

Facultad: Ingeniería
Escuela: Computación
Asignatura: Base de datos II

Tema: PL/SQL y PROCEDIMIENTOS ALMACENADOS

Objetivo

Conocer la definición y utilización de funciones y procedimientos en Oracle.

Conocer la definición y utilización de las secuencias en Oracle.

Materiales

- Oracle 12 c
- Guía Número 6

Introducción

SQL es un lenguaje de conjuntos muy poderoso, cuyo único objetivo es manipular el contenido de bases de datos relacionales. Sin embargo, SQL no se puede utilizar para implementar toda la lógica de negocios y la funcionalidad que el usuario final necesita en nuestras aplicaciones. Esto nos lleva a PL/SQL.

PL/SQL significa Procedural Language/Structured Query Language (una extensión de programación estructurada sobre SQL). PL/SQL ofrece un conjunto de instrucciones clásicas de la programación estructurada (instrucción condicional IF, loops o iteraciones, asignaciones), organizado dentro de bloques (lo que se explica más adelante), que complementan y amplían el alcance de SQL.

Sin duda que es posible crear aplicaciones sobre Oracle y SQL sin usar PL/SQL. Sin embargo, utilizar PL/SQL para realizar operaciones específicas de bases de datos, particularmente la ejecución de sentencias SQL, ofrece varias ventajas, incluyendo una estrecha integración con SQL, un mejor rendimiento a través del tráfico de red reducido, y la portabilidad (los programas PL/SQL pueden correr en cualquier instancia de base de datos Oracle). Por lo tanto, el código del front-end de muchas aplicaciones ejecuta tanto sentencias SQL como bloques PL/SQL, para maximizar el rendimiento al tiempo que mejora la capacidad de mantenimiento de las aplicaciones.

Construyendo bloques de programas PL/SQL

PL/SQL es un lenguaje estructurado con bloques. Un bloque PL/SQL es definido por las palabras clave DECLARE, BEGIN, EXCEPTION, y END, que dividen el bloque en tres secciones

1. **Declarativa:** sentencias que declaran variables, constantes y otros elementos de código, que después pueden ser usados dentro del bloque
2. **Ejecutable:** sentencias que se ejecutan cuando se ejecuta el bloque

3. Manejo de excepciones: una sección especialmente estructurada para atrapar y manejar cualquier excepción que se produzca durante la ejecución de la sección ejecutable

Sólo la sección ejecutable es obligatoria. No es necesario que usted declare nada en un bloque, ni que maneje las excepciones que se puedan lanzar.

Un bloque es en sí mismo una sentencia ejecutable, por lo que se pueden anidar los bloques unos dentro de otros.

Ejemplos:

El clásico “¡Hola Mundo!” es un bloque con una sección ejecutable que llama al procedimiento DBMS_OUTPUT.PUT_LINE para mostrar texto en pantalla:

```
BEGIN
  DBMS_OUTPUT.put_line('¡Hola Mundo!');
END;
```

Las funciones y procedimientos —tipos de bloques con un nombre— son discutidos con mayor detalle más adelante en este artículo, así como los paquetes. En pocas palabras, sin embargo, un paquete es un contenedor de múltiples funciones y procedimientos. Oracle extiende PL/SQL con muchos paquetes incorporados en el lenguaje.

El siguiente bloque declara una variable de tipo VARCHAR2 (un string) con un largo máximo de 100 bytes para contener el string ‘¡Hola Mundo!’. Después, el procedimiento DBMS_OUTPUT.PUT_LINE acepta la variable, en lugar del literal, para desplegarlo:

```
DECLARE
  l_mensaje VARCHAR2(100) := '¡Hola Mundo!';
BEGIN
  DBMS_OUTPUT.put_line(l_mensaje);
END;
```

Note que he llamado a la variable l_mensaje. Normalmente uso el prefijo l_ para variables locales —variables definidas dentro del código de un bloque— y el prefijo g_ para variables globales definidas en un paquete.

El siguiente ejemplo de bloque agrega una sección de manejo de excepciones que atrapa cualquier excepción (WHEN OTHERS) que pueda ser lanzada y muestra el mensaje de error, que es retornado por la función SQLERRM (provista por Oracle).

```
DECLARE
  l_mensaje VARCHAR2(100) := '¡Hola Mundo!';
BEGIN
  DBMS_OUTPUT.put_line(l_mensaje);
EXCEPTION
  WHEN OTHERS
  THEN
    DBMS_OUTPUT.put_line(SQLERRM);
END;
```

El siguiente ejemplo de bloque demuestra la habilidad de PL/SQL de anidar bloques dentro de bloques así como el uso del operador de concatenación (||) para unir múltiples strings.

```

DECLARE
    l_mensaje VARCHAR2(100) := '¡Hola';
BEGIN
    DECLARE
        l_mensaje2 VARCHAR2(100) := l_mensaje || ' Mundo!';
    BEGIN
        DBMS_OUTPUT.put_line(l_mensaje2);
    END;
EXCEPTION
    WHEN OTHERS
    THEN
        DBMS_OUTPUT.put_line(DBMS_UTILITY.format_error_stack);
END;

```

PROCEDIMIENTOS ALMACENADOS

Oracle permite acceder y manipular información de la base de datos definiendo objetos procedurales (subprogramas) que se almacenan en la base de datos. Estos objetos procedurales son unidades de programa PL/SQL: Funciones y Procedimientos almacenados.

Los procedimientos o funciones son bloques PL/SQL con nombre, que pueden recibir parámetros y pueden ser invocados desde distintos entornos: SQL*PLUS, Oracle*Forms, desde otros procedimientos y funciones y desde otras herramientas Oracle y aplicaciones.

Los procedimientos y funciones llevan a cabo tareas específicas, y su mayor diferencia radica en que las funciones devuelven un valor.

Procedimientos y funciones

```

CREATE [OR REPLACE] PROCEDURE [esquema].nombre-procedimiento
    (nombre-parámetro {IN | OUT | IN OUT} tipo de dato, ..) {[S] AS}
    Declaración de variables;
    Declaración de constantes;
    Declaración de cursores;
    BEGIN
        Cuerpo del subprograma PL/SQL;
    EXCEPTION
        Bloque de excepciones PL/SQL;
    END;

```

Sintaxis funciones

```

CREATE [OR REPLACE] FUNCTION [esquema].nombre-función
(nombre-parámetro {IN | OUT | IN OUT} tipo-de-dato, ...)
RETURN tipo-de-dato {IS | AS}
  Declaración de variables;
  Declaración de constantes;
  Declaración de cursores;
BEGIN
  Cuerpo del subprograma PL/SQL;
EXCEPTION
  Bloque de excepciones PL/SQL;
END;

```

Descripción de la sintaxis:

Nombre-parámetro: es el nombre que queramos dar al parámetro. Podemos utilizar múltiples parámetros. En caso de no necesitarlos, podemos omitir los paréntesis.

IN: especifica que el parámetro es de entrada y que por tanto dicho parámetro tiene que tener un valor en el momento de llamar a la función o procedimiento. Si no se especifica nada, los parámetros son por defecto de tipo entrada.

OUT: especifica que se trata de un parámetro de salida. Son parámetros cuyo valor es devuelto después de la ejecución el procedimiento al bloque PL/SQL que lo llamó. Las funciones PLSQL no admiten parámetros de salida.

IN OUT: Son parámetros de entrada y salida a la vez.

Tipo-de-dato: Indica el tipo de dato PLSQL que corresponde al parámetro (NUMBER, VARCHAR2, etc)

PROCEDIMIENTO

Ejemplo 1

La Figura 1 muestra un ejemplo de ejecución del más simple de los bloques de ejemplo de nuestro “¡Hola Mundo!” en SQL*Plus.



```
SQL*Plus: Release 10.2.0.5.0 - Production on Mon Jan 23 15:34:39 2012
Copyright (c) 1982, 2010, Oracle. All Rights Reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

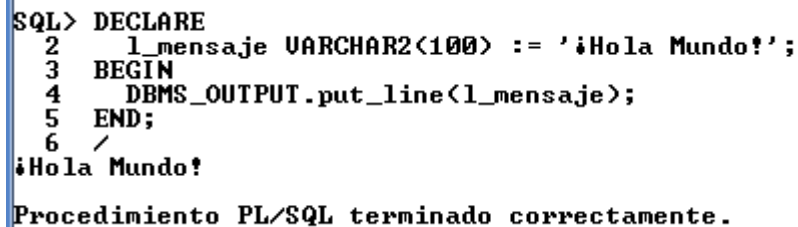
SQL> set serveroutput on
SQL> begin
  2     dbms_output.put_line('¡Hola Mundo!');
  3 end;
  4 /
¡Hola Mundo!

PL/SQL procedure successfully completed.
```

Figura 1: Ejecutando “¡Hola Mundo!” en SQL*Plus

Lo primero que hacemos después de conectarnos a la base mediante SQL*Plus es habilitar la salida del servidor, por lo que las llamadas a DBMS_OUTPUT.PUT_LINE resultarán en la visualización de texto en la pantalla. Luego escribimos el código que constituye nuestro bloque. Finalmente, ingresamos una barra (/) para decirle a SQL*Plus que ejecute ese código.

Ejemplo 2



```
SQL> DECLARE
  2     l_mensaje VARCHAR2(100) := '¡Hola Mundo!';
  3 BEGIN
  4     DBMS_OUTPUT.put_line(l_mensaje);
  5 END;
  6 /
¡Hola Mundo!

Procedimiento PL/SQL terminado correctamente.
```

Ejemplo 3

```

SQL> DECLARE
2   l_mensaje VARCHAR2(100) := '¡Hola';
3   BEGIN
4   DECLARE
5   l_mensaje2 VARCHAR2(100) := l_mensaje || ' Mundo!';
6   BEGIN
7   DBMS_OUTPUT.put_line(l_mensaje2);
8   END;
9   EXCEPTION
10  WHEN OTHERS
11  THEN
12  DBMS_OUTPUT.put_line(DBMS_UTILITY.format_error_stack);
13  END;
14
15 /
¡Hola Mundo!
Procedimiento PL/SQL terminado correctamente.

```

Ejemplo 4

Procedimiento almacenado Hola Mundo

```

SQL> CREATE OR REPLACE PROCEDURE hola_mundo IS
2   l_mensaje VARCHAR2(100) := '¡Hola Mundo!';
3   BEGIN
4   DBMS_OUTPUT.put_line(l_mensaje);
5   END hola_mundo;
6   /

```

Si al final aparece un mensaje de Procedimiento compilado con advertencia para ver los errores digite **show errors;** para ver los errores que del código

Para llamar al procedimiento se utiliza la siguiente sintaxis

```

SQL> BEGIN
2   hola_mundo;
3   END;
4   /
¡Hola Mundo!
Procedimiento PL/SQL terminado correctamente.

```

Ejemplo 5

```

SQL> CREATE OR REPLACE PROCEDURE hola_mundo2 IS
2   l_mensaje VARCHAR2(100) := '¡Hola mundo2!';
3   BEGIN
4   DBMS_OUTPUT.put_line(l_mensaje);
5   END hola_mundo2;
6   /
Procedimiento creado.

```

```

1   Begin
2   hola_mundo2;
3* end;
SQL>
SQL> /
¡Hola mundo2!

```

Ejemplo 5 utilizando parámetros de entrada

```
1 CREATE OR REPLACE PROCEDURE hola_lugar (lugar_in IN VARCHAR2) IS
2   l_mensaje VARCHAR2(100);
3 BEGIN
4   l_mensaje := '¡Hola ' || lugar_in;
5   DBMS_OUTPUT.put_line(l_mensaje);
6* END hola_lugar;
SQL> /
```

Procedimiento creado.

```
SQL> BEGIN
2   hola_lugar('UDB');
3   hola_lugar('El Salvador');
4 END;
5 /
¡Hola UDB
¡Hola El Salvador
Procedimiento PL/SQL terminado correctamente.
```

Ejemplo 6.

Para este ejemplo crearemos la siguiente tabla

```
SQL> create table empleado
2   (codigo_emp number,
3   nombre varchar2(30),
4   fecha_ingreso date
5   );
```

Tabla creada.

```
SQL> insert into empleado values(12,'carlos', '14-11-2015');
```

1 fila creada.

```
SQL> commit
2   ;
```

Confirmación terminada.

```
SQL>
```

Ahora crearemos el siguiente procedimiento

```
1 CREATE OR REPLACE PROCEDURE insertar_empleado
2   (w_codigo_emp IN empleado.codigo_emp%TYPE,
3   w_nombre IN empleado.nombre%TYPE,
4   w_fecha_ingreso IN empleado.fecha_ingreso%TYPE) IS
5 BEGIN
6   INSERT INTO empleado (codigo_emp,nombre , fecha_ingreso)
7   VALUES (w_codigo_emp, w_nombre, w_fecha_ingreso);
8   COMMIT WORK;
9* END insertar_empleado;
SQL> /
```

Procedimiento creado.

En este procedimiento se ha definido el tipo de dato de los parámetros de entrada como del mismo tipo que los campos de la tabla "empleado", es decir: nombreParametro IN nombreTabla.nombreColumna%TYPE. Sería equivalente a poner: w_codigo_emp number, w_nombre varchar2

```
SQL>      begin
2  insertar_empleado(13,'Jaime','12-01-2017');
3  insertar_empleado(14,'Rafael','11-03-2015');
4  end;
5  /
```

Procedimiento PL/SQL terminado correctamente.

```
SQL> select * from empleado;
```

CODIGO_EMP	NOMBRE	FECHA_IN
12	carlos	14/11/15
13	Jaime	12/01/17
14	Rafael	11/03/15

Para ejecutar el procedimiento también puede utilizar la siguiente sintaxis

```
EXECUTE insertar_empleado(14,Juan,'11-05-2011');
```

ANALISIS DE RESULTADOS

Crear la siguiente tabla

```
SQL> create table pedido(
2  cod_producto number,
3  nombre_producto varchar2(30),
4  cant_pedido number,
5  precio_unit number(5,2),
6  total number(5,3));
```

Tabla creada.

1. Crear un procedimiento almacenado llamada ingresoP1 que permita ingresar los datos a dicha tabla, debe tomar en cuenta que la columna total es igual al precio por la cantidad

Cree la siguiente tabla

```
SQL> create table auditoria(
2  codigo number primary key,
3  producto varchar2(30),
4  total number(5,3));
```

Tabla creada.

Ahora crearemos una secuencia para hacer el código autoincremental


```
SQL> CREATE SEQUENCE caudi
2 START WITH 1
3 INCREMENT BY 1;
```

Secuencia creada.

Probando la secuencia

```
1 INSERT INTO auditoria (codigo, producto, total)
2* VALUES (caudi.nextval, 'peras', 25.3)
SQL> /
```

1 fila creada.

```
SQL>
SQL> select * from auditoria;
```

CODIGO	PRODUCTO	TOTAL
1	peras	25,3

2. Cree un nuevo procedimiento llamado ingreso_pedido2, el procedimiento realizara la misma acción del procedimiento ingresoP1, con la diferencia que cuando se inserten datos en la tabla pedidos, también deberán insertarse datos en la tabla auditoria, para que quede registro de las ventas totales de los productos vendidos