-

TEMA 8.
PL SQL. PROCEDIMIENTOS Y FUNCIONES

Base de Datos
CFGS DAW

Francisco Aldarias Raya
paco.aldarias@ceedcv.es
2019/2020
Fecha 05/05/20
Versión:200505.1302

Licencia

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

TEMA 8. PL SQL. PROCEDIMIENTOS Y FUNCIONES

1. INTRODUCCIÓN

Como hemos visto en la parte de introducción al PL/SQL existen bloques de PL/SQL. Estos bloques PL/SQL pueden ser bloques anónimos (scripts) y subprogramas.
Hasta ahora hemos estado creando bloques anónimos que hemos guardado en ficheros para ejecutarlos como scripts.
Sin embargo Oracle permite la creación procedimientos y funciones que quedarán almacenados en la base de datos y podrán ser llamados desde la línea de comandos o desde otras funciones o procedimientos para su ejecución.

> ♦ Los subprogramas son bloques de PL/SQL a los que asignamos un nombre identificativo.

Además,

> ~ Hay que tener en cuenta que los subprogramas pueden recibir parámetros.

Los subprogramas pueden ser de varios tipos:
• Procedimientos almacenados
• Funciones
• Triggers o cursores (que veremos en la siguiente parte de de esta unidad)

2. PROCEDIMIENTOS

> ♦ Un procedimiento es un subprograma que ejecuta una acción especifica y que no devuelve ningún valor.

Los procedimientos son compilados y almacenados en la base de datos. Gracias a ellos se consigue una reutilización eficiente del código, ya que se puede invocar al procedimiento las veces que haga falta desde otro código o desde una herramienta de desarrollo como Oracle Developer. Una vez almacenados pueden ser modificados de nuevo.

Un procedimiento tiene un nombre que le asignamos tal y como acabamos de decir, así como un conjunto de parámetros (opcional) y un bloque de código. Veamos su sintaxis:

```
CREATE [OR REPLACE] PROCEDURE
```

```
[esquema].nombreprocedimiento (nombre_parámetro {IN, OUT, IN OUT} tipo_de_dato, ..)

{IS ,AS}

Declaración de variables;

Declaración de constantes;

Declaración de cursores;

BEGIN

Cuerpo del procedimiento PL/SQL;

[EXCEPTION

Bloque de excepciones PL/SQL;]

END;
```

Comenzamos con la cláusula CREATE PROCEDURE para crear un nuevo procedimiento con el nombre que indiquemos y si añadimos OR REPLACE, si ya existe un procedimiento con ese nombre, será reemplazado por éste.
Además los procedimientos pueden llevar parámetros y para cada parámetro debemos indicar si es de entrada (IN, el parámetro debe tener un valor en el momento de llamar al procedimiento) de salida (OUT, tomará el valor de salida durante la ejecución del procedimiento) o de entrada/salida (IN OUT) y además debemos indicar el tipo de dato de ese parámetro, aunque no su tamaño. Es decir sería VARCHAR2 y no VARCHAR2(50) .
•Parámetros IN. Son los parámetros que en otros lenguajes se denominan como parámetros por valor. El procedimiento recibe una copia del valor o variable que se utliia como parámetro al llamar al procedimiento. Estos parámetros pueden ser: valores literales (ej. 18), variables (ej. v_num) o expresiones (ej. v_num+18). A estos parámetros se les puede asignar un valor por defecto.
• Parámetros OUT. Relacionados con el paso por variable de otros lenguajes. Sólo pueden ser variables y no pueden tener un valor por defecto. Se utliian para que el procedimiento almacene en ellas algún valor. Es decir, los parámetros OUT son variables sin declarar que se envían al procedimiento de modo que si en el procedimiento cambian su valor, ese valor permanece en ellas cuando el procedimiento termina.
• Parámetros IN OUT. Son una mezcla de los dos anteriores. Se trata de variables declaradas anteriormente cuyo valor puede ser utilizado por el procedimiento que, además, puede almacenar un valor en ellas. No se les puede asignar un valor por defecto.

Si hay más de un parámetro irán separados por comas. Todos los parámetros irán encerrados entre paréntesis (los paréntesis son obligatorios aunque no haya parámetros en la definición del procedimiento, no así en su llamada).
Después tenemos las palabras reservadas IS/AS, que podrán utilizarse indistintamente y seguidamente la declaración de variables, constantes y cursores (en breve veremos lo que son los cursores). A continuación el cuerpo del procedimiento y las excepciones. Las excepciones

representan el código a ejecutar en caso de que ocurra algún suceso inesperado, también las veremos más profundamente en los apartados siguientes.

Veamos unos ejemplos de procedimiento y cómo utilizarlo.

2.1 Ejemplo 1

Vamos a hacer un procedimiento que sume dos números. Recibirá tres parámetros, dos números a sumar y el parámetro donde se guardará el resultado.

```
CREATE OR REPLACE PROCEDURE
SUMA_DOS_NUMEROS (primero IN NUMBER, segundo IN NUMBER, resultado OUT NUMBER) IS
BEGIN
resultado:=primero+segundo;
END;
/
```

El procedimientos se llama SUMA_DOS_NUMEROS y recibe tres parámetros: primero que es de entrada de tipo NUMBER, segundo que también es de entrada de tipo NUMBER y por último resultado que es de salida (OUT) y también de tipo NUMBER.
Ahora para ejecutarlo crearemos el archivo pl19a.sql desde donde vamos a llamar al procedimiento simplemente indicando su nombre y pasando los parámetro indicados en el paso anterior:

```
SET SERVEROUTPUT ON
DECLARE
A INT :=5;
B INT :=7;
R INT :=0;
BEGIN
SUMA_DOS_NUMEROS(A,B,R);
DBMS_OUTPUT.PUT_LINE('La suma de '||A||' y '||B||' es '||R);
END;
/
```

Cuando hacemos la llamada al procedimiento SUMA_DOS_NUMEROS(A,B,R) la asignación de valores se realiza de forma posicional, es decir por el lugar que ocupan en la llamada, de esta forma el valor de A (5 en este caso) es pasado al parámetro primero, el valor de B (7 en este caso) es pasado al parámetro segundo y por último el valor que calcula el procedimiento es devuelto en la variable R que tomará el valor final de la variable resultado del procedimiento.

El resultado será:

```
SQL> @ c:\src\pl19.sql

Procedure created.

SQL> @ c:\src\pl19a.sql
La suma de 5 y 7 es 12

PL/SQL procedure successfully completed.
```

Oracle proporciona la vista USER_SOURCE donde podemos ver los procedimientos y demás elementos de código que crea el usuario.

La descripción de la vista es:

```
SQL> desc user_source;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------

 NAME                                               VARCHAR2(30)
 TYPE                                               VARCHAR2(12)
 LINE                                               NUMBER
 TEXT                                               VARCHAR2(4000)
```

Puedes ver el código fuente del procedimiento con el campo TEXT.

```
SQL> select text from user_source
  2  where name='SUMA_DOS_NUMEROS';

TEXT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------------------------------
PROCEDURE
SUMA_DOS_NUMEROS (primero IN NUMBER, segundo IN NUMBER, resultado OUT NUMBER)
IS
BEGIN
resultado:=primero+segundo;
END;

6 rows selected.
```

Disponemos también de la vista USER_PROCEDURES que contiene una fla por cada procedimiento o función que tenga almacenado el usuario actual.

Por último, se puede eliminar un procedimiento creado utilizando la instrucción:

DROP PROCEDURE nombre_procedimiento;

```
SQL> DROP PROCEDURE SUMA_DOS_NUMEROS;

Procedure dropped.
```

2.2 Ejemplo 2

Vamos a realizar un procedimiento (pl20.sql) que nos dé el precio medio de los artículos de una gama que recibirá como parámetro. Si la gama no existe devolverá 0 como resultado. Después realizaremos un script (pl20a.sql) para llamar a ese procedimientos dos veces y mostrar el resultado en pantalla.

(Ojo, este procedimiento tene errores)

```
CREATE OR REPLACE PROCEDURE
PM_VENTA_GAMA (LA_GAMA IN VARCHAR2(50), PM OUT NUMBER(10,2))
IS
BEGIN
SELECT AVG(PRECIOVENTA) INTO PM
FROM PRODUCTOS
WHERE GAMA = LA_GAMA;
EXCEPTION
WHEN NO_DATA_FOUND THEN
PM := 0;
END;
/
```

El procedimiento recibe una gama como parámetro de entrada (LA_GAMA) y devuelve el precio medio de los artículos de esa gama (PM). Vamos a ejecutar el script de creación del procedimiento:

```
SQL> @ c:\src\pl20.sql

Warning: Procedure created with compilation errors.

SQL> show errors;
Errors for PROCEDURE PM_VENTA_GAMA:

LINE/COL ERROR
-------- -----------------------------------------------------------------
2/35     PLS-00103: Encountered the symbol "(" when expecting one of the
         following:
         := . ) , @ % default character
         The symbol ":=" was substituted for "(" to continue.

2/54     PLS-00103: Encountered the symbol "(" when expecting one of the
         following:
         := . ) , @ % default character
         The symbol ":=" was substituted for "(" to continue.
```

Al ejecutarlo se han producido errores de compilación. Para ver los errores utilizaremos SHOW ERRORS que nos mostrará las líneas y las columnas donde se ha encontrado los errores. En nuestro caso ambos errores son en la línea 2 en las columnas 35 y 54. Los errores que hemos cometido ha sido dimensionar los parámetros (en lugar de VARCHAR2(50) solo hay que poner VARCHAR2, y en lugar de NUMBER(10,2) solo hay que poner NUMBER).

Después de realizar las modificaciones y guardarlas ejecutaremos de nuevo el script, ahora ya sin problemas.

```
SQL> @ c:\src\pl20.sql

Procedure created.
```

Ahora crearemos el script donde vamos a utilizar el procedimiento que acabamos de crear:

```
SET SERVEROUTPUT ON
DECLARE
MI_GAMA VARCHAR2(50);
MEDIA NUMBER (10,2);
BEGIN
MI_GAMA:='Frutales';
PM_VENTA_GAMA(MI_GAMA,MEDIA);
DBMS_OUTPUT.PUT_LINE('El precio medio de la gama '||MI_GAMA||' es '||
MEDIA);
MI_GAMA:='Ornamentales';
PM_VENTA_GAMA(MI_GAMA,MEDIA);
DBMS_OUTPUT.PUT_LINE('El precio medio de la gama '||MI_GAMA||' es '||
MEDIA);
END;
/
```

Podemos observar que llamamos dos veces al procedimiento cambiando el valor de la variable MI_GAMA.
El resultado será:

```
SQL> @ c:\src\pl20a.sql
El precio medio de la gama Frutales es 23,93
El precio medio de la gama Ornamentales es 23,31

PL/SQL procedure successfully completed.
```

3. FUNCIONES

♦ La función es un tipo especial de procedimiento. La diferencia estriba sólo en que éstas devuelven un valor y esa devolución la realiia directamente la función, no es necesario emplear un parámetro OUT para recogerlo, tal y como hemos realizado hasta ahora en los procedimientos.

Para devolver el valor desde dentro de la función se utilizará la instrucción RETURN. Su sintaxis es la siguiente:

```
CREATE [OR REPLACE] FUNCTION
[esquema].nombrefuncion (nombre_parámetro tipo_de_dato, ..)
RETURN tipo_de_dato
{IS, AS}

Declaración de variables;

Declaración de constantes;

Declaración de cursores;
BEGIN
Cuerpo del subprograma PL/SQL;
RETURN resultado;
[EXCEPTION
Bloque de excepciones PL/SQL;]
END;
```

También puedes eliminar una función creada utilizando la instrucción:

```
DROP FUNCTION nombreFunción;
```

3.1 Ejemplo 3

Vamos a realizar una función que reciba dos valores numéricos y nos devuelva la suma de ellos.

```
CREATE OR REPLACE FUNCTION
SUMA_2_NUMEROS (PRIMERO IN NUMBER, SEGUNDO IN NUMBER)
RETURN NUMBER
IS
RESULTADO NUMBER := 0;
BEGIN
RESULTADO := PRIMERO + SEGUNDO;
```

```
RETURN RESULTADO;
END;
/
```

Podemos observar como ahora sólo recibe dos parámetros de entrada (PRIMERO y SEGUNDO) e indica que la función va a devolver un valor de tipo NUMBER con (RETURN NUMBER).

Además después de calcular el resultado se emplea la instrucción RETURN RESULTADO para devolver el cálculo realizado.

```
SQL> @ c:\src\pl21.sql

Function created.
```

Para utilizar la función que acabamos de crear utilizaremos el script:

```
SET SERVEROUTPUT ON
DECLARE
A INT :=9;
B INT :=7;
R INT :=0;
BEGIN
R := SUMA_2_NUMEROS(A,B);
DBMS_OUTPUT.PUT_LINE('La suma de '||A||' y '||B||' es '||R);
END;
/
```

Donde llamamos a la función con la instrucción R := SUMA_2_NUMEROS(A,B). En este caso se ejecutará la función con los parámetros A y B, es decir el valor de A (9 en este caso) pasará al parámetro PRIMERO y el valor de B (7 en este caso) pasará al parámetro SEGUNDO.

Una vez ejecutada la función, el valor que devuelve mediante la instrucción RETURN será recogido por la variable R.

```
SQL> @ c:\src\pl21a.sql
La suma de 9 y 7 es 16

PL/SQL procedure successfully completed.
```

Aunque las funciones sólo devuelven un valor, también podemos utilizar sus parámetros de salida (OUT) como en los procedimientos para recibir más de un dato (aunque perderían su filosofía). Veamos un ejemplo:

3.2 Ejemplo 4

```
CREATE OR REPLACE FUNCTION DIVISION (DIVIDENDO IN NUMBER, DIVISOR IN
NUMBER, RESTO OUT NUMBER)
RETURN NUMBER
IS
BEGIN
RESTO := MOD(DIVIDENDO,DIVISOR);
RETURN DIVIDENDO / DIVISOR;
END;
/
```

Para utilizar la función anterior crearemos el script:

```
SET SERVEROUTPUT ON
DECLARE
N1 INT := 10;
N2 INT := 3;
C NUMBER;
R NUMBER;
BEGIN
C := DIVISION(N1,N2,R);
DBMS_OUTPUT.PUT_LINE('EL COCIENTE ES: '||C||' Y EL RESTO: '||R);
END;
/
```

El resultado de la ejecución será:

```
SQL> @ c:\src\pl21b.sql

Function created.

SQL> @ c:\src\pl21c.sql
EL COCIENTE ES: 3,33333333333333333333333333333333333333 Y EL RESTO: 1

PL/SQL procedure successfully completed.
```

3.3 Ejemplo 5

Se trata de realizar una función que reciba como parámetro un número de pedido y devuelva el importe total de ese pedido.

```
CREATE OR REPLACE FUNCTION TOTAL_PEDIDO (NUMPEDIDO IN NUMBER)
RETURN NUMBER
IS
RESULTADO NUMBER := 0;
BEGIN
SELECT SUM(CANTIDAD * PRECIOUNIDAD) INTO RESULTADO
```

```
FROM DETALLEPEDIDOS
WHERE CODIGOPEDIDO = NUMPEDIDO;
RETURN RESULTADO;
END;
/
```

Si la ejecutamos obtenemos:

```
SQL> @ C:\SRC\PL22.SQL

Function created.
```

Para probar el funcionamiento crearemos el archivo pl22a.sql con el siguiente contenido que nos mostrará el total de cada pedido cuyo código de pedido se encuentre entre el 100 y el 105.

```
SET SERVEROUTPUT ON
BEGIN
FOR I IN 100 .. 105 LOOP
DBMS_OUTPUT.PUT_LINE('El importe del pedido '||I||' es '|| TOTAL_PEDIDO(I));
END LOOP;
END;
/
```

Al ejecutarlo obtendremos:

```
SQL> @ C:\SRC\PL22A.SQL
El importe del pedido 100 es 800
El importe del pedido 101 es 209
El importe del pedido 102 es 660
El importe del pedido 103 es 304
El importe del pedido 104 es 1760
El importe del pedido 105 es 1506

PL/SQL procedure successfully completed.
```

Y La gran ventaja que ofrecen las funciones frente a los procedimientos es que una vez que se han almacenado en la base de datos pueden ser utilizadas en nuestras consultas.

Por ejemplo en un SELECT de la siguiente forma:

```
SQL> SELECT CODIGOPEDIDO, TOTAL_PEDIDO(CODIGOPEDIDO)
  2  FROM PEDIDOS
  3  WHERE CODIGOPEDIDO BETWEEN 100 AND 105
  4  ORDER BY CODIGOPEDIDO;

CODIGOPEDIDO TOTAL_PEDIDO(CODIGOPEDIDO)
------------ --------------------------
         100                        800
         101                        209
         102                        660
         103                        304
         104                       1760
         105                       1506

6 rows selected.
```

Hay que tener en cuenta que para que las funciones puedan ser invocadas desde SQL, éstas tienen que cumplir que:

- Sólo valen funciones que se hayan compilado y almacenado en la bbdd.
- Sólo pueden utilizar parámetros de tipo IN.
- Sus parámetros deben ser de tipos compatibles con el lenguaje SQL (no valen tipos específicos de PL/SQL como BOOLEAN, por ejemplo).
- El tipo devuelto debe ser compatible con SQL.
- No pueden contiener instrucciones DML.
- Si una instrucción DML modifica una determinada tabla, en dicha instrucción no se puede invocar a una función que realice consultas sobre la misma tabla.
- No pueden utilizar instrucciones de transacciones (COMMIT, ROLLBACK ,...)
- La función no puede invocar a otra función que se salte alguna de las reglas anteriores.

3.4 Recursividad

En PL/SQL la recursividad (el hecho de que una función pueda llamarse a sí misma) está permitida.

Veamos un ejemplo:

```
CREATE FUNCTION Factorial (n IN NUMBER)
IS
BEGIN
IF (n<=1) THEN
RETURN 1
ELSE
RETURN n * Factorial(n-1);
END IF;
END;
/
```