

Procedimientos y Funciones

Los subprogramas son bloques de instrucciones de PL/SQL que pueden ser invocados por otros y recibir parámetros. En PL/SQL existen dos tipos de subprogramas: Los Procedimientos y las Funciones. Por regla general, se utiliza un procedimiento para ejecutar una acción específica y una función para calcular un valor.

Los subprogramas también constan de una sección de declaraciones, un cuerpo que se ejecuta y una sección opcional de manejo de excepciones.

Procedimientos

Un procedimiento es un bloque PL nombrado que realiza una determinada acción. Está almacenado en la Base de Datos como un objeto de la misma y puede ser ejecutado múltiples veces.

Los procedimientos promueven la reutilización y el mantenimiento de los programas, y una vez validados, pueden utilizarse en muchas aplicaciones. Si la definición cambia, solo se ve afectado el procedimiento y esto simplifica en gran medida el mantenimiento.

Sintaxis:

```
CREATE [OR REPLACE] PROCEDURE nb_procedimiento
    (argumento1 [ modo 1] tipo_de_dato1,
    argumento2 [ modo2] tipo_de_dato2,
    ...)
IS | AS
Bloque PL/SQL;
```

Donde:

ARGUMENTO: nombre de la variable PL cuyo valor se recibe

MODO: IN, OUT, IN OUT.

TIPO_DE_DATO: tipo de dato del argumento.

EJEMPLO:

```
CREATE OR REPLACE PROCEDURE PROC1
    (v_ID in emp.empno%type) is
Begin
    UPDATE EMP SET SAL= SAL*1.10 WHERE EMPNO = V_ID;
End PROC1;
```

Borrado de un procedimiento.

```
DROP PROCEDURE nb_procedimiento;
```

Para volver a compilar un procedimiento utilizamos la orden ALTER

```
ALTER { PROCEDURE|FUNCTION} nb_subprograma COMPILE;
```

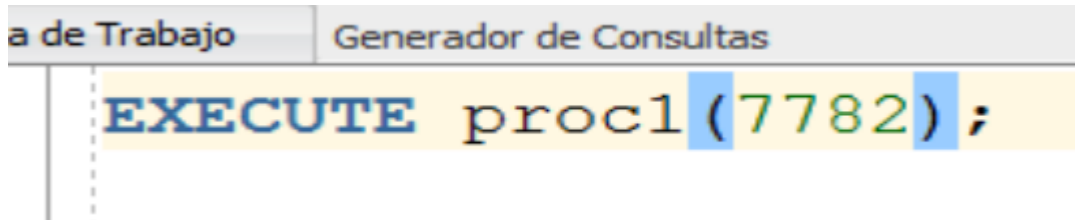
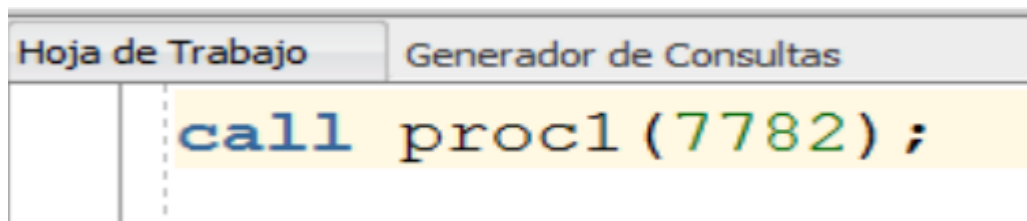
Esto vale tanto para procedimientos como para funciones.

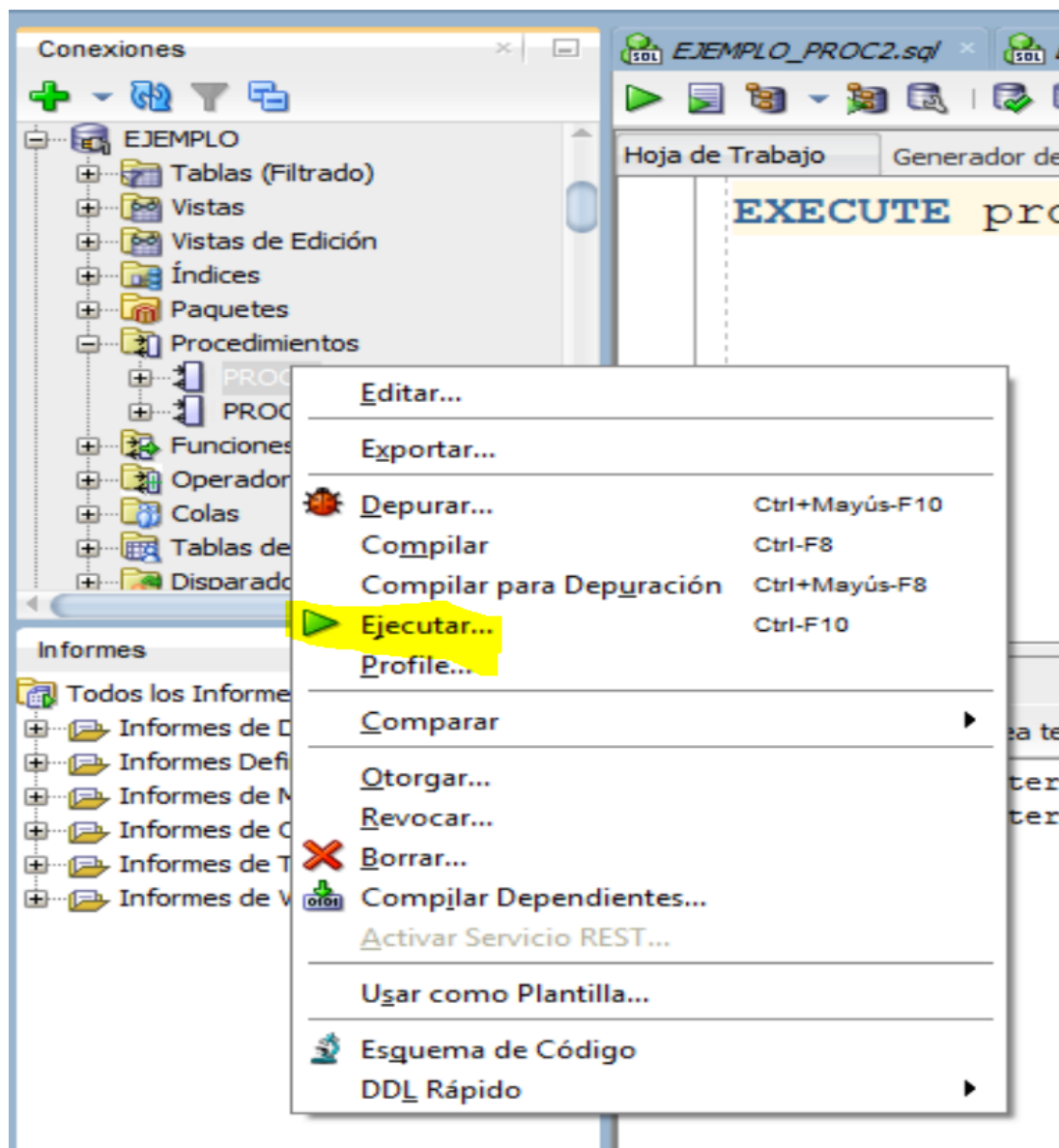
Como llamar a un procedimiento

En general un procedimiento se hace para ser llamado desde otro programa (bien de la propia BBDD o bien externo).

Para llamarlo desde otro programa de la BBDD simplemente hay que poner el nombre del procedimiento, pasarle los parámetros y ya está. Lo veremos más adelante con ejemplos.

Para llamarlo desde un cliente de BBDD y poder hacer pruebas hay varias formas. En general se hará un CALL o un EXECUTE del procedimiento, aunque el cliente de BBDD seguro que ofrece una forma de ejecutarlo. Lo vemos con el SQL Developer.





Tipos de parámetros de un procedimiento

| IN | OUT | IN OUT |
|-----------------------------------|---------------------------------|---|
| Por defecto. | Tiene que especificarse. | Tiene que especificarse. |
| Valor que se pasa al subprograma. | Devuelve al entorno de llamada. | Valor que se pasa al subprograma. Devuelve al entorno de llamada. |

EJEMPLO OUT:

```

a de Trabajo  Generador de Consultas
CREATE OR REPLACE PROCEDURE PROC2
  (v_ID in emp.empno%type,
   v_sal out emp.sal%type,
   v_job out emp.job%type) is
Begin
  SELECT SAL, JOB INTO V_SAL,V_JOB FROM EMP
  WHERE EMPNO = V_ID;
End PROC2;

```

Cuando hay parámetros de salida o de entrada / salida, para poder invocar al procedimiento debes “recoger” estos parámetros en variables:

```

a de Trabajo  Generador de Consultas
Declare
  v_salario emp.sal%type;
  v_trabajo emp.job%type;
  v_num emp.empno%type := &id;
Begin
  proc2(v_num,v_salario,v_trabajo);
  DBMS_OUTPUT.put_line('Salario:'||v_salario);
  DBMS_OUTPUT.put_line('Trabajo:'||v_trabajo);
End;

```

Métodos para pasar parámetros

Existen tres métodos para pasar parámetros:

1. **Posicional:** Lista de valores en el orden que se declaran los parámetros
2. **Asociación nombrada:** Lista de valores en orden arbitrario mediante la asignación (=>).
3. **Mixto:** Mezcla de los dos anteriores. Lista de los primeros valores posicionalmente, y el resto, utilizando la sintaxis especial del método nombrado.

```

a de Trabajo  Generador de Consultas
Declare
  v_salario emp.sal%type;
  v_trabajo emp.job%type;
  v_num emp.empno%type := &id;
Begin
  proc2(v_sal => v_salario,v_id => v_num ,v_job=> v_trabajo);
  DBMS_OUTPUT.put_line('Salario:'||v_salario);
  DBMS_OUTPUT.put_line('Trabajo:'||v_trabajo);
End;

```

Funciones

Una función es un bloque nombrado PL/SQL que devuelve un valor, solamente tiene parámetros de entrada. Está almacenada en la Base de Datos como un objeto de la misma para repetidas ejecuciones.

Sintaxis:

```
CREATE [ OR REPLACE] FUNCTION nb_función
    (argumento1 [modo1] tipo_dato1,
    argumento2 [modo2] tipo_dato2,
    ...)
RETURN tipo_de_dato
IS | AS
Bloque PL/SQL;
/
```

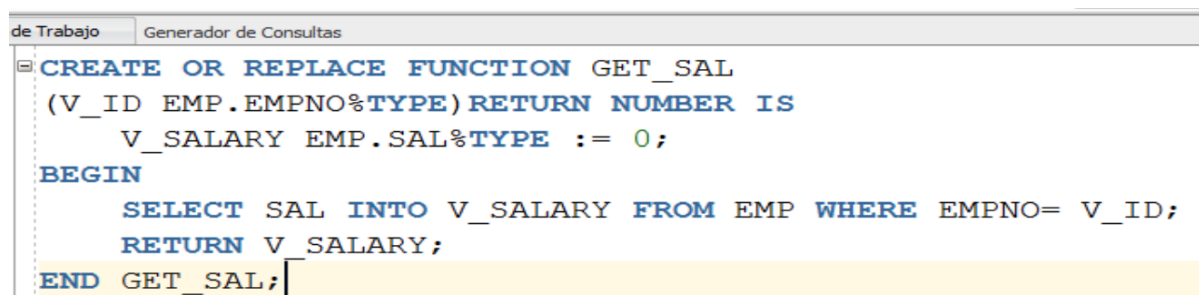
Donde:

ARGUMENTO: Nombre de la variable PL cuyo valor se pasa en la función.

TIPO_DATO: tipo de dato del parámetro.

RETURN TIPO_DE_DATO: Indica el tipo de dato que la función debe devolver.

MODO: siempre es IN y se puede omitir.



The screenshot shows a SQL Developer window titled "de Trabajo" and "Generador de Consultas". It contains the following SQL code:

```
CREATE OR REPLACE FUNCTION GET_SAL
(V_ID EMP.EMPNO%TYPE) RETURN NUMBER IS
    V_SALARY EMP.SAL%TYPE := 0;
BEGIN
    SELECT SAL INTO V_SALARY FROM EMP WHERE EMPNO= V_ID;
    RETURN V_SALARY;
END GET_SAL;
```

Para borrar funciones se utiliza la siguiente orden:

```
DROP FUNCTION nb_funcion;
```

Como llamar a una función

Igual que los procedimientos, las funciones están hechas para ser llamadas desde otros programa, sólo que en este caso la salida debe almacenarse en una variable del tipo que retorna la función:

```

de Trabajo  Generador de Consultas
-- Declare
v_salario emp.sal%type;
Begin
v_salario := get_sal(&id);

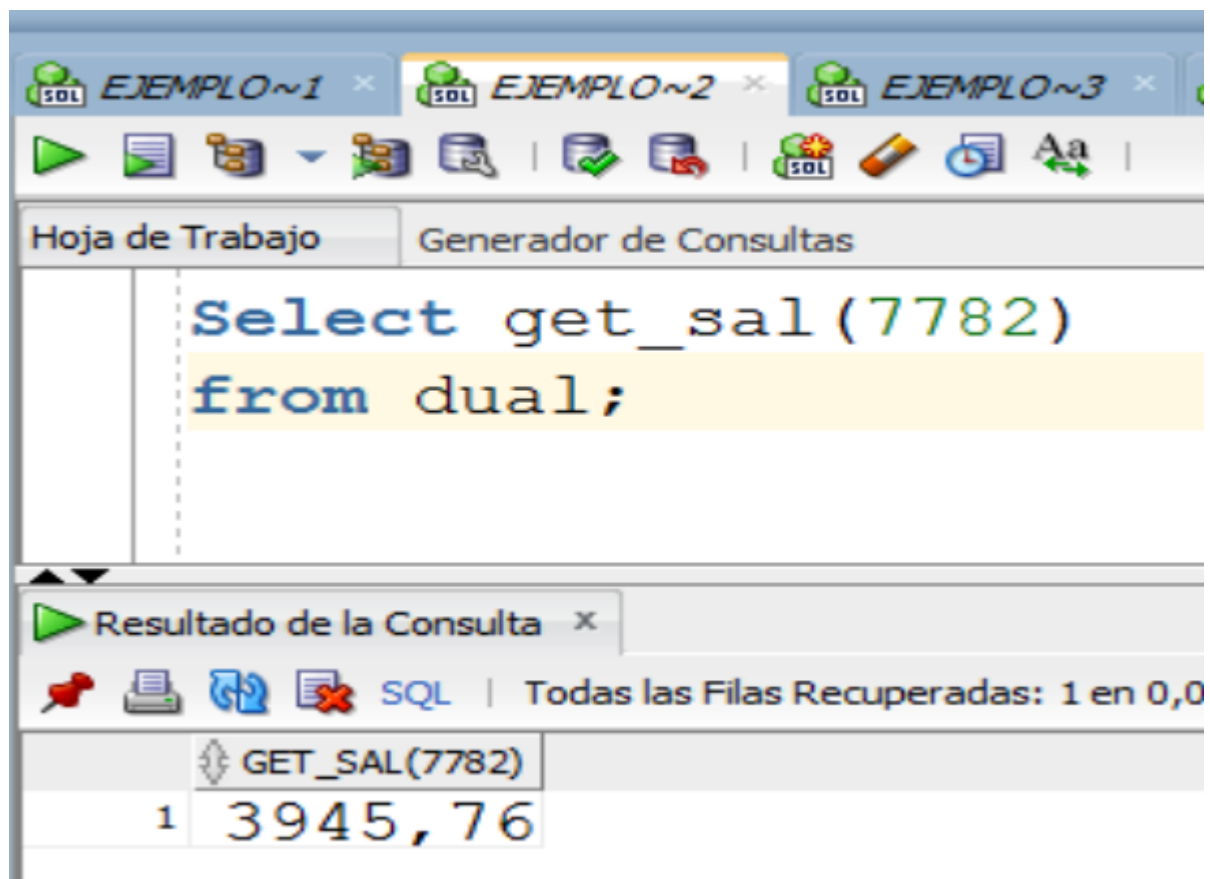
DBMS_OUTPUT.put_line('Salario: '||v_salario);
End;

```

En el caso de las funciones, se les puede llamar dentro de una select, bien una select sobre una tabla o bien una select from dual:

The screenshot shows the SQL Developer interface. The top pane displays a query: `Select get_sal(7782), d.* from dept d;`. The bottom pane, titled 'Resultado de la Consulta', shows the results of the query. The results are displayed in a table with 5 rows and 4 columns: GET_SAL(7782), DEPTNO, DNAME, and LOC. The first column contains the value 3945,76 for all rows. The other columns show department details for departments 10, 20, 30, 40, and 50.

| | GET_SAL(7782) | DEPTNO | DNAME | LOC |
|---|---------------|--------|------------|----------|
| 1 | 3945,76 | 10 | ACCOUNTING | NEW YORK |
| 2 | 3945,76 | 20 | RESEARCH | DALLAS |
| 3 | 3945,76 | 30 | SALES | CHICAGO |
| 4 | 3945,76 | 40 | DAM 6 | SORIA |
| 5 | 3945,76 | 50 | ASIR 1 | TOLEDO |



Y también se pueden ejecutar, como los procedimientos, desde el propio cliente de BBDD.

Paquetes (PACKAGE)

Los paquetes en Oracle sirven para agrupar y organizar funcionalidades en una base de datos. Son estructuras que agrupan objetos PL/SQL (funciones, procedimientos, tipos, etc.). Ello nos va a permitir tener programas estructurados agrupados por funcionalidades.

Los paquetes están divididos en 2 partes: **especificación** (obligatoria) y **cuerpo** (no obligatoria).

La **especificación** o encabezado es la interfaz entre el paquete y las aplicaciones que lo utilizan, y es allí donde se declaran los procedimientos y funciones que podrán ser invocados desde fuera del paquete.

En el **cuerpo** del paquete se implementa la especificación del mismo. El cuerpo contiene los detalles de implementación y declaraciones privadas, manteniéndose todo esto oculto a las aplicaciones externas, siguiendo el conocido concepto de "caja negra". Sólo las declaraciones hechas en la especificación del paquete son visibles y accesibles desde fuera del paquete (por otras aplicaciones o procedimientos almacenados) quedando los detalles de implementación del cuerpo del paquete totalmente ocultos e inaccesibles para el exterior.

Definimos la cabecera de nuestro paquete al que llamaré primer paquete:


```

de Trabajo  Generador de Consultas
CREATE OR REPLACE PACKAGE primer_paquete IS
-- Declaraciones
    PROCEDURE PROC2 (v_ID in emp.empno%type,
        v_sal out emp.sal%type,
        v_job out emp.job%type);

    FUNCTION GET_SAL (V_ID EMP.EMPNO%TYPE) RETURN NUMBER;
END;

```

Y a continuación definimos el cuerpo de este paquete, ya con la lógica:

```

CREATE OR REPLACE PACKAGE BODY primer_paquete IS
-- Bloques de código
    PROCEDURE PROC2 (v_ID in emp.empno%type,
        v_sal out emp.sal%type,
        v_job out emp.job%type) is
    Begin
        SELECT SAL, JOB INTO V_SAL,V_JOB FROM EMP WHERE EMPNO = V_ID;
    End PROC2;

    FUNCTION GET_SAL (V_ID EMP.EMPNO%TYPE) RETURN NUMBER IS
        V_SALARY EMP.SAL%TYPE := 0;
    BEGIN
        SELECT SAL INTO V_SALARY FROM EMP WHERE EMPNO= V_ID;
        RETURN V_SALARY;
    END GET_SAL;
END;

```

Y para utilizarlo, nombre_paquete.nombre_procedimiento_o_función:

The screenshot shows the SQL Developer interface. The 'Hoja de Trabajo' (Worksheet) tab is active, displaying the query: `Select primer_paquete.get_sal(7782) from dual;`. Below the query, the 'Resultado de la Consulta' (Query Result) tab shows the output. The result is a single row with the value 3945,76. The status bar at the bottom indicates 'Todas las Filas Recuperadas: 1 en 0,003 segundos'.

| PRIMER_PAQUETE.GET_SAL(7782) |
|------------------------------|
| 3945,76 |

Ventajas del uso de paquetes

- Permite modularizar el diseño de nuestra aplicación: El uso de Paquetes permite encapsular elementos relacionados entre sí (tipos, variables, procedimientos, funciones) en un único módulo PL/Sql que llevará un nombre que identifique la funcionalidad del conjunto.
- Otorga flexibilidad al momento de diseñar la aplicación: En el momento de diseñar una aplicación todo lo que necesitaremos inicialmente es la información de interfaz en la especificación del paquete. Puede codificarse y compilarse la especificación sin su cuerpo para posibilitar que otros sub-programas que referencian a estos elementos declarados puedan compilarse sin errores. De esta manera podremos armar un "prototipo" de nuestro sistema antes de codificar el detalle del mismo.
- Permite ocultar los detalles de implementación: Pueden especificarse cuáles tipos, variables y sub-programas dentro del paquete son públicos (visibles y accesibles por

otras aplicaciones y sub-programas fuera del paquete) o privados (ocultos e inaccesibles fuera del paquete). Por ejemplo, dentro del paquete pueden existir procedimientos y funciones que serán invocados por otros programas, así como también otras rutinas de uso interno del paquete que no tendrán posibilidad de ser accedidas fuera del mismo. Esto asegura que cualquier cambio en la definición de estas rutinas internas afectará sólo al paquete donde se encuentran, simplificando el mantenimiento y protegiendo la integridad del conjunto.

- Agrega mayor funcionalidad a nuestro desarrollo: Las definiciones públicas de tipos, variables y cursores hechas en la especificación de un paquete persisten a lo largo de una sesión. Por lo tanto pueden ser compartidas por todos los sub-programas y/o paquetes que se ejecutan en ese entorno durante esa sesión. Por ejemplo, puede utilizarse esta técnica (en dónde sólo se define una especificación de paquete y no un cuerpo) para mantener tipos y variables globales a todo el sistema.
- Introduce mejoras al rendimiento: En relación a su ejecución, cuando un procedimiento o función que está definido dentro de un paquete es llamado por primera vez, todo el paquete es “subido” a memoria. Por lo tanto, posteriores llamadas al mismo u otros sub-programas dentro de ese paquete realizarán un acceso a memoria en lugar de a disco. Esto no sucede con procedimientos y funciones estándares.
- Permite la “Sobrecarga de funciones” (Overloading): PL/SQL nos permite que varios procedimientos o funciones almacenadas, declaradas dentro de un mismo paquete, tengan el mismo nombre. Esto nos es muy útil cuando necesitamos que los sub-programas puedan aceptar parámetros que contengan diferentes tipos de datos en diferentes instancias. En este caso Oracle ejecutará la “versión” de la función o procedimiento cuyo encabezado se corresponda con la lista de parámetros recibidos.