PL/SQL

Autor:

Kevin Cárdenas.

2023

Índice

1. Introducción

PL/SQL (Procedural Language/Structured Query Language) es un lenguaje de programación procedural desarrollado por Oracle Corporation para su uso con el sistema de gestión de bases de datos Oracle. Fue creado en la década de 1990 como una extensión del lenguaje SQL estándar, con el objetivo de proporcionar capacidades avanzadas de programación y lógica empresarial en el contexto de una base de datos relacional.

PL/SQL combina elementos de lenguajes de programación procedurales tradicionales, como variables, estructuras de control de flujo y subrutinas, con la potencia del lenguaje SQL para interactuar con la base de datos. Esto permite a los desarrolladores crear aplicaciones más complejas y sofisticadas que pueden aprovechar todas las capacidades de una base de datos relacional.

Una de las principales ventajas de PL/SQL es su estrecha integración con Oracle Database. Los programas PL/SQL se ejecutan directamente en el servidor de la base de datos, lo que reduce la necesidad de enviar múltiples consultas desde una aplicación cliente y minimiza la cantidad de datos transferidos a través de la red. Esto mejora significativamente el rendimiento y la eficiencia de las aplicaciones, especialmente en entornos empresariales donde el acceso a la base de datos es fundamental.

PL/SQL se utiliza ampliamente para desarrollar funciones, procedimientos almacenados, desencadenadores (triggers) y paquetes, que encapsulan la lógica de negocio y proporcionan una capa de abstracción adicional sobre los datos almacenados en la base de datos. Estas construcciones permiten una mejor organización y modularidad del código, promoviendo la reutilización y el mantenimiento eficiente de la lógica empresarial.

1.1. Bloques de PL/SQL

Un bloque de PL/SQL es una unidad básica de código en PL/SQL. Puede contener declaraciones, sentencias SQL y lógica de programación. Los bloques de PL/SQL se utilizan para encapsular la lógica de negocio y pueden ser anónimos o nombrados.

Los bloques anónimos se ejecutan de forma inmediata, mientras que los bloques nombrados se almacenan en la base de datos y se pueden invocar desde otras partes del sistema.

A continuación, se muestra un ejemplo de un bloque de PL/SQL anónimo:

```
DECLARE
nombre VARCHAR2(50) := 'Juan';
edad NUMBER := 30;
BEGIN
```

```
-- Logic of program
IF edad >= 18 THEN
DBMS_OUTPUT.PUT_LINE(nombre || ' es mayor de edad');
ELSE
DBMS_OUTPUT.PUT_LINE(nombre || ' es menor de edad');
END IF;
END;

-- Sentencias SQL
INSERT INTO empleados (nombre, salario)
VALUES ('Ana', 5000);

COMMIT;
END;
/
```

En este ejemplo, el bloque de PL/SQL anónimo comienza con la palabra clave "BEGIN" y finaliza con "/" para indicar el final del bloque. Dentro del bloque, se pueden realizar declaraciones como la declaración de variables y constantes. Además, se puede incluir lógica de programación como condicionales y sentencias SQL para manipular la base de datos.

Por otro lado, los bloques de PL/SQL nombrados se almacenan en la base de datos y se pueden invocar desde otras partes del sistema. A continuación se muestra un ejemplo de un procedimiento almacenado, que es un tipo de bloque de PL/SQL nombrado:

```
CREATE OR REPLACE PROCEDURE calcular_salario(p_empleado_id NUMBER) AS
v_salario NUMBER;
BEGIN
-- Logica para calcular el salario del empleado
SELECT salario INTO v_salario
FROM empleados
WHERE empleado_id = p_empleado_id;

DBMS_OUTPUT.PUT_LINE('El salario del empleado ' || p_empleado_id || ' es: ' || v_salario);
END;
/
```

En este ejemplo, se crea un procedimiento almacenado llamado "calcular_salario" que acepta un parámetro de empleado_id. Dentro del procedimiento, se realiza una consulta para obtener el salario del empleado correspondiente al empleado_id proporcionado. Luego, se muestra el salario utilizando la función DBMS_OUTPUT.PUT_LINE.

Además de los bloques anónimos y los procedimientos almacenados, PL/SQL también ofrece la posibilidad de crear disparadores (triggers). Los disparadores son bloques de PL/SQL que se ejecutan automáticamente en respuesta a eventos específicos que ocurren en la base de datos, como la inserción, actualización o eliminación de datos en una tabla.

A continuación se muestra un ejemplo de un disparador (trigger) que se activa después de insertar una nueva fila en la tabla "empleados":

```
CREATE OR REPLACE TRIGGER insertar_empleado_trigger
AFTER INSERT ON empleados
FOR EACH ROW
BEGIN
DBMS_OUTPUT.PUT_LINE('Nuevo empleado insertado: ' || :NEW.nombre);
END;
/
```

En este ejemplo, el disparador "insertar_empleado_trigger" se ejecuta después de cada inserción en la tabla "empleados". El bloque de PL/SQL dentro del disparador muestra un mensaje que indica el nombre del nuevo empleado que se ha insertado.

Los disparadores son una poderosa herramienta en PL/SQL que permiten automatizar acciones y aplicar lógica adicional en la base de datos en respuesta a eventos específicos.

1.2. Variables y constantes

En PL/SQL, se pueden declarar variables y constantes para almacenar y manipular datos. Las variables se utilizan para almacenar valores temporales y pueden cambiar durante la ejecución del programa. Las constantes, por otro lado, son valores fijos que no pueden modificarse una vez que se les ha asignado un valor. Tanto las variables como las constantes pueden tener diferentes tipos de datos, como enteros, caracteres, fechas, etc.

Algunos de los tipos de datos más comunes en PL/SQL. Cada tipo de dato tiene características específicas y se debe elegir según el tipo de datos que se desea almacenar y manipular en la base de datos.

- **VARCHAR2**: se utiliza para almacenar cadenas de caracteres.

```
DECLARE
    nombre VARCHAR2(50) := 'Juan';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Nombre: ' || nombre);
END;
```

- **NUMBER**: se utiliza para almacenar números enteros o decimales.

```
DECLARE
    sueldo NUMBER(10, 2) := 5000.50;
    extras NUMBER(10, 2) := 200.50;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Saldo: ' || TO_CHAR(sueldo + extras));
END;
```

- **DATE**: se utiliza para almacenar fechas y horas.

```
DECLARE
    fecha_nacimiento DATE := TO_DATE('1990/01/01', 'YYYY/MM/DD');
BEGIN
    DBMS_OUTPUT.PUT_LINE('Fecha de nacimiento: ' || TO_CHAR(fecha_nacimiento, 'DD/MM/YYYY'));
END;
```

- **BOOLEAN**: se utiliza para almacenar valores de verdadero o falso.

```
DECLARE
    es_mayor BOOLEAN := TRUE;
BEGIN
    IF es_mayor THEN
        DBMS_OUTPUT.PUT_LINE('Es mayor de edad');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Es menor de edad');
    END IF;
END;
```

- **CHAR**: se utiliza para almacenar caracteres de longitud fija.

```
DECLARE
    inicial CHAR(1) := 'A';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Inicial: ' || inicial);
END;
```

- **LONG**: se utiliza para almacenar cadenas de longitud variable.

```
DECLARE
    descripcion LONG := 'Esta es una descripcion larga';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Descripcion: ' || descripcion);
END;
```

- **RAW**: se utiliza para almacenar datos binarios.

```
DECLARE
    datos RAW(100) := UTL_RAW.CAST_TO_RAW('010101');
BEGIN
    DBMS_OUTPUT.PUT_LINE('Datos: ' || UTL_RAW.CAST_TO_VARCHAR2(datos));
END;
```

- **BLOB**: se utiliza para almacenar datos binarios grandes.

```
DECLARE
    es_mayor BOOLEAN := FALSE;
    edad INTEGER := 18;
BEGIN
    es_mayor := (edad >= 18);
    IF es_mayor THEN
        DBMS_OUTPUT.PUT_LINE('Es mayor de edad');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Es menor de edad');
    END IF;
END;
```

- **CLOB**: se utiliza para almacenar cadenas de caracteres grandes.

```
DECLARE
    descripcion CLOB := 'Esta es una descripcion larga';
BEGIN
    DBMS_OUTPUT.PUT_LINE('Descripcion: ' || descripcion);
END;
```

- **TIMESTAMP**: se utiliza para almacenar fechas y horas con precisión de fracciones de segundo.

```
DECLARE
    fecha_hora TIMESTAMP := SYSTIMESTAMP;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Fecha y hora: ' || TO_CHAR(fecha_hora, 'DD/MM/YYYY HH24:MI:SS.FF'));
```

- **INTERVAL**: se utiliza para almacenar intervalos de tiempo.

```
DECLARE
    duracion INTERVAL DAY TO SECOND := INTERVAL '3' HOUR;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Duracion: ' || duracion);
END;
```

- **RECORD**: se utiliza para almacenar un conjunto de valores relacionados.

```
DECLARE
    TYPE tipo_empleado IS RECORD (
        nombre VARCHAR2(50),
        edad NUMBER,
        sueldo NUMBER(10, 2)
    );

    empleado tipo_empleado;
BEGIN
    empleado.nombre := 'Juan';
    empleado.edad := 30;
    empleado.sueldo := 5000.50;

    DBMS_OUTPUT.PUT_LINE('Nombre: ' || empleado.nombre);
    DBMS_OUTPUT.PUT_LINE('Edad: ' || empleado.edad);
    DBMS_OUTPUT.PUT_LINE('Sueldo: ' || empleado.sueldo);
END;
```

- **TABLE**: se utiliza para almacenar conjuntos de datos en forma de tabla.

```
DECLARE
    TYPE tipo_empleados IS TABLE OF VARCHAR2(50);
    empleados tipo_empleados := tipo_empleados('Juan', 'Maria', 'Pedro');
BEGIN
    FOR i IN 1..empleados.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('Empleado ' || i || ': ' || empleados(i));
    END LOOP;
END;
```

1.3. Condicionales y bucles

PL/SQL proporciona estructuras de control de flujo, como condicionales y bucles, para tomar decisiones y repetir acciones. Los condicionales, como IF-THEN-ELSE, permiten ejecutar diferentes bloques de código según una condición específica. Los bucles, como FOR, WHILE y LOOP, permiten repetir un bloque de código hasta que se cumpla una condición o se alcance un criterio de finalización.

1.4. Matrices

Las matrices o arrays son estructuras de datos utilizadas para almacenar múltiples valores del mismo tipo en una sola variable. En PL/SQL, se pueden declarar y manipular matrices para procesar conjuntos de datos de manera eficiente. Las matrices se acceden mediante índices y se pueden utilizar en bucles y operaciones aritméticas.

1.4.1. Ejemplo de declaración y uso de matrices:

```
DECLARE
    TYPE numeros_array IS VARRAY(5) OF NUMBER;
    numeros numeros_array := numeros_array(1, 2, 3, 4, 5);
BEGIN
    FOR i IN 1..numeros.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('Numero en la posicion ' || i || ': ' || numeros(i));
    END LOOP;
END;
```

2. Procedimientos almacenados

Los procedimientos almacenados son una característica clave de PL/SQL. Son bloques de código con nombre que se almacenan en la base de datos y se utilizan para encapsular lógica de negocio compleja. Los procedimientos almacenados se pueden invocar desde otras partes del sistema, lo que permite una reutilización eficiente del código y mejora el rendimiento de las aplicaciones al reducir la necesidad de enviar múltiples consultas al servidor de base de datos.

2.1. Ejemplo de procedimiento almacenado:

```
CREATE OR REPLACE PROCEDURE calcular_salario(p_empleado_id NUMBER) AS
v_salario NUMBER;
BEGIN
    -- Logica para calcular el salario del empleado
    -- y asignarlo a la variable v_salario
    -- ...

    DBMS_OUTPUT.PUT_LINE('El salario del empleado ' || p_empleado_id || ' es: ' || v_salario);
END;
```