

## UT8. FUNDAMENTOS DEL LENGUAJE PL/SQL

Este lenguaje, basado en el lenguaje ADA, incorpora todas las características propias de los lenguajes de tercera generación: manejo de variables, estructura modular (procedimientos y funciones), estructuras de control (bifurcaciones, bucles y demás estructuras), control de excepciones. También incorpora un completo soporte para la Programación Orientada a Objetos (POO), por lo que puede ser considerado como un lenguaje procedimental y orientado a objetos.

Los programas creados con PL/SQL se pueden almacenar en la base de datos como cualquier otro objeto quedando así disponibles para su ejecución por los usuarios.

### 1. TIPOS DE DATOS

Podremos trabajar con los principales tipos de datos de Oracle vistos en SQL, no existe una equivalencia exacta entre los tipos de PL/SQL y los mismos tipos de las columnas de Oracle, especialmente en lo relativo a las longitudes máximas que pueden almacenar.

CHAR	Cadena de caracteres de longitud fija. De 1 a 32767 caracteres. El dato se rellena con blancos a la derecha hasta alcanzar la longitud definida Si no damos longitud permite un único carácter
VARCHAR2	Cadena de caracteres de longitud variable De 1 a 32767 caracteres. Siempre hay que darle longitud. (Existe el tipo VARCHAR y funciona como VARCHAR2 pero Oracle recomienda el uso de este último por compatibilidad con futuras versiones)
NUMBER	Para datos numéricos, números enteros y reales. Admite hasta 38 dígitos Puede indicarse la precisión deseada NUMBER (p,s) “p” es el total de dígitos y “s” es el número de decimales.
DATE	Tipo de datos fecha Incluye día, mes, año, hora, minutos y segundos.
LONG	Cadena de caracteres de longitud variable. Longitud máxima 2Gb

INTEGER INT	Para números enteros
DECIMAL(p,s)	Como NUMBER
FLOAT REAL SMALLINT	Numéricos

Y otros propios de PL/SQL

BINARY_INTEGER	Numérico que se utiliza para índices, contadores Admite valores entre -21,47483647 y + 2747483647
BOOLEAN	Almacena valores TRUE, FALSE y NULL

Se pueden hacer conversiones entre datos utilizando las funciones correspondientes:  
TO\_CHAR, TO\_NUMBER, TO\_DATE

PL/SQL dispone además de tipos de datos más complejos como registros, tablas y arrays.

## 2. IDENTIFICADORES

Para nombrar variables, constantes, funciones, procedimientos...

Pueden tener hasta 30 caracteres, empezando siempre por letra, que puede ir seguida por letras, números, \$, #, \_

No diferencia entre minúsculas y mayúsculas.

## 3. VARIABLES

Por cada variable se tiene que especificar el tipo, no admite lista de variables separadas por comas y después el tipo como en otros lenguajes.

```
Importe NUMBER (6,2);
Nombre VARCHAR2(30);
Existe BOOLEAN;
```

A la vez que se declara se puede inicializar y determinar que no puede ser nula.

```
Contador NUMBER (3) := 0;
Suma NUMBER (3) NOT NULL := 0;
```

## 4. Uso de atributos %TYPE y %ROWTYPE

Permiten indicar el tipo de una variable para que sean del mismo tipo de otros objetos ya definidos.

- %TYPE el mismo tipo de otra variable ya definida, o que una columna de una tabla
- %ROWTYPE crea una variable de registro cuyos campos se corresponden con las columnas de una tabla

```

Importe NUMBER (6,2);
Total importe%TYPE;      -- declara total como del mismo tipo que importe

Vemple EMPLE%ROWTYPE ;
    -- crea una variable que podrá contener una fila de la tabla EMPLE
    -- para hacer referencia a cada uno de los campos variable.campo
    Vemple.num_emple
    
```

## 5. CONSTANTES

Se declaran como variables pero se le añade la palabra reservada **CONSTANT** y siempre hay que asignarles un valor en la declaración que después no se podrá cambiar.

```
PI  CONSTANT REAL := 3.1416;
```

## 6. OPERADORES

- **Asignación:**                    :=  
                   Por ejemplo     importe:=40;
- **Lógicos:**                    **AND, OR, NOT**
- **Concatenación:**            ||
- **Comparación:**               =, !=, <>, <, >, <=, >=,  
                                   **IS NULL, BETWEEN, LIKE, IN**
- **Aritméticos:**               +, -, \*, /, \*\*

Incluyendo que algunos se pueden utilizar con fechas:

f1 – f2     devuelve el número de días entre f1 y f2

f + n     devuelve la fecha resultado de sumar n días a f

f - n     devuelve la fecha resultado de restar n días a f

## 7. Estructuras de control IF

- **Alternativa simple:**  
     **IF** <condición> **THEN**  
         Instrucciones;  
         ...  
     **END IF;**

- **Alternativa doble:**  

```

IF <condición> THEN
    Instrucciones;
    ...
ELSE
    Instrucciones;
    ...
END IF;
        
```
- **Alternativa múltiple:**  

```

IF <condición1> THEN
    Instrucciones1;
    ...
ELSIF
    <condición2> THEN
    Instrucciones2;
    ...
ELSIF
    <condición3> THEN
    Instrucciones3;
    ...
...
[ELSE
    Instrucciones;
    ...;]
END IF;
        
```

## 8. Bucles

- **LOOP**  

```

Instrucciones;
...
IF <condición> THEN
    EXIT;
END IF;
Instrucciones;
...
END LOOP;
        
```

También podemos salir del bucle con **EXIT WHEN** <condición>

- **WHILE** <condición> **LOOP**  
Instrucciones;  
...  
**END LOOP;**
- **FOR** <variablecontrol> **IN** <valorInicio> .. <valorFinal> **LOOP**  
Instrucciones;  
...  
**END LOOP;**

La <variablecontrol> es una variable que se declara de manera implícita como variable local al bucle de tipo BYNARY\_INTEGER, por lo que nosotros ni podemos declararla ni utilizarla fuera del for. Se va incrementando de unidad en unidad en cada iteración del bucle. Se puede utilizar dentro del bucle pero no asignarle valor.

El incremento siempre es en una unidad pero puede ser negativo utilizando la opción **REVERSE**

```
FOR <variablecontrol> IN REVERSE <valorInicio> .. <valorFinal> LOOP
    Instrucciones;
...
END LOOP;
```

## 9. BLOQUES PL/SQL

El bloque es la estructura básica característica de todos los programas PL/SQL. Tiene tres zonas claramente definidas:

- Una zona de declaraciones donde se declaran objetos locales (variables, constantes, etc). Suele ir precedida por la cláusula DECLARE (o IS/AS en los procedimientos y funciones). Es opcional.
- Un conjunto de instrucciones precedido por la cláusula BEGIN.
- Una zona de tratamiento de excepciones precedido por la cláusula EXCEPTION. Es opcional.

E[ formato genérico del bloque es:

```
[ DECLARE
<declaraciones>]
BEGIN
<órdenes>
[ EXCEPTION
<gestión de excepciones>]
END;
```

Vamos a utilizar la función **DBMS\_OUTPUT.PUT\_LINE ()** para mostrar mensajes por pantalla.

Para que estos sean visibles hay que poner el parámetro **SERVEROUTPUT** a **ON**

Al comenzar la sesión haremos:

**SET SERVEROUTPUT ON**

**Podemos realizar bloques anidados**

```
DECLARE
<declaraciones>
BEGIN
<órdenes>
    DECLARE
    <declaraciones>
    BEGIN
    <órdenes>
    EXCEPTION
    <gestión de excepciones>
    END;

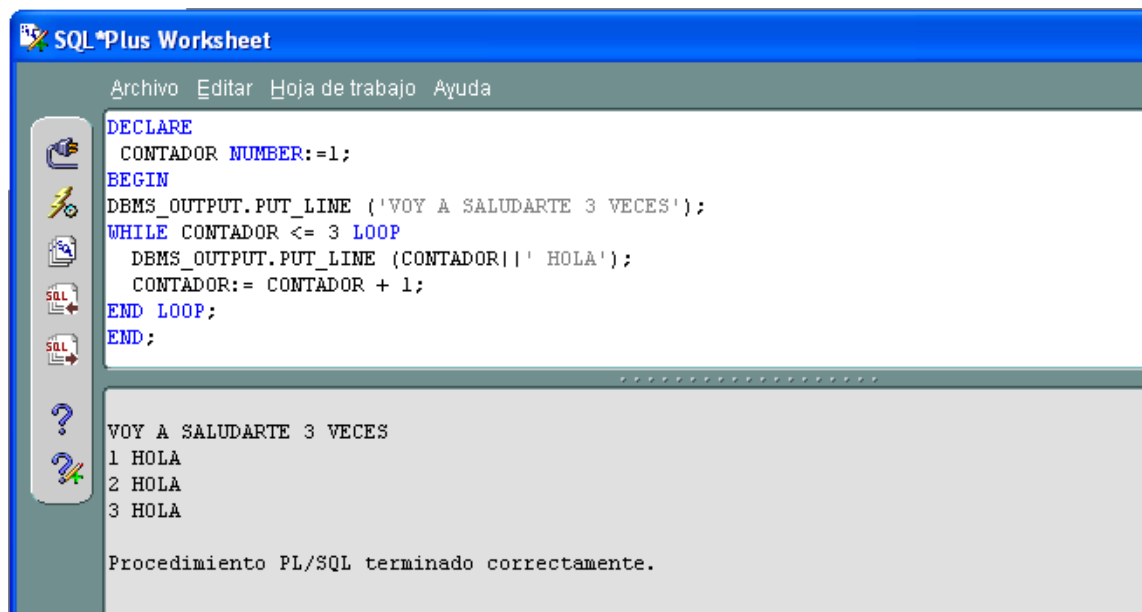
...
EXCEPTION
<gestión de excepciones>
END;
```

Podemos utilizar cualquiera de las funciones vistas en SQL (de cadenas de caracteres, fechas, numéricas...)

Podemos utilizar la instrucción **NULL** para indicar que no se realiza ninguna instrucción.

## 10. Bloques anónimos

Son bloques que se crean y ejecutan en el momento pero no quedan almacenados.



```

SQL*Plus Worksheet
Archivo  Editar  Hoja de trabajo  Ayuda

DECLARE
  CONTADOR NUMBER:=1;
BEGIN
  DBMS_OUTPUT.PUT_LINE ('VOY A SALUDARTE 3 VECES');
  WHILE CONTADOR <= 3 LOOP
    DBMS_OUTPUT.PUT_LINE (CONTADOR||' HOLA');
    CONTADOR:= CONTADOR + 1;
  END LOOP;
END;

VOY A SALUDARTE 3 VECES
1 HOLA
2 HOLA
3 HOLA

Procedimiento PL/SQL terminado correctamente.
    
```

## 11. Subprogramas: procedimientos y funciones

Los subprogramas son bloques PL/SQL que tienen un nombre, pueden recibir parámetros y, en el caso de las funciones, también devolver un valor.

Se guardan en la base de datos y podemos ejecutarlos invocándolos desde otros subprogramas o herramientas.

En todo subprograma podemos distinguir:

- **La cabecera o especificación del subprograma**  
 Contiene el nombre del subprograma  
 La definición de los parámetros con sus tipos (nunca los tamaños)  
 En el caso de las funciones el tipo del valor de retorno
- **El cuerpo del subprograma**  
 Es un bloque PL/SQL que incluye declaraciones (opcional), instrucciones y manejo de excepciones (opcional)

## PROCEDIMIENTOS

Tienen la siguiente estructura general:

```

PROCEDURE <nombreprocedimiento> [( <lista de parámetros> )]
AS (o IS)
    <declaraciones>;
BEGIN
    <instrucciones>;
[EXCEPTION
    <excepciones>;]
    
```

END [<nombreprocedimiento>];

Para crear un procedimiento lo haremos utilizando la orden CREATE

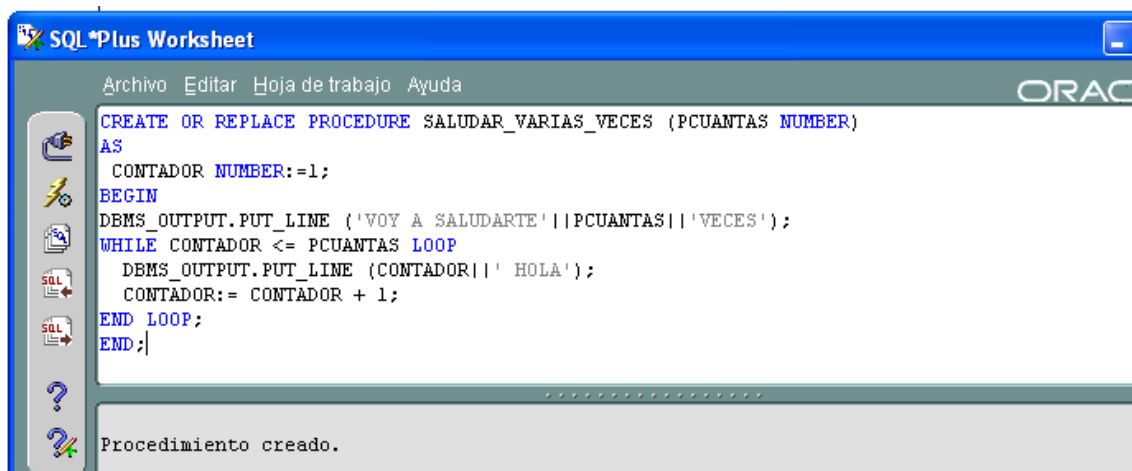
CREATE [OR REPLACE] PROCEDURE <nombreprocedimiento>

```

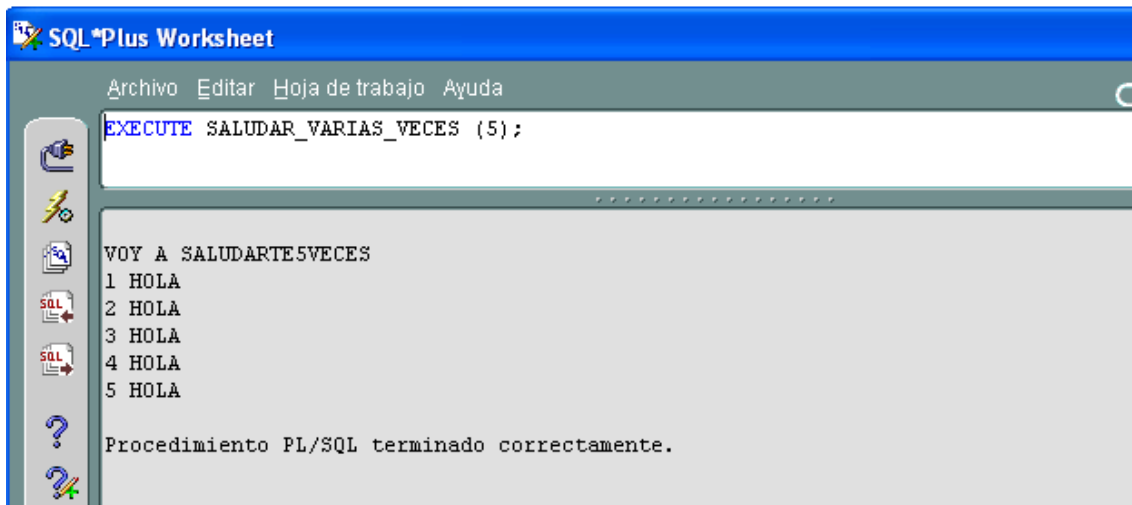
CREATE OR REPLACE PROCEDURE SALUDAR_VARIAS_VECES (PCUANTAS NUMBER)
AS
  CONTADOR NUMBER:=1;
BEGIN
  DBMS_OUTPUT.PUT_LINE ('VOY A SALUDARTE VECES');
  WHILE CONTADOR <= PCUANTAS LOOP
    DBMS_OUTPUT.PUT_LINE (CONTADOR||' HOLA');
    CONTADOR:= CONTADOR + 1;
  END LOOP;
END;
```

Para ejecutar un procedimiento hay que invocarlo desde cualquier herramienta Oracle, por ejemplo desde SQL

EXECUTE SALUDAR\_VARIAS\_VECES (5);







SQL\*Plus Worksheet

Archivo Editar Hoja de trabajo Ayuda

```
EXECUTE SALUDAR_VARIAS_VECES (5);
```

VOY A SALUDARTESVECES

```
1 HOLA
2 HOLA
3 HOLA
4 HOLA
5 HOLA
```

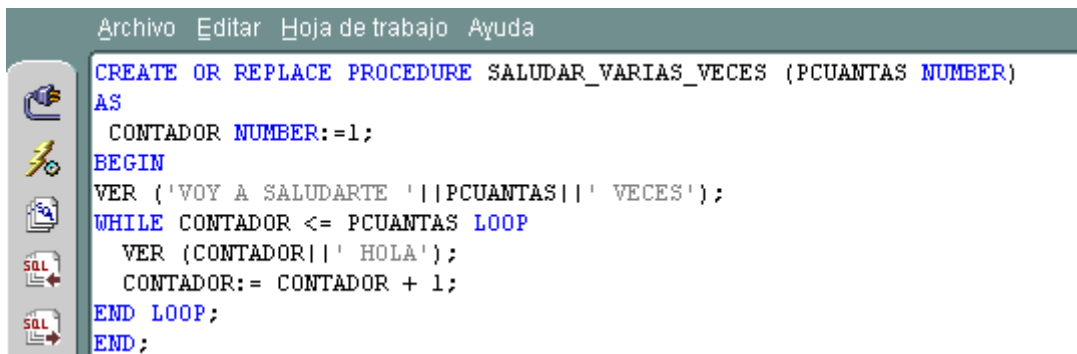
Procedimiento PL/SQL terminado correctamente.

También se puede invocar a un procedimiento o función desde otro procedimiento o función.

Crearemos este procedimiento para que nos sea más cómodo mostrar datos por pantalla.

```
CREATE OR REPLACE PROCEDURE VER (A VARCHAR2)
AS
BEGIN
  DBMS_OUTPUT.PUT_LINE (A);
END;
```

Y ahora lo utilizamos en nuestro procedimiento anterior.



Archivo Editar Hoja de trabajo Ayuda

```
CREATE OR REPLACE PROCEDURE SALUDAR_VARIAS_VECES (PCUANTAS NUMBER)
AS
  CONTADOR NUMBER:=1;
BEGIN
  VER ('VOY A SALUDARTE '||PCUANTAS||' VECES');
  WHILE CONTADOR <= PCUANTAS LOOP
    VER (CONTADOR||' HOLA');
    CONTADOR:= CONTADOR + 1;
  END LOOP;
END;
```