# Indice

[Indice 1](#_Toc99200867)

[Introducción a PL SQL 3](#_Toc99200868)

[Fundamentos 4](#_Toc99200869)

[Tipos de Datos 7](#_Toc99200870)

[Numéricos 8](#_Toc99200871)

[Texto 9](#_Toc99200872)

[Lógicos 10](#_Toc99200873)

[Fecha y hora 10](#_Toc99200874)

[Tipo especial 10](#_Toc99200875)

[Conclusiones 11](#_Toc99200876)

[Operadores 11](#_Toc99200877)

[Operadores aritméticos 12](#_Toc99200878)

[Operadores relacionales 12](#_Toc99200879)

[Operadores de comparación 13](#_Toc99200880)

[Operadores lógicos 14](#_Toc99200881)

[Funciones integradas 15](#_Toc99200882)

[Clasificación de comandos 18](#_Toc99200883)

[Estructura de PL SQL 19](#_Toc99200884)

[Sentencias de control 20](#_Toc99200885)

[Condicional 21](#_Toc99200886)

[IF - THEN 21](#_Toc99200887)

[IF - THEN - ELSE 21](#_Toc99200888)

[IF - THEN - ELSIF 22](#_Toc99200889)

[CASE 22](#_Toc99200890)

[ITERATIVOS - BUCLES 23](#_Toc99200891)

[FOR-LOOP 23](#_Toc99200892)

[WHILE-LOOP 25](#_Toc99200893)

[LOOP 26](#_Toc99200894)

[SECUENCIAL 26](#_Toc99200895)

[Cursores 27](#_Toc99200896)

[Cursores implícitos 28](#_Toc99200897)

[Excepciones asociadas a los cursores implícitos 29](#_Toc99200898)

[Atributos en cursores implícitos 29](#_Toc99200899)

[Cursores explícitos 30](#_Toc99200900)

[Ejemplos de Cursor Explícitos con estructura FOR-LOOP 31](#_Toc99200901)

[Ejemplos de Cursor Explícitos con estructura WHILE-LOOP 31](#_Toc99200902)

[Ejemplos de Cursor Explícitos con estructura LOOP 32](#_Toc99200903)

[Cursores de actualización 33](#_Toc99200904)

[Excepciones 34](#_Toc99200905)

[Excepciones predefinidas 35](#_Toc99200906)

[Excepciones definidas por el usuario 37](#_Toc99200907)

[Reglas de alcance 40](#_Toc99200908)

[Subprogramas en PL/SQL 40](#_Toc99200909)

[Procedimiento 40](#_Toc99200910)

[Función 42](#_Toc99200911)

[Triggers 43](#_Toc99200912)

[Paquetes - Packages 44](#_Toc99200913)

[Especificación del paquete 45](#_Toc99200914)

[Cuerpo del paquete 45](#_Toc99200915)

[Usando el paquete 48](#_Toc99200916)

[Transacciones 48](#_Toc99200917)

[Secuencias 50](#_Toc99200918)

# Introducción a 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. 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](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero" \l "term570) para extender el SQL estándar con otro tipo de instrucciones y elementos propios de los lenguajes de programación.

PL/SQL (Procedural Language/Structured Query Language) apareció por primera vez en la versión 6 de Oracle (1988) y amplia SQL con los elementos característicos de los lenguajes de programación: variables, sentencias de control de flujo, bucles, etc.

Con PL/SQL vamos a poder programar las unidades de programa de la base de datos [Oracle](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero" \l "term570):

* Procedimientos almacenados
* Funciones
* Triggers
* Scripts

Pero además, PL/SQL nos permite realizar programas sobre las siguientes herramientas de [Oracle](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero" \l "term570):

* [Oracle](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero#term570) Forms
* [Oracle](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero#term570) Reports
* [Oracle](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero#term570) Graphics
* [Oracle](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero#term570) Aplication Server

## Fundamentos

Para programar en PL/SQL es necesario conocer sus fundamentos. 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 la búsqueda de texto.
* Una línea en PL/SQL contiene grupos de caracteres conocidos como UNIDADES LEXICAS, que pueden ser clasificadas como:
  + DELIMITADOR: Es un símbolo simple o compuesto que tiene una función especial en PL/SQL. Estos pueden ser:
    - Operadores Aritméticos
    - Operadores Lógicos
    - Operadores Relaciónales

**Ejemplo de símbolos simple**s:

+ Operador de suma

% Indicador de Atributo

‘ Carácter delimitador de String

. Selector

/ Operador de división

( Expresión o delimitador de lista

) Expresión o delimitador de lista

: Indicador de variable host

, Separador de Items

\* Operador de multiplicación

“ Delimitador de identificadores

= Operador relacional

< Operador relacional

> Operador relacional

@ Indicador de acceso remoto

; Terminador de sentencia

- Resta/Operador de negación

**Ejemplo de símbolos compuestos:**

\*\* Operador de exponenciación

<> Operador relacional

!= Operador relacional

~= Operador relacional

<= Operador relacional

>= Operador relacional

:= Operador de Asignación

=> Operador de asociación

.. Operador de rango

|| Operador de concatenación

<< (Comienzo) delimitador de etiqueta

>> (Fin) delimitador de etiqueta

-- Indicador de comentario para una sola línea

/\* (Comienzo) delimitador de comentario de varias líneas

\*/ (Fin) delimitador de comentario de varias líneas

* + IDENTIFICADOR: Se emplean para dar nombre a los objetos PL/SQL, tales como variables, cursores, tipos y subprogramas. Los identificadores constan de una letra, seguida por una secuencia opcional de caracteres, que pueden incluir letras, números, signos de dólar ($), caracteres de subrayado y símbolos de almohadilla (#). Los demás caracteres no pueden emplearse. La longitud máxima de un identificador es de 30 caracteres y todos los caracteres son significativos:
    - * Constantes
      * Cursores
      * Variables
      * Subprogramas
      * Excepciones
      * Paquetes

Ejemplo:

*mi\_variable -- Identificador legal*

*mi variable -- Identificador Ilegal*

*mi-variable -- Identificador Ilegal*

* + 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 2 estilos de comentarios, el de línea simple y de multilínea, para lo cual son empleados ciertos caracteres especiales como son:

*-- Linea simple*

*/\**

*Conjunto de Líneas*

*\*/*

Por ejemplo, la siguiente sentencia:

*bonificacion := salario \* 0.10; -- Cálculo de Bonus*

Contiene las siguientes unidades léxicas:

• identificadores: bonificacion y salario

• símbolo compuesto: :=

• símbolos simples: \* y ;

• literal numérico: 0.10

• comentario: -- Cálculo de Bonus

* Cuando se escribe código en PL/SQL, este puede estar agrupado en unidades denominadas “conjunto de instrucciones”. Un conjunto de instrucciones puede contener otros subconjuntos y así sucesivamente. Un conjunto de instrucciones queda delimitado por las palabras reservadas BEGIN y END.

**BEGIN**

Sentencias . . .

Sentencias . . .

**BEGIN**

Sentencias . . .

Sentencias . . .

Sentencias . . .

**END;**

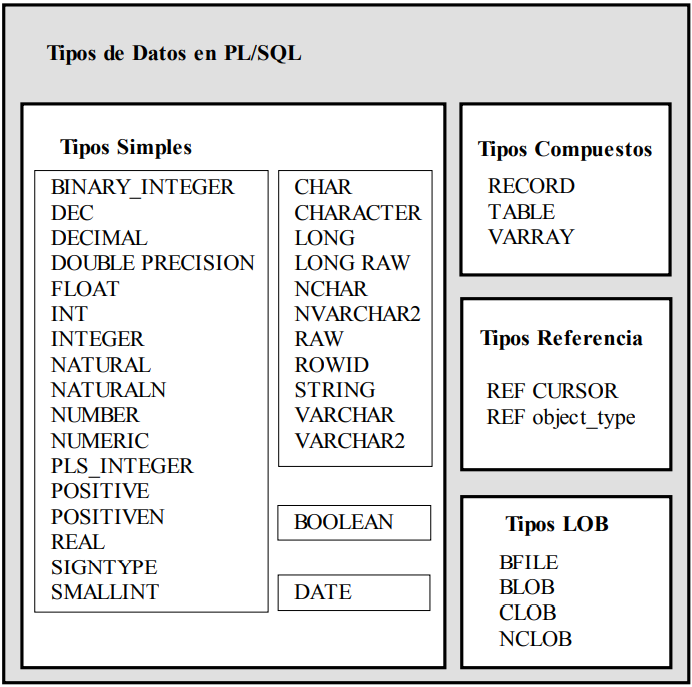
Sentencias . . .

Sentencias . . .

**END;**

# 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. Los tipos soportados por PL/SQL se muestran en la siguiente tabla:



Para fines específicos de este tutorial, sólo se especificarán los tipos de datos que más se suelen usar. Para tal fin, se va a realizar una especie de agrupamiento:

## Numéricos

| Tipos de dato en Oracle | | |
| --- | --- | --- |
| **Tipo** | **Almacena** | **Rango** |
| PLS\_INTEGER | Entero con signo | -2,147,483,648  a   2,147,483,647 |
| NATURAL | Entero PLS\_INTEGER no negativo |  |
| NUMBER | Números de coma flotante. Se debe especificar la precisión | 1e-130 a 1.0e126 |

## Texto

| Tipos de dato para caracteres en Oracle | | |
| --- | --- | --- |
| **Tipo** | **Almacena** | **Rango** |
| CHAR | Cadena de texto de tamaño fijo. | 32,767 bytes |
| VARCHAR2 | Cadena de texto de tamaño variable. | 32,767 bytes |
| NCHAR | Cadena de texto de tamaño fijo. Incluye caracteres UNICODE. | 32,767 bytes |
| NVARCHAR2 | Cadena de texto de tamaño variable. Incluye caracteres UNICODE. | 32,767 bytes |

## Lógicos

| Tipos de dato para valores lógicos en Oracle | | |
| --- | --- | --- |
| **Tipo** | **Almacena** | **Rango** |
| BOOLEAN | Almacena valores lógicos. Incluye el valor NULO | TRUE, FALSE o NULL |

## Fecha y hora

| Tipos de dato para tiempo en Oracle | | |
| --- | --- | --- |
| **Tipo** | **Almacena** | **Rango** |
| DATE | Fechas fijas, incluye año, mes, día, horas, minutos y segundos | 01/01/1,4712 BC  a  31/12/9999 |
| TIMESTAMP | Extiende el tipo DATE, incluye año, mes, día, horas, minutos y segundos fraccionados. |  |
| TIMESTAMP WITH TIME ZONE | Extiende el tipo de dato TIMESTAMP adicionando una zona horaria |  |

### Tipo especial

Se puede agregar un tipo más que es particular de PL SQL y es 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.

Atributo %TYPE:

* Permite declarar una variable basada en:
  + Otras variables previamente declaradas
  + La definición de una columna de la base de datos
* Preceder de %TYPE por:
  + La tabla y la columna de la base de datos
  + El nombre de la variable definida con anterioridad
* Ejemplo:  
  v\_ename  
  emp.ename%TYPE;

### Conclusiones

El manejo de variables es muy sencillo, aunque el hecho de que deben ser declaradas antes de ser usadas resulte incómodo para algunos. Los tipos de datos son muy amplios en Oracle, sólo se han presentado los más utilizados.

# Operadores

Un operador es un símbolo que le dice al compilador que realice una manipulación matemática o lógica específica. El lenguaje PL / SQL es rico en operadores integrados y proporciona los siguientes tipos de operadores:

* Operadores aritméticos
* Operadores relacionales
* Operadores de comparación
* Operadores logicos
* Operadores de cadena

Aquí se expilcarán los operadores aritméticos, relacionales, de comparación y lógicos uno por uno. Los operadores de cadena sólo se lo mencionan a modo nominativo.

## Operadores aritméticos

La siguiente tabla muestra todos los operadores aritméticos compatibles con PL / SQL. Supongamos que la variable A tiene 10 y la variable B tiene 5, entonces:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descripción** | **Ejemplo** |
| + | Agrega dos operandos | A + B dará 15 |
| - | Resta el segundo operando del primero | A - B dará 5 |
| \* | Multiplica ambos operandos | A \* B dará 50 |
| / | Divide el numerador por el numerador | A / B dará 2 |
| \*\* | Operador de exponenciación, eleva un operando a la potencia de otro | A \*\* B dará 100000 |

## Operadores relacionales

Los operadores relacionales comparan dos expresiones o valores y devuelven un resultado booleano. La siguiente tabla muestra todos los operadores relacionales compatibles con PL / SQL. Supongamos que la variable A tiene 10 y la variable B tiene 20, entonces:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descripción** | **Ejemplo** |
| = | Comprueba si los valores de dos operandos son iguales o no, si es así, la condición se convierte en verdadera. | (A = B) no es cierto. |
| ! =  <>  ~ = | Comprueba si los valores de dos operandos son iguales o no, si los valores no son iguales, la condición se convierte en verdadera. | (A! = B) es cierto. |
| > | Comprueba si el valor del operando izquierdo es mayor que el valor del operando derecho, si es así, la condición se convierte en verdadera. | (A> B) no es cierto. |
| < | Comprueba si el valor del operando izquierdo es menor que el valor del operando derecho, si es así, la condición se convierte en verdadera. | (A <B) es cierto. |
| > = | Comprueba si el valor del operando izquierdo es mayor o igual que el valor del operando derecho, si es así, la condición se convierte en verdadera. | (A> = B) no es cierto. |
| <= | Comprueba si el valor del operando izquierdo es menor o igual que el valor del operando derecho, si es así, la condición se convierte en verdadera. | (A <= B) es cierto |

## Operadores de comparación

Los operadores de comparación se utilizan para comparar una expresión con otra. El resultado es siempre VERDADERO, FALSO o NULO.

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descripción** | **Ejemplo** |
| LIKE | El operador LIKE compara un carácter, cadena o valor CLOB con un patrón y devuelve VERDADERO si el valor coincide con el patrón y FALSO si no lo hace. | Si 'Zara Ali' como 'Z% A\_i' devuelve un booleano verdadero, mientras que 'Nuha Ali' como 'Z% A\_i' devuelve un booleano falso. |
| BETWEEN | El operador BETWEEN comprueba si un valor se encuentra en un rango especificado. x BETWEEN ayb significa que x> = ay x <= b. | Si x = 10 entonces, x entre 5 y 20 devuelve verdadero, x entre 5 y 10 devuelve verdadero, pero x entre 11 y 20 devuelve falso. |
| IN | El operador IN prueba la pertenencia al conjunto. x IN (conjunto) significa que x es igual a cualquier miembro del conjunto. | Si x = 'm' entonces, x in ('a', 'b', 'c') devuelve booleano falso pero x in ('m', 'n', 'o') devuelve booleano verdadero. |
| IS NULL | El operador IS NULL devuelve el valor BOOLEAN TRUE si su operando es NULL o FALSE si no es NULL. Las comparaciones que involucran valores NULL siempre dan como resultado NULL. | Si x = 'm', entonces 'x es nulo' devuelve booleano falso. |

## Operadores lógicos

La siguiente tabla muestra los operadores lógicos compatibles con PL / SQL. Todos estos operadores funcionan con operandos booleanos y producen resultados booleanos. Supongamos que la variable A es verdadera y la variable B es falsa, entonces:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Descripción** | **Ejemplos de** |
| and | Llamado operador lógico AND. Si ambos operandos son verdaderos, la condición se convierte en verdadera. | (A y B) es falso. |
| or | Llamado operador OR lógico. Si alguno de los dos operandos es verdadero, la condición se vuelve verdadera. | (A o B) es cierto. |
| not | Llamado operador NOT lógico. Se utiliza para invertir el estado lógico de su operando. Si una condición es verdadera, el operador NOT lógico la convertirá en falsa. | no (A y B) es cierto. |

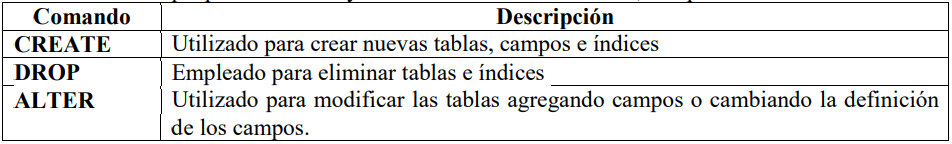
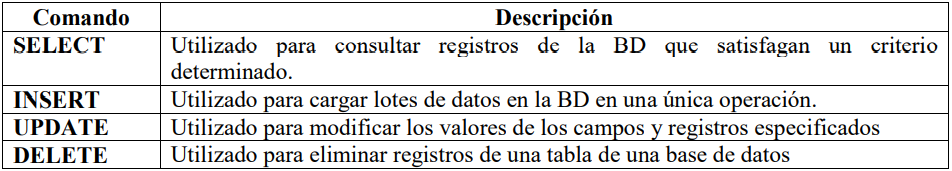
# Funciones integradas

Las funciones integradas son funciones predefinidas en PL SQL y que son de gran ayuda para obtener ciertos valores en forma inmediata. Existen infinidad de funciones. Sólo vamos a nombrar alguna de las más relevantes:

* CONCAT(cadena1, cadena2): concatena dos cadenas de caracteres, es equivalente al operador ||. Ejemplo: SELECT concat('Buenas', ' tardes') FROM dual; /\*retorna 'Buenas tardes'.\*/ SELECT ‘Buenas’ || ‘ tardes’ FROM dual
* LOWER(cadena): retorna la cadena enviada como argumento en minúsculas. Ejemplo: SELECT lower('Buenas tardes ALUMNO') FROM dual; /\*retorna "buenas tardes alumno".\*/
* UPPER(cadena): retorna la cadena con todos los caracteres en mayúsculas. Ejemplo: SELECT upper('www.oracle.com') FROM dual; /\*'WWW.ORACLE.COM'\*/
* LTRIM(cadena1): elimina todos los espacios en blanco que se encuentren a la izquierda de cadena1. Ejemplo: SELECT ltrim(' la casa') FROM dual; /\*'la casa'\*/
* RTRIM(cadena1): elimina todos los espacios en blanco que se encuentren a la derecha de cadena1. Ejemplo: SELECT rtrim('la casa lila ') FROM dual; /\*'la casa lila'\*/
* TRIM(cadena): retorna la cadena con los espacios de la izquierda y derecha eliminados. Ejemplo: SELECT TRIM(' oracle ') FROM dual; /\*'oracle'\*/
* REPLACE(cadena1, subcadena1, subcadena2): retorna la cadena con todas las ocurrencias de la subcadena de reemplazo (subcadena2) por la subcadena a reemplazar (subcadena1). Ejemplo: SELECT REPLACE('xxx.oracle.com', 'x', 'w') FROM dual; /\*retorna "www.oracle.com'.\*/
* SUBSTR(cadena, inicio, longitud): devuelve una parte de la cadena especificada como primer argumento, empezando desde la posición especificada por el segundo argumento y de tantos caracteres de longitud como indica el tercer argumento. Ejemplos: SELECT substr('www.oracle.com', 1, 10) FROM dual; /\*'www.oracle'\*/ SELECT substr('www.oracle.com', 5, 6) FROM dual; /\*'oracle'\*/
* LENGTH(cadena): retorna la longitud de la cadena enviada como argumento. Ejemplo: SELECT length('www.oracle.com') FROM dual; /\*devuelve 14.\*/
* INSTR(cadena, subcadena): devuelve la posición de comienzo (de la primera ocurrencia) de la subcadena especificada en la cadena enviada como primer argumento. Si no la encuentra retorna 0. Ejemplos: SELECT instr('Jorge Luis Borges', 'or') FROM dual; /\*2\*/ SELECT instr('Jorge Luis Borges', 'ar') FROM dual; /\*0, no se encuentra\*/
* ABS(x): retorna el valor absoluto del argumento “x”. Ejemplo: SELECT abs(-20) FROM dual; --retorna 20.
* CEIL(x): redondea a entero, hacia arriba, al argumento “x”. Ejemplo: SELECT ceil(12.34) FROM dual; --retorna 13.
* FLOOR(x): redondea a entero, hacia abajo, el argumento “x”. Ejemplo: SELECT floor(12.34) FROM dual; --12
* MOD(x, y): devuelve el resto de la división x/y. Ejemplos: SELECT MOD(10, 3) FROM dual; --retorna 1. SELECT MOD(10, 2) FROM dual; --retorna 0.
* POWER(x, y): retorna el valor de “x” elevado a la “y” potencia. Ejemplo: SELECT power(2, 3) FROM dual; --retorna 8.
* ROUND(n, d): retorna “n” redondeado a “d” decimales, si se omite el segundo argumento, redondea todos los decimales. Si el segundo argumento es positivo, el número de decimales es redondeado según “d”, si es negativo, el número es redondeado desde la parte entera según el valor de “d”. Ejemplos: SELECT round(123.456, 2) FROM dual; /\*retorna "123.46", es decir, redondea desde el segundo decimal.\*/ SELECT round(123.456, 1) FROM dual; /\*123.5, es decir, redondea desde el primer decimal.\*/ SELECT round(123.456, -1) FROM dual; /\*120, redondea desde el primer valor entero (hacia la izquierda).\*/ SELECT round(123.456, -2) FROM dual; /\*100, redondea desde el segundo valor entero (hacia la izquierda).\*/ SELECT round(123.456) FROM dual; /\*123.\*/
* TRUNC(n, d): trunca un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si “d” es negativo, el número es truncado desde la parte entera. Ejemplos: SELECT trunc(1234.5678, 2) FROM dual; --retorna 1234.56 SELECT trunc(1234.5678, -2) FROM dual; --retorna 1200 SELECT trunc(1234.5678, -1) FROM dual; --retorna 1230 SELECT trunc(1234.5678) FROM dual; --retorna 1234
* SQRT(x): devuelve la raíz cuadrada del valor enviado como argumento. Ejemplo: SELECT sqrt(9) FROM dual; --retorna 3
* ADD\_MONTHS(f, n): agrega a una fecha, un número de meses. Si el segundo argumento es positivo, se le suma a la fecha enviada tal cantidad de meses, si es negativo, se le resta a la fecha enviada tal cantidad de meses. Ejemplos: SELECT add\_months('10/06/2007', 5) FROM dual; /\*retorna "10/11/07"\*/ SELECT add\_months('10/06/2007', -5) FROM dual; /\*retorna "10/01/07"\*/ SELECT add\_months('30/01/2007', 1) FROM dual; /\*retorna "28/02/07" ya que es el último día de ese mes.\*/
* SYSDATE: retorna la fecha y hora actuales en el servidor de Oracle.
* SYSTIMESTAMP: retorna fecha y hora actuales.
* TO\_DATE: convierte una cadena a tipo de dato “date”. Ejemplo: SELECT to\_date('05-SEP-2016 10:00 AM', 'DD-MON-YYYY HH:MI AM') FROM dual;
* TO\_CHAR: convierte una fecha a cadena de caracteres. Ejemplo: SELECT to\_char('10/10/2016') FROM dual;

# Clasificación de comandos

Las sentencias SQL pertenecen a dos categorías principales: Lenguaje de Definición de Datos (DDL) y Lenguaje de Manipulación de Datos (DML). Estos dos lenguajes no son lenguajes en sí mismos, sino que es una forma de clasificar las sentencias de lenguaje SQL en función de su cometido. La diferencia principal reside en que el DDL crea objetos en la base de datos y sus efectos se pueden ver en el diccionario de la base de datos; mientras que el DML es el que permite consultar, insertar, modificar y eliminar la información almacenada en los objetos de la base de datos. Entonces tenemos:

* Los DDL que permiten crear y definir nuevas bases de datos, campos e índices.
* Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

# Estructura de PL SQL

La sintaxis básica de PL / SQL es un lenguaje estructurado en bloques; esto significa que los programas PL / SQL se dividen y escriben en bloques lógicos de código. Cada bloque consta de tres subpartes:

|  |  |
| --- | --- |
| **Nro.** | **Secciones y descripción** |
| 1 | **Declaraciones**  Esta sección comienza con la palabra clave **DECLARE** . Es una sección opcional y define todas las variables, cursores, subprogramas y otros elementos que se utilizarán en el programa. |
| 2 | **Comandos ejecutables**  Esta sección se incluye entre las palabras clave **BEGIN** y **END** y es una sección obligatoria. Consiste en las sentencias PL / SQL ejecutables del programa. Debe tener al menos una línea de código ejecutable, que puede ser solo un **comando NULL** para indicar que no se debe ejecutar nada. |
| 3 | **Manejo de excepciones**  Esta sección comienza con la palabra clave **EXCEPCTION** . Esta sección opcional contiene **excepciones** que manejan errores en el programa. |

Cada instrucción PL / SQL termina con un punto y coma (;). Los bloques PL / SQL se pueden anidar dentro de otros bloques PL / SQL usando BEGIN y END . A continuación se muestra la estructura básica de un bloque PL / SQL:

DECLARE

<declarations section>

BEGIN

<executable command(s)>

EXCEPTION

<exception handling>

END;

El ejemplo de 'Hola mundo'

DECLARE

message varchar2(20):= 'Hello, World!';

BEGIN

dbms\_output.put\_line(message);

END;

/

# Sentencias de control

Las sentencias de control permiten cambiar el flujo de ejecución de las instrucciones de nuestro código PL/SQL, de esta manera se puede indicar la ejecución de un bloque de instrucciones cuando se cumpla una condición o simplemente hacer que se repita en tiempo de ejecución.

En PL/SQL tenemos grupos de sentencias de control:

* Condicional
* Iterativo - Bucles
* Secuencial

## Condicional

La estructura condicional está representada por las sentencias IF y CASE

### IF - THEN

Se evalúa la condición y si resulta **verdadera**, se ejecutan uno o más líneas de código de programa. En el caso de que la condición resulte **falsa o nula**, NO se realiza NINGUNA acción.

**IF** fecha\_nac **<** '1-01-1970' **THEN** *--No termina con un ;*

Salario :**=** salario **\***1.15; *--aumento de salario en un 15%*

**END** **IF**;

Se pueden anidar varias instrucciones:

**IF** fecha\_nac **<** ‘1**-**01**-**1970’ **THEN**

**IF** apellido **=**‘Martínez’ **THEN**

salario:**=** salario **\***1.15;

**END** **IF**;

**END** **IF**;

### IF - THEN - ELSE

Se evalúa la condición y si resulta **verdadera**, se ejecutan uno o más líneas de código de programa. En el caso de que la condición resulte **falsa**, se ejecutan las instrucciones que siguen a la instrucción ELSE. Sólo se permite una instrucción ELSE en cada instrucción IF.

**IF** fecha\_nac **<**’1**-**01**-**1970’ **THEN**

salario:**=** salario **\***1.15;

**ELSE**

salario:**=** salario**\*** 1.05;

**END** **IF**;

### IF - THEN - ELSIF

Se evalúa la condición y si resulta **verdadera**, se ejecutan uno o más líneas de código de programa. En el caso de que la condición resulte ser **falsa**, se evalúa la condición especificada en el ELSIF.

**IF** condicion **THEN**

instrucciones;

**ELSE**

**IF** condicion2 **THEN**

instrucciones;

**ELSE**

**IF** condicion3 **THEN**

instrucciones;

**END** **IF**;

**END** **IF**;

**END** **IF**;

**IF** apellido **= ‘**Pérez’ **THEN**

salario:**=** salario **\***1.10; *--aumento de salario en un 10%*

**ELSIF** apellido **=**‘Martínez’ **THEN**

salario:**=** salario **\***1.15; *--aumento de salario en un 15%*

**ELSIF** apellido**=**‘Alvarez’ **THEN**

salario:**=** salario **\***1.20; *--aumento de salario en un 20%*

**ELSE**

salario:**=** salario**\*** 1.05; *--aumento de salario en un 5%*

**END** **IF**; *--Sólo se necesita un único END IF*

### CASE

La instrucción CASE puede evaluar múltiples expresiones y devolver para cada una de ellas un valor/bloque de instrucciones. El resultado de cada WHEN puede ser un valor o una sentencia, en el primer caso el resultado de una sentencia CASE se puede guardar en una variable.

Su sintaxis:

**CASE**

**WHEN** expresión1 **THEN** valor1**/**bloque de instrucciones

**WHEN** expresión2 **THEN** valor2**/**bloque de instrucciones

**WHEN** expresión3 **THEN** valor3**/**bloque de instrucciones

**WHEN** expresión4 **THEN** valor4**/**bloque de instrucciones

**ELSE** valor5**/**bloque de instrucciones

**END CASE**

## ITERATIVOS - BUCLES

Las sentencias iterativas permiten ejecutar varias instrucciones múltiples veces. A éstas sentencias también se les conoce por el nombre de bucles repetitivos

### FOR-LOOP

La sentencia FOR-LOOP permite especificar un rango de números enteros, finalmente ejecuta una secuencia de instrucciones para cada número entero dentro de la lista de números.

**FOR** contador **IN** limite\_inferior..limite\_superior LOOP

sentencias

**END** LOOP;

Contador deberá ser una variable de tipo numérico que sea capaz de contener los valores comprendidos entre limite\_inferior y limite\_superior, los cuales deberán ser expresiones numéricas, ya sean constantes (1,10…) o funciones (ROUND(max,0), ASCII(‘A’)…) .

Si la variable contador no está definida, PL/SQL definirá una variable de tipo INTEGER al iniciar el bucle, y la liberará al finalizar el bucle.

Veámos un ejemplo:

|  |  |
| --- | --- |
| 1 | BEGIN |
| 2 | FOR i IN 1..8 LOOP | |

|  |  |  |
| --- | --- | --- |
| 3 | DBMS\_OUTPUT.PUT\_LINE('El número de esta iteración es:' || i); | |
| 4 | END LOOP; |

|  |  |
| --- | --- |
| 5 | END; |
| El número de ésta iteración es:1 | | |

|  |
| --- |
| El número de ésta iteración es:2 |
| El número de ésta iteración es:3 |

|  |
| --- |
| El número de ésta iteración es:4 |
| El número de ésta iteración es:5 |

|  |
| --- |
| El número de ésta iteración es:6 |
| El número de ésta iteración es:7 |

|  |
| --- |
| El número de ésta iteración es:8 |

En el ejemplo anterior, sólo existe una instrucción para imprimir un mensaje en la sentencia LOOP, pero sin embargo en la salida el mensaje se repite 8 veces con los números del 1 al 8. En la cláusula IN la expresión 1..8 genera una lista de números, para éste caso , del 1 al 8.

Probemos ahora generando la tabla de multiplicar del 7 del 1 al 12:

|  |  |
| --- | --- |
| 1 | BEGIN |
| 2 | FOR i IN 1..12 LOOP | |

|  |  |  |
| --- | --- | --- |
| 3 | DBMS\_OUTPUT.PUT\_LINE('7\*' || i || '='|| (7\*i)); | |
| 4 | END LOOP; |

|  |  |
| --- | --- |
| 5 | END; |

|  |
| --- |
| 7\*1=7 |
| 7\*2=14 | |

|  |
| --- |
| 7\*3=21 |
| 7\*4=28 |

|  |
| --- |
| 7\*5=35 |
| 7\*6=42 |

|  |
| --- |
| 7\*7=49 |
| 7\*8=56 |

|  |
| --- |
| 7\*9=63 |
| 7\*10=70 | |

|  |
| --- |
| 7\*11=77 |
| 7\*12=84 |

### WHILE-LOOP

La sentencia de control WHILE LOOP, al igual que FOR-LOOP, permite la ejecución de una o varias instrucciones. A diferencia de la sentencia FOR-LOOP, en WHILE-LOOP la ejecución se define a partir de una condición. Veamos un ejemplo:

|  |  |
| --- | --- |
| 1 | DECLARE |
| 2 | V\_NUM NUMBER := 1; | |

|  |  |
| --- | --- |
| 3 | BEGIN |
| 4 | WHILE V\_NUM <= 13 LOOP | |

|  |  |  |
| --- | --- | --- |
| 5 | DBMS\_OUTPUT.PUT\_LINE('El valor V\_NUM es :' || V\_NUM); | |
| 6 | V\_NUM := V\_NUM + 2.1; |

|  |  |  |
| --- | --- | --- |
| 7 | END LOOP; | |
| 8 | END; |
| 1 | El valor V\_NUM es :1 | | |

|  |  |
| --- | --- |
| 2 | El valor V\_NUM es :3,1 |
| 3 | El valor V\_NUM es :5,2 |

|  |  |
| --- | --- |
| 4 | El valor V\_NUM es :7,3 |
| 5 | El valor V\_NUM es :9,4 |

|  |  |
| --- | --- |
| 6 | El valor V\_NUM es :11,5 |
|  |  |

En el ejemplo anterior, al igual que en la sentencia FOR-LOOP, el mensaje "El valor de V\_NUM es ..." se repite varias veces, pero ésta vez la variable V\_NUM se ve incrementada en 2.1 cada vez que la sentencia WHILE-LOOP se ejecuta. En la cláusula WHILE, mientras la condición **V\_NUM <= 13** es verdadera se ejecutará el código dentro de la cláusula LOOP. Finalmente debido al incremento realizado en la línea 6 (V\_NUM := V\_NUM + 2.1) el código sólo se repetirá 6 veces.

### LOOP

La sentencia LOOP permite ejecutar un bloque de código de manera infinita, a diferencia de las otras sentencias iterativas, en la sentencia LOOP se debe utilizar la cláusula EXIT WHEN para colocar una condición para terminar el bucle:

|  |  |
| --- | --- |
| 01 | DECLARE |
| 02 | V\_TOTAL    NUMBER := 0; | |

|  |  |  |
| --- | --- | --- |
| 03 | V\_CONTADOR NUMBER := 0; | |
| 04 | BEGIN |

|  |  |
| --- | --- |
| 05 | LOOP |
| 06 | V\_CONTADOR := V\_CONTADOR + 1; | |

|  |  |
| --- | --- |
| 07 | V\_TOTAL := V\_TOTAL + V\_CONTADOR; |
| 08 | DBMS\_OUTPUT.PUT\_LINE('Valor de V\_TOTAL:' || V\_TOTAL); | |

|  |  |  |
| --- | --- | --- |
| 09 | EXIT WHEN V\_TOTAL > 20; | |
| 10 | END LOOP; |

|  |  |
| --- | --- |
| 11 | END; |
| Valor de V\_TOTAL:1 | | |

|  |
| --- |
| Valor de V\_TOTAL:3 |
| Valor de V\_TOTAL:6 |

|  |
| --- |
| Valor de V\_TOTAL:10 |
| Valor de V\_TOTAL:15 |

|  |
| --- |
| Valor de V\_TOTAL:21 |
|  |

En el ejemplo anterior, la sentencia LOOP terminará cuando el valor de la variables V\_TOTAL sea mayor a 20.

## SECUENCIAL

La sentencia GOTO permite cambiar la línea secuencial de la ejecución hacia una etiqueta. Las etiquetas son marcadas con los símbolos << y >>. Para ir dirigir la ejecución hacia una etiqueta simplemente hay que usar la sentencia GOTO:

|  |  |
| --- | --- |
| 01 | DECLARE |
| 02 | V\_TOTAL    NUMBER(9) := 0; | |

|  |  |  |
| --- | --- | --- |
| 03 | V\_CONTADOR NUMBER(6) := 0; | |
| 04 | BEGIN |

|  |  |
| --- | --- |
| 05 |  |
| 06 | <<calcular>> | |

|  |  |
| --- | --- |
| 07 | V\_CONTADOR := V\_CONTADOR + 1; |
| 08 | V\_TOTAL := V\_TOTAL + V\_CONTADOR; | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | IF V\_TOTAL <= 20 THEN | |

|  |  |  |
| --- | --- | --- |
| 11 | DBMS\_OUTPUT.PUT\_LINE('Valor de V\_TOTAL:' || V\_TOTAL); | |
| 12 | GOTO calcular; |

|  |  |  |
| --- | --- | --- |
| 13 | END IF; | |
| 14 | END; |

|  |
| --- |
| Valor de V\_TOTAL:1 |
| Valor de V\_TOTAL:3 |

|  |
| --- |
| Valor de V\_TOTAL:6 |
| Valor de V\_TOTAL:10 | |

|  |
| --- |
| Valor de V\_TOTAL:15 |

En el ejemplo anterior, la sentencia GOTO redireccionará la ejecución hacia la línea 6 mientras el valor de V\_TOTAL sea menor a 20.

# Cursores

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. |

### Atributos en cursores implícitos

Los cursores implícitos no se pueden manipular por el usuario, pero Oracle sí permite el uso de sus atributos. Las sentencia a través de las que podemos obtener información de estos atributos son: SELECT ... INTO, [INSERT, UPDATE, DELETE].

En este caso, se debe anteponer al nombre del atributo el prefijo SQL, en lugar del nombre del cursor.

* **SQL%NOTFOUND** devuelve TRUE cuando la última sentencia SELECT no recuperó ninguna fila, o cuando INSERT, DELETE o UPDATE no afectan a ninguna fila
* **SQL%FOUND** devuelve TRUE cuando la última sentencia SELECT devuelve alguna fila, o cuando INSERT, DELETE o UPDATE afectan a alguna fila
* **SQL%ROWCOUNT** devuelve el número de filas afectadas por INSERT, DELETE o UPDATE o las filas devueltas por una sentencia SELECT
* **SQL%ISOPEN** siempre devuelve FALSE, porque Oracle cierra automáticamente el cursor implícito cuando termina la ejecución de la sentencia SELECT

## 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;

### Ejemplos de Cursor Explícitos con estructura FOR-LOOP

DECLARE

CURSOR cemp IS

SELECT salario

, dni

FROM empleados;

BEGIN

FOR cemp\_rec IN cemp LOOP

Dbms\_output.put\_line (cemp\_rec.dni || ' ' || cemp\_rec.salario);

END LOOP;

END;

Como se puede observar, el bucle FOR es el único que no necesita un OPEN, FETCH y CLOSE. Cuando inicia el bucle, ya se ejecutan las dos primeras sentencias. Y a finalizar el bucle, ya se hace ejecuta la última sentencia automáticamente.

### Ejemplos de Cursor Explícitos con estructura WHILE-LOOP

DECLARE

CURSOR cemp(pnombre IN VARCHAR2) IS

SELECT \*

FROM empleados

WHERE nombre = pnombre;

cemp\_rec empleados%ROWTYPE;

vnombre VARCHAR2(20);

BEGIN

vnombre := 'Juan';

dbms\_output.put\_line ('Sueldo de los empleados con nombre ' || vnombre);

OPEN cemp(vnombre);

FETCH cemp

INTO cemp\_rec;

WHILE cemp%FOUND LOOP

dbms\_output.put\_line (cemp\_rec.dni || ' ' || cemp\_rec.salario);

FETCH cemp

INTO cemp\_rec;

END LOOP;

dbms\_output.put\_line ('Número de empleados procesados ' || cemp%ROWCOUNT);

CLOSE cemp;

END;

### Ejemplos de Cursor Explícitos con estructura LOOP

DECLARE

CURSOR cemp(pnombre IN VARCHAR2) IS

SELECT \*

FROM empleados

WHERE nombre = pnombre;

cemp\_rec empleados%ROWTYPE;

vnombre VARCHAR2(20);

BEGIN

vnombre := 'Juan';

dbms\_output.put\_line ('Sueldo de los empleados con nombre ' || vnombre);

OPEN cemp(vnombre);

LOOP

FETCH cemp

INTO cemp\_rec;

dbms\_output.put\_line (cemp\_rec.dni || ' ' || cemp\_rec.salario);

EXIT WHEN cemp%NOTFOUND; -- Último registro.

END LOOP;

dbms\_output.put\_line ('Número de empleados procesados ' || cemp%ROWCOUNT);

CLOSE cemp;

END;

## Cursores de actualización

Los cursores de actualización se declaran igual que los cursores explícitos, añadiendo FOR UPDATE al final de la sentencia SELECT.

**CURSOR** nombre\_cursor **IS** instrucción\_SELECT **FOR** **UPDATE**

Para actualizar los datos del cursor hay que ejecutar una sentencia UPDATE especificando la cláusula WHERE CURRENT OF <cursor\_name>.

**UPDATE** **<**nombre\_tabla**>** **SET** **<**campo\_1**>** **=** **<**valor\_1**>**[,**<**campo\_2**>** **=** **<**valor\_2**>**]

**WHERE** **CURRENT** **OF** **<cursor\_name>**

Cuando trabajamos con cursores de actualización debemos tener en cuenta que la sentencia UPDATE genera bloqueos en la base de datos ( [transacciones](https://elbauldelprogramador.com/introduccion-plsql-transacciones/), disparadores,etc).

**DECLARE**

**CURSOR** cpaises **IS**

**select** CO\_PAIS, DESCRIPCION, CONTINENTE **from** paises

**FOR** **UPDATE**;

co\_pais VARCHAR2(3);

descripcion VARCHAR2(50);

continente VARCHAR2(25);

**BEGIN**

**OPEN** cpaises;

**FETCH** cpaises **INTO** co\_pais,descripcion,continente;

WHILE cpaises**%found**

LOOP

**UPDATE** PAISES **SET** CONTINENTE **=** CONTINENTE **||** '.'

**WHERE** **CURRENT** **OF** cpaises;

**FETCH** cpaises **INTO** co\_pais,descripcion,continente;

**END** LOOP;

**CLOSE** cpaises;

**COMMIT**;

**END**;

# 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](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero" \l "term570) 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](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero" \l "term570) 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](http://www.juntadeandalucia.es/servicios/madeja/glossary/12/lettero" \l "term570) -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;

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.

Cuando se produce un error, se ejecuta el bloque EXCEPTION. Si existe un bloque de excepción apropiado para el tipo de error producido se ejecuta dicho bloque. Si este último no existe, se ejecutará el bloque de excepción WHEN OTHERS THEN ( en el caso de haberlo definido, este bloque debe ser el último manejador de excepciones ). Una vez finalizada la ejecución del bloque de EXCEPTION no se continúa ejecutando el bloque anterior.

En ocasiones queremos enviar un mensaje de error personalizado al producirse una excepción PL/SQL. Para ello es necesario utilizar la instrucción RAISE\_APPLICATION\_ERROR.

RAISE\_APPLICATION\_ERROR(**<**error\_num**>**,**<**mensaje**>**);

Donde:

* **error\_num** es un entero negativo comprendido entre -20001 y -20999.
* **mensaje** es la descripción del error.

Ejemplo:

**DECLARE**

v\_div NUMBER;

**BEGIN**

**SELECT** 1**/**0 **INTO** v\_div **FROM** DUAL;

**EXCEPTION**

**WHEN** OTHERS **THEN**

RAISE\_APPLICATION\_ERROR(**-**20001,'No se puede dividir por cero');

**END**;

Dentro del bloque de excepciones conviene recordar la existencia de la excepción OTHERS, que simboliza cualquier condición de excepción que no ha sido declarada. Se utiliza comúnmente para controlar cualquier tipo de error que no ha sido previsto. En ese caso, es común observar la sentencia ROLLBACK en el grupo de sentencias de la excepción o alguna de las funciones SQLCODE – SQLERRM.

* SQLCODE devuelve el número del error de Oracle y un 0 (cero) en caso de éxito al ejecutarse una sentencia SQL.
* SQLERRM devuelve la descripción del correspondiente mensaje de error. También es posible entregarle a la función SQLERRM un número negativo que represente un error de Oracle y ésta devolverá el mensaje asociado.

Estas funciones no pueden ser utilizadas directamente en una sentencia SQL, pero sí se puede asignar su valor a alguna variable de programa y luego usar esta última en alguna sentencia.

**SET** SERVEROUTPUT **ON**;

**DECLARE**

err\_num NUMBER;

err\_msg VARCHAR2(255);

**result** NUMBER;

msg VARCHAR2(255);

**BEGIN**

msg :**=** SQLERRM(**-**1403);

DBMS\_OUTPUT.put\_line(MSG);

**SELECT** 1**/**0 **INTO** **result** **FROM** DUAL;

**EXCEPTION**

**WHEN** OTHERS **THEN**

err\_num :**=** **SQLCODE**;

err\_msg :**=** SQLERRM;

DBMS\_OUTPUT.put\_line('Error:'**||**TO\_CHAR(err\_num));

DBMS\_OUTPUT.put\_line(err\_msg);

**END**;

**DECLARE**

e\_sinreg **EXCEPTION**;

a number(10) :**=** 25;

b number(10) :**=** 0;

**c** number(10);

**BEGIN**

**Select** **count**(**\***) **INTO** a **FROM** Articulos;

**If** a **<** 10 **THEN**

RAISE e\_sinreg;

**END** **IF**;

**c** :**=** a **/** b;

DBMS\_OUTPUT.PUT\_LINE(' Esto nunca llegará a mostrarse. ');

**EXCEPTION**

**WHEN** ZERO\_DIVIDE **THEN** DBMS\_OUTPUT.PUT\_LINE('No se puede dividir por 0');

**WHEN** e\_sinreg **THEN** DBMS\_OUTPUT.PUT\_LINE('Hay menos de 10 articulos.');

**WHEN** OTHERS **THEN** DBMS\_OUTPUT.PUT\_LINE('Se ha producido otra excepción.');

**END**;

Las líneas de código debajo del manejador específico se ejecutarán cuando esa excepción se produzca. Algunas excepciones se lanzarán automáticamente cuando se produzcan ciertos tipos de errores en la ejecución del bloque de código. Cada excepción automática tiene asociado un código de error ORA-XXXX el cual si se produce, hará que se lance la excepción correspondiente.

## Reglas de alcance

Una excepción es válida dentro de su ámbito 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 sub-bloques que comprende.

# Subprogramas en PL/SQL

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.

## Procedimiento

Un procedimiento es un subprograma que ejecuta una acción específica 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;

El procedimiento puede ser llamado desde un bloque de PL SQL

**BEGIN**

Actualiza\_Saldo(200501,2500);

**COMMIT**;

**END**;

Ejemplos de Procedimiento

--Crear Procedimiento sin Parámetro

CREATE OR REPLACE PROCEDURE HolaMundoNoParam IS

BEGIN

dbms\_output.put\_line('Hola Mundo Sin Parámetros');

END;

--Llamar al procedimiento

BEGIN

HolaMundoNoParam;

END;

--Crear Procedimiento con Parámetro

CREATE OR REPLACE PROCEDURE HolaMundoConParam(v\_desc varchar2) IS

BEGIN

dbms\_output.put\_line(v\_desc);

END;

--Llamar al procedimiento con parámetro

DECLARE

v\_mensaje varchar2(50):='Hola Mundo con Parámetro';

BEGIN

HolaMundoConParam(v\_mensaje);

END;

## 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

Ejemplo de Función

--Crear una función que realice la suma de dos números y devolver el resultado

CREATE OR REPLACE FUNCTION suma(v\_num1 number, v\_num2 number) return number IS

v\_res number;

BEGIN

v\_res := v\_num1 + v\_num2;

return v\_res;

END;

--Llamar a la función suma en un bloque

DECLARE

v\_resultado number;

v\_otra number;

BEGIN

v\_resultado:=suma(2, 8);

v\_otra := v\_resultado \* 5;

dbms\_output.put\_line('La suma es: ' || v\_resultado);

END;

## 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.

# Paquetes - Packages

Los paquetes son objetos de esquema que agrupan tipos, variables y subprogramas de PL / SQL relacionados lógicamente.

Un paquete tendrá dos partes obligatorias:

* Especificación del paquete
* Cuerpo o definición del paquete

## Especificación del paquete

La especificación es la interfaz del paquete. Simplemente DECLARA los tipos, variables, constantes, excepciones, cursores y subprogramas a los que se puede hacer referencia desde fuera del paquete. En otras palabras, contiene toda la información sobre el contenido del paquete, pero excluye el código de los subprogramas.

Todos los objetos colocados en la especificación se denominan objetos públicos . Cualquier subprograma que no esté en la especificación del paquete pero que esté codificado en el cuerpo del paquete se denomina objeto privado .

El siguiente fragmento de código muestra una especificación de paquete con un solo procedimiento. Puede tener muchas variables globales definidas y múltiples procedimientos o funciones dentro de un paquete.

CREATE OR REPLACE PACKAGE cust\_sal AS

PROCEDURE find\_sal(c\_id customers.id%type);

END cust\_sal;

/

Cuando el código anterior se ejecuta en el indicador de SQL, produce el siguiente resultado:

Package created.

## Cuerpo del paquete

El cuerpo del paquete tiene los códigos para varios métodos declarados en la especificación del paquete y otras declaraciones privadas, que están ocultas al código fuera del paquete.

La instrucción CREATE PACKAGE BODY se utiliza para crear el cuerpo del paquete. El siguiente fragmento de código muestra la declaración del cuerpo del paquete para el paquete cust\_sal creado anteriormente.

CREATE OR REPLACE PACKAGE BODY cust\_sal AS

PROCEDURE find\_sal(c\_id customers.id%TYPE) IS

c\_sal customers.salary%TYPE;

BEGIN

SELECT salary INTO c\_sal

FROM customers

WHERE id = c\_id;

dbms\_output.put\_line('Salary: '|| c\_sal);

END find\_sal;

END cust\_sal;

/

Cuando el código anterior se ejecuta en el indicador de SQL, produce el siguiente resultado:

Package body created.

Ejemplo

El siguiente programa proporciona un paquete más completo. Usaremos la tabla CLIENTES almacenada en nuestra base de datos con los siguientes registros:

Select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 3000.00 |

| 2 | Khilan | 25 | Delhi | 3000.00 |

| 3 | kaushik | 23 | Kota | 3000.00 |

| 4 | Chaitali | 25 | Mumbai | 7500.00 |

| 5 | Hardik | 27 | Bhopal | 9500.00 |

| 6 | Komal | 22 | MP | 5500.00 |

+----+----------+-----+-----------+----------+

La especificación del paquete

CREATE OR REPLACE PACKAGE c\_package AS

-- Adds a customer

PROCEDURE addCustomer(c\_id customers.id%type,

c\_name customers.name%type,

c\_age customers.age%type,

c\_addr customers.address%type,

c\_sal customers.salary%type);

-- Removes a customer

PROCEDURE delCustomer(c\_id customers.id%TYPE);

--Lists all customers

PROCEDURE listCustomer;

END c\_package;

/

Cuando el código anterior se ejecuta en el indicador de SQL, crea el paquete anterior y muestra el siguiente resultado:

Package created.

Crear el cuerpo del paquete

CREATE OR REPLACE PACKAGE BODY c\_package AS

PROCEDURE addCustomer(c\_id customers.id%type,

c\_name customers.name%type,

c\_age customers.age%type,

c\_addr customers.address%type,

c\_sal customers.salary%type)

IS

BEGIN

INSERT INTO customers (id,name,age,address,salary)

VALUES(c\_id, c\_name, c\_age, c\_addr, c\_sal);

END addCustomer;

PROCEDURE delCustomer(c\_id customers.id%type) IS

BEGIN

DELETE FROM customers

WHERE id = c\_id;

END delCustomer;

PROCEDURE listCustomer IS

CURSOR c\_customers is

SELECT name FROM customers;

TYPE c\_list is TABLE OF customers.Name%type;

name\_list c\_list := c\_list();

counter integer :=0;

BEGIN

FOR n IN c\_customers LOOP

counter := counter +1;

name\_list.extend;

name\_list(counter) := n.name;

dbms\_output.put\_line('Customer(' ||counter|| ')'||name\_list(counter));

END LOOP;

END listCustomer;

END c\_package;

/

Cuando el código anterior se ejecuta en el indicador de SQL, produce el siguiente resultado:

Package body created.

## Usando el paquete

El siguiente programa utiliza los métodos declarados y definidos en el paquete c\_package .

DECLARE

code customers.id%type:= 8;

BEGIN

c\_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);

c\_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);

c\_package.listcustomer;

c\_package.delcustomer(code);

c\_package.listcustomer;

END;

/

Cuando el código anterior se ejecuta en el indicador de SQL, produce el siguiente resultado:

Customer(1): Ramesh

Customer(2): Khilan

Customer(3): kaushik

Customer(4): Chaitali

Customer(5): Hardik

Customer(6): Komal

Customer(7): Rajnish

Customer(8): Subham

Customer(1): Ramesh

Customer(2): Khilan

Customer(3): kaushik

Customer(4): Chaitali

Customer(5): Hardik

Customer(6): Komal

Customer(7): Rajnish

PL/SQL procedure successfully completed

# Transacciones

Oracle es un sistema de base de datos puramente transaccional, de tal forma, que la instrucción BEGIN TRANSACTION no existe.

Una transacción es un conjunto de sentencias SQL que se ejecutan en una base de datos como una única operación, confirmándose o deshaciéndose todo el conjunto de sentencias SQL. La transacción puede quedar finalizada (con las sentencias apropiadas) o implícitamente (terminando la sesión).

Durante la transacción, todas las modificaciones que hagamos sobre base de datos, no son definitivas, más concretamente, se realizan sobre un tablespace especial que se denomina tablespace de ROLLBACK, o RBS (RollBack Segment). Este tablespace tiene reservado un espacio para cada sesión activa en el servidor, y es en ese espacio donde se almacenan todas las modificaciones de cada transacción. Una vez que la transacción se ha finalizado, las modificaciones temporales almacenadas en el RBS, se vuelcan al tablespace original, donde está almacenada nuestra tabla. Esto permite que ciertas modificaciones que se realizan en varias sentencias, se puedan validar todas a la vez, o rechazar todas a la vez.

Dentro de una transacción se pueden crear los llamados “punto de control” mediante la sentencia:

SAVEPOINT Nombre\_punto\_control;

Las sentencias de finalización de transacción son:

* **COMMIT**: la transacción termina correctamente, se vuelcan los datos al tablespace original y se vacía el RBS.
* **ROLLBACK**: se rechaza la transacción y se vacía el RBS. Cualquier cambio realizado desde que se inició la transacción se deshace, quedando la base de datos en el mismo estado que antes de iniciarse la transacción.

A la hora de hacer un ROLLBACK o un COMMIT se podrá hacer hasta cierto punto con la sintaxis:

**COMMIT** **TO** punto\_control;

**ROLLBACK** **TO** punto\_control;

Cuando tenemos abierta una sesión (WorkSheet de Oracle por ejemplo), los cambios que realizamos no son visibles a otra sesión hasta que no hagamos un COMMIT. Este se puede realizar de forma manual, ejecutando el comando COMMIT; o bien, de forma automática, cuando cerramos la sesión.

En una transacción los datos modificados no son visibles por el resto de usuarios hasta que se confirme la transacción.

Si alguna de las tablas afectadas por la transacción tiene [triggers](https://elbauldelprogramador.com/plsql-disparadores-o-triggers/), las operaciones que realiza el [trigger](https://elbauldelprogramador.com/plsql-disparadores-o-triggers/) están dentro del ámbito de la transacción, y son confirmadas o deshechas conjuntamente con la transacción. Durante la ejecución de una transacción, una segunda transacción no podrá ver los cambios realizados por la primera transacción hasta que esta se confirme.

El siguiente ejemplo muestra una supuesta transacción bancaria:

**DECLARE**

importe NUMBER;

ctaOrigen VARCHAR2(23);

ctaDestino VARCHAR2(23);

**BEGIN**

importe :**=** 100;

ctaOrigen :**=** '2530 10 2000 1234567890';

ctaDestino :**=** '2532 10 2010 0987654321';

**UPDATE** CUENTAS **SET** SALDO **=** SALDO – importe **WHERE** CUENTA **=** ctaOrigen;

**UPDATE** CUENTAS **SET** SALDO **=** SALDO **+** importe **WHERE** CUENTA **=** ctaDestino;

**INSERT** **INTO** MOVIMIENTOS **VALUES** (ctaOrigen, ctaDestino, importe**\***(**-**1), SYSDATE);

**INSERT** **INTO** MOVIMIENTOS **VALUES** (ctaDestino,ctaOrigen, importe, SYSDATE);

**COMMIT**;

**EXCEPTION**

**WHEN** OTHERS **THEN**

dbms\_output.put\_line('Error en la transaccion:'**||**SQLERRM);

dbms\_output.put\_line('Se deshacen las modificaciones);

ROLLBACK;

END;

# Secuencias

Otro objeto de base de datos es la secuencia.

Una secuencia (sequence) se emplea para generar valores enteros secuenciales únicos y asignárselos a campos numéricos; se utilizan generalmente para las claves primarias de las tablas garantizando que sus valores no se repitan.

Una secuencia es una tabla con un campo numérico en el cual se almacena un valor y cada vez que se consulta, se incrementa tal valor para la próxima consulta.

Sintaxis general:

***create sequence NOMBRESECUENCIA***

***start with VALORENTERO***

***increment by VALORENTERO***

***maxvalue VALORENTERO***

***minvalue VALORENTERO***

***cycle | nocycle;***

- La cláusula "start with" indica el valor desde el cual comenzará la generación de números secuenciales. Si no se especifica, se inicia con el valor que indique "minvalue".

- La cláusula "increment by" especifica el incremento, es decir, la diferencia entre los números de la secuencia; debe ser un valor numérico entero positivo o negativo diferente de 0. Si no se indica, por defecto es 1.

- "maxvalue" define el valor máximo para la secuencia. Si se omite, por defecto es 99999999999999999999999999.

- "minvalue" establece el valor mínimo de la secuencia. Si se omite será 1.

- La cláusula "cycle" indica que, cuando la secuencia llegue a máximo valor (valor de "maxvalue") se reinicie, comenzando con el mínimo valor ("minvalue") nuevamente, es decir, la secuencia vuelve a utilizar los números. Si se omite, por defecto la secuencia se crea "nocycle".

Si no se especifica ninguna cláusula, excepto el nombre de la secuencia, por defecto, comenzará en 1, se incrementará en 1, el mínimo valor será 1, el máximo será 999999999999999999999999999 y "nocycle".

En el siguiente ejemplo creamos una secuencia llamada "sec\_codigolibros", estableciendo que comience en 1, sus valores estén entre 1 y 99999 y se incrementen en 1, por defecto, será "nocycle":

***create sequence sec\_codigolibros***

***start with 1***

***increment by 1***

***maxvalue 99999***

***minvalue 1;***

Si bien, las secuencias son independientes de las tablas, se utilizarán generalmente para una tabla específica, por lo tanto, es conveniente darle un nombre que referencie a la misma.

Otro ejemplo:

***create sequence sec\_numerosocios***

***increment by 5***

***maxvalue 1000000***

***cycle;***

La secuencia anterior, "sec\_numerosocios", incrementa sus valores en 5 y al llegar al máximo valor recomenzará la secuencia desde el valor mínimo; no se especifican las otras cláusulas, por lo tanto, por defecto, el valor mínimo es 1 y el valor inicial es 1.

Dijimos que las secuencias son tablas; por lo tanto se accede a ellas mediante consultas, empleando "select". La diferencia es que utilizamos pseudocolumnas para recuperar el valor actual y el siguiente de la secuencia. Estas pseudocolumnas pueden incluirse en el "from" de una consulta a otra tabla o de la tabla "dual".

Para recuperar los valores de una secuencia empleamos las pseudocolumnas "currval" y "nextval".

Primero debe inicializarse la secuencia con "nextval". La primera vez que se referencia "nextval" retorna el valor de inicio de la secuencia; las siguientes veces, incrementa la secuencia y nos retorna el nuevo valor:

NOMBRESECUENCIA.NEXTVAL;

se coloca el nombre de la secuencia seguido de un punto y la pseudocolumna "nextval" (que es una forma abreviada de "next value", siguiente valor).

Para recuperar el valor actual de una secuencia usamos:

NOMBRESECUENCIA.CURRVAL;

es decir, el nombre de la secuencia, un punto y la pseudocolumna "currval" (que es una forma abreviada de "current value", valor actual).

Los valores retornados por "currval" y "nextval" pueden usarse en sentencias "insert" y "update".

Veamos un ejemplo completo:

Creamos una secuencia para el código de la tabla "libros", especificando el valor máximo, el incremento y que no sea circular:

***create sequence sec\_codigolibros***

***maxvalue 999999***

***increment by 1***

***nocycle;***

Luego inicializamos la secuencia

***select sec\_codigolibros.nextval from dual;***

Recuerde que la primera vez que se referencie la secuencia debe emplearse "nextval" para inicializarla.

Ingresamos un registro en "libros", almacenando en el campo "codigo" el valor actual de la secuencia:

***insert into libros values***

***(sec\_codigolibros.currval,'El aleph', 'Borges','Emece');***

Ingresamos otro registro en "libros", almacenando en el campo "codigo" el valor siguiente de la secuencia:

***insert into libros values***

***(sec\_codigolibros.nextval,'Matematica estas ahi', 'Paenza','Nuevo siglo');***

Para ver todas las secuencias de la base de datos actual realizamos la siguiente consulta:

***select \*from all\_sequences;***

Nos muestra el propietario de la secuencia, el nombre de la misma, los valores mínimo y máximo, el valor de incremento y si es circular o no, entre otros datos que no analizaremos por el momento.

También podemos ver todos los objetos de la base de datos actual tipeando;

***select \*from all\_objects;***

En la tabla resultado aparecen todos los objetos de la base de datos, incluidas las secuencias; si es una secuencia en la columna OBJECT\_TYPE se muestra "SEQUENCE".

Podemos consultar "all\_objects" especificando que nos muestre el nombre de todas las secuencias:

***select object\_name from all\_objects***

***where object\_type='SEQUENCE';***

Para eliminar una secuencia empleamos "drop sequence". Sintaxis:

drop sequence NOMBRESECUENCIA;

Si la secuencia no existe aparecerá un mensaje indicando tal situación.

En el siguiente ejemplo se elimina la secuencia "sec\_codigolibros":

***drop sequence sec\_codigolibros;***