

### **Prácticas sobre conceptos básicos de: PL-SQL**

#### **(Subprogramas : procedimientos y funciones)**

Los subprogramas son bloques PL/SQL que tienen un nombre y pueden recibir y devolver valores. Normalmente se guardan en la base de datos y podemos ejecutarlos invocándolos desde otros subprogramas o herramientas.

#### **PROCEDIMIENTOS:**

Usamos el siguiente formato:

**CREATE (OR REPLACE) PROCEDURE <nombre del procedimiento>**  
**(lista de parámetros)**  
**AS .....**

A continuación, se introducirá el bloque de código PL/SQL sin la palabra DECLARE.

La opción de **REPLACE** actúa en el caso de que hubiese un subprograma almacenado con ese nombre, sustituyéndolo por el nuevo.

En la lista de parámetros , se encuentra la declaración de cada uno de los parámetros que se utilizan para pasar valores al programa separados por comas:

Veamos un ejemplo:

#### **Ejemplo1**

```
create or replace procedure cambio_oficio
( v_numempleado number, v_nuevooficio varchar2)
as
v_oficio emple.oficio%type;
begin
select oficio into v_oficio from emple
where emp_no =v_numempleado;
update emple set oficio =v_nuevooficio
where emp_no= v_numempleado;
dbms_output.put_line(v_numempleado || '*Oficio anterior: '|| v_oficio ||
'*Oficio nuevo: ' || v_nuevooficio);
end cambio_oficio;
```



```
1 create or replace procedure cambio_oficio
2 ( v_numempleado number, v_nuevooficio varchar2)
3 as
4 v_oficio emp.oficio%type;
5 begin
6 select oficio into v_oficio from emple
7 where emp_no =v_numempleado;
8 update emple set oficio =v_nuevooficio
9 where emp_no= v_numempleado;
10 dbms_output.put_line(v_numempleado || '*Oficio anterior: ' || v_oficio || '*Oficio nuevo: ' || v_nuevooficio);
11 end cambio_oficio;
12
13
```

Resultados   Explicar   Describir   SQL Guardado   Historial

Procedimiento creado.

0,01 segundos

Repasando el ejemplo del procedimiento con nombre: cambio\_oficio podemos apreciar lo siguiente:  
Definimos dos variables :

v\_numempleado number , v\_nuevooficio varchar2

donde vendrán los valores uno numérico y otro alfanumérico con los que vamos a operar en el procedimiento.

v\_oficio emp.oficio%type

Definimos un %type para almacenar la información de la FILA que obtengamos con el SELECT en la tabla EMPLE.

Gestionamos la búsqueda en la tabla EMPLE con el where para encontrar el empleado que recibimos en la variable de la lista de parámetros del procedimiento: v\_numempleado number

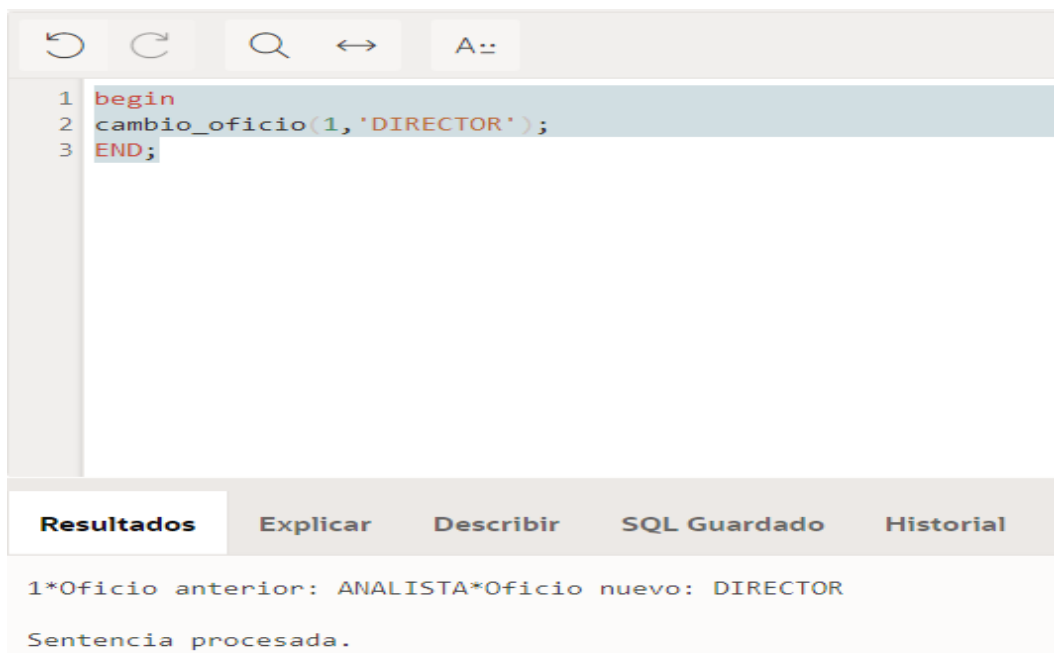
Si coincide actualizamos el atributo oficio con el segundo valor de la lista de parámetros del procedimiento: v\_nuevooficio varchar2

Podemos ejecutar el procedimiento anterior con el proceso siguiente.

```
begin  
cambio_oficio(1,'DIRECTOR');  
END;
```

Donde ponemos los parámetros del empleado que queremos cambiar su oficio y el nombre de ese nuevo oficio.

Y el resultado es el siguiente:



The screenshot shows a SQL IDE interface. At the top, there is a toolbar with icons for undo, redo, search, and a dropdown menu. Below the toolbar, the SQL editor contains the following code:

```
1 begin  
2 cambio_oficio(1,'DIRECTOR');  
3 END;
```

Below the editor, there is a tabbed interface with five tabs: "Resultados", "Explicar", "Describir", "SQL Guardado", and "Historial". The "Resultados" tab is selected, and it displays the following output:

```
1*Oficio anterior: ANALISTA*Oficio nuevo: DIRECTOR  
  
Sentencia procesada.
```

La llamada a un procedimiento es una instrucción por si misma. Cuando se produce la llamada, el control pasa al procedimiento llamado hasta que finaliza su ejecución y el control retorna a la línea siguiente a la llamada

Veamos en este segundo ejemplo un nuevo procedimiento desde donde se produce la llamada al anterior procedimiento

### Ejemplo2

```
create or replace procedure cambio_oficio1
( v_apellido varchar2, v_oficio varchar2)
as
v_empleado emple.oficio%type;
begin
select emp_no into v_empleado from emple
where apellido =v_apellido;
cambio_oficio (v_empleado, v_oficio);
dbms_output.put_line(v_empleado || '*Oficio anterior: ' || v_oficio || '*Oficio
nuevo: ' || v_oficio);
end cambio_oficio1;
```



```
1 create or replace procedure cambio_oficio1
2 ( v_apellido varchar2, v_oficio varchar2)
3 as
4 v_empleado emple.oficio%type;
5 begin
6 select emp_no into v_empleado from emple
7 where apellido =v_apellido;
8 cambio_oficio (v_empleado, v_oficio);
9 dbms_output.put_line(v_empleado || '*Oficio anterior: ' || v_oficio || '*Oficio nuevo: ' || v_oficio);
10 end cambio_oficio1;
11 |
```

Resultados   Explicar   Describir   SQL Guardado   Historial

Procedimiento creado.

0,01 segundos

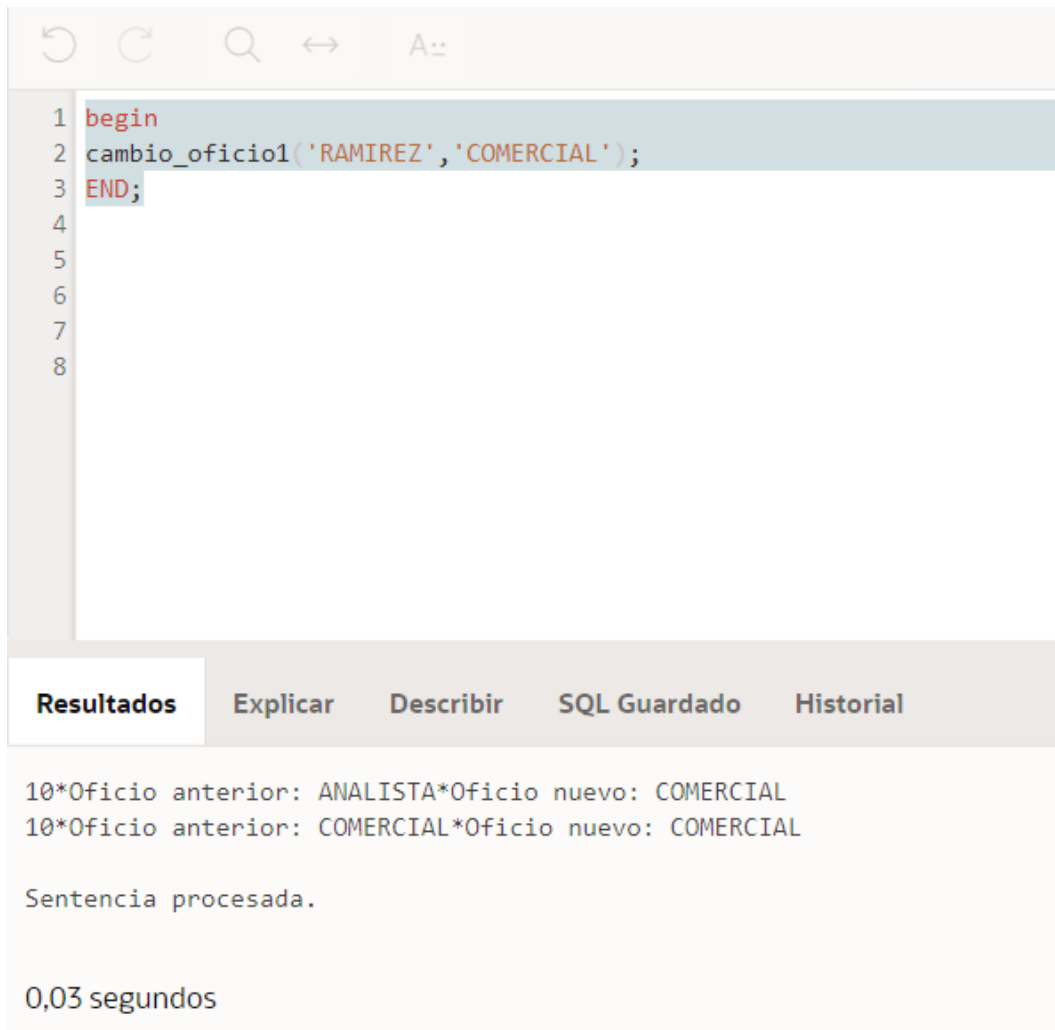
En esta ocasión, el procedimiento tiene otro nombre: cambio\_oficio1 y admite dos valores externos :  
v\_apellido varchar2 y v\_oficio varchar2

Gestionamos la búsqueda en la tabla EMPLE con el where para encontrar si coincide el apellido del empleado con el valor que se recibe. Si es el caso la siguiente sentencia es la llamada al anterior procedimiento

**cambio\_oficio (v\_empleado, v\_oficio);**  
con los datos de que ya disponemos: el número de empleado y su oficio en las variables v\_empleado y v\_oficio.  
Se ejecutaría y obtendríamos la impresión necesaria

Para ello haríamos el proceso siguiente y cambiaríamos el oficio del empleado que se apellida APELLIDO7

```
begin  
cambio_oficio1('APELLIDO7','COMERCIAL');  
END;
```



```
1 begin  
2 cambio_oficio1('RAMIREZ','COMERCIAL');  
3 END;
```

**Resultados** Explicar Describir SQL Guardado Historial

10\*Oficio anterior: ANALISTA\*Oficio nuevo: COMERCIAL  
10\*Oficio anterior: COMERCIAL\*Oficio nuevo: COMERCIAL

Sentencia procesada.

0,03 segundos

Se imprime dos veces : la primera es la del cambio en el procedimiento : cambio\_oficio y la segunda es la visualización en el procedimiento cambio\_oficio1.

### **FUNCIONES:**

Tienen una estructura y funcionalidad similar a los procedimientos pero, a diferencia de estos, las funciones devuelven siempre un valor.

Usamos el siguiente formato:

```
CREATE (OR REPLACE) FUNCTION <nombre de la función>  
(lista de parámetros)  
RETURN <tipo de valor devuelto>....  
AS  
Declaración de variables  
BEGIN  
.....  
RETURN devuelve el control al programa que llamó a la función  
END <nombre de la función>
```

En el cuerpo del programa, el comando RETURN devuelve el control al programa que llamó a la función

Empezamos con **create or replace function** y el nombre de la función, el **replace** es opcional, después van los parámetros que le vamos a pasar y el primer **return** que especifica el tipo de valor que devuelve la función.

Después del AS se hacen las declaraciones de variables que usaremos en el proceso de la función.

A continuación vendría el bloque **Begin/End** y al final devolvemos con **return** al programa que llamó a la función.

Veamos un ejemplo:

#### Ejemplo3

```
create or replace function busco_empleado  
( v_apellido varchar2)  
return varchar  
as  
v_empleado emple.emp_no%type;  
begin  
select emp_no into v_empleado from emple  
where apellido =v_apellido;  
return v_empleado;  
end busco_empleado;
```

La función recibe un parámetro : v\_apellido varchar2 de tipo varchar2

El proceso de la función lo que hace es realizar un proceso de lectura de todos los números de empleados de la tabla empleados guardando cada fila leída en una variable %type llamada

v\_empleado emple.emp\_no%type;

Cuando en el proceso de lectura con la sentencia SELECT encuentra el apellido que ha recibido a través del parámetro, devuelvo el código del empleado con la sentencia Return en la variable v\_empleado de tipo %type.



```
1 create or replace function busco_empleado
2 ( v_apellido varchar2)
3 return varchar
4 as
5 v_empleado emple.emp_no%type;
6 begin
7 select emp_no into v_empleado from emple
8 where apellido =v_apellido;
9 return v_empleado;
10 end busco_empleado;
11
```

Resultados   Explicar   Describir   SQL Guardado   Historial

Función creada.

0,02 segundos

Una vez que el procedimiento ha sido validado y almacenado podemos utilizar la función de la manera que vienen a continuación:

El formato de llamada a una función consiste en utilizarla como parte de una expresión:

<variable> := <nombre función> (lista de parámetros);

En las sentencias posteriores vemos como con un bloque sencillo PL/SQL hacemos uso de la función descrita anteriormente utilizando la expresión de llamada de la función :

V\_empleado := busco\_empleado('APELLIDO7');

Obteniendo el resultado que nos muestra la imagen.

```
Declare
v_employado emple.emp_no%type;
Begin
V_employado := busco_employado('APELLIDO7');
dbms_output.put_line('El número de empleado es : ' || v_employado);
END;
```

The screenshot shows a SQL IDE interface. At the top, there's a toolbar with a 'Filas' dropdown set to '10', a help icon, and buttons for 'Borrar Comando' and 'Buscar Tablas'. Below the toolbar is a code editor with a PL/SQL script. The script is as follows:

```
1 Declare
2 v_employado emple.emp_no%type;
3 Begin
4 V_employado := busco_employado('APELLIDO7');
5 dbms_output.put_line('El número de empleado es : ' || v_employado);
6 END;
7
8 |
9
```

Below the code editor is a tabbed interface with four tabs: 'Resultados' (selected), 'Explicar', 'Describir', and 'SQL Guardado'. The 'Resultados' tab displays the output of the script:

```
El número de empleado es : 7

Sentencia procesada.

0,01 segundos
```