PROCEDIMIENTOS

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.

En todo subprograma podemos distinguir:

- **La cabecera o especificación del subprograma**, que contiene:
  - Nombre del subprograma.
  - Los parámetros con sus tipos (opcional).
  - Tipo de valor de retorno (en el caso de las funciones).
- **El cuerpo del subprograma**. Es un bloque PL/SQL que incluye:
  - Declaraciones (opcional).
  - Instrucciones.
  - Manejo de excepciones (opcional).

En PL/SQL podemos distinguir dos tipos de subprogramas: *procedimientos* y *funciones*. Veámoslos:

A. Procedimientos

Tienen la siguiente estructura general:

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

Podemos apreciar dos partes:

- **La cabecera o especificación del procedimiento.** Comienza con la palabra PROCEDURE y termina después de la declaración de parámetros.
- **El cuerpo del procedimiento.** Corresponde con un bloque PL/SQL. Comienza a continuación de la palabra IS (o AS) y termina con la palabra END, opcionalmente seguida del nombre del procedimiento. En el formato genérico anterior corresponde a la zona sombreada.

Para crear un procedimiento desde SQL*Plus usaremos el siguiente formato:

```
CREATE [OR REPLACE] PROCEDURE <nombreprocedimiento>
[(listadeparametros)]
AS ...
```

Por concordancia gramatical se utilizará AS en lugar de IS al crear un procedimiento o función con la orden CREATE OR REPLACE. Pero, a efectos del compilador, se puede utilizar cualquiera de las dos.

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

La **opción 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:

```
(nombreparam1 TIPOP1, nombreparam2 TIPOP2, nombreparam3
TIPOP3, ...)
```

Caso práctico

4 Crearemos un procedimiento que reciba un número de empleado y una cadena correspondiente a su nuevo oficio. El procedimiento deberá localizar el empleado, modificar el oficio y visualizar los cambios realizados.

```
CREATE OR REPLACE PROCEDURE cambiar_oficio (
    num_empleado NUMBER,     -- En los parámetros ..
    nuevo_oficio VARCHAR2)   -- ..no se especifica tamaño
AS
    v_anterior_oficio emple.oficio%TYPE;
BEGIN
    SELECT oficio INTO v_anterior_oficio FROM emple
      WHERE emp_no = num_empleado;

    UPDATE emple SET oficio = nuevo_oficio
        WHERE emp_no = num_empleado;
    DBMS_OUTPUT.PUT_LINE(num_empleado||'*Oficio Anterior:'||v_anterior_oficio||
                                    '*Oficio Nuevo   :'||nuevo_oficio );
END cambiar_oficio;
/
```

El sistema responderá:

```
Procedimiento creado.
```

Ahora el procedimiento está creado y almacenado en la base de datos. Para ejecutarlo podemos invocar el procedimiento desde cualquier herramienta de Oracle, por ejemplo, desde SQL*Plus:

```
SQL>  EXECUTE CAMBIAR_OFICIO(7902,'DIRECTOR');
7902*Oficio Anterior:ANALISTA*Oficio Nuevo   :DIRECTOR
Procedimiento PL/SQL terminado con éxito.
```

Actividades propuestas

3 Escribe un procedimiento con funcionalidad similar al ejemplo anterior, que recibirá un número de empleado y un número de departamento y asignará al empleado el departamento indicado en el segundo parámetro.

Podemos invocar al procedimiento desde otro bloque, procedimiento o función. Para ello escribiremos el nombre del procedimiento seguido de la lista de parámetros entre paréntesis, y de un punto y coma:

```
nombreprocedimientoalquellamamos(listadeparametros);
```

La llamada a un procedimiento es una instrucción por sí 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.

Por ejemplo, podemos crear un bloque que reciba el apellido y el oficio nuevo. El programa buscará el número de empleado y utilizará una llamada al procedimiento anterior para cambiar oficio:

No es obligatorio escribir el nombre de subprograma detrás del END, pero es aconsejable por razones de legibilidad.

```
CREATE OR REPLACE PROCEDURE cam_ofi_ (
  v_apellido  VARCHAR,
  nue_oficio VARCHAR2)
IS
  v_n_empleado emple.emp_no%TYPE;
BEGIN
    SELECT emp_no INTO v_n_empleado FROM emple
      WHERE  apellido = v_apellido;
    cambiar_oficio(v_n_empleado, nue_oficio);
END cam_ofi_;
```

La ejecución del nuevo programa será:

```
SQL> EXECUTE  cam_ofi_('FERNANDEZ','ANALISTA');
7902*Oficio Anterior:DIRECTOR*Oficio Nuevo   :ANALISTA
Procedimiento PL/SQL terminado con éxito.
```

B. Funciones

Las **funciones** tienen una estructura y funcionalidad similar a los procedimientos pero, a diferencia de éstos, las funciones devuelven siempre un valor:

```
FUNCTION <nombredefunción>
     [(<lista de parámetros>)]
RETURN <tipo de valor devuelto >
IS
```

```
     [<declaraciones>;]
BEGIN
     <instrucciones>;
     RETURN <expresión>;
     …
[EXCEPTION
     <excepciones>;]
END [<nombredefunción>];
```

La lista de parámetros es opcional. Si no hay parámetros no debemos poner paréntesis pues dará error igual que en los procedimientos. Sin embargo, es obligatorio el uso de la cláusula **RETURN** en la cabecera y el comando **RETURN** en el cuerpo del programa. No debemos confundirlas:

- La cláusula **RETURN** de la cabecera especifica el tipo del valor que retorna la función.
- En el cuerpo del programa, el comando **RETURN** devuelve el control al programa que llamó a la función, asignando el valor de la expresión que sigue al RETURN a la variable que figura en la llamada a la función.

Antes de ejecutar un subprograma almacenado, Oracle marca un punto de salvaguarda implícito, de forma que si el subprograma falla durante la ejecución, se desharán todos los cambios realizados por él.

De manera análoga a los procedimientos, para crear o modificar una función utilizaremos el comando **CREATE OR REPLACE FUNCTION:**

```
CREATE OR REPLACE FUNCTION Encontrar_Num_empleado (
    v_apellido  VARCHAR2)
    RETURN REAL
AS
    N_empleado emple.emp_no%TYPE;
BEGIN
    SELECT emp_no INTO N_empleado FROM emple
      WHERE  apellido = v_apellido;
    RETURN N_empleado;
END Encontrar_num_empleado;
```

Un procedimiento también puede usar la cláusula RETURN (en este caso, sin devolver ningún valor) para devolver el control al programa que lo llamó, pero no es una técnica recomendable.

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

```
<variable> :=  <nombredefunción>[(listadeparámetros)];
```

Para invocar a una función desde SQL también tenemos que «hacer algo» con el valor que devuelve, por ejemplo, utilizarlo como parámetro para otra llamada:

```
SQL> BEGIN DBMS_OUTPUT.PUT_LINE(ENCONTRAR_NUM_EMPLEADO
                                ('GIL'));END;
2  /
7788
Procedimiento PL/SQL terminado con éxito.
```

Una función puede tener varios RETURN pero solo se ejecutará uno de ellos en cada llamada a la función:

```
...
IF nota < 5 THEN
  RETURN 'SUSPENSO';
ELSE
  RETURN 'APROBADO';
END IF;
...
```

C. Parámetros

Los subprogramas utilizan parámetros para pasar y recibir información. Hay dos clases:

- **Parámetros actuales o reales.** Son las variables o expresiones indicadas en la llamada a un subprograma.
- **Parámetros formales.** Son variables declaradas en la especificación del subprograma.

Las declaraciones de variables locales se hacen después del IS, que equivale al DECLARE. En este caso, sí se deberá indicar la longitud pues ya no se trata de parámetros.

Si es necesario, PL/SQL hará la conversión automática de tipos; sin embargo, los tipos de los parámetros actuales y los correspondientes parámetros formales deben ser compatibles.

Podemos hacer el paso de parámetros utilizando la *notación posicional, nominal* o *mixta* (ambas):

- **Notación posicional:** El compilador asocia los parámetros actuales a los formales basándose en su posición.
- **Notación nominal:** El símbolo => después del parámetro actual y antes del nombre del formal indica al compilador la correspondencia.
- **Notación mixta:** Consiste en usar ambas notaciones con la restricción de que la notación posicional debe preceder a la nominal.

Por ejemplo, dada la siguiente especificación del procedimiento `ges_dept`:

```
PROCEDURE ges_dept (
  N_departamento INTEGER,
  Localidad VARCHAR2
IS...
```

Desde el siguiente bloque se podrán realizar las llamadas indicadas:

```
DECLARE
  Num_dep INTEGER;
  Local   VARCHAR(14)
BEGIN
...
  -- posicional  ges_dept(Num_dep, local);
  -- nominal     ges_dept(Num_dep   =>  N_departamento,
                 local => localidad);
```

```
  -- nominal     ges_dept(local => localidad, Num_dep =>
                 N_departamento);
  -- mixta       ges_dept(Num_dept, Local => localidad);
...
END;
```

Actividades propuestas

4 **Dado el siguiente procedimiento:**

```
PROCEDURE crear_depart (
    v_num_dept depart.dept_no%TYPE,
    v_dnombre depart.dnombre%TYPE DEFAULT 'PROVISIONAL',
    v_loc     depart.loc%TYPE DEFALUT 'PROVISIONAL')
IS
BEGIN
    INSERT INTO depart
      VALUES (v_num_dept, v_dnombre, v_loc);
END crear_depart;
```

Indica cuáles de las siguientes llamadas son correctas y cuáles incorrectas. En el caso de que sean incorrectas, escribe la llamada correcta usando la notación posicional, siempre que sea posible:

```
crear_depart;
crear_depart(50);
crear_depart('COMPRAS');
crear_depart(50,'COMPRAS');
crear_depart('COMPRAS', 50);
crear_depart('COMPRAS', 'VALENCIA');
crear_depart(50,'COMPRAS', 'VALENCIA');
crear_depart('COMPRAS',50, 'VALENCIA');
crear_depart('VALENCIA', 'COMPRAS');
crear_depart('VALENCIA', 50);
```

PL/SQL soporta tres tipos de parámetros:

Tipo	Características y utilización
IN	Son parámetros de **ENTRADA;** se usan para pasar valores al subprograma. Dentro del subprograma el parámetro actúa como una constante, es decir, no se le puede asignar ningún valor. Por tanto, se sitúa siempre a la derecha del operador de asignación. El parámetro actual puede ser una variable, constante, literal o expresión.
OUT	Son parámetros de **SALIDA;** se usan para devolver valores al programa que hizo la llamada. Dentro del subprograma, el parámetro actúa como una variable no inicializada y no puede intervenir en ninguna expresión, salvo para tomar un valor. Se sitúa siempre a la izquierda del operador de asignación. El parámetro actual debe ser una variable.
IN OUT	Son parámetros de **ENTRADA/SALIDA;** permiten pasar un **valor inicial** y devolver un **valor actualizado.** Dentro del subprograma actúa como una variable inicializada. Puede intervenir en otras expresiones y puede tomar nuevos valores. El **parámetro actual** debe ser una variable.

No podemos asignar valores nuevos a los parámetros formales de entrada pues nos encontraremos con el error:

```
PLS-00363: expression 'VPa-
rametro' cannot be used as
an assignment target.
```

Si queremos cambiar el valor de un parámetro formal de entrada deberemos declararlo de **tipo IN OUT.**

El formato genérico de la declaración de cada uno de los parámetros es:

```
<nombrevariable> [ IN | OUT | IN OUT ] < tipodedato>
                 [ { := | DEFAULT } <valor>]
```

Debemos tener en cuenta, además, las siguientes reglas:

- Al indicar los parámetros debemos especificar el tipo, pero no el tamaño.
- En el caso de que el subprograma no tenga parámetros no se pondrán los paréntesis.
- Cuando un subprograma recibe un parámetro en modo OUT y se produce una excepción no tratada, el parámetro actual correspondiente queda sin ningún valor.

Valores por defecto en el paso de parámetros de entrada (modo IN): los parámetros de entrada (todos los que hemos manejado hasta este momento) se pueden inicializar con valores por omisión, es decir, indicando al subprograma que en el caso de que no se pase el parámetro correspondiente, asuma un valor por defecto. Para ello, se utiliza la opción DEFAULT <valor>, o bien := <valor>.

Caso práctico

5 Supongamos que nos han solicitado un programa de cambio de divisas para un banco que cumpla las siguientes especificaciones:

- Recibirá una cantidad en euros y el cambio (divisas/euro) de la divisa.
- También podrá recibir una cantidad correspondiente a la comisión que se cobrará por la transacción. En el caso de que no reciba dicha cantidad el programa calculará la comisión que será de un 0,2% del importe, con un mínimo de 3 euros.
- El programa calculará la comisión, la deducirá de la cantidad inicial y calculará el cambio en la moneda deseada, retornando estos dos valores (comisión y cambio) a los parámetros actuales del programa que realice la llamada para solicitar el cambio de divisas.

```
CREATE OR REPLACE PROCEDURE cambiar_divisas (
    cantidad_euros    IN     NUMBER, -- parámetro entrada
    cambio_actual     IN     NUMBER, -- parámetro entrada
    cantidad_comision IN OUT  NUMBER,-- parámetro de e/s
    cantidad_divisas  OUT    NUMBER) -- parámetro de salida
AS
    pct_comision    CONSTANT NUMBER (3,2) := 0.2;
    minimo_comision CONSTANT NUMBER (6) DEFAULT 3;
BEGIN
    IF cantidad_comision IS NULL THEN
        cantidad_comision := GREATEST(cantidad_euros/100*pct_comision,
                                      minimo_comision);
```

(Continúa)

(Continuación)

```
    END IF;
    cantidad_divisas := (cantidad_euros - cantidad_comision) * cambio_actual;
END;
/
```

Una vez creado el procedimiento podremos diseñar programas que hagan uso de él teniendo en cuenta que los parámetros formales para llamar al programa deberán ser cuatro. De éstos, los dos últimos deberán ser variables, que recibirán los valores de la ejecución del programa, tal como aparece en el siguiente procedimiento:

```
CREATE OR REPLACE PROCEDURE mostrar_cambio_divisas (
    eur NUMBER,
    cambio NUMBER)
AS
    v_comision NUMBER (9);
    v_divisas NUMBER (9);
BEGIN
    Cambiar_divisas(eur, cambio, v_comision, v_divisas);
    DBMS_OUTPUT.PUT_LINE ('Euros           : '||
                TO_CHAR( Eur, '999,999,999.999'));
     DBMS_OUTPUT.PUT_LINE ('Divisas X 1 euro : '||
                TO_CHAR( cambio, '999,999,999.999'));
     DBMS_OUTPUT.PUT_LINE ('Euros Comisión   : '||
              TO_CHAR( v_comision, '999,999,999.999'));
     DBMS_OUTPUT.PUT_LINE ('Cantidad divisas : '||
              TO_CHAR( v_divisas, '999,999,999.999'));
END;
/
```

Llamamos al programa pasándole la cantidad y el cambio respecto al euro de la divisa queremos cambiar a euros.

```
SQL> EXECUTE MOSTRAR_CAMBIO_DIVISAS(2500, 1.220);
Euros           :       2,500.000
Divisas X 1 euro :           1.220
Euros Comisión   :           5.000
Cantidad divisas :       3,044.000

Procedimiento PL/SQL terminado con éxito.
```

D. Subprogramas almacenados

Los subprogramas (procedimientos y funciones) que hemos visto hasta ahora se pueden compilar independientemente y almacenar en la base de datos Oracle.

Cuando creamos procedimientos o funciones almacenados desde SQL*Plus utilizando los comandos CREATE PROCEDURE o CREATE FUNCTION, Oracle automáticamente compila el código fuente, genera el código objeto (llamado *P-código*) y los guarda en el diccionario de datos. De este modo, quedan disponibles para su utilización.

Oracle dispone de opciones avanzadas y funcionalidades especiales para el paso de parámetros que pueden resultar interesantes para el manejo de estructuras complejas. Estas opciones exceden del objetivo de este libro.

Los programas almacenados tienen dos estados: *disponible* (*valid*) y *no disponible* (*invalid*). Si alguno de los objetos referenciados por el programa ha sido borrado o alterado desde la última compilación del programa, quedará en situación de «no disponible» y se compilará de nuevo automáticamente en la próxima llamada. Al compilar de nuevo, Oracle determina si hay que compilar algún otro subprograma referido por el actual. Se puede producir una cascada de compilaciones.

Estos estados se pueden comprobar en la vista USER_OBJECTS:

```
SQL> SELECT OBJECT_NAME, OBJECT_TYPE, STATUS FROM
USER_OBJECTS WHERE OBJECT_NAME='CAMBIAR_OFICIO';

OBJECT_NAME          OBJECT_TYPE          STATUS
-------------------  ---------------      -------
CAMBIAR_OFICIO       PROCEDURE            VALID
```

Podemos acceder al código fuente almacenado mediante la vista **USER_SOURCE:**

```
SQL> SELECT LINE, SUBSTR(TEXT,1,60) FROM USER_SOURCE WHERE
NAME = 'CAMBIAR_OFICIO';
 LINE  SUBSTR(TEXT,1,60)
------ ------------------------------------------------------
   1   PROCEDURE cambiar_oficio (
   2       num_empleado NUMBER,     -- En los parámetros ..
   3       nuevo_oficio VARCHAR2)   -- ..no se especifica tamaño
   4   AS
   5       v_anterior_oficio emple.oficio%TYPE;
   6   BEGIN
   7       SELECT oficio INTO v_anterior_oficio FROM emple
   8         WHERE emp_no = num_empleado;
   9
  10       UPDATE emple SET oficio = nuevo_oficio
  11           WHERE emp_no = num_empleado;
  12       DBMS_OUTPUT.PUT_LINE(num_empleado||
  13         '*Oficio Anterior:'||v_anterior_oficio||
  14         '*Oficio Nuevo   :'||nuevo_oficio );
  15   END cambiar_oficio;
```

Para volver a compilar un subprograma almacenado en la base de datos se emplea la orden ALTER con la opción COMPILE, indicando PROCEDURE o FUNCTION, según el tipo de subprograma:

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

Para borrar un subprograma, igual que para eliminar otros objetos, se usa la orden DROP seguida del tipo de subprograma (PROCEDURE o FUNCTION):

```
DROP {PROCEDURE|FUNCTION} nombresubprograma;
```

E. Subprogramas locales

Los **subprogramas locales** son procedimientos y funciones que se crean dentro de otro subprograma o de un bloque, al final de la sección declarativa:

```
CREATE OR REPLACE pr_ejem1 /* programa ppal. (contiene el
local)  */
  AS
  ... /* lista de declaraciones: variables, etc. */
```

```
PROGRAM sprloc1 /* comienza el subprograma local */
...  /*lista de parámetros del subprograma local */
IS
...  /*declaraciones locales al subprograma local */
BEGIN
...  /*instrucciones del subprograma local */
END;
```

```
BEGIN
  ...
  sprloc1; /* llamada al subprograma local */
  ...
END pr_ejem1;
```

Estos subprogramas tienen las siguientes particularidades:

- Se declaran al final de la sección declarativa de otro subprograma o bloque.
- Sólo están disponibles en el bloque en que se crearon y los bloques hijos de éste. Se les aplica las mismas reglas de ámbito y visibilidad que a las variables declaradas en el mismo bloque.
- Se utilizará este tipo de subprogramas cuando no se contemple su reutilización por otros subprogramas (distintos de aquel en el que se declaran).
- En el caso de subprogramas locales con referencias cruzadas o de subprogramas mutuamente recursivos, hay que realizar *declaraciones anticipadas*, tal como se explica a continuación.

PL/SQL necesita que todos los identificadores, incluidos los subprogramas, estén declarados antes de usarlos. Por eso, la llamada que aparece en el siguiente subprograma es ilegal:

```
DECLARE
   ...
   PROCEDURE subprograma1 ( ... ) IS
   BEGIN
      ...
      Subprograma2( ... );  -- > ERR. identificador no
declarado
      ...
   END;
```

Las **declaraciones anticipadas** permiten definir subprogramas en el orden que queramos, incluso programas mutuamente recursivos.

```
PROCEDURE subprograma2 ( ... ) IS
BEGIN
   ...
END;
...
```

Se podía haber invertido el orden de declaración de los procedimientos anteriores, lo cual hubiera resuelto este caso, pero no sirve para procedimientos mutuamente recursivos. El problema se resuelve utilizando declaraciones anticipadas de subprogramas.

```
DECLARE
   PROCEDURE subprograma2 (...);  -- declaración anticipada
       PROCEDURE subprograma1 ( ... ) IS
   BEGIN
      Subprograma2( ... );      -- > ahora no dará error.
      ...
   END;
   PROCEDURE subprograma2 ( ... ) IS
   BEGIN
      ...
   END;
   ...
```

F. Recursividad

No se deben usar algoritmos recursivos en conjunción con cursores FOR LOOP, ya que se puede exceder el número máximo de cursores abiertos. Siempre se pueden sustituir las estructuras recursivas por bucles.

PL/SQL implementa, al igual que la mayoría de los lenguajes de programación, la posibilidad de escribir subprogramas recursivos:

```
CREATE OR REPLACE FUNCTION factorial
  (n POSITIVE)
RETURN INTEGER                      -- > devuelve n!
AS
BEGIN
   IF n = 1 THEN        -- > condición de terminación
      RETURN 1;
   ELSE
      RETURN n * factorial(n - 1); -- > llamada recursiva
   END IF;
END factorial;
```

G. Sobrecarga en los nombres de subprogramas

Teniendo en cuenta las conversiones automáticas y la posibilidad de valores por defecto, la sobrecarga de nombres de programas se debe usar solamente cuando hay total seguridad.

PL/SQL permite sobrecarga en los nombres de subprogramas dentro de paquetes (los veremos más adelante) siempre que los parámetros formales de los subprogramas difieran en número, orden y/o tipo de dato (familia de tipo de dato). Por ejemplo, podemos crear las variables:

- **buscar_emple(NUMBER)** que recibe un número de empleado y lo busca en la base de datos.
- **buscar_emple(VARCHAR2)** recibe un nombre, lo busca y muestra los datos encontrados.

En realidad, son dos procedimientos distintos (aunque tengan similitudes, el código es distinto). Oracle los diferencia en las llamadas (y otros comandos) por el contexto, en este caso por los parámetros.