

## TEMA8

### PL\_SQL. INTRO A LA PROGRAMACIÓ

Bases de Datos  
CFGS DAW

Autor: Raquel Torres  
Revisado Por: Pau Miñana


## Licencia




**Reconocimiento - NoComercial - CompartirIgual (by-nc-sa):** No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

## Revisiones

-

# ÍNDICE DE CONTENIDO

<b>1. Introducción.....</b>	<b>4</b>
1.1 Conceptos básicos.....	5
<b>2. Escritura de PL/SQL.....</b>	<b>5</b>
2.1 Estructura de un bloque PL/SQL.....	5
2.2 Escritura de instrucciones PL/SQL.....	6
2.2.1 Normas básicas.....	6
2.2.2 Comentarios.....	6
<b>3. Variables.....</b>	<b>7</b>
3.1 Tipos de variables.....	7
3.2 Declaración de variables.....	10
3.3 Atributos de tipo.....	11
<b>4. Operadores.....</b>	<b>12</b>
4.1 Operador de asignación.....	13
4.2 Operadores matemáticos.....	13
4.3 Operadores de relación.....	14
4.4 Operadores lógicos.....	14
4.5 Operador de concatenación.....	14
<b>5. Salida de Datos.....</b>	<b>14</b>
5.1 El paquete <i>DBMS_OUTPUT</i> .....	14
5.1.1 Ejemplo 1.....	15
5.1.2 Ejemplo 2.....	15
5.1.3 Ejemplo 3.....	16
<b>6. Estructuras de control.....</b>	<b>17</b>
6.1 Alternativa.....	17
6.1.1 Ejemplo 4.....	18
6.1.2 Ejemplo 5.....	19
6.2 Alternativa múltiple. Sentencia <i>CASE</i> .....	20
6.2.1 Ejemplo 6.....	21
6.2.2 Ejemplo 6a.....	22
6.3 Bucles.....	22
6.3.1 Bucle <i>WHILE</i> .....	22
6.3.1.1 Ejemplo 7.....	23
6.3.1.2 Ejemplo 8.....	23
6.3.2 El bucle <i>FOR</i> .....	24
6.3.2.1 Ejemplo 9.....	24
6.3.2.2 Ejemplo 10.....	25
<b>7. SQL en PL/SQL.....</b>	<b>25</b>
7.1 Instrucción <i>SELECT</i> .....	25
7.2 Instrucciones DML y de transacción.....	26
7.3 Ejemplo 11.....	26
7.4 Ejemplo 12.....	27
7.5 Entrada de datos por teclado.....	27

## UD8 PL\_SQL. INTRO A LA PROGRAMACIÓN

### 1. INTRODUCCIÓN

Casi todos los grandes Sistemas Gestores de Datos incorporan utilidades que permiten ampliar el lenguaje SQL para añadirle mejoras como las que proporciona la programación estructurada (bucles, condiciones, funciones,...etc.).

La razón es que hay diversas acciones a realizar en las bases de datos para las que SQL no es suficiente. Por ello todas las bases de datos incorporan algún lenguaje de tipo procedimental que permite manipular de forma más avanzada los datos de la base de datos.



**PL/SQL** es el lenguaje procedimental implementado por el precompilador de **Oracle**.

PL/SQL (Procedural Language/ Structured Query Language) es una extensión procedimental del lenguaje SQL; es decir, se trata de un lenguaje creado para dar a SQL nuevas posibilidades. Esas posibilidades permiten utilizar condiciones y bucles al estilo de los lenguajes de tercera generación (como Basic, Cobol, C++, Java, etc.).

En otros sistemas gestores de bases de datos existen otros lenguajes procedimentales: SQL Server utiliza Transact SQL, Informix usa Informix 4GL, etc.

La creación de aplicaciones sobre la base de datos se realiza con herramientas como Oracle Developer o con lenguajes externos como Visual Basic o Java.

Lo interesante del lenguaje PL/SQL es que integra SQL, por lo que gran parte de su sintaxis procede de dicho lenguaje. Algunas de sus características son las siguientes:

- Las sentencias SQL de consulta y manipulación de datos pueden ser incluidas en unidades procedurales de código, pero no pueden usarse instrucciones DDL ni DCL.
- Se ejecuta en el lado del servidor y los procedimientos y funciones se almacenan en la BD o en archivos externos.
- Incluye los tipos de datos y los operadores de SQL.

Las funciones más destacadas que pueden realizar los programas PL/SQL son:

- Facilitar la realización de tareas administrativas sobre la base de datos (copia de valores antiguos, auditorías, control de usuarios, etc.)
- Validación y verificación avanzada de usuarios.
- Consultas muy avanzadas.
- Tareas imposibles de realizar con SQL.

### 1.1 Conceptos básicos

- Bloque PL/SQL  
Se trata de un trozo de código que puede ser interpretado por la BD. Se encuentra inmerso dentro de las palabras BEGIN y END.
- Programa PL/SQL  
Conjunto de bloques que realizan una determinada labor.
- Procedimiento  
Programa PL/SQL almacenado en la base de datos y que puede ser ejecutado si se desea con solo saber su nombre (y teniendo permiso para su acceso).
- Función  
Programa PL/SQL que a partir de unos datos de entrada obtiene un resultado (datos de salida). Una función puede ser utilizada desde cualquier otro programa PL/SQL e incluso desde una instrucción SQL.
- Trigger (disparador)  
Programa PL/SQL que se ejecuta automáticamente cuando ocurre un determinado suceso a un objeto de la base de datos.
- Paquete  
Colección de procedimientos y funciones agrupados dentro de la misma estructura. Similar a las bibliotecas y librerías de los lenguajes convencionales.

## 2. ESCRITURA DE PL/SQL

### 2.1 Estructura de un bloque PL/SQL

Ya se ha comentado antes que los programas PL/SQL se agrupan en estructuras llamadas bloques. Cuando un bloque no tiene nombre, se le llama bloque anónimo. Un bloque consta de tres

secciones:

**Declaraciones.** Define e inicializa las variables, constantes, excepciones de usuario y cursores utilizados en el bloque. Va precedida de la palabra **DECLARE**

**Comandos ejecutables.** Sentencias para manipular la base de datos y los datos del programa. Todas estas sentencias van precedidas por la palabra **BEGIN**.

**Tratamiento de excepciones.** Para indicar las acciones a realizar en caso de error. Van precedidas por la palabra **EXCEPTION**.

**Final del bloque.** La palabra **END** da fin al bloque.

Quedando una estructura como esta:

```
[DECLARE
    variables, cursores, excepciones definidas por el usuario ]
BEGIN
    sentencias SQL y sentencias de control PL/SQL
    [EXCEPTION
        instrucciones de manejo de errores ]
END;
```

## 2.2 Escritura de instrucciones PL/SQL

### 2.2.1 Normas básicas

La mayor parte de las normas de escritura en PL/SQL proceden de SQL, además:

- Todas las instrucciones finalizan con el signo del punto y coma (;), excepto las que encabezan un bloque.
- Los bloques comienzan con la palabra **BEGIN** y terminan con **END**.
- Las instrucciones pueden ocupar varias líneas.

### 2.2.2 Comentarios

Pueden ser de dos tipos:

- Comentarios de varias líneas.  
Comienzan con **/\*** y terminan con **\*/**.
- Comentarios de línea simple.

Son los que utilizan los signos -- (doble guión). El texto a la derecha de los guiones se considera comentario (el de la izquierda no).

### 3. VARIABLES



Una **variable** es un espacio de memoria reservado para guardar información.

A dicho espacio de memoria accederemos desde los programas a través del nombre que le hayamos dado. Por supuesto, como su nombre indica es variable, es decir su valor puede variar a lo largo de la ejecución del programa.

#### 3.1 Tipos de variables

Los tipos de datos más utilizados en PL/SQL para las variables son en su mayoría los mismos que estudiamos para SQL. Recordemos algunos de ellos:

Tipo Dato	Descripción
VARCHAR2(size [BYTE   CHAR])	Cadena de caracteres de longitud variable que tiene como tamaño máximo el valor de size en BYTE o CHAR. El tamaño máximo es de 4000 bytes o caracteres, y la mínima es de 1 byte o un carácter. Se debe especificar el tamaño de VARCHAR2.
NVARCHAR2(size)	Cadena de caracteres Unicode de longitud variable con size como máximo tamaño de longitud. El número de bytes que pueden ser hasta dos veces el tamaño de codificación AL16UTF16 y tres veces el tamaño de la codificación UTF8. El tamaño máximo está determinado por la definición del juego de caracteres nacional, con un límite máximo de 4000 bytes. Se debe especificar el size de NVARCHAR2.
NUMBER [ (p [, s]) ]	Número con p precisión (parte entera) y s escala (parte decimal). La precisión p puede variar de 1 a 38. La s escala puede variar desde -84 hasta 127. Tanto la precisión y la escala se encuentran en dígitos decimales. Un valor numérico requiere 1 a 22 bytes.
FLOAT [(p)]	Un subtipo del tipo de datos NUMBER con precisión p. Un valor de coma flotante se representa internamente como un NUMBER. La precisión p puede variar desde 1 hasta 126 dígitos binarios. Un valor flotante requiere 1 a 22 bytes.
LONG	Tipo de datos de caracteres de longitud variable de hasta 2 gigabytes, o $2^{31} - 1$ bytes. Permanece para compatibilidad con versiones anteriores de Oracle.

DATE	Intervalo de fechas válidas del 1 de enero de 4712 antes de Cristo al 31 de diciembre de 9999. El formato por defecto se determina explícitamente por el parámetro NLS_DATE_FORMAT o implícitamente por el parámetro NLS_TERRITORY. El tamaño es de 7 bytes. Este tipo de datos contiene los campos de fecha y hora AÑO, MES, día, hora, minuto y segundo. No tiene fracciones de segundo o de una zona horaria.
BINARY_FLOAT	Número en coma flotante de 32 bits. Este tipo de datos requiere 4 bytes.
BINARY_DOUBLE	Número en coma flotante de 64 bits. Este tipo de datos se requieren de 8 bytes.
TIMESTAMP [(fractional_seconds_precision)]	Año, mes y día como valores de la fecha, así como la hora, minutos y segundos como valores de tiempo, donde fractional_seconds_precision es el número de dígitos en la parte fraccionaria del segundo del campo datetime. Los valores aceptados de fractional_seconds_precision son del 0 al 9. El valor por defecto es 6. El formato por defecto se determina explícitamente por el parámetro NLS_TIMESTAMP_FORMAT o implícitamente por el parámetro NLS_TERRITORY. El tamaño es de 7 o 11 bytes, dependiendo de la precisión. Este tipo de datos contiene los campos datetime AÑO, MES, DIA, HORA, MINUTO y SEGUNDO. Contiene las fracciones de segundo, pero no tiene una zona horaria.
TIMESTAMP [(fractional_seconds_precision)] WITH TIME ZONE	Todos los valores de TIMESTAMP, así como el valor de tiempo de desplazamiento de la zona, donde fractional_seconds_precision es el número de dígitos en la parte fraccionaria del segundo del campo datetime. Los valores aceptados son del 0 al 9. El valor por defecto es 6. El formato por defecto se determina explícitamente por el parámetro NLS_TIMESTAMP_FORMAT o implícitamente por el parámetro NLS_TERRITORY. El tamaño se fija en 13 bytes. Este tipo de datos contiene los campos datetime AÑO, MES, DIA, HORA, MINUTO, SEGUNDO, TIMEZONE_HOUR y TIMEZONE_MINUTE. Cuenta con las fracciones de segundo y una zona horaria explícita.
TIMESTAMP [(fractional_seconds)] WITH LOCAL TIME ZONE	<p>Todos los valores de TIMESTAMP WITH TIME ZONE, con las siguientes excepciones:</p> <ul style="list-style-type: none"> <li>* Los datos se normalizan con la zona horaria de base de datos cuando se almacenan en la base de datos.</li> <li>* Cuando se recuperan los datos, los usuarios ven los datos en la zona de tiempo de la sesión.</li> </ul> <p>El formato por defecto se determina explícitamente por el parámetro NLS_TIMESTAMP_FORMAT o implícitamente por el parámetro NLS_TERRITORY. El tamaño es de 7 o 11 bytes, dependiendo de la precisión.</p>



INTERVAL YEAR [(year_precision)] TO MONTH	Almacena un período de tiempo en años y meses, donde year_precision es el número de dígitos en el campo datetime AÑO. Los valores aceptados son del 0 al 9. El valor predeterminado es 2. El tamaño se fija en 5 bytes.
INTERVAL DAY [(day_precision)] TO SECOND [(fractional_seconds)]	Almacena un período de tiempo en días, horas, minutos y segundos, donde  * day_precision es el número máximo de dígitos en el campo datetime DÍA. Los valores aceptados son del 0 al 9. El valor predeterminado es 2. * fractional_seconds_precision es el número de dígitos en la parte fraccionaria del campo SEGUNDO. Los valores aceptados son del 0 al 9. El valor por defecto es 6.  El tamaño se fija en 11 bytes.
RAW(size)	Datos binarios sin formato de longitud size. El tamaño máximo es de 2000 bytes. Se debe especificar el tamaño de un valor RAW.
LONG RAW	Datos binarios de tipo RAW de longitud variable hasta 2 gigabytes.
ROWID	Cadena en base 64 que representa la dirección única de una fila en la tabla. Este tipo de datos es principalmente para los valores devueltos por la pseudo columna ROWID.
UROWID [(size)]	Cadena en base 64 que representa la dirección lógica de una fila de una tabla de índice organizado. El tamaño opcional es el tamaño de una columna de tipo UROWID de Oracle. El tamaño máximo y por defecto es de 4000 bytes.
CHAR [(size [BYTE   CHAR])]	Cadena de caracteres de longitud fija de size bytes de tamaño o size de caracteres. El tamaño máximo es de 2000 bytes o caracteres, el tamaño predeterminado y mínimo es de 1 byte.
NCHAR[(size)]	Cadena de caracteres de longitud fija de size caracteres de tamaño de largo. El número de bytes pueden ser hasta dos veces el tamaño de codificación AL16UTF16 y tres veces el tamaño de la codificación UTF8. El tamaño máximo está determinado por la definición del juego de caracteres nacional, con un límite máximo de 2000 bytes. El tamaño predeterminado y mínimo es de un carácter.
CLOB	Un objeto de tipo LOB que contiene caracteres de un byte o multibyte. Son compatibles tanto de ancho fijo y conjuntos de ancho variable de caracteres, con el carácter de base de datos establecida. El tamaño máximo es (4 gigabytes - 1) * (tamaño del bloque de la base de datos).
NCLOB	Un objeto de tipo LOB que contiene caracteres Unicode. Son compatible tanto de ancho fijo y conjuntos de ancho variable de caracteres, con el conjunto base de datos de carácter nacional. El

	tamaño máximo es (4 gigabytes - 1) * (tamaño del bloque de la base de datos). Guarda los datos nacionales sobre el conjunto de caracteres.
BLOB	Un objeto de tipo LOB binario. El tamaño máximo es (4 gigabytes - 1) * (tamaño del bloque de la base de datos).
BFILE	Contiene un localizador a un archivo binario almacenado fuera de la base de datos. Permite flujo de bytes de E/S para el acceso a LOB externos que residen en el servidor de base de datos. El tamaño máximo es de 4 gigabytes.

### 3.2 Declaración de variables

Las variables se declaran en el apartado **DECLARE** del bloque. PL/SQL. La sintaxis de la declaración de variables es:

#### **DECLARE**

```
identificador [CONSTANT] tipoDeDatos [NOT NULL][:= valorInicial | DEFAULT  
valorInicial];  
[siguienteVariable... ]
```

El operador `:=` o la palabra **DEFAULT** sirven para asignar valores a una variable. Permiten inicializar la variable con un valor determinado o con la expresión que inicializará la variable que estamos declarando.

La palabra *CONSTANT* indica que la variable no puede ser modificada (es una constante). Si no se inicia la variable, ésta contendrá el valor *NULL*.

Además podemos forzar a que la variable no pueda tener valores nulos colocando *NOT NULL* detrás del tipo de dato.

⚡ Los nombres de las variables comenzarán siempre con una letra y después pueden ir seguidos de letras y números.

Si una variable está formada por dos palabras es aconsejable unir las con un subrayado (underscore `_`), por ejemplo *Mi Numero* quedaría como *Mi\_Numero*.

Además,

⚡ PL/SQL no distingue entre mayúsculas y minúsculas (NO es Case Sensitive)

por lo que podemos escribir las variables en mayúsculas o minúsculas y estaremos haciendo referencia al mismo espacio de memoria. Es decir *MI\_NUMERO*, *Mi\_Numero* y *mi\_numero* hacen referencia a la misma variable.

También Oracle tiene un conjunto de palabras reservadas que no pueden ser utilizadas como nombres de variables, por ejemplo *LOOP*, *BEGIN*, *END*, etc.

⚡ En PL/SQL sólo se puede declarar una variable por línea.

Ejemplos:

```
DECLARE
```

```
pi CONSTANT NUMBER(9,7):=3.1415927;
```

```
radio NUMBER(5);
```

```
area NUMBER(14,2) := 23.12;
```

```
mi_fecha DATE;
```

```
mi_num_dpto NUMBER(2) NOT NULL DEFAULT 10;
```

```
mi_ciudad VARCHAR2(20) := 'TALAVERA';
```

### 3.3 Atributos de tipo

Cuando declaramos una variable también podemos decir que sea del mismo tipo que otra variable que ya existe, para ello se emplean los 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.

¿Y para qué se usan? Declarar las variables que vamos a utilizar en nuestro código es obligatorio. Tal como hemos visto en el punto anterior, podemos definir variables de todos los tipos que se pueden utilizar en las tablas de las bases de datos. Sin embargo, las tablas de datos pueden ser alteradas (como ya sabemos) y se puede cambiar el tipo y el tamaño de los campos, lo cual puede ser un problema que nos obligue a cambiar el código que hemos generado si el tipo de las variables que hemos definido ya no coincide con el de los campos de la tabla con la que vamos a operar.

Para evitar estos problemas y facilitarnos la declaración de las variables que van a recoger los datos de las consultas que vamos a realizar Oracle suministra los atributos de tipo (*%TYPE* y *%ROWTYPE*) que permiten asignar a nuestras variables el mismo tipo y tamaño que tenga un campo de una tabla (*%TYPE*) o bien todas las columnas de la tabla (*%ROWTYPE*).

En este apartado vamos a ver como utilizar *%TYPE* y más adelante, en el apartado de registros, veremos como emplear *%ROWTYPE*.

#### Ejemplos:

```
nom personas.nombre%TYPE;
```

```
precio NUMBER(9,2);
```

```
precio_iva precio%TYPE;
```

La variable *nom* tomará el tipo de datos asignado a la columna *nombre* de la tabla *personas*. La variable *precio\_iva* tomará el tipo de la variable *precio* (es decir *NUMBER(9,2)*).

## 4. OPERADORES

En PL/SQL contamos con operadores de varios tipos:

- Operadores de asignación.
- Operadores matemáticos.
- Operadores de relación.
- Operadores lógicos.
- Operador de concatenación.

### 4.1 Operador de asignación

El operador de asignación nos permite asignar valores a las variables que hemos declarado. El valor que asignamos puede ser un valor fijo, el contenido de otra variable o el resultado de una expresión.



El operador de asignación es el signo dos puntos seguido de un igual ( := )

Por ejemplo:

```
mi_num_dpto := 20;
mi_fecha := '21-10-2012';
mi_ciudad := 'Barcelona';
mi_altitud := 3100;
encontrado := false;
```

Pero para asignar un valor a una variable como resultado de una consulta a la base de datos no utilizaremos el operador anterior, sino que emplearemos dentro de la *SELECT* el predicado *INTO* para dejar el resultado en la variable que especifiquemos. La forma de realizar esta asignación es la siguiente:

```
SELECT AVG(sueldo) INTO Sueldo_Medio
FROM empleados where Dpto = 'IT';
```

La instrucción anterior calcula el sueldo medio de los empleados del departamento de Informática (IT) y lo guarda en la variable *Sueldo\_Medio* (que deberíamos haber declarado previamente) mediante el predicado *INTO*.

### 4.2 Operadores matemáticos

Los operadores matemáticos son los que ya conocemos:

Operadores matemáticos PL/SQL	
+	Suma
-	Resta
*	Producto
/	División
**	Potenciación. 3**2 es 9. (3 elevado a 2)
MOD(Dividendo,Divisor)	Resto de la división entera.

### 4.3 Operadores de relación

Los operadores de relación también son conocidos ya:

Operadores de relación PL/SQL.	
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
=	Igual a
<>	Distinto de (también se puede emplear != )

### 4.4 Operadores lógicos

Lo mismo ocurre con los operadores lógicos que ya conocemos:

Operadores lógicos PL/SQL.	
AND	Y lógico
OR	O lógico
NOT	Negación.

### 4.5 Operador de concatenación

Este operador es nuevo, no lo hemos utilizado aún. El operador de concatenación nos permitirá unir cadenas de caracteres.

Operador de concatenación.	
	Concatenación de cadenas de caracteres.

## 5. SALIDA DE DATOS

### 5.1 El paquete *DBMS\_OUTPUT*

Oracle suministra el paquete *DBMS\_OUTPUT* que nos permite enviar mensajes desde las funciones, los procedimientos y los triggers. La principal utilidad de este paquete es la **depuración** de nuestros programas, poder mostrar valores intermedios durante la ejecución de una acción para ver si todo se desarrolla de forma correcta y así poder detectar errores durante la ejecución.

Para poder visualizar los mensajes es necesario ejecutar el comando:

```
SET SERVEROUTPUT ON
```

Nosotros emplearemos principalmente el comando:

```
DBMS_OUTPUT.PUT_LINE ('texto o variables a mostrar')
```

para mostrar la información de nuestros procedimientos.

### 5.1.1 Ejemplo 1

Vamos a crear el archivo *p101.sql* con el siguiente contenido:

```
SET SERVEROUTPUT ON

BEGIN

    DBMS_OUTPUT.PUT_LINE('HOLA ESTAS CON EL USUARIO: '||user);

END;

/
```

Después ejecutaremos el script y el resultado de la ejecución será:

```
SQL> @ c:\src\p101.sql
HOLA ESTAS CON EL USUARIO: JARDINERIA
PL/SQL procedure successfully completed.
```

Recuerda, el operador `||` servía para concatenar, hemos unido la cadena literal *HOLA ESTAS CON EL USUARIO:* al contenido de la variable *user* que es donde estamos conectados.

⚡ Fijaos que hemos terminado nuestro bloque de programación con el símbolo `/`. Este símbolo es necesario incluirlo al final de nuestros bloques de programación para indicar al intérprete de comandos SQL (sqlplus) que se han acabado las sentencias PL/SQL y hay que ejecutarlas.

Por tanto sólo será necesario incluir `/` al final del bloque cuando ejecutemos desde consola, no será necesario si lo ejecutamos, por ejemplo, desde SQL Developer.

Vamos a crear otro par de ejemplos para practicar con los mensajes que muestre *DBMS\_OUTPUT*.

### 5.1.2 Ejemplo 2

Se trata de declarar dos variables enteras y asignarles un valor inicial (*A* con valor 2 y *B* con valor 5) y una tercera (*NUM*) para guardar los resultados de las operaciones. Después realizar la suma de

esas variables y mostrar el resultado en pantalla.

```
SET SERVEROUTPUT ON

DECLARE

    NUM INT;

    A INT:=2;

    B INT:=5;

BEGIN

    NUM:=A+B;

    DBMS_OUTPUT.PUT_LINE(A||'+'||B||'='||NUM);

END;

/
```

Una vez guardado lo ejecutamos y vemos el resultado.

```
SQL> @ c:\src\pl02.sql
2 + 5 = 7
PL/SQL procedure successfully completed.
```

### 5.1.3 Ejemplo 3

Ahora vamos a declarar dos variables enteras y asignarles un valor inicial (*A* con valor 10 y *B* con valor 2) y una tercera (*NUM*) para guardar los resultados de las operaciones. Después realizaremos las operaciones de suma, resta, multiplicación, división, potenciación y resto, mostrando el resultado de cada una por pantalla.

```
SET SERVEROUTPUT ON

DECLARE

    NUM INT;

    A INT:=10;

    B INT:=2;

BEGIN

    NUM:=A+B;

    DBMS_OUTPUT.PUT_LINE (A||'+'||B||'='||NUM);
```



```
NUM:=A-B;

DBMS_OUTPUT.PUT_LINE (A||' - '||B||' = '||NUM);

NUM:=A*B;

DBMS_OUTPUT.PUT_LINE (A||' X '||B||' = '||NUM);

NUM:=A/B;

DBMS_OUTPUT.PUT_LINE (A||' / '||B||' = '||NUM);

NUM:=A**B;

DBMS_OUTPUT.PUT_LINE (A||' ^ '||B||' = '||NUM);

NUM:=MOD(A,B);

DBMS_OUTPUT.PUT_LINE (A||' MOD '||B||' = '||NUM);

END;

/
```

Una vez guardado, lo ejecutamos y vemos el resultado:

```
SQL> @ c:\src\pl03.sql
10 + 2 = 12
10 - 2 = 8
10 X 2 = 20
10 / 2 = 5
10 ^ 2 = 100
10 MOD 2 = 0

PL/SQL procedure successfully completed.
```

## 6. ESTRUCTURAS DE CONTROL

PL/SQL es un lenguaje de programación y cuenta con las estructuras de control típicas de los lenguajes: alternativas, alternativas múltiples, bucles, etc. Veamos cómo utilizarlas.

### 6.1 Alternativa

La sentencia alternativa es el *IF*, su sintaxis es la siguiente:

```
IF condición THEN
    instrucciones;
[ELSIF condición THEN
```

```
        instrucciones;]  
[ELSE  
        instrucciones;]  
END IF;
```

Las condiciones estarán formadas por los operadores de relación y los operadores lógicos que vimos anteriormente. Los elementos que van entre corchetes, como siempre, son opcionales, es decir, opcionalmente puede llevar cláusulas ELSIF y opcionalmente puede llevar un ELSE final en el que entrará si no ha entrado por ninguna otra opción anterior.

#### 6.1.1 Ejemplo 4

Se trata de un ejemplo que declara dos variables enteras y nos dice cuál es mayor o si son iguales.

```
SET SERVEROUTPUT ON  
  
DECLARE  
  
    A INT:=5;  
    B INT:=5;  
  
BEGIN  
  
    IF A > B THEN  
        DBMS_OUTPUT.PUT_LINE(' EL MAYOR ES A QUE VALE: '||A);  
    ELSIF B > A THEN  
        DBMS_OUTPUT.PUT_LINE(' EL MAYOR ES B, Y VALE: '||B);  
    ELSE  
        DBMS_OUTPUT.PUT_LINE(' SON IGUALES Y VALEN: '||A);  
    END IF;  
  
END;  
  
/
```

El resultado de la ejecución será:

```
SQL> @ c:\src\pl04.sql
SON IGUALES Y VALEN:5
PL/SQL procedure successfully completed.
```

Prueba a cambiar los valores de las variables por otros diferentes y ejecuta de nuevo el script.

### 6.1.2 Ejemplo 5

¿Qué hace este Script? Pruébalo.

```
SET SERVEROUTPUT ON
DECLARE
    nota NUMBER(3,1):=6.5;
BEGIN
    IF nota < 5 and nota >0 THEN
        DBMS_OUTPUT.PUT_LINE(' Insuficiente: '||nota);
    ELSIF nota >=5 AND nota < 6 THEN
        DBMS_OUTPUT.PUT_LINE(' Suficiente: '||nota);
    ELSIF nota >=6 AND nota < 7 THEN
        DBMS_OUTPUT.PUT_LINE(' Bien: '||nota);
    ELSIF nota >=7 AND nota < 9 THEN
        DBMS_OUTPUT.PUT_LINE(' Notable: '||nota);
    ELSIF nota >=9 AND nota <= 10 THEN
        DBMS_OUTPUT.PUT_LINE(' Sobresaliente: '||nota);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Nota incorrecta: '||nota);
    END IF;
END;
/
```

Resultado de la ejecución:

```
SQL> @ c:\src\pl05.sql
Bien: 6.5
PL/SQL procedure successfully completed.
```

Bien, seguro que no has tenido mucha dificultad para ver lo que hace, pero te presento aquí otra versión del ejemplo:

```
SET SERVEROUTPUT ON
DECLARE
    nota NUMBER(3,1):=6.5;
BEGIN
    IF nota < 5 and nota >0 THEN
        DBMS_OUTPUT.PUT_LINE(' Insuficiente: '||nota);
    ELSIF nota < 6 THEN
        DBMS_OUTPUT.PUT_LINE(' Suficiente: '||nota);
    ELSIF nota < 7 THEN
        DBMS_OUTPUT.PUT_LINE(' Bien: '||nota);
    ELSIF nota < 9 THEN
        DBMS_OUTPUT.PUT_LINE(' Notable: '||nota);
    ELSIF nota <= 10 THEN
        DBMS_OUTPUT.PUT_LINE(' Sobresaliente: '||nota);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Nota incorrecta: '||nota);
    END IF;
END;
/
```

¿Hace lo mismo que la versión anterior? ¿Por qué?

## 6.2 Alternativa múltiple. Sentencia CASE

La sintaxis de la sentencia CASE es:

```
CASE [expresion]
WHEN {condicion1|valor1} THEN
    bloque_instrucciones_1
WHEN {condicion2|valor2} THEN
    bloque_instrucciones_2
....
ELSE
    bloque_instrucciones_por_defecto
END CASE;
```

Esta instrucción de PL/SQL es muy versátil, más que los CASE de otros lenguajes de programación. Esta instrucción puede comparar el valor de una variable o el resultado de una expresión con los valores que siguen a la palabra reservada *WHEN*, o bien puede omitir ese valor o expresión inicial y plantear condiciones independientes en cada uno de los *WHEN*.

### 6.2.1 Ejemplo 6

```
SET SERVEROUTPUT ON
DECLARE
    nota INT:=8;
BEGIN
    CASE nota
        WHEN 0 THEN
            DBMS_OUTPUT.PUT_LINE(' Insuficiente:' || nota);
        WHEN 1 THEN
            DBMS_OUTPUT.PUT_LINE(' Insuficiente:' || nota);
        WHEN 2 THEN
            DBMS_OUTPUT.PUT_LINE(' Insuficiente:' || nota);
        WHEN 3 THEN
            DBMS_OUTPUT.PUT_LINE(' Insuficiente:' || nota);
        WHEN 4 THEN
            DBMS_OUTPUT.PUT_LINE(' Insuficiente:' || nota);
        WHEN 5 THEN
            DBMS_OUTPUT.PUT_LINE(' Suficiente:' || nota);
        WHEN 6 THEN
            DBMS_OUTPUT.PUT_LINE(' Bien:' || nota);
        WHEN 7 THEN
            DBMS_OUTPUT.PUT_LINE(' Notable:' || nota);
        WHEN 8 THEN
            DBMS_OUTPUT.PUT_LINE(' Notable:' || nota);
        WHEN 9 THEN
            DBMS_OUTPUT.PUT_LINE(' Sobresaliente:' || nota);
        WHEN 10 THEN
            DBMS_OUTPUT.PUT_LINE(' Sobresaliente:' || nota);
        ELSE
            DBMS_OUTPUT.PUT_LINE(' Nota incorrecta:' || nota);
    END CASE;
END;
/
```

El resultado será:

```
SQL> @ c:\src\pl06.sql
Notable:8
PL/SQL procedure successfully completed.
```

### 6.2.2 Ejemplo 6a

Veamos el mismo ejercicio pero utilizando el *CASE* con condiciones:

```
SET SERVEROUTPUT ON
DECLARE
    nota INT:=8;
BEGIN
    CASE
        WHEN nota >=0 and nota < 5 THEN
            DBMS_OUTPUT.PUT_LINE(' Insuficiente:' || nota);
        WHEN nota < 6 THEN
            DBMS_OUTPUT.PUT_LINE(' Suficiente:' || nota);
        WHEN nota < 7 THEN
            DBMS_OUTPUT.PUT_LINE(' Bien:' || nota);
        WHEN nota < 9 THEN
            DBMS_OUTPUT.PUT_LINE(' Notable:' || nota);
        WHEN nota <=10 THEN
            DBMS_OUTPUT.PUT_LINE(' Sobresaliente:' || nota);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Nota incorrecta: ' || nota);
    END CASE;
END;
/
```

Si guardáis y ejecutáis veréis que se obtiene el mismo resultado.

## 6.3 Bucles

Aunque PL/SQL tiene más tipos de bucles, en este curso solamente vamos a presentar el bucle *WHILE* y el *FOR*, ya que con ellos podremos cubrir todas las necesidades que no van a aparecer.

### 6.3.1 Bucle *WHILE*

Este bucle ejecuta las instrucciones que están incluidas en él mientras la condición sea verdadera. Su sintaxis es la siguiente:

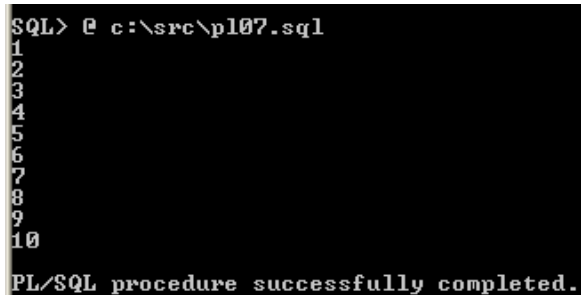
```
WHILE condición LOOP
    instrucciones;
.....
END LOOP;
```

### 6.3.1.1 Ejemplo 7

Vamos a realizar un programa que muestre en pantalla del 1 al 10.

```
SET SERVEROUTPUT ON
DECLARE
    NUM INT:=1;
BEGIN
    WHILE NUM<=10 LOOP
        DBMS_OUTPUT.PUT_LINE(NUM);
        NUM:=NUM+1;
    END LOOP;
END;
/
```

El resultado será:



```
SQL> @ c:\src\pl07.sql
1
2
3
4
5
6
7
8
9
10
PL/SQL procedure successfully completed.
```

### 6.3.1.2 Ejemplo 8

Ahora vamos a mostrar en pantalla los números impares comprendidos entre 1 y 10.

```
SET SERVEROUTPUT ON
DECLARE
    NUM INT:=1;
BEGIN
    WHILE NUM<=10 LOOP
        DBMS_OUTPUT.PUT_LINE(NUM);
        NUM:=NUM+2;
    END LOOP;
END;
/
```

Resultado de la ejecución:

```
$SQL> @ c:\src\pl08.sql
1
3
5
7
9
PL/SQL procedure successfully completed.
```

¿Cómo deberías modificar el código si en lugar de los impares entre 1 y 10 queremos mostrar los impares entre 1 y 100? ¿Y entre 100 y 200?

### 6.3.2 El bucle *FOR*

El bucle *FOR* se emplea cuando sabemos el número de repeticiones que debe realizar el bucle. La sintaxis es la siguiente:

```
FOR índice IN [REVERSE] valor_inicial .. valor_final LOOP
instrucciones;
.....
END LOOP;
```

La variable índice no es necesario declararla de forma explícita, al utilizarla en el bucle se declara de forma implícita pero solo estará disponible dentro del bucle. Los valores iniciales (*valor\_inicial*) y finales (*valor\_final*) del bucle también pueden ser variables o expresiones matemáticas.

#### 6.3.2.1 Ejemplo 9

Mostrar en pantalla los números del 1 al 10 con un bucle *FOR*.

```
SET SERVEROUTPUT ON
BEGIN
    FOR I IN 1 .. 10 LOOP
        DBMS_OUTPUT.PUT_LINE(I);
    END LOOP;
END;
/
```

El resultado será:



```
SQL> @ c:\src\pl09.sql
1
2
3
4
5
6
7
8
9
10
PL/SQL procedure successfully completed.
```

### 6.3.2.2 Ejemplo 10

Ahora vamos a mostrar del 10 al 1 utilizando un bucle *FOR*. Para ello emplearemos la opción *REVERSE*.

```
SET SERVEROUTPUT ON
BEGIN
    FOR I IN REVERSE 1 .. 10 LOOP
        DBMS_OUTPUT.PUT_LINE(I);
    END LOOP;
END;
/
```

El resultado será:

```
SQL> @ c:\src\pl10.sql
10
9
8
7
6
5
4
3
2
1
PL/SQL procedure successfully completed.
```

## 7. SQL EN PL/SQL

Podemos utilizar instrucciones SQL en nuestro código para obtener información de la base de datos o para actualizarla. Por ahora para consultas que solamente devuelvan un valor o una fila. En breve veremos cómo trabajar con consultas que devuelvan un conjunto de filas y su tratamiento (cursores).

### 7.1 Instrucción *SELECT*

PL/SQL admite el uso de un *SELECT* que permite almacenar valores en variables. Es el llamado *SELECT INTO*.

Su sintaxis es:

```
SELECT listaDeCampos INTO listaDeVariables
FROM tabla
[JOIN ...]
[WHERE condición]
```

La cláusula *INTO* es obligatoria en PL/SQL y además la expresión *SELECT con lo visto* sólo puede devolver una única fila ya que la almacenamos en una variable; ampliaremos funcionalidades cuando veamos cursores. De momento veamos un ejemplo:

```
DECLARE
    v_salario NUMBER(9,2);
    v_nombre VARCHAR2(50);
BEGIN
    SELECT salario,nombre INTO v_salario, v_nombre
    FROM empleados WHERE id_empleado=12344;
    SYSTEM_OUTPUT.PUT_LINE ('El nuevo salario de ' || v_nombre || ' será de ' || v_salario*1.2 ||
    'euros');
END;
/
```

## 7.2 Instrucciones DML y de transacción

Se pueden utilizar instrucciones *DML* dentro del código ejecutable. Se permiten las instrucciones *INSERT*, *UPDATE*, *DELETE* y *MERGE*; con la ventaja de que en PL/SQL pueden utilizar variables. Las instrucciones de transacción *ROLLBACK* y *COMMIT* también están permitidas para anular o confirmar instrucciones.

## 7.3 Ejemplo 11

Si estamos trabajando con la base de datos de *Jardinería* podríamos comprobar el precio medio de todos los artículos:

```
SET SERVEROUTPUT ON
DECLARE
```

```
PRECIO_MEDIO NUMBER(10,2):=0;
BEGIN
    SELECT AVG(PRECIOVENTA) INTO PRECIO_MEDIO
    FROM PRODUCTOS;
    DBMS_OUTPUT.PUT_LINE('El precio medio es: '||PRECIO_MEDIO);
END;
/
```

El resultado será:

```
SQL> select avg<precioventa> precio_medio from productos;
PRECIO_MEDIO
-----
    22,5942029

SQL> @ c:\src\pl15.sql
El precio medio es: 22,59

PL/SQL procedure successfully completed.
```

Fíjate que el valor medio calculado con la *SELECT* se coloca en la variable *PRECIO\_MEDIO* mediante la palabra reservada *INTO* dentro de la *SELECT*.

## 7.4 Ejemplo 12

En este ejemplo vamos a crear 5 nuevas gamas con los nombres *GAMA\_1*, *GAMA\_2*, ... *GAMA\_5*, y con las descripciones de "Esta es la gama n" donde n será el número correspondiente. (Utilizaremos un bucle *WHILE* para el ejercicio).

```
SET SERVEROUTPUT ON
DECLARE
    CONT INT := 1;
BEGIN
    WHILE CONT <= 5 LOOP
        INSERT INTO GAMASPRODUCTOS (GAMA, DESCRIPCIONTEXTO)
        VALUES ('GAMA_'||CONT,'Esta es la gama '|| CONT);
        CONT := CONT + 1;
    END LOOP;
END;
/
```

El resultado será:

```
SQL> @ c:\src\pl16.sql
PL/SQL procedure successfully completed.
SQL> SELECT GAMA,DESCRIPCIONTEXTO FROM GAMASPRODUCTOS
  2  WHERE GAMA LIKE 'GAMA%';

GAMA
-----
DESCRIPCIONTEXTO
-----
GAMA_1
Esta es la gama 1
GAMA_2
Esta es la gama 2
GAMA_3
Esta es la gama 3
GAMA_4
Esta es la gama 4
GAMA_5
Esta es la gama 5
```

Comprobamos el resultado para verificar que se han realizado los *INSERT* deseados.

### 7.5 Entrada de datos por teclado

Si se desea introducir mediante el teclado el valor de alguna variable en el momento de la ejecución esto puede realizarse mediante el uso de:

**&Nombre**

De este modo se muestra en la consola "Enter value for Nombre:" y podemos escribir el valor que deseamos. Esto substituye en el momento de la ejecución "Nombre" por el valor que hayamos escrito.

Se debe tener presente que para escribir texto lo deberemos hacer entre comillas.

Su uso más habitual es para introducir valores de variables y para especificar condiciones en una consulta. Esto último puede realizarse en cualquier consulta habitual incluso fuera de los bloques o programas de PL/SQL.

Vamos a ver un par de ejemplos:

En este primer ejemplo simplemente se almacena el nombre que se escriba en la variable nombre y se muestra por la pantalla un saludo con ese nombre. Además se realiza otra vez directamente en el output sin almacenar el nombre en ningún sitio

```
SET SERVEROUTPUT ON
DECLARE
    NOMBRE VARCHAR(10):= &Escribe_tu_Nombre;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hola, ' || Nombre);
    DBMS_OUTPUT.PUT_LINE('Directamente tambien se puede:');
    DBMS_OUTPUT.PUT_LINE('Hola, ' || &Escribe_Otro_Nombre);
END;
/
```

```

SQL> @ /usr/lib/oracle/xe/oradata/XE/nombre.sql
Enter value for escribe_tu_nombre: 'tu nombre'
old 2: NOMBRE VARCHAR(10):=&Escribe_tu_Nombre;
new 2: NOMBRE VARCHAR(10):='tu nombre';
Enter value for escribe_otro_nombre: 'otro nombre'
old 6: DBMS_OUTPUT.PUT_LINE('Hola, ' || &Escribe_Otro_Nombre);
new 6: DBMS_OUTPUT.PUT_LINE('Hola, ' || 'otro nombre');
Hola, tu nombre
Directamente tambien se puede:
Hola, otro nombre

PL/SQL procedure successfully completed.

```

Como se puede observar, el programa simplemente pregunta antes de empezar por todos los parámetros con & y substituye en el código por el texto introducido por el teclado.

Vamos ahora con un segundo ejemplo en el que se realiza una consulta en la tabla de empleados de la BD teoriaud6. Se pregunta ahora tanto el campo que mostrará la consulta como el dpto que se usará para filtrar el resultado. Aunque como "resultado" se declara de tipo DNI se está forzando a que el campo a consultar sea DNI o alguno con un formato compatible.

```

SET SERVEROUTPUT ON
DECLARE
    resultado empleados.DNI%TYPE;
BEGIN
    select &campo into resultado from empleados where dpto=&departamento;
    DBMS_OUTPUT.PUT_LINE(resultado);
END;
/

```

```

SQL> @ /usr/lib/oracle/xe/oradata/XE/consulta.sql
Enter value for campo: DNI
Enter value for departamento: 'CONT'
old 4:      select &campo into resultado from empleados where dpto=&departamento;
new 4:      select DNI into resultado from empleados where dpto='CONT';
12345678A

PL/SQL procedure successfully completed.

```

Atención al campo DNI; puesto que se sustituye directamente en la consulta, en este caso no se usan comillas.

Para finalizar, cómo se ha comentado, consultas como la anterior se pueden ejecutar en un script o comando de SQL cualquiera (sin el INTO), el uso de estos parámetros de entrada no está limitado a PL/SQL.

```

SQL> select &campo from empleados where dpto=&departamento;
Enter value for campo: nombre
Enter value for departamento: 'IT'
old 1: select &campo from empleados where dpto=&departamento
new 1: select nombre from empleados where dpto='IT'

NOMBRE
-----
Mariano Sanz
Ana Silvan
Rafael Colmenar

```