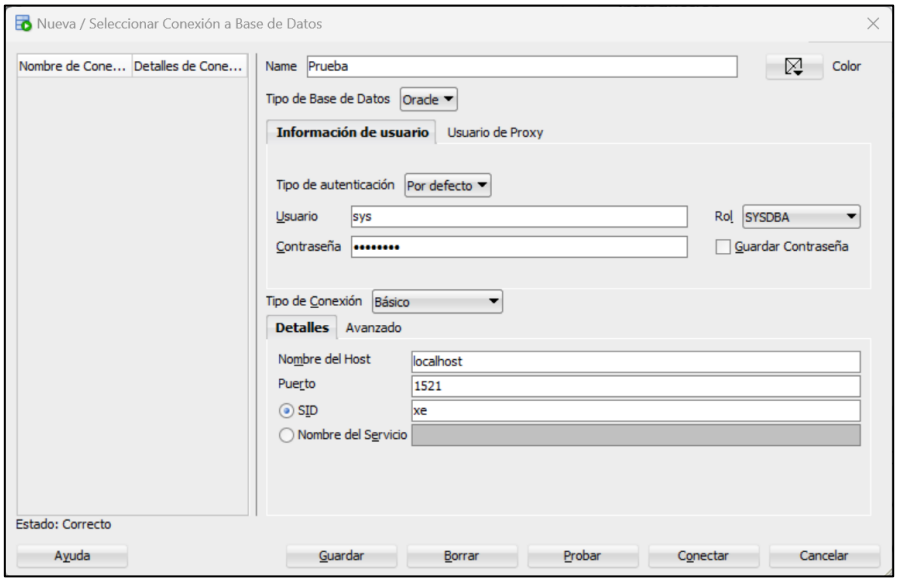
3. PRIMEROS PASOS

3.1. Crear una conexión

Los usuarios por defecto creados al instalar Oracle son SYS, SYSTEM Y PDBADMIN.

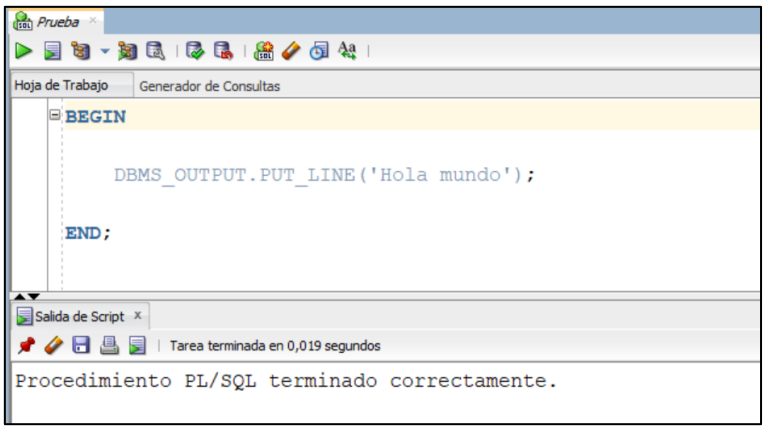


Por lo tanto, serán los que usemos cuando creemos nuestra primera conexión.

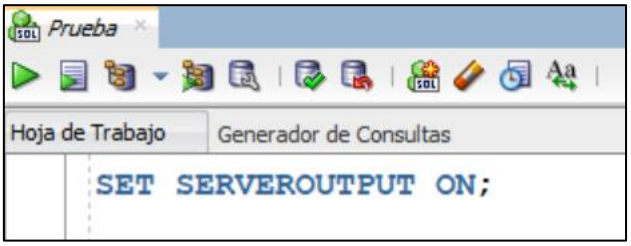


3.2. Primer programa

La primera vez que ejecutemos un script en Sqldeveloper, veremos que se ejecuta pero no me muestra nada en pantalla.



Eso es porque la salida no está redirigida hacia la salida estándar. Por lo tanto, para poder visualizar los resultados de la ejecución del script tenemos que redirigir la salida a la salida estándar mediante la siguiente sentencia:



Comando SET SERVEROUTPUT	
Especifica si la salida del almacenamiento intermedio de mensajes DBMS_OUTPUT se redirige a una salida estándar.	
Parámetros	
ON	Establece que los mensajes del almacenamiento intermedio de mensajes se redirijan a la salida estándar.
OFF	Establece que los mensajes del almacenamiento intermedio de mensajes no se redirijan a la salida estándar.

3.3. Lectura y Escritura

Para mostrar información en teclado, utilizaremos la siguiente sentencia:

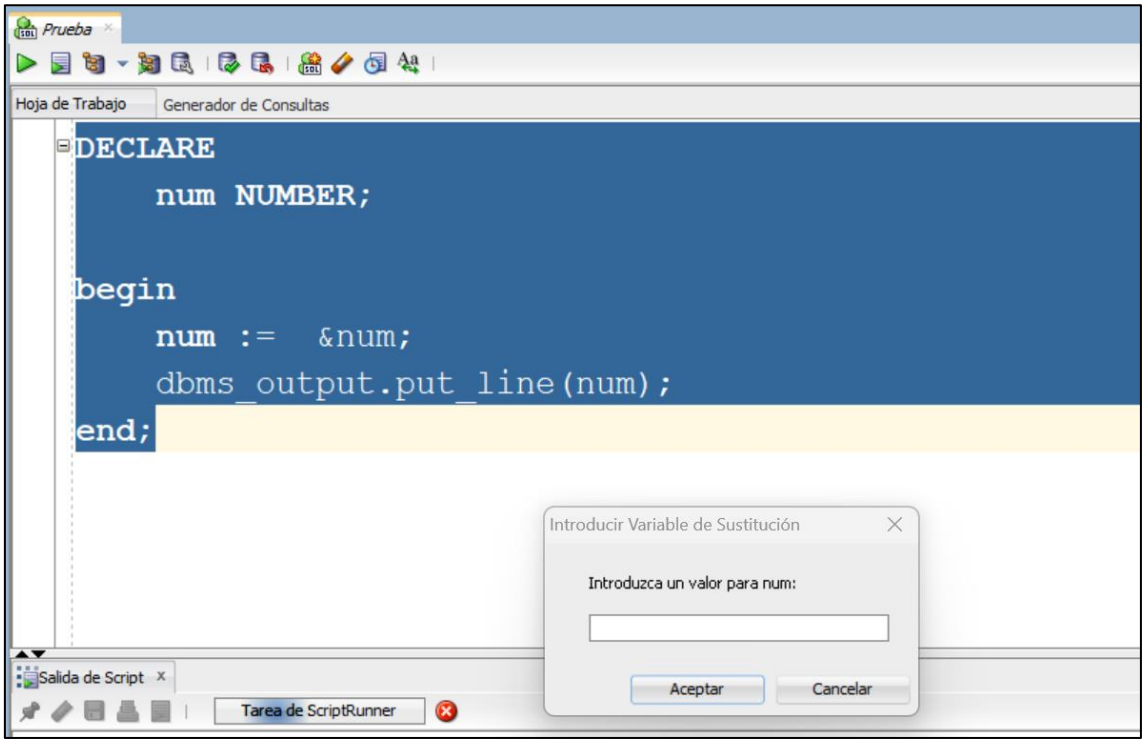
```
DBMS_OUTPUT.PUT_LINE('texto');
```

Para concatenar valores, se utiliza el símbolo ||:

```
begin
    num :=  &num;
    dbms_output.put_line('Num: ' || num);
end;
```

En PL/SQL para leer datos desde teclado se usan las **variables de sustitución**. En el código, estas variables van precedidas del símbolo **&**. El nombre de la variable de sustitución aparecerá en el prompt para que sepamos qué datos vamos a introducir.

```
variable := &variable de sustitución;
```



4. TIPOS DE DATOS

4.1. Tipos de Datos Simples

En PL/SQL contamos con todos los tipos de datos simples utilizados en SQL y algunos más. Vamos a ver los más utilizados.

Numéricos

NUMBER

Numérico. PL/SQL tiene predefinidos los siguientes subtipos:

NUMBER	(General)
DEC **DECIMAL** **NUMERIC**	Números con punto fijo con precisión máxima de 38 dígitos decimales
DOUBLE PRECISION **FLOAT**	Números con punto flotante con precisión máxima de 38 dígitos decimales
REAL	Números con punto flotante con precisión máxima de 18 dígitos decimales
INTEGER **INT** **SMALLINT**	Números enteros con una precisión máxima de 38 dígitos

BINARY_INTEGER

Binario con desbordamiento a number. Usado para almacenar enteros con signo. PL/SQL tiene predefinidos los siguientes subtipos de binary_integer:

BYNARY_INTEGER	(General)
NATURAL	No negativo.
NATURALN	No negativo, no admite nulos.
POSITIVE	Positivo.
POSITIVEN	Positivo, no admite nulos.
SIGNTYPE	-1, 0 y 1, usado en lógica trivaluada.

PLS_INTEGER

Binario sin desbordamiento. Es más actual que su equivalente binary_integer, el cual se mantiene por razones de compatibilidad.

Resumen

Tipo	Subtipos
NUMBER	DEC
	DECIMAL
	NUMERIC
	DOUBLE PRECISION
	FLOAT
	REAL
	INTEGER
	INT
	SMALLINT
BINARY_INTEGER	NATURAL

	NATURALN
	POSITIVE
	POSITIVEN
	SIGNTYPE
PLS_INTEGER	

Alfanuméricos

CHAR	Array de n caracteres. Máximo 2000 bytes. Si no especificamos la longitud sería 1.
LONG	Array de caracteres con un máximo de 32760 bytes
RAW	Array de bytes con un máximo de 2000
LONG RAW	Array de bytes con un máximo de 32760
VARCHAR 2	Almacena cadenas de longitud variable con un máximo de 32760

Otros

BOOLEAN

Solo pueden tomar los valores true/false.

DATE

Almacena valores día/hora desde el 1 de enero de 4712 ac hasta el 31 de diciembre de 4712 dc.

4.1.1. Conversiones

Además de las conversiones de tipo de datos realizadas explícitamente por las funciones de conversión (`TO_CHAR, TO_NUMBER`…), cuando se hace necesario, PL/SQL puede convertir un tipo de dato a otro en forma implícita. Esto significa que la interpretación que se dará a algún dato será la que mejor se adecue dependiendo del contexto en que se encuentre, por ejemplo cuando variables de tipo char se operan matemáticamente para obtener un resultado numérico.

Si PL/SQL no puede decidir a qué tipos de dato de destino puede convertir una variable se generará un **error de compilación**.

Hasta	**BIN_INT**	**CHAR**	**DATE**	**LONG**	**NUMBER**	**PLS_INT**	**RAW**	**ROWID**	**VARCHAR2**
Desde									
BIN_INT		X		X	X	X			X
CHAR	X		X	X	X	X	X	X	X
DATE		X		X					X
LONG		X					X		X
NUMBER	X	X		X		X			X
PLS_INT	X	X		X	X				X
RAW		X		X					X

ROWID		X							X
VARCHAR2	X	X	X	X	X	X	X	X	

4.1.2. Delimitadores

PL/SQL tiene un conjunto de símbolos denominados delimitadores utilizados para representar operaciones entre tipos de datos, delimitar comentarios, etc. A continuación podemos ver un resumen de los mismos.

Delimitadores Simples		**Delimitadores Compuestos**	
Símbolo	**Significado**	**Símbolo**	**Significado**
+	Suma	**	Exponenciación
-	Resta / negación	:=	Asignación
*	Producto	<>	Distinto
/	División	!=	Distinto
=	Igual relacional	<=	Menor o igual
<	Menor	>=	Mayor o igual
>	Mayor	..	Rango
(	Delimitador de lista	||	Concatenación
)	Delimitador de lista	<<	Delimitador de etiquetas
:	Variable host	>>	Delimitador de etiquetas
,	Separador de elementos	--	Comentario de una línea
.	Selector	/*	Comentario de varias líneas
%	Indicador de atributo	*/	Comentario de varias líneas
“	Delimitador de identificador acotado	=>	Selector de nombre de parámetro
@	Indicador de acceso remoto		
;	Terminador de sentencias		

4.2. Tipos de Datos Compuestos

4.2.1. Cursores

Los cursores son un tipo de datos cuyo uso es exclusivo en la programación de las bases de datos. Un cursor es una estructura que almacena el conjunto de filas devuelto por una consulta a la base de datos.

Oracle tiene dos tipos de cursores:

- **Implícitos:** PL/SQL declara implícitamente un cursor para todas las sentencias SQL de manipulación de datos, incluyendo consultas que devuelven una sola fila.
- **Explícitos:** Para las consultas que devuelven más de una fila, se debe declarar explícitamente un cursor para procesarlas individualmente.

Por otra parte, los podemos clasificar en:

- **Estáticos:** se declaran en la DECLARE con su cláusula SELECT asociada, y se abren, ejecutan y cierran en la zona BEGIN. Pueden ser simples o parametrizados.
- **Dinámicos:** la sentencia SELECT asociada al cursor no se especifica en la zona DECLARE, sino en BEGIN, con lo que el resultado de su ejecución es dinámico. Según se declare o no el registro que recibirá los datos de la SELECT, los cursores dinámicos pueden ser prefijados o no prefijados.

4.2.2. Registros

Un registro es un grupo de elementos relacionados almacenados en campos, cada uno de los cuales tiene su propio nombre y tipo de datos.

Las variables de este tipo de datos pueden recibir la información de un cursor.

Se declaran mediante la siguiente sintaxis:

```
TYPE nombre_registro IS RECORD(
      campo1 tabla.columna%TYPE,
      …
      );
```

El tipo de datos de los campos se puede realizar a mano, siguiendo la estructura de la definición de los campos de las tablas en SQL, pero es mucho más cómodo hacer uso de TYPE o ROWTYPE.

```
TYPE reg is RECORD(
        empleado temple.nomem%TYPE,
        salario  temple.salar%TYPE
    );
r_empleado reg;
r_empleado2 temple%ROWTYPE;
```

5. EL BLOQUE PL/SQL

La unidad básica que vamos a utilizar en PL/SQL es el bloque.

Un bloque PL/SQL consta de tres zonas:

- **Declaraciones**: donde se definen las variables, cursores y excepciones de usuario que se necesiten.
- **Proceso**: donde se escriben todas las sentencias ejecutables.
- **Excepciones**: donde se define el control de errores.

```
DECLARE
Declaración de variables y ctes ;
Declaración de cursores ;
Declaración de excepciones ;

BEGIN
Sentencias ejecutables ;

EXCEPTION
Control de excepciones ;
```

```
DECLARE
.........
BEGIN
....
        DECLARE
        BEGIN
        ....
        EXCEPTION
        ....
        END ;
.....

EXCEPTION
............
END ;
```

De estas secciones, tan solo la ejecutable es obligatoria. No es necesario declarar nada en un bloque ni levantar excepciones.

Los bloques PL/SQL **se pueden anidar** tanto en la zona BEGIN como en la EXCEPTION, pero no en la zona DECLARE, que es única para cada bloque. Cada bloque debe acabar con el carácter '/'.

6. ZONA DE DECLARACIÓN

6.1. Declaración de Variables y Constantes

La declaración de variables y constantes sigue la siguiente sintaxis:

```
Identificador [Modificador] Tipo de dato [Inicialización]
```

Donde `Identificador` es el nombre de la variable o la constante.

Donde `Modificador` puede ser:

`CONSTANT`	Palabra reservada para la definición de constantes
`NOT NULL`	Obliga a que siempre tenga un valor. Si restringimos una variable con la palabra `NOT NULL` deberemos asignarle un valor al declararla; de lo contrario, PL/SQL lanzará una excepción `VALUE_ERROR`.

Donde `Tipo de dato` puede ser:

`tipo de dato`	Tipo de dato de la variable
`identificador%TYPE`	Declara la variable o constante con el mismo tipo de dato que una variable definida anteriormente o que una columna de una tabla. `Identificador` es el nombre de la variable PL/SQL definida anteriormente, o el nombre de la tabla y la columna de la base de datos.
`identificador%ROWTYPE`	Declara una variable fila con campos con los mismos nombres y tipos que las columnas de una tabla o de una fila recuperada de un cursor. Al declarar una fila `%ROWTYPE`, no se admite ni la constante, ni la asignación de valores.

Donde `Inicialización` puede ser:

`:= <expresión_plsql / literal`	Asigna a la variable o constante el valor inicial como resultado de una operación o con un literal
`DEFAULT valor`	Asigna a la variable el valor indicado en `valor`

```
DECLARE
 id SMALLINT;
 hoy DATE := sysdate;
 pi CONSTANT REAL := 3.14;
 num1 SMALLINT NOT NULL; -- Generaría un error, ya que no está inicializado
 num2 SMALLINT NOT NULL := 10;
 num3 SMALLINT DEFAULT -5;
```

6.2. Declaración de Registros

Tras declarar el tipo registro, simplemente declaramos una variable de ese tipo.

```
TYPE reg is RECORD(
        empleado temple.nomem%TYPE,
        salario  temple.salar%TYPE
    );
r_empleado reg;
```

6.3. Declaración de Cursores

Como vimos anteriormente, nos podemos encontrar con dos tipos de cursores, los estáticos y los dinámicos, que a su vez podían ser simples o parametrizados.

6.3.1. Estático simple

Un cursor estático simple se declara en el bloque DECLARE con su claúsula SELECT y se abre, ejecuta y cierra en la zona BEGIN.

```
DECLARE
      CURSOR nombre IS Sentencia_SELECT;
```

```
DECLARE
    CURSOR departamento IS
        SELECT numde,nomde FROM tdepto;
```

6.3.2. Estático parametrizado

Los cursores estáticos parametrizados permiten obtener con el mismo cursor diferentes resultados en función del valor que se le pase al parámetro. Su sintaxis es:

```
CURSOR nombre_cursor(nombre_parámetro tipo_parámetro) IS
      sentencia_SELECT_utilizando_los_parámtetros;
```

Los **parámetros son variables únicamente de entrada**, nunca para recuperación de datos. Estas variables son locales para el cursor y solo se referencian en la SELECT.

Si se definen parámetros, estos deben especificarse en la sentencia SELECT y se utilizan igual que si utilizásemos un valor constante. Al abrir el cursor se sustituyen los parámetros por los valores correspondientes.

```
DECLARE
    n NUMBER;
    CURSOR empleados (dept NUMBER) IS
        SELECT numem,nomem
        FROM temple
        WHERE numde = dept;

BEGIN
    OPEN empleados(&n); -- Apertura del cursor y paso del parámetro
```

Si en cualquier cursor quisiéramos modificar las filas que nos devuelve, deberíamos añadir el siguiente código a la cláusula select:

```
FOR UPDATE OF nombre_columna;
```

```
DECLARE
    n NUMBER;
    CURSOR empleados IS
        SELECT numem,nomem
        FROM temple
        FOR UPDATE OF nomem;
```

6.3.3. Dinámicos

Los cursores dinámicos son aquellos en los que la sentencia SELECT no aparece en la zona DECLARE. Este tipo de cursores solo se pueden definir dentro de paquetes y no están soportados en rutinas, triggers o bloques anónimos. Por ello, no van a ser objeto de estudio este curso.

6.4. Declaración de excepciones

Si se van a utilizar excepciones definidas por el usuario, es necesario declararlas en la zona DECLARE. Para crearlas, simplemente escribimos la palabra EXCEPTION detrás del nombre asignado a la excepción.

Sintaxis	Ejemplo
nombre_excepción EXCEPTION	mi_excepcion **EXCEPTION;**

7. ZONA DE PROCESO

En esta zona se escriben todas las sentencias ejecutables. El comienzo del bloque PL/SQL se especifica con la palabra BEGIN. En el bloque se permiten:

- Sentencias propias de PL/SQL.
- Sentencias DML de SQL (SELECT, INSERT, UPDATE, DELETE)
- Sentencias transaccionales (COMMIT, ROLLBACK, SAVEPOINT)

7.1. Sentencias Propias de PL/SQL

Las sentencias propias del lenguaje PL/SQL se agrupan en:

- Asignaciones.
- Control condicional.
- Bucles
- Manejo de cursores
- Sentencias GOTO, EXIT y NULL.

7.1.1. Condicionales

7.1.1.1. Condicional simple: IF-THEN

En esta estructura de control se evalúa la condición y, si es verdadera, se ejecutan las sentencias encerradas en ella.

Su sintaxis es:

Sintaxis	Ejemplo

`IF condición THEN`
`  Sentencias;`
`END IF;`
	`BEGIN`
	`  num := 5;`
	`  IF (num>0) THEN`
	`  DBMS_OUTPUT.PUT_LINE('Mayor que 0');`
	`  END IF;`
	`END;`

7.1.1.2. Condicional doble: IF-THEN-ELSE

Con esta estructura evaluamos una condición. Si se evalúa a verdadero ejecutaremos el primer conjunto de sentencias. Si no, se ejecutará el segundo.

Sintaxis	Ejemplo

`IF condición THEN`
`  Sentencias 1;`
`ELSE`
`  Sentencias 2;`
`END IF;`
	`BEGIN`
	`  num := 5;`
	`  IF (MOD(num,2)=0) THEN`
	`  DBMS_OUTPUT.PUT_LINE('Par');`
	`  ELSE`
	`  DBMS_OUTPUT.PUT_LINE('Impar');`
	`  END IF;`
	`END;`

7.1.1.3. Condicional múltiple: IF-THE-ELSIF

Con esta forma de condicional podemos hacer una selección múltiple, de manera que se ejecutará el bloque de instrucciones de la condición que se cumpla.

Sintaxis	Ejemplo

IF condición THEN
  Sentencias 1;
ELSIF
  Sentencias 2;
…
[ELSE
  Sentencias n;]
END IF;
	BEGIN
	  num := 5;
	  IF (num>0) THEN
	  DBMS_OUTPUT.PUT_LINE('Mayor que cero');
	  ELSIF (num<0) THEN
	  DBMS_OUTPUT.PUT_LINE('Menor que cero');
	  ELSE
	  DBMS_OUTPUT.PUT_LINE('Cero');
	  END IF;
	END;

7.1.2. Bucles

Existen 5 tipos de bucles:

- Bucles básicos (Loop).
- Bucles condicionales (While).
- Bucles numéricos (For).
- Bucles sobre cursores.
- Bucles para un select.

En este apartado, veremos los 3 primeros, y los demás se verán cuando se estudien esos conceptos.

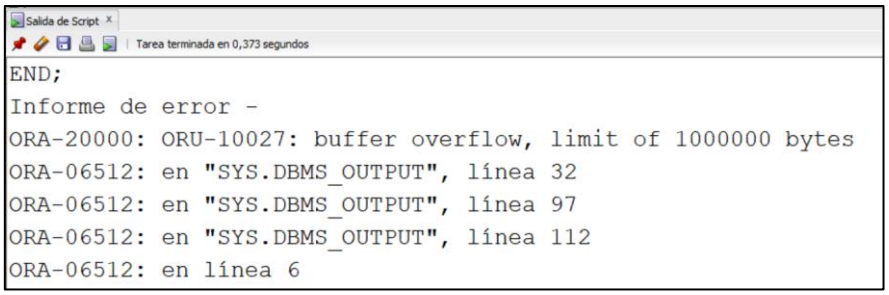
7.1.2.1. Loop

Es la forma más simple de bucle infinito.

Sintaxis	Ejemplo

LOOP
  Sentencias;
END LOOP;
	BEGIN
	  LOOP
	  DBMS_OUTPUT.PUT_LINE('Ejecutando...');
	  END LOOP;
	END;

Si no se sale de ellos de alguna manera, generarán una excepción:



Para salir de ellos, deberemos utilizar los comandos EXIT, GOTO, RAISE o EXIT WHEN *condicion*.

Sintaxis	Ejemplo

LOOP
Sentencias;
END LOOP;
	BEGIN
	i:=0;
	LOOP
	DBMS_OUTPUT.PUT_LINE('Ejecutando...');
	i:=i+1;
	EXIT WHEN i>100;
	END LOOP;
	END;

7.1.2.2. Bucle While

En este tipo de bucles, la condición se evalúa antes de cada entrada al bucle. Si la condición se cumple se ejecutan las sentencias, y si no es así, se para el control a la siguiente sentencia al final del bucle.

Sintaxis	Ejemplo

WHILE condición LOOP
Sentencias;
END LOOP;
	BEGIN
	i:=0;
	WHILE i<100 **LOOP**
	dbms_output.put_line(i);
	i:=i+1;
	END LOOP;
	END;

7.1.2.3. Bucle For

Son bucles que se ejecutan mientras el elemento definido se encuentre dentro del rango numérico.

Sintaxis	Ejemplo

`FOR índice IN [REVERSE] n1..n2 LOOP`
`  Sentencias;`
`END LOOP;`
	`BEGIN`
	`  FOR i IN 1..10 LOOP`
	`  DBMS_OUTPUT.PUT_LINE(i);`
	`  END LOOP;`
	`END;`

7.1.2.4. Bucles sobre cursores

Son bucles que se ejecutan para cada fila que recupera el cursor. Cuando se inicia un bucle sobre un cursor, automáticamente se realizan los siguientes pasos:

- Se declara implícitamente el registro especificado como `nombre_cursor%ROWTYPE`.
- Se abre el cursor.
- Se realiza la lectura y se ejecutan las sentencias del bucle hasta que no hay más filas.
- Se cierra el cursor.

Sintaxis:

```
FOR  nombre_registro IN nombre_cursor LOOP
     Sentencias;
END LOOP;
```

```
DECLARE
    CURSOR curdep IS
        SELECT numde, nomde FROM tdepto;
    -- reg_dept curdep%ROWTYPE;
    -- Esta sentencia sobra, pues es declarada implícitamente por este
    -- tipo de bucle

BEGIN
    FOR reg_dept IN curdep LOOP
        DBMS_OUTPUT.PUT_LINE('Número: ' || reg_dept.numde);
    END LOOP;
END;
```

La variable de control del bucle se incrementa automáticamente y no es necesario inicializarla, pues lo está implícitamente como variable local de tipo `integer`. El ámbito del contador es el bucle y no puede accederse a su valor fuera de él. Dentro del bucle, el contador puede referenciarse como una constante pero no se le puede asignar un valor.

También le podemos pasar parámetros al cursor. En este caso, los parámetros se indican entre paréntesis tras el nombre del cursor:

```
FOR nombre_registro IN nombre_cursor(lista_de_parametros)
```

Si se sale del bucle prematuramente (por ejemplo, con EXIT), o se detecta un error (excepción), el cursor se cierra.

7.1.2.5. Bucles para Sentencias Select

Es el mismo concepto que el del cursor, excepto que en vez de declarar un cursor con su SELECT, se escribe directamente la sentencia SELECT y Oracle utiliza un cursor interno para hacer la declaración.

Su sintaxis es:

```
FOR nombre_registro IN sentencia_SELECT LOOP
     sentencias ;
END LOOP ;
```

```
BEGIN
    FOR reg_dept IN (SELECT numde FROM tdepto) LOOP
        DBMS_OUTPUT.PUT_LINE('Número: ' || reg_dept.numde);
    END LOOP;
END;
```

Si se sale del bucle prematuramente, o se detecta un error, el cursor se cierra. Hay que resaltar que las columnas del cursor no pueden ser usadas con `<registro.columna>` fuera de estos dos últimos bucles, ya que cierran automáticamente el cursor que tratan.

7.1.3. Etiquetas de Control

7.1.3.1. EXIT

Nos permite salir de un bucle de forma radical o evaluando una condición. Su sintaxis es:

```
EXIT [nombre_bucle] [WHEN condición]
```

Si se omite el nombre del bucle sale del bucle actual. Si se especifica WHEN, solo sale si se cumple la condición. Si se omiten ambos, sale del bucle actual incondicionalmente.

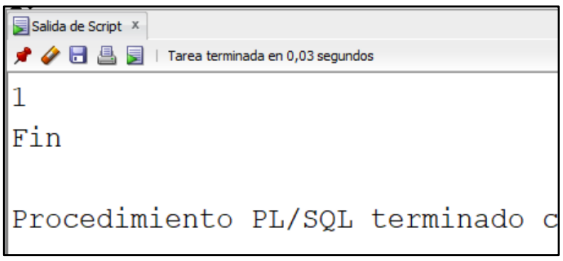
7.1.3.2. GOTO

Esta sentencia transfiere el control a la sentencia o bloque PL/SQL siguiente a la etiqueta indicada. Su sintaxis es:

```
GOTO etiqueta;
```

La etiqueta se especifica entre los símbolos << y >>.

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(1);
    GOTO salir;
    DBMS_OUTPUT.PUT_LINE(2);
    DBMS_OUTPUT.PUT_LINE(3);
    <<salir>>
    DBMS_OUTPUT.PUT_LINE('Fin');
END;
```



7.1.3.3. NULL

Significa inacción. El único objeto que tiene es pasar el control a la siguiente sentencia. Su sintaxis es:

```
NULL
```

```
BEGIN
    i:=0;
    IF i<10 THEN
        i:=i+1;
    ELSE
        NULL;
    END IF;
END;
```

7.1.4. Manejo de cursores

Antes de manejar un cursor, este ha tenido que ser definido en la zona DECLARE.

Para poder realizar una lectura de todas las filas recuperadas en un cursor es necesario realizar un bucle. Existen bucles ya predefinidos para recorrer cursores, aunque también se pueden utilizar bucles simples y hacer el recorrido de forma más controlada.

Para el manejo de los cursores, existen los siguientes **atributos predefinidos**:

%NOTFOUND	Devuelve TRUE si la última lectura falla porque no hay más filas disponibles o FALSE si recupera una fila. Se utiliza para detectar el final y romper el bucle de lectura de un cursor.
%FOUND	Es lo contrario de %NOTFOUND
%ROWCOUNT	Devuelve el número de fila, procesada por el cursor.
%ISOPEN	Devuelve TRUE si el cursor está abierto y FALSE si está cerrado.

Para manejar un cursor primero hay que abrirlo (OPEN), luego leerlo (FETCH) y por último cerrarlo (CLOSE).

Abrir el cursor: OPEN

La sentencia OPEN, evalúa la sentencia SELECT asociada al cursor y lo prepara para permitir su lectura. Abre el cursor y ejecuta la consulta asociada a dicho cursor.

Tipo de cursor	Sintaxis
cursor estático simple	OPEN nombre_cursor ;
cursor estático parametrizado	OPEN nombre_cursor (parm1 [,parm2] ..) ;
cursor dinámico	OPEN nombre_cursor FOR sentencia_select ;

Los parámetros se pueden pasar posicionalmente: el primer parámetro sustituye al primero que espera el cursor y así sucesivamente, o por asociación de nombres:

```
DECLARE
    CURSOR departamentos (dep_pl NUMBER, t_director_pl CHAR) IS
        SELECT numde, nomde, tidir from tdepto WHERE numde = dep_pl AND tidir = t_director_pl;

BEGIN
    OPEN departamentos(10,'F'); -- por orden
    OPEN departamentos(t_director_pl=>'F',dep_pl=>10); -- por asociación de nombres
```

Leer el Cursor: FETCH

La sentencia FETCH, recupera la siguiente fila del cursor hasta detectar el final. Los datos recuperados deben almacenarse en variables o registros.

- ✓ Sintaxis para recuperación en variables:

```
FETCH  nombre_cursor  INTO var1 [,var2] ;
```

Todas las variables que aparecen en la cláusula INTO deben haber sido definidas previamente en la zona de declaración de variables DECLARE.

Será necesario tener tantas variables como columnas estemos recuperando en la SELECT asociada al cursor. La recuperación es posicional, es decir la primera columna seleccionada se almacenará en la primera variable, la segunda en la segunda, y así sucesivamente, por lo tanto, los tipos de las variables deberán ser compatibles con los valores de las columnas que van a almacenar.

- ✓ Sintaxis para recuperación en registro:

```
FETCH nombre_cursor INTO nombre_registro ;
```

Es la sintaxis más usada debido a la facilidad para declarar el registro en la zona DECLARE con la opción de atributo `%ROWTYPE<nombre_tabla>/<nombre_cursor>`.

En la zona BEGIN, y mientras el cursor esté abierto, las columnas del cursor pueden utilizarse haciendo referencia al nombre del registro, del que son variables miembro. En el supuesto del párrafo anterior, y caso de que la select asociada al cursor tenga columnas con expresiones distintas a nombres de columna (SUM(salar), por ejemplo), deberán usarse alias de columna, por cuyo nombre serán referenciadas.

```
DECLARE
    CURSOR curdep IS
        SELECT numde, nomde, presu FROM tdepto;
    reg_dept curdep%ROWTYPE;

BEGIN
    OPEN curdep;
    FETCH curdep INTO reg_dept;
```

Cerrar el Cursor: CLOSE

La sentencia CLOSE cierra el cursor. Una vez cerrado no se puede volver a leer (FETCH), pero si se puede volver a abrir. Cualquier operación que se intenta realizar con un cursor que está cerrado provoca la excepción predefinida `INVALID_CURSOR`. Su sintaxis es:

Sintaxis	Ejemplo
`CLOSE nombre_cursor`	`CLOSE curdep;`

7.2. Sentencias DML

Oracle abre un cursor implícito por cada sentencia SQL que tenga que procesar. PL/SQL nos permite referimos al cursor implícito más reciente como "SQL%". Estos cursores los maneja Oracle, por lo que no se pueden abrir, leer o cerrar, pero sí podemos utilizar algunos atributos que nos dan información sobre la ejecución de la sentencia SQL (INSERT, UPDATE, DELETE, SELECT) para la que se haya abierto. Estos atributos son:

%NOTFOUND	Devuelve TRUE si un INSERT, UPDATE o DELETE no ha procesado ninguna fila o si la sentencia SELECT no recupera nada. En este último caso se provoca la excepción predefinida `NO_DATA_FOUND`
%FOUND	Devuelve TRUE si un INSERT, UPDATE o DELETE ha procesado alguna fila o una sentencia SELECT recupera datos
%ROWCOUNT	Devuelve el número de filas procesadas por un INSERT, UPDATE o DELETE o las recuperadas por una SELECT

7.2.1. SELECT

La sentencia SELECT recupera valores de la Base de Datos que serán almacenados en variables PL/SQL.

Para poder recuperar valores en variables, la consulta sólo debe devolver una fila (no funciona con predicados de grupo). Su sintaxis es:

```
SELECT {* / col1 [,coI2] } INTO {reg / var1 [,var2] }
FROM tabla
resto_de_la_select ;
```

Las excepciones predefinidas más comunes son:

A.- Si la consulta no devuelve ninguna fila:

Se produce la excepción predefinida `NO_DATA_FOUND`
El error Oracle `(SQLCODE) ORA-01403`
El mensaje de error `(SQLERRM)` "*no data found*"
`SQL%NOTFOUND` es TRUE
`SQL%FOUND` es FALSE
SQL%ROWCOUNT es 0

B.- Si la consulta devuelve más de una fila:

Se produce la excepción predefinida `TOO_MANY _ROWS`
El error Oracle `(SQLCODE) ORA-O1422`
El mensaje de error `(SQLEERM)` "*single-row query returns more than one row*"
`SQL%NOTFOUND` es FALSE
`SQL%FOUND` es TRUE
`SQL%ROWCOUNT` es 1 (solo puede devolver 0 o 1 fila)

7.2.2. INSERT

La sentencia INSERT crea filas en la tabla o vista especificada. Su sintaxis es idéntica a la de SQL.

Se puede realizar el INSERT con valores de las variables PL/SQL o con los datos de una SELECT. En este caso hay que tener en cuenta que esta última no lleva cláusula INTO.

En ambos casos, el número de columnas a insertar deberá ser igual al número de valores y/o variables o a las columnas especificadas en la SELECT. En caso de omitir las columnas de la tabla en la que vamos a insertar, se deberán especificar valores para todas las columnas de la tabla y en el mismo orden en el que esta haya sido creada.

Su sintaxis es la misma de SQL:

```
INSERT INTO tabla [ (col1 [,col2] .....) ]
{ VALUES (exp1 [,exp2]..... ) /
   Sentencia_Select } ;
```

Donde

VALUES permite insertar una tupla indicando expresiones constantes o variables.

Sentencia_Select permite insertar varias tuplas a la vez.

```
DECLARE
    num NUMBER(5) := 140;
    nombre tdepto.nomde%TYPE := 'RRHH';

BEGIN
    INSERT INTO tdepto (numde,nomde) VALUES (num,nombre);
END;
```

Si no se inserta ninguna fila, los atributos devuelven los siguientes valores:

`SQL%NOTFOUND` es TRUE
`SQL%FOUND` es FALSE
`SQL%ROWCOUNT` es 0

Si se dan de alta una o más filas, los atributos devuelven los siguientes valores:

`SQL%NOTFOUND` es FALSE
`SQL%FOUND` es TRUE
`SQL%ROWCOUNT` es número de filas insertadas

7.2.3. UPDATE

La sentencia UPDA TE permite modificar los datos almacenados en las tablas o vistas.

Sintaxis:

```
UPDATE tabla
SET columna = {sentencia select /
               expresión_plsql /
               constante /
               variable_plsql}
[ WHERE {condición  / CURRENT OF nombre_cursor} ] ;
```

```
DECLARE
    num NUMBER(5) := 140;
    nombre tdepto.nomde%TYPE := 'RRHH';

BEGIN
    UPDATE temple SET salar = salar*0.1 WHERE numde = 100;
END;
```

La cláusula `WHERE CURRENT OF` debe ser utilizada después de haber realizado la lectura del cursor que debe haber sido definido con la opción `FOR UPDATE OF`. Esta cláusula no es admitida en cursores cuya select utilice varias tablas.

Si no se actualiza ninguna fila, los atributos devuelven los siguientes valores:

`SQL%NOTFOUND` es TRUE
`SQL%FOUND` es FALSE
`SQL %ROWCOUNT` es 0

Si se actualizan una o más filas, los atributos devuelven los siguientes valores:

`SQL%NOTFOUND` es FALSE
`SQL%FOUND` es TRUE
`SQL%ROWCOUNT` el número de filas actualizadas

7.2.4. DELETE

La sentencia DELETE permite borrar filas de la tabla o vista especificada.

Sintaxis:

```
DELETE [FROM] tabla
[ WHERE {condición / CURRENT OF nombre_cursor} ] ;
```

```
BEGIN
    DELETE FROM temple WHERE numde = 100;
END;
```

Al igual que con UPDATE, la cláusula `WHERE CURRENT OF` debe ser utilizada después de haber leído el cursor que debe haber sido definido con la opción `FOR UPDATE OF`.

Si no se borra ninguna fila, los atributos devuelven los siguientes valores:

`SQL%NOTFOUND` es TRUE
`SQL%FOUND` es FALSE
`SQL%ROWCOUNT` es 0

Si se borran una o más filas, los atributos devuelven los siguientes valores:

`SQL%NOTFOUND` es FALSE
`SQL%FOUND` es TRUE
`SQL%ROWCOUNT` es número de filas borradas

7.2.5. Sentencias Transaccionales

Una transacción o Unidad Lógica de Trabajo (ULT) es una secuencia de operaciones de actualización que forman un todo, de forma que o se ejecutan todas o no se ejecuta ninguna, debiendo dejar la base de datos en estado coherente.

Una transacción comienza en la primera sentencia SQL tras una sentencia COMMIT, una sentencia ROLLBACK o una conexión a la base de datos.

Una transacción termina con una sentencia COMMIT, una sentencia ROLLBACK o una desconexión, intencionada o no, a la base de datos. El SGBD realiza un COMMIT implícito antes de ejecutar cualquier sentencia de LDD (create, alter, ...) o al realizar una desconexión que no haya sido precedida de un error.

A continuación veremos las sentencias SQL que permiten gestionar explícitamente las transacciones

7.2.5.1. SAVEPOINT.

Los puntos de salvaguarda son marcas que va poniendo el usuario durante la transacción. Estas marcas permiten deshacer los cambios por partes en vez de deshacer toda la transacción.

Sintaxis:

```
SAVEPOINT  <punto_de_salvaguarda> ;
```

Los nombres de los puntos de salvaguarda pueden reutilizarse durante la transacción. Al reutilizarlo el anterior punto se pierde.

Al ejecutar un `ROLLBACK` sin parámetros o un `COMMIT`, se eliminan todos los puntos de salvaguarda. `ROLLBACK TO <punto_de_salvaguarda>` hace que solo se borren los puntos posteriores al indicado. Así, por ejemplo:

```
INSERT INTO ...
SAVEPOINT A;
DELETE ...
ROLLBACK TO SAVEPOINT A;
```

deshace solo el borrado, permaneciendo pendiente la inserción realizada con INSERT .

7.2.5.2. COMMIT.

Señala el final de una transacción y el principio de otra, indicándole al sistema que se deben validar los cambios que se produjeron desde el principio de la transacción que se da por concluida, haciéndolos visibles para los demás usuarios.

Su sintaxis es:

```
COMMIT [WORK] ;
```

La palabra reservada `WORK` es opcional, y no tiene ninguna trascendencia.

Al hacer un `COMMIT`, se liberan todos los `SAVEPOINT` indicados hasta el momento.

Si se hace el `COMMIT` cuando tenemos abierto un cursor declarado con la opción `FOR UPDATE OF`, el siguiente `FETCH` provoca un error, por lo que se debe cerrar el cursor.

7.2.5.3. ROLLBACK.

Señala el final de una transacción y el principio de otra, indicándole al sistema que se deben restaurar el estado de la base de datos tal y como estaba al comenzar la transacción, es decir, deshace todos los cambios pendientes de validación de la transacción actual.

Su sintaxis es:

```
ROLLBACK  [WORK]  [TO SAVEPOINT punto_salvaguarda] ;
```

`WORK` es opcional, y no tiene ninguna trascendencia.

`TO SAVEPOINT punto_salvaguarda` deshace sólo los cambios efectuados desde el punto de salvaguarda indicado.

7.2.5.4. SET TRANSACTION READ ONLY.

Permite establecer una transacción de solo lectura.

Sintaxis:

```
SET TRANSACTION READ ONLY ;
```

Una vez especificado este comando, las siguientes consultas que se realicen solo verán los cambios efectuados antes de que comenzase la transacción.

Oracle SQL Developer dispone de un comando para controlar las validaciones de forma automática. Su sintaxis es:

```
SET AUTOCOMMIT ON/OFF
```

Si se especifica ON los cambios se validarán automáticamente después de cada operación de actualización de datos. La opción por defecto es OFF y permite que sea el usuario el que controle la validación de los cambios con los comandos anteriores.

Las sentencias del LMD y la desconexión de Oracle llevan implícita una validación (commit). Si se produce la caída del sistema o la terminación anormal de una aplicación, se llevará a cabo una restauración automática, mediante la consulta al fichero diario o log.

Oracle almacena temporalmente, en los llamados segmentos de rollback, la información necesaria para deshacer los cambios si se ejecuta un `ROLLBACK` y dejar la información en estado consistente.

Puede comprobarse que el propietario de la tabla modificada, tiene constancia al instante de las modificaciones que se produzcan. Bastará hacer una consulta y observar el nuevo valor.

8. EXCEPCIONES

8.1. Conceptos Generales

La zona de excepciones es la última parte del bloque PL/SQL y en ella se realiza la

gestión y el control de errores.

En PL/SQL, cualquier situación de error es llamada excepción. Cuando se detecta un error en la ejecución del código, se para la ejecución normal del programa y el control se transfiere a la parte de manejo de excepciones. 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 en forma separada de la lógica del negocio.

Las excepciones se pueden producir de dos formas:

Automáticamente Oracle detecta un error, para el proceso y pasa el control a la zona EXCEPTION, buscando si existe tratamiento al error detectado.

Manualmente En el proceso llega un punto en el que nos interesa realizar lo mismo que si se hubiese detectado un error y ejecutamos la excepción. Este tipo de excepciones deberán definirse y lanzarse explícitamente.

La zona EXCEPTION es opcional y se puede omitir. En este caso, si se detecta algún error, el bloque PL/SQL termina incorrectamente, terminando el proceso y devolviendo el control al punto de partida.

La sintaxis para el control de excepciones es el siguiente:

```
EXCEPTION
      WHEN nombre_exception THEN
            Sentencias;
      …
      WHEN OTHERS THEN
            Sentencias;
END;
```

Las excepciones de usuario tendremos que definirlas en el bloque `DECLARE`. Para lanzarlas, se utilizará la sentencia `RAISE`.

La parte `OTHERS` recoge cualquier excepción no capturada.

```
DECLARE
    sueldo NUMBER;
    sueldo_erroneo EXCEPTION;

BEGIN
    sueldo := &sueldo;
    IF sueldo<0 THEN
        RAISE sueldo_erroneo;
    END IF;
EXCEPTION
    WHEN sueldo_erroneo THEN
        DBMS_OUTPUT.PUT_LINE('El sueldo no puede ser negativo');
END;
```

Existen dos variables para poder recuperar los errores Oracle que se pueden producir: SQLCODE y SQLERRM.

8.2. Control de Errores

Cuando manejamos una excepción no podemos continuar por la siguiente sentencia a la que se lanzó. Si por algún motivo nos interesara hacerlo, lo que podemos hacer en encerrar la sentencia dentro de un bloque y ahí capturar las posibles excepciones, para continuar con las siguientes sentencias.

```
BEGIN
    sueldo := &sueldo;
    IF sueldo<0 THEN
        RAISE sueldo_erroneo;
    END IF;

    comision := &comision;
    BEGIN
        IF comision<0 THEN
            RAISE comision_erronea;
        END IF;
    EXCEPTION
        WHEN comision_erronea THEN
            dbms_output.put_line('Comisión errónea.');
            comision := 0;
    END;

    sueldo_total := sueldo + comision;
    dbms_output.put_line(sueldo_total);

EXCEPTION
    WHEN sueldo_erroneo THEN
```

8.3. Excepciones Predefinidas

En Oracle existen las siguientes excepciones predefinidas:

Nombre de excepción	Tipo de error indicado	SQLCODE
CURSOR_ALREADY_OPEN	Cursor abierto previamente	-6511
DUP_VAL_ON_INDEX	Valor duplicado en índice	-1
INVALID_CURSOR	Cursor no válido	-1001
INVALID_NUMBER	Número no válido	-1722
LOGIN_DENIED	Denegada la conexión a Oracle	-1017
NO_DATA_FOUND	La consulta no recupera ninguna fila	+100
NOT_LOGGED_ON	No conectado a Oracle	-1012
PROGRAM_ERROR	Problema interno	-6501
STORAGE_ERROR	Fuera de memoria o error de memoria	-6500
TIMEOUT_ON_RESOURCE	Exceso de recursos consumidos	-51
TOO_MANY_ROWS	La consulta devuelve más de una fila	-1422
VALUE_ERROR	Valor incorrecto	-6502
ZERO_DIVIDE	División por cero	-1476
OTHERS	Cualquier otro error no especificado	

8.4. Excepciones definidas por el usuario

Existen dos tipos de excepciones a definir por los usuarios:

- **Basadas en errores ORACLE (EXCEPTION INIT)**

  Asigna un nombre a un error ORACLE existente.

  Sintaxis:

```
PRAGMA EXCEPTION_INIT (nombre_excepción, -número_error)
```

  donde

  nombre_excepción deberá estar definida como excepción de usuario.

  número_error es el número del error Oracle que se desea controlar (el que nos devuelve el SQLCODE).

  La definición se realiza en la zona del DECLARE y el control de la excepción en la zona EXCEPTION. Solo debe haber una excepción por cada error Oracle.

- **Otras excepciones.**

Se trata de un control de errores para el usuario. Este tipo de excepciones se deben declarar según vimos en el apartado de DECLARE. Una vez declarada la excepción, la podemos provocar cuando sea necesario utilizando la función RAISE que se

detalla a continuación. El tratamiento dentro de la zona de excepciones es exactamente igual que el de una excepción predefinida.

Sintaxis:

```
DECLARE
   A EXCEPTION;
BEGIN
     RAISE A;
EXCEPTION
     WHEN A THEN .... ;
END;
```

8.5. Ejecución de excepciones: RAISE

Para la ejecución del bloque PL/SQL y pasa el control a la zona de excepciones.

Sintaxis:

```
RAISE nombre_excepción
```

nombre_excepción es el nombre de la excepción que se desea ejecutar. Pueden ser excepciones predefinidas o excepciones de usuario.

```
DECLARE
    error EXCEPTION;

BEGIN
    RAISE error;
    RAISE TOO_MANY_ROWS;
EXCEPTION
    WHEN error THEN ...;
    WHEN TOO_MANY_ROWS THEN ...;
END;
```

8.6. SQLCODE

La función SQLCODE nos devuelve el número del error que se ha producido. Sólo tiene valor cuando ocurre un error Oracle.

Esta función solo se habilita en la zona de Excepciones, ya que es el único sitio donde se pueden controlan los errores.

No se puede utilizar directamente, pero si se puede guardar su valor en una variable.

8.7. SQLERRM.-

La función `SQLERRM` devuelve el mensaje del error del valor actual del `SQLCODE`.

Al igual que `SQLCODE`, no se puede utilizar directamente, sino que debemos declarar una variable alfanumérica lo suficientemente grande para contener el mensaje de error.