

**FUNDAMENTOS**

**Y**

**AVANCES**

DE

**PL/SQL & SQL PSM**

(Trabajo realizado por Carlos, Melisa y Cova)

***Conceptos Básicos***

2.1. Bloques PL/SQL: Estructura y propósito de los bloques PL/SQL.

2.2. Unidades Léxicas: Descripción y ejemplos de unidades léxicas en PL/SQL.

2.3 Tipos de Datos Simples: Exploración de los tipos de datos disponibles en PL/SQL.

2.4. Variables y Constantes: Definición y uso de variables y constantes.

2.5. Operadores y Expresiones: Cómo se utilizan los operadores y expresiones en PL/SQL.

2.6. Etiquetas: Uso y propósito de las etiquetas en el código.

2.7. Ámbito de las Variables: Explicación del alcance de las variables dentro de un programa PL/SQL.

2.8. Entrada y Salida: Interacción con el Usuario: Mecanismos de entrada/salida en PL/SQL.

2.9. Estructuras de Control: Detalle sobre las estructuras de control como: o IF -> THEN, CASE -> WHEN -> THEN, etc.

***Cursores***

3.1. Cursores Implícitos: Definición y cómo se utilizan.

3.2. Cursores Explícitos: Diferencias con los implícitos y ejemplos de uso.

***PARTE II PL/SQL AVANCES***

4.1. Tipos de Excepciones: Clasificación y ejemplos.

4.2. Propagación de Excepciones: Cómo se propagan las excepciones en PL/SQL.

4.3. Ámbito de las Excepciones: Dónde se capturan y se manejan las excepciones.

5.1. Procedimientos: Creación y uso de procedimientos en PL/SQL.

5.2. Funciones: Diferencias con los procedimientos y ejemplos de uso.

5.3. Uso de Parámetros en los Subprogramas: Cómo se pasan los parámetros a funciones y procedimientos.

5.4. Sobrecarga de Subprogramas y Recursividad: Explicación y ejemplos.

5.5. Procedimientos frente a Funciones: Cuándo usar cada uno.

6.1. Creación de Paquetes: Cómo y por qué crear paquetes en PL/SQL

* **2.1Bloques SQL-PSM: Estructura y propósito de los bloques en SQL-PSM.**

**Bloques SQL-PSM: Estructura y propósito**

Los bloques SQL-PSM (Structured Query Language-Procedural Standard) son unidades de código que permiten definir y ejecutar procedimientos almacenados y funciones dentro de una base de datos. Aquí se presenta una breve descripción de su estructura y propósito:

**Estructura de los bloques SQL-PSM:**

* Inician con las palabras clave **BEGIN** y finalizan con **END**.
* Pueden contener declaraciones de variables, instrucciones de control de flujo (como bucles y condicionales), y sentencias SQL para manipular datos.
* Se pueden anidar dentro de otros bloques.

**Propósito de los bloques SQL-PSM:**

* Permiten encapsular lógica empresarial compleja dentro de la base de datos.
* Facilitan la reutilización de código, ya que los bloques pueden ser llamados desde diferentes partes de la aplicación.
* Mejoran la seguridad al evitar la exposición directa de la lógica de la aplicación.
* Ayudan a optimizar el rendimiento al reducir el tráfico de red al mover la lógica del lado del cliente al servidor de base de datos.

En resumen, los bloques SQL-PSM proporcionan una forma estructurada y eficiente de implementar la lógica procedimental dentro de una base de datos, lo que contribuye a la modularidad, seguridad y rendimiento de las aplicaciones que interactúan con ella.

EJEMPLO:

CREATE PROCEDURE calcular\_salario\_anual\_promedio()

BEGIN

DECLARE total\_salarios DECIMAL(10, 2);

DECLARE total\_empleados INT;

DECLARE salario\_promedio DECIMAL(10, 2);

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Calcular el salario anual promedio

IF total\_empleados > 0 THEN

SET salario\_promedio = total\_salarios / total\_empleados \* 12;

SELECT salario\_promedio;

ELSE

SELECT 'No hay empleados en la tabla.';

END IF;

END;

En este ejemplo, hemos creado un procedimiento almacenado llamado **calcular\_salario\_anual\_promedio**. Este procedimiento calcula el salario anual promedio de todos los empleados en una tabla llamada **empleados**.

El bloque comienza con la palabra clave **BEGIN** y termina con **END**. Dentro del bloque, hemos declarado variables para almacenar el total de salarios, el número de empleados y el salario promedio. Luego, utilizamos una consulta SQL para obtener el total de salarios y el número de empleados en la tabla.

Después, calculamos el salario anual promedio y mostramos el resultado. Si no hay empleados en la tabla, se muestra un mensaje indicando que no hay empleados. Este es un ejemplo básico para ilustrar la estructura y el propósito de un bloque SQL-PSM.

* **2.1Bloques PL-SQL: Estructura y propósito de los bloques en PL-SQL**

**Bloques PL-SQL: Estructura y propósito**

Los bloques PL-SQL (Procedural Language/Structured Query Language) son unidades de código que permiten la ejecución de procedimientos almacenados, funciones y otros programas dentro de una base de datos Oracle.

Aquí se presenta una breve descripción de su estructura y propósito:

**Estructura de los bloques PL-SQL:**

* Inician con las palabras clave **DECLARE** o **BEGIN** y finalizan con **END**.
* Pueden contener declaraciones de variables, instrucciones de control de flujo (como bucles y condicionales), y sentencias SQL para manipular datos.
* Se pueden anidar dentro de otros bloques.
* Las excepciones se pueden manejar con bloques **EXCEPTION**.

**Propósito de los bloques PL-SQL:**

* Permiten encapsular lógica empresarial compleja dentro de la base de datos Oracle.
* Facilitan la reutilización de código y la modularidad en el desarrollo de aplicaciones.
* Mejoran la seguridad al centralizar la lógica de negocio en la base de datos.
* Ayudan a optimizar el rendimiento al ejecutar operaciones directamente en el servidor de base de datos.

En resumen, los bloques PL-SQL proporcionan una manera estructurada y eficiente de implementar la lógica procedimental dentro de una base de datos Oracle, lo que contribuye a la modularidad, seguridad y rendimiento de las aplicaciones que interactúan con ella.

EJEMPLO:

DECLARE

total\_salarios NUMBER;

total\_empleados NUMBER;

salario\_promedio NUMBER;

BEGIN

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Calcular el salario anual promedio

IF total\_empleados > 0 THEN

salario\_promedio := total\_salarios / total\_empleados \* 12;

DBMS\_OUTPUT.PUT\_LINE('El salario anual promedio es: ' || salario\_promedio);

ELSE

DBMS\_OUTPUT.PUT\_LINE('No hay empleados en la tabla.');

END IF;

END;

/

En este ejemplo, el bloque PL-SQL comienza con **DECLARE** para declarar las variables necesarias, seguido de **BEGIN** y finaliza con **END**.

Dentro del bloque, se realiza una consulta para obtener el total de salarios y el número de empleados.

Luego, se calcula el salario anual promedio y se muestra el resultado utilizando **DBMS\_OUTPUT.PUT\_LINE**. Este es un ejemplo básico para ilustrar la estructura y el propósito de un bloque PL-SQL.

* **2.2 Unidades Léxicas: Descripción y ejemplos de unidades léxicas en PL/SQL y SQL/PSM:**
* **PL/SQL**

Al igual que en nuestra lengua podemos identificar diferentes unidades léxicas para escribir código en PL/SQL.

Como cualquier lenguaje de programación, PL/SQL cuenta con sus propias **unidades léxicas.**

Son las que establecen los elementos más pequeños con significado, elementos que, al combinarlos, atendiendo a las reglas de sintaxis, se convierten en sentencias.

**PL/SQL** es un lenguaje **no sensible a las mayúsculas**, por lo que será equivalente escribir en mayúsculas o minúsculas,

excepto cuando hablemos de literales de tipo cadena o de tipo carácter.

Cada unidad léxica puede estar separada por espacios (debe estar separada por espacios si se trata de 2 identificadores), por saltos de línea o por tabuladores para aumentar la legibilidad del código escrito.

Por lo que:

IF A = CLAVE THEN

ENCONTRADO := TRUE;

ELSE

ENCONTRADO := FALSE;

END IF;

Es equivalente a escribir la siguiente línea:

if a = clave then

encontrado := true;

else

encontrado := false;

end if;

Y también equivalente a escribirlo de esta otra forma:

IF a = clave THEN

encontrado := TRUE;

ELSE

encontrado := FALSE;

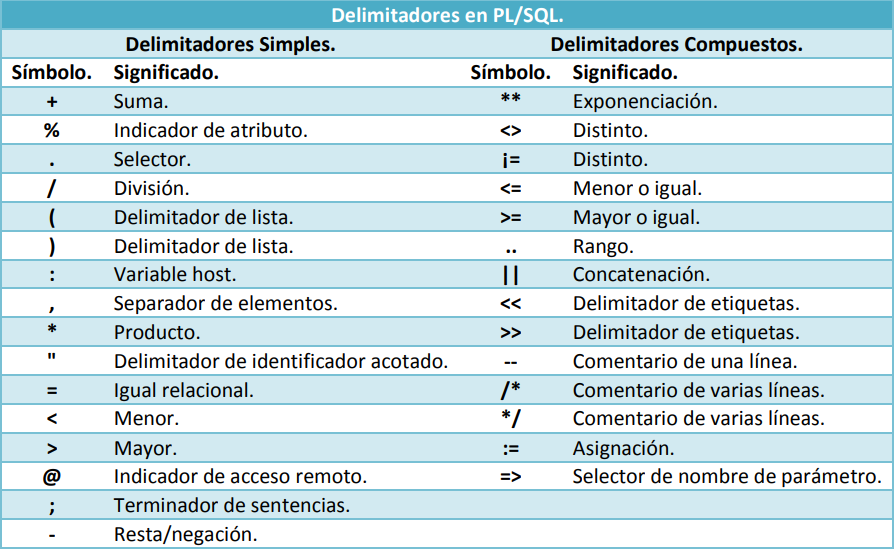
END IF;

Aclarado esto, nos adentramos en su clasificación:

En PL/SQL se clasifican en:

* **DELIMITADORES**: PL/SQL tiene un conjunto de símbolos denominados delimitadores utilizados para representar operaciones entre tipos de datos, delimitar comentarios, etc.

En la siguiente tabla nos podemos encontrar un resumen de estos:



Un ejemplo de cómo utilizar un delimitador en PL/SQL para representar datos y realizar operaciones, en este ejemplo el delimitador utilizado es “/” que marca el final del bloque.

DECLARE

nombre VARCHAR2(50);

BEGIN

-- Asignar un valor a la variable 'nombre'

nombre := 'Chema';

-- Imprimir el valor de 'nombre'

DBMS\_OUTPUT.PUT\_LINE('El nombre es: ' || nombre);

END;

/

* **LITERALES**: Los literales se utilizan en las comparaciones de valores o para asignar valores concretos a los identificadores que actúan como variables o constantes.

Para expresar estos literales se ha de tener en cuenta que:

* Los literales numéricos se expresarán con notación decimal o exponencial.

Ejemplos: 234, +341, 2e3, -2E-3, 7.45, 8.1e3.

* Los literales de tipo carácter y de tipo cadena se deben delimitar con unas comillas simples.
* Los literales lógicos son **TRUE** y **FALSE**.
* El literal **NULL** que expresa que una variable no tiene ningún valor asignado.

Un ejemplo de cómo utilizar un tipo de literal en PL/SQL para representar datos y realizar operaciones, por ejemplo, **literal de cadena (string)**:

-- Declaración de variables con nombres y departamento.

DECLARE

nombre VARCHAR2(50) := 'Juan Pérez';

departamento VARCHAR2(50) := 'Recursos Humanos';

BEGIN

-- Impresión de los datos del empleado.

DBMS\_OUTPUT.PUT\_LINE('Empleado: ' || nombre || ', Departamento: ' || departamento);

END;

* **COMENTARIOS**: En los lenguajes de programación es muy recomendable utilizar comentarios en mitad del código.

Los comentarios no afectan al código, pero ayudan mucho al programador a recordar qué se intenta hacer en cada caso (más aún cuando el código es compartido entre personas que se dedican a mejorarlo o corregirlo).

En PL/SQL podemos utilizar distintos tipos de comentarios:

* Los comentarios de una línea se expresarán por medio del delimitador:

Ejemplo: a:=b; --asignación

* Los comentarios de varias líneas se acotarán por medio de los delimitadores /\* y \*/

Ejemplo: /\* Primera línea de comentarios.

Segunda línea de comentarios. \*/

Un ejemplo de cómo utilizar un tipo de comentario en PL/SQL para representar datos y realizar operaciones, por ejemplo:

-- Declaración de variables con nombres y departamento.

DECLARE

nombre VARCHAR2(50) := 'Juan Pérez'; -- Nombre del empleado

departamento VARCHAR2(50) := 'Recursos Humanos'; -- Departamento al que pertenece

BEGIN

-- Impresión de los datos del empleado.

DBMS\_OUTPUT.PUT\_LINE('Empleado: ' || nombre || ', Departamento: ' || departamento);

END;

* **IDENTIFICADORES**: Los identificadores en PL/SQL, como en cualquier otro lenguaje de programación, son utilizados para nombrar elementos de nuestros programas.

A la hora de utilizar los identificadores debemos tener en cuenta los siguientes aspectos:

* Un identificador es una letra seguida opcionalmente de letras, números, $, \_, #.
* No podemos utilizar como identificador una palabra reservada.
* Ejemplos válidos: X, A1, codigo\_postal.
* Ejemplos no válidos: rock&roll, on/off.
* PL/SQL nos permite además definir los identificadores acotados, en los que podemos usar cualquier carácter con una longitud máxima de 30 y deben estar delimitados por ". Ejemplo: "X\*Y".
* En PL/SQL existen algunos identificadores predefinidos y que tienen un significado especial ya que nos permitirán darle sentido sintáctico a nuestros programas. Estos identificadores son las palabras reservadas y no las podemos utilizar como identificadores en nuestros programas. Ejemplo: IF, THEN, ELSE, etc
* Algunas palabras reservadas para PL/SQL no lo son para SQL, por lo que podríamos tener una tabla con una columna llamada 'type' por ejemplo, que nos daría un error de compilación al referirnos a ella en PL/SQL. La solución sería acotarlos. SELECT "TYPE", etc

Un ejemplo de cómo utilizar un tipo de identificador en PL/SQL para representar datos y realizar operaciones, representa una variable, por ejemplo:

DECLARE

mi\_variable NUMBER := 10;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('El valor de mi\_variable es: ' || mi\_variable);

END;

/

* **SQL/PSM**

En **SQL/PSM**, una unidad léxica se refiere a los elementos básicos que componen las instrucciones y comandos en SQL.

Estos elementos son reconocidos por el motor de SQL y se utilizan para construir consultas y manipular datos en bases de datos relacionales.

Sabiendo esto, nos adentramos en su clasificación:

* **PALABRAS CLAVE (KEYWORDS)**: Las palabras clave son términos reservados con un significado específico en SQL y que no pueden usarse como nombres de tablas, columnas o variables.

Algunos ejemplos de palabras clave en SQL son SELECT, INSERT, UPDATE, etc:

Un ejemplo de esta unidad léxica la encontramos en:

CREAR PROCEDIMIENTO ObtenerPromedio AS

INICIO

SELECCIONAR AVG(calificacion) COMO promedio\_calificacion DE ResultadosExamenes;

FIN;

Explicación: En este ejemplo, se utiliza la palabra clave CREAR PROCEDIMIENTO para definir un nuevo procedimiento almacenado llamado ObtenerPromedio.

El procedimiento calcula el promedio de las calificaciones de la tabla ResultadosExamenes y lo devuelve como promedio\_calificacion.

* **LITERALES**: Los literales son valores constantes que se utilizan directamente en las consultas SQL.

Pueden ser literales numéricos, literales de cadena de caracteres, etc.

Un ejemplo de esta unidad léxica lo encontramos en:

INSERTAR EN Estudiantes (ID\_estudiante, Nombre, Apellido, Edad) VALORES (101, 'Juan', 'Pérez', 25);

*Explicación: Este ejemplo inserta una nueva fila en la tabla Estudiantes. Los valores entre paréntesis (101, 'Juan', 'Pérez', 25) son literales que representan el ID del estudiante, su nombre, apellido y edad respectivamente.*

* **OPERADORES**: Los operadores se utilizan para realizar operaciones en los datos, como operadores aritméticos, de comparación, lógicos, etc.

Un ejemplo de esta unidad léxica lo encontramos en:

SELECCIONAR \* DE Productos DONDE precio < 50;

Explicación: En este ejemplo, el operador < se utiliza para realizar una comparación en la cláusula WHERE. Selecciona todas las filas de la tabla Productos donde el valor de la columna precio es menor que 50.

* **DELIMITADORES**: Los delimitadores se utilizan para separar elementos en una consulta SQL.

Por ejemplo, el punto y coma (;) se utiliza para separar instrucciones, comas (,) para separar columnas en una lista de selección, etc.

Un ejemplo de esta unidad léxica lo encontramos en:

CREAR PROCEDIMIENTO ActualizarSalario(IN idEmpleado INT, IN nuevoSalario DECIMAL(10,2)) COMO

INICIO

ACTUALIZAR Empleados SET salario = nuevoSalario DONDE id\_empleado = idEmpleado;

FIN;

*Explicación: En este ejemplo, los delimitadores ( ), ,, ; e INICIO...FIN se utilizan para definir el procedimiento almacenado ActualizarSalario.*

Los paréntesis se usan para agrupar los parámetros de entrada, la coma se utiliza para separar los parámetros, y ; marca el final de la sentencia SQL dentro del procedimiento. INICIO y FIN delimitan el bloque de código del procedimiento.

* **¿Comparación unidades léxicas PL/SQL y SQL/PSM con ejemplos?**

|  |  |  |
| --- | --- | --- |
|  | **PL/SQL** | **SQL/PSM** |
| **Palabras clave** | En PL/SQL, las palabras clave son específicas de su propio lenguaje y se utilizan para construir bloques de código, definiciones de funciones, procedimientos, etc. | En SQL/PSM, las palabras clave son específicas del estándar SQL y se utilizan para construir consultas y procedimientos almacenados. |
| **Identificadores** | Tanto en SQL/PSM como en PL/SQL, los identificadores se utilizan para nombrar objetos como tablas, columnas, variables, procedimientos, funciones, etc.  Sin embargo, la sintaxis exacta para los identificadores y las reglas de nomenclatura pueden diferir ligeramente entre SQL/PSM y PL/SQL. | |
| **Literales** | Los literales, como las cadenas de texto, los números y las fechas, se utilizan de manera similar en ambas variantes para representar valores constantes en consultas y procedimientos. | |
| **Operadores** | Los operadores aritméticos, de comparación, lógicos, etc., son comunes tanto en SQL/PSM como en PL/SQL. | |
| **Delimitadores** | Los delimitadores como comas, puntos y comas, paréntesis, etc., se utilizan en ambos lenguajes para separar elementos en instrucciones y expresiones. | |
| **Ejemplos código** | CREATE OR REPLACE PROCEDURE CalcularTotalVentas IS  total\_ventas NUMBER;  BEGIN  SELECT SUM(monto) INTO total\_ventas FROM Ventas;  DBMS\_OUTPUT.PUT\_LINE('Total de ventas: ' || total\_ventas);  END;  /  ---------------------------------   * **Palabras clave**: CREATE, OR, REPLACE, PROCEDURE, IS, BEGIN, SELECT, INTO, FROM, END. * **Identificadores**: CalcularTotalVentas, total\_ventas. * **Literales**: 'Total de ventas: '. * **Operadores**: SUM, || (concatenación). * **Delimitadores**: ;. | CREATE PROCEDURE CalcularTotalVentas()  BEGIN  DECLARE total\_ventas INT;  SELECT SUM(monto) INTO total\_ventas FROM Ventas;  PRINT 'Total de ventas: ' || total\_ventas;  END;  ---------------------------------   * Palabras clave: CREATE, PROCEDURE, BEGIN, DECLARE, INT, SELECT, INTO, FROM, END. * Identificadores: CalcularTotalVentas, total\_ventas. * Literales: 'Total de ventas: '. * Operadores: SUM, || (concatenación). * Delimitadores: ;. |
| **Código** | * Las palabras clave, identificadores, literales, operadores y delimitadores son comunes tanto en PL/SQL como en SQL/PSM. * Sin embargo, la sintaxis para declarar procedimientos almacenados y algunas instrucciones específicas, como la impresión de resultados (DBMS\_OUTPUT.PUT\_LINE en PL/SQL y PRINT en SQL/PSM), pueden variar entre los dos lenguajes. | |

* **2.1Bloques SQL-PSM: Estructura y propósito de los bloques en SQL-PSM**

**Bloques SQL-PSM: Estructura y propósito**

Los bloques SQL-PSM (Structured Query Language-Procedural Standard) son unidades de código que permiten definir y ejecutar procedimientos almacenados y funciones dentro de una base de datos. Aquí se presenta una breve descripción de su estructura y propósito:

**Estructura de los bloques SQL-PSM:**

* Inician con las palabras clave **BEGIN** y finalizan con **END**.
* Pueden contener declaraciones de variables, instrucciones de control de flujo (como bucles y condicionales), y sentencias SQL para manipular datos.
* Se pueden anidar dentro de otros bloques.

**Propósito de los bloques SQL-PSM:**

* Permiten encapsular lógica empresarial compleja dentro de la base de datos.
* Facilitan la reutilización de código, ya que los bloques pueden ser llamados desde diferentes partes de la aplicación.
* Mejoran la seguridad al evitar la exposición directa de la lógica de la aplicación.
* Ayudan a optimizar el rendimiento al reducir el tráfico de red al mover la lógica del lado del cliente al servidor de base de datos.

En resumen, los bloques SQL-PSM proporcionan una forma estructurada y eficiente de implementar la lógica procedimental dentro de una base de datos, lo que contribuye a la modularidad, seguridad y rendimiento de las aplicaciones que interactúan con ella.

EJEMPLO:

CREATE PROCEDURE calcular\_salario\_anual\_promedio()

BEGIN

DECLARE total\_salarios DECIMAL(10, 2);

DECLARE total\_empleados INT;

DECLARE salario\_promedio DECIMAL(10, 2);

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Calcular el salario anual promedio

IF total\_empleados > 0 THEN

SET salario\_promedio = total\_salarios / total\_empleados \* 12;

SELECT salario\_promedio;

ELSE

SELECT 'No hay empleados en la tabla.';

END IF;

END;

En este ejemplo, hemos creado un procedimiento almacenado llamado **calcular\_salario\_anual\_promedio**.

Este procedimiento calcula el salario anual promedio de todos los empleados en una tabla llamada **empleados**.

El bloque comienza con la palabra clave **BEGIN** y termina con **END**. Dentro del bloque, hemos declarado variables para almacenar el total de salarios, el número de empleados y el salario promedio.

Luego, utilizamos una consulta SQL para obtener el total de salarios y el número de empleados en la tabla. Después, calculamos el salario anual promedio y mostramos el resultado. Si no hay empleados en la tabla, se muestra un mensaje indicando que no hay empleados.

Este es un ejemplo básico para ilustrar la estructura y el propósito de un bloque SQL-PSM.

* **2.1Bloques PL-SQL: Estructura y propósito de los bloques en PL-SQL**

**Bloques PL-SQL: Estructura y propósito**

Los bloques PL-SQL (Procedural Language/Structured Query Language) son unidades de código que permiten la ejecución de procedimientos almacenados, funciones y otros programas dentro de una base de datos Oracle. Aquí se presenta una breve descripción de su estructura y propósito:

**Estructura de los bloques PL-SQL:**

* Inician con las palabras clave **DECLARE** o **BEGIN** y finalizan con **END**.
* Pueden contener declaraciones de variables, instrucciones de control de flujo (como bucles y condicionales), y sentencias SQL para manipular datos.
* Se pueden anidar dentro de otros bloques.
* Las excepciones se pueden manejar con bloques **EXCEPTION**.

**Propósito de los bloques PL-SQL:**

* Permiten encapsular lógica empresarial compleja dentro de la base de datos Oracle.
* Facilitan la reutilización de código y la modularidad en el desarrollo de aplicaciones.
* Mejoran la seguridad al centralizar la lógica de negocio en la base de datos.
* Ayudan a optimizar el rendimiento al ejecutar operaciones directamente en el servidor de base de datos.

En resumen, los bloques PL-SQL proporcionan una manera estructurada y eficiente de implementar la lógica procedimental dentro de una base de datos Oracle, lo que contribuye a la modularidad, seguridad y rendimiento de las aplicaciones que interactúan con ella.

**EJEMPLO**:

DECLARE

total\_salarios NUMBER;

total\_empleados NUMBER;

salario\_promedio NUMBER;

BEGIN

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Calcular el salario anual promedio

IF total\_empleados > 0 THEN

salario\_promedio := total\_salarios / total\_empleados \* 12;

DBMS\_OUTPUT.PUT\_LINE('El salario anual promedio es: ' || salario\_promedio);

ELSE

DBMS\_OUTPUT.PUT\_LINE('No hay empleados en la tabla.');

END IF;

END;

/

En este ejemplo, el bloque PL-SQL comienza con **DECLARE** para declarar las variables necesarias, seguido de **BEGIN** y finaliza con **END**. Dentro del bloque, se realiza una consulta para obtener el total de salarios y el número de empleados. Luego, se calcula el salario anual promedio y se muestra el resultado utilizando **DBMS\_OUTPUT.PUT\_LINE**. Este es un ejemplo básico para ilustrar la estructura y el propósito de un bloque PL-SQL.

* **2.4 Variables y Constantes: Definición y uso de variables y constantes en SQL-PSM**

**Variables y Constantes en SQL-PSM**

En SQL-PSM (Structured Query Language-Procedural Standard), las variables y constantes se utilizan para almacenar y manipular valores dentro de los bloques de código. Aquí está una definición y ejemplo de cómo se definen y usan en SQL-PSM:

Variables:

* **Definición:** Las variables son objetos que pueden almacenar valores que pueden cambiar durante la ejecución de un bloque de código.
* **Uso:** Se declaran con la palabra clave **DECLARE** y pueden asignarse valores usando la operación **:=**.

Constantes:

* **Definición:** Las constantes son objetos que almacenan valores que no cambian durante la ejecución de un bloque de código.
* **Uso:** Se definen utilizando la palabra clave **CONSTANT** o **CONST**, y su valor se establece en la declaración.

EJEMPLO:

CREATE PROCEDURE calcular\_salario\_anual\_promedio()

BEGIN

-- Declaración de variables

DECLARE total\_salarios DECIMAL(10, 2);

DECLARE total\_empleados INT;

DECLARE salario\_promedio DECIMAL(10, 2);

DECLARE mensaje VARCHAR(100);

-- Declaración de constantes

CONSTANT meses\_en\_anio INT DEFAULT 12;

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Calcular el salario anual promedio

IF total\_empleados > 0 THEN

salario\_promedio := total\_salarios / total\_empleados \* meses\_en\_anio;

SET mensaje = 'El salario anual promedio es: ' || salario\_promedio;

ELSE

SET mensaje = 'No hay empleados en la tabla.';

END IF;

-- Mostrar el mensaje

SELECT mensaje;

END;

En este ejemplo, hemos creado un procedimiento almacenado llamado **calcular\_salario\_anual\_promedio**.

Dentro de este procedimiento, hemos declarado variables para almacenar valores que cambiarán durante la ejecución (**total\_salarios**, **total\_empleados**, **salario\_promedio**, **mensaje**) y una constante (**meses\_en\_anio**) que no cambiará. Luego, realizamos cálculos y asignaciones de valores a las variables según sea necesario dentro del bloque de código del procedimiento.

Finalmente, mostramos un mensaje que depende del resultado de los cálculos utilizando una variable.

* **2.4 Variables y Constantes: Definición y uso de variables y constantes en PL-SQL**

**Variables y Constantes en PL/SQL**

En PL/SQL (Procedural Language/Structured Query Language), las variables y constantes se utilizan para almacenar y manipular valores dentro de bloques de código. Aquí está una definición y ejemplo de cómo se definen y usan en PL/SQL:

Variables:

* **Definición:** Las variables son objetos que pueden almacenar valores que pueden cambiar durante la ejecución de un bloque de código.
* **Uso:** Se declaran con la palabra clave **DECLARE** y pueden asignarse valores usando la operación **:=**.

Constantes:

* **Definición:** Las constantes son objetos que almacenan valores que no cambian durante la ejecución de un bloque de código.
* **Uso:** Se definen utilizando la palabra clave **CONSTANT** o **CONST** y su valor se establece en la declaración.

EJEMPLO:

DECLARE

-- Declaración de variables

total\_salarios NUMBER(8, 2);

total\_empleados INT;

salario\_promedio NUMBER(8, 2);

mensaje VARCHAR2(100);

-- Declaración de constantes

meses\_en\_anio CONSTANT INT := 12;

BEGIN

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Calcular el salario anual promedio

IF total\_empleados > 0 THEN

salario\_promedio := total\_salarios / total\_empleados \* meses\_en\_anio;

mensaje := 'El salario anual promedio es: ' || salario\_promedio;

ELSE

mensaje := 'No hay empleados en la tabla.';

END IF;

-- Mostrar el mensaje

DBMS\_OUTPUT.PUT\_LINE(mensaje);

END;

/

En este ejemplo, hemos creado un bloque anónimo en PL/SQL que comienza con **DECLARE** y termina con **END;**. Dentro de este bloque, hemos declarado variables para almacenar valores que pueden cambiar durante la ejecución (**total\_salarios**, **total\_empleados**, **salario\_promedio**, **mensaje**) y una constante (**meses\_en\_anio**) que no cambiará.

Luego, realizamos cálculos y asignaciones de valores a las variables según sea necesario dentro del bloque de código.

Finalmente, mostramos un mensaje que depende del resultado de los cálculos utilizando la función **DBMS\_OUTPUT.PUT\_LINE**.

* **2.5. Operadores Y expresiones: Cómo se utilizan los operadores y expresiones en PL/SQL y SQL/PSM:**

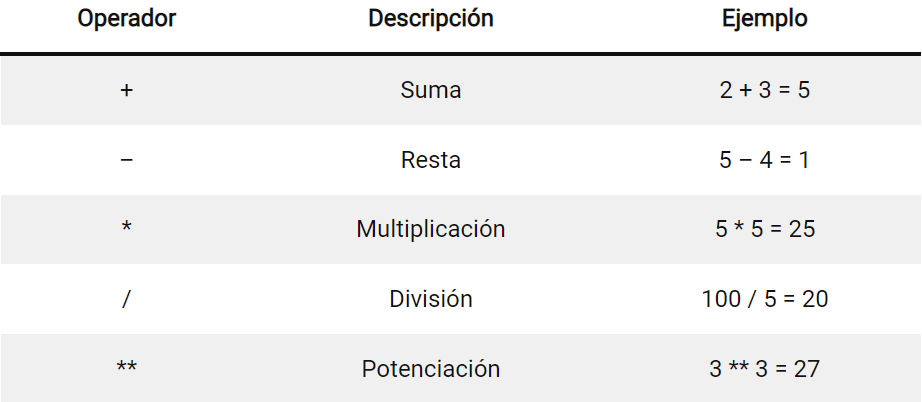
**PL/SQL**

Los operadores en PL/SQL son símbolos que le dicen al compilador qué hacer con los datos en las variables y las expresiones.

Los operadores nos facilitan hacer cálculos, comparaciones, concatenaciones y otras operaciones lógicas con los datos.

Estos son los tipos de operadores en PL/SQL:

* **OPERADORES ARITMÉTICOS**: Los operadores aritméticos en PL/SQL son símbolos que se emplean para hacer operaciones matemáticas básicas con los datos numéricos.



**EJEMPLO**

BEGIN

DBMS\_OUTPUT.PUT\_LINE(2 + 3); -- operador de suma

DBMS\_OUTPUT.PUT\_LINE(5 - 4); -- operador de resta

DBMS\_OUTPUT.PUT\_LINE (5 \* 5); -- operador de multiplicación

DBMS\_OUTPUT.PUT\_LINE (100 / 4); -- operador de división

DBMS\_OUTPUT.PUT\_LINE(3 \*\* 3); -- operador de potenciación

END;

El orden de precedencia de los operadores aritméticos en PL/SQL es el orden en que se calculan las expresiones que contienen estos operadores.

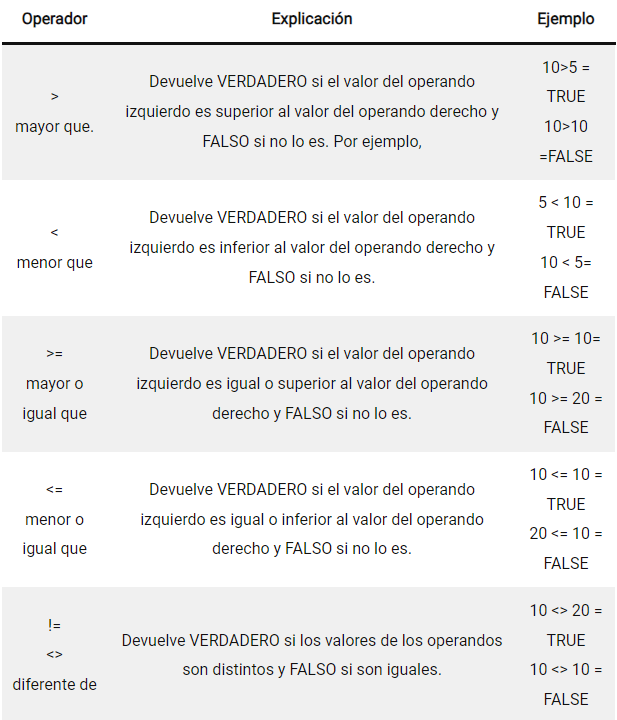
Los operadores con más precedencia se calculan antes que los operadores con menos precedencia.

El orden de precedencia de los operadores aritméticos en PL/SQL es el siguiente:

* \*\* (exponenciación)
* \*, / (multiplicación, división)
* +, – (suma, resta)
* || (concatenación)

Por ejemplo, la expresión 3 + 4 \* 6 \*\* 2 se calcula así:

* Primero se calcula la exponenciación: 6 \*\* 2 = 36
* Luego se calcula la multiplicación: 4 \* 36 = 144
* Luego se calcula la suma: 3 + 144 = 147
* **OPERADORES DE COMPARACIÓN**: Los operadores de comparación son usados para evaluar si dos valores numéricos o de cadena de caracteres son iguales o diferentes y devolver un valor booleano (VERDADERO o FALSO) según el resultado de la evaluación.



**EJEMPLO**

-- Declaramos dos variables numéricas y les asignamos valores

DECLARE

v\_num1 NUMBER := 10;

v\_num2 NUMBER := 20;

BEGIN

-- Usamos el operador > para comparar las variables y mostrar un mensaje según el resultado

IF v\_num1 > v\_num2 THEN

DBMS\_OUTPUT.PUT\_LINE('v\_num1 es mayor que v\_num2');

ELSE

DBMS\_OUTPUT.PUT\_LINE('v\_num1 no es mayor que v\_num2');

END IF;

-- Usamos el operador < para comparar las variables y mostrar un mensaje según el resultado

IF v\_num1 < v\_num2 THEN

DBMS\_OUTPUT.PUT\_LINE('v\_num1 es menor que v\_num2');

ELSE

DBMS\_OUTPUT.PUT\_LINE('v\_num1 no es menor que v\_num2');

END IF;

-- Usamos el operador >= para comparar las variables y mostrar un mensaje según el resultado

IF v\_num1 >= v\_num2 THEN

DBMS\_OUTPUT.PUT\_LINE('v\_num1 es mayor o igual que v\_num2');

ELSE

DBMS\_OUTPUT.PUT\_LINE('v\_num1 no es mayor o igual que v\_num2');

END IF;

-- Usamos el operador <= para comparar las variables y mostrar un mensaje según el resultado

IF v\_num1 <= v\_num2 THEN

DBMS\_OUTPUT.PUT\_LINE('v\_num1 es menor o igual que v\_num2');

ELSE

DBMS\_OUTPUT.PUT\_LINE('v\_num1 no es menor o igual que v\_num2');

END IF;

-- Usamos el operador <> para comparar las variables y mostrar un mensaje según el resultado

IF v\_num1 <> v\_num2 THEN

DBMS\_OUTPUT.PUT\_LINE('v\_num1 es diferente de v\_num2');

ELSE

DBMS\_OUTPUT.PUT\_LINE('v\_num1 no es diferente de v\_num2');

END IF;

-- Usamos el operador != para comparar las variables y mostrar un mensaje según el resultado

IF v\_num1 != v\_num2 THEN

DBMS\_OUTPUT.PUT\_LINE('v\_num1 es diferente de v\_num2');

ELSE

DBMS\_OUTPUT.PUT\_LINE('v\_num1 no es diferente de v\_num2');

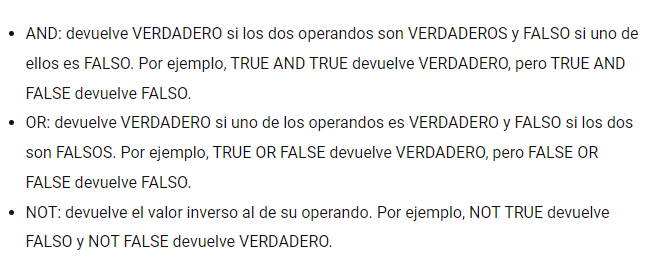
END IF;

END;

/

* **OPERADORES LÓGICOS**: Los operadores lógicos en PL/SQL son símbolos que se usan para calcular el valor de expresiones booleanas o lógicas.

Estas expresiones son aquellas que devuelven un valor VERDADERO o FALSO.



Los operadores lógicos se usan para unir o invertir condiciones en las expresiones booleanas o lógicas.

Estas expresiones se pueden usar para controlar el flujo del programa o para filtrar los datos en las consultas SQL.

**EJEMPLO**

-- Este programa muestra el uso de los operadores lógicos AND, OR y NOT en PL/SQL

DECLARE

x NUMBER := 10; -- Variable x con valor 10

y NUMBER := 20; -- Variable y con valor 20

BEGIN

-- Uso del operador AND: devuelve TRUE si ambas condiciones son verdaderas

IF x > 0 AND y > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('x e y son positivos');

END IF;

-- Uso del operador OR: devuelve TRUE si al menos una condición es verdadera

IF x > y OR x = y THEN

DBMS\_OUTPUT.PUT\_LINE('x es mayor o igual que y');

ELSE

DBMS\_OUTPUT.PUT\_LINE('x es menor que y');

END IF;

-- Uso del operador NOT: devuelve TRUE si la condición es falsa

IF NOT (x < 0) THEN

DBMS\_OUTPUT.PUT\_LINE('x no es negativo');

END IF;

END;

**EXPRESIONES**

Las expresiones pueden combinar operadores aritméticos para realizar cálculos.

**EJEMPLO**

DECLARE

tipo\_cliente CHAR := 'VIP';

descuento NUMBER;

BEGIN

CASE tipo\_cliente

WHEN 'VIP' THEN descuento := 0.2;

WHEN 'Regular' THEN descuento := 0.1;

ELSE descuento := 0;

END CASE;

DBMS\_OUTPUT.PUT\_LINE('Descuento: ' || descuento);

END;

* **SQL/PSM**

En SQL/PSM los operadores y expresiones se utilizan de igual manera a como se usan en PL/SQL u otros lenguajes de programación, por lo tanto, es lo mismo que lo de arriba.

* **¿Comparación operadores y expresiones PL/SQL y SQL/PSM con ejemplos?**

|  |  |  |
| --- | --- | --- |
|  | **PL/SQL** | **SQL/PSM** |
| **Operadores Aritméticos** | Se usan de la misma manera en ambas. | |
|  |  |  |
| **Operadores de Comparación** | Se usan de la misma manera en ambas. | |
|  |  |  |
| **Operadores Lógicos** | Se usan de la misma manera en ambas. | |
|  |  |  |
| **Expresiones** | Las expresiones aritméticas son similares en ambos lenguajes, pero la asignación de valores a las variables se realiza de manera diferente.  ---------------------------------  salario\_total := salario\_base + bono; | Las expresiones aritméticas son similares en ambos lenguajes, pero la asignación de valores a las variables se realiza de manera diferente.  ---------------------------------  SET salario\_total = salario\_base + bono; |
|  |  |  |

* **2.6 Etiquetas: Uso y propósito de las etiquetas en el código en SQL-PSM**

**Uso y Propósito de las Etiquetas en SQL-PSM:**

**Propósito:**

* Las etiquetas proporcionan un nombre único para identificar un bloque de código en SQL-PSM.
* Se usan junto con instrucciones de control de flujo para indicar el inicio y el fin de un bloque de código.
* Permiten usar instrucciones GOTO para saltar a un bloque específico en un procedimiento almacenado.

**Uso:**

* Se definen utilizando cualquier identificador válido seguido de dos puntos (**:**).
* Se colocan antes de un bloque de código que se desea etiquetar.
* Pueden ser utilizadas en combinación con instrucciones de control de flujo como **GOTO**, **LEAVE**, etc., para especificar el destino del flujo de ejecución.

EJEMPLO:

CREATE PROCEDURE ejemplo\_etiquetas()

BEGIN

-- Etiqueta 1

etiqueta\_inicio:

DECLARE total\_salarios DECIMAL(10, 2);

DECLARE total\_empleados INT;

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Verificar si hay empleados

IF total\_empleados = 0 THEN

-- Ir a la etiqueta final si no hay empleados

GOTO etiqueta\_final;

END IF;

-- Realizar cálculos si hay empleados

DECLARE salario\_promedio DECIMAL(10, 2);

salario\_promedio := total\_salarios / total\_empleados;

DBMS\_OUTPUT.PUT\_LINE('El salario promedio es: ' || salario\_promedio);

-- Etiqueta 2

etiqueta\_final:

DBMS\_OUTPUT.PUT\_LINE('El proceso ha finalizado.');

END;

/

En este ejemplo, se ha creado un procedimiento almacenado llamado **ejemplo\_etiquetas**. Dentro del procedimiento, se han utilizado dos etiquetas: **etiqueta\_inicio** y **etiqueta\_final**. La etiqueta **etiqueta\_inicio** se coloca al inicio del bloque de código, mientras que **etiqueta\_final** se coloca antes del bloque final del procedimiento. Se utiliza la instrucción **GOTO** para saltar al bloque **etiqueta\_final** si no hay empleados en la tabla.

Esto ilustra cómo se pueden usar las etiquetas para controlar el flujo de ejecución en un procedimiento almacenado en SQL-PSM.

* **2.6 Etiquetas: Uso y propósito de las etiquetas en el código en PL-SQL**

**Uso y Propósito de las Etiquetas en PL/SQL:**

**Propósito:**

* Las etiquetas proporcionan un nombre único para identificar un bloque de código en PL/SQL.
* Se usan junto con instrucciones de control de flujo para indicar el inicio y el fin de un bloque de código.
* Permiten el uso de instrucciones GOTO para saltar a un bloque en un procedimiento o función.

**Uso:**

* Se definen utilizando cualquier identificador válido seguido de dos puntos (**:**).
* Se colocan antes de un bloque de código que se desea etiquetar.
* Pueden ser utilizadas en combinación con instrucciones de control de flujo como **GOTO**, **EXIT**, etc., para especificar el destino del flujo de ejecución.

EJEMPLO:

DECLARE

total\_salarios NUMBER(8, 2);

total\_empleados INT;

BEGIN

-- Etiqueta 1

etiqueta\_inicio:

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Verificar si hay empleados

IF total\_empleados = 0 THEN

-- Ir a la etiqueta final si no hay empleados

GOTO etiqueta\_final;

END IF;

-- Realizar cálculos si hay empleados

DECLARE salario\_promedio NUMBER(8, 2);

salario\_promedio := total\_salarios / total\_empleados;

DBMS\_OUTPUT.PUT\_LINE('El salario promedio es: ' || salario\_promedio);

-- Etiqueta 2

etiqueta\_final:

DBMS\_OUTPUT.PUT\_LINE('El proceso ha finalizado.');

END;

/

En este ejemplo, se ha creado un bloque anónimo en PL/SQL. Dentro del bloque, se han utilizado dos etiquetas: **etiqueta\_inicio** y **etiqueta\_final**. La etiqueta **etiqueta\_inicio** se coloca al inicio del bloque de código, mientras que **etiqueta\_final** se coloca antes del bloque final del bloque anónimo.

Se utiliza la instrucción **GOTO** para saltar al bloque **etiqueta\_final** si no hay empleados en la tabla. Esto ilustra cómo se pueden usar las etiquetas para controlar el flujo de ejecución en un bloque anónimo en PL/SQL.

* **2.7 Ámbito de las Variables: Explicación del alcance de las variables dentro de un programa SQL-PSM**

**Alcance de las Variables en SQL-PSM:**

1. **Alcance Local:**
   * Las variables declaradas dentro de un bloque de código, como un procedimiento almacenado, función o un bloque anónimo, tienen un alcance local.
   * Estas variables solo son accesibles dentro del bloque en el que se han declarado y no pueden ser referenciadas desde fuera de ese bloque.
   * Una vez que el bloque de código finaliza su ejecución, las variables locales se desreferencian y su valor se libera de la memoria.
2. **Alcance de Bloque Anidado:**
   * Si un bloque de código está anidado dentro de otro bloque, las variables declaradas en el bloque externo no son visibles dentro del bloque interno.
   * Sin embargo, las variables declaradas dentro del bloque interno tienen visibilidad dentro de su propio bloque y no interfieren con las variables del bloque externo que pueden tener el mismo nombre.
3. **Alcance Global:**
   * SQL-PSM no admite variables globales como otros lenguajes de programación, como PL/SQL.
   * No se pueden declarar variables fuera de un bloque de código que tengan un alcance global y estén disponibles en todo el programa.
   * Para compartir valores entre diferentes partes de un programa en SQL-PSM, generalmente se utilizan parámetros de entrada y salida en procedimientos almacenados y funciones.

EJEMPLO:

CREATE PROCEDURE calcular\_salario\_promedio\_detallado()

BEGIN

-- Declaración de variables locales

DECLARE total\_salarios DECIMAL(10, 2);

DECLARE total\_empleados INT;

DECLARE salario\_promedio DECIMAL(10, 2);

-- Obtener el total de salarios y el número de empleados

SELECT SUM(salario), COUNT(\*) INTO total\_salarios, total\_empleados

FROM empleados;

-- Calcular el salario promedio

IF total\_empleados > 0 THEN

salario\_promedio := total\_salarios / total\_empleados;

ELSE

-- Si no hay empleados, mostrar un mensaje y finalizar el procedimiento

DBMS\_OUTPUT.PUT\_LINE('No hay empleados en la tabla.');

RETURN;

END IF;

-- Mostrar el salario promedio

DBMS\_OUTPUT.PUT\_LINE('El salario promedio es: ' || salario\_promedio);

-- Obtener detalles de los empleados cuyo salario es mayor que el promedio

FOR empleado IN (SELECT id\_empleado, nombre, salario FROM empleados WHERE salario > salario\_promedio) LOOP

DBMS\_OUTPUT.PUT\_LINE('ID Empleado: ' || empleado.id\_empleado || ', Nombre: ' || empleado.nombre || ', Salario: ' || empleado.salario);

END LOOP;

END;

/

En este ejemplo, hemos creado un procedimiento almacenado llamado **calcular\_salario\_promedio\_detallado**. Dentro de este procedimiento, hemos declarado varias variables locales para almacenar el total de salarios, el número de empleados y el salario promedio.

Luego, calculamos el salario promedio y lo mostramos. Después, utilizamos un bucle **FOR** para recorrer los registros de empleados cuyo salario es mayor que el salario promedio y mostramos detalles sobre cada uno de ellos.

* **2.7 Ámbito de las Variables: Explicación del alcance de las variables dentro de un programa PL-SQL**

**Alcance de las Variables en PL/SQL:**

1. **Alcance Local:**
   * Las variables declaradas dentro de un bloque de código, como un procedimiento almacenado, función o un bloque anónimo, tienen un alcance local.
   * Estas variables solo son accesibles dentro del bloque en el que se han declarado y no pueden ser referenciadas desde fuera de ese bloque.
   * Una vez que el bloque de código finaliza su ejecución, las variables locales se desreferencian y su valor se libera de la memoria.
2. **Alcance de Bloque Anidado:**
   * Si un bloque de código está anidado dentro de otro bloque, las variables declaradas en el bloque externo no son visibles dentro del bloque interno.
   * Sin embargo, las variables declaradas dentro del bloque interno tienen visibilidad dentro de su propio bloque y no interfieren con las variables del bloque externo que pueden tener el mismo nombre.
3. **Alcance de Subprogramas:**
   * En PL/SQL, los subprogramas como procedimientos y funciones tienen sus propios ámbitos.
   * Las variables declaradas dentro de un subprograma son locales a ese subprograma y no son accesibles desde fuera de él.
   * Sin embargo, los subprogramas pueden acceder a variables globales y a parámetros pasados a ellos.
4. **Alcance Global:**
   * PL/SQL admite variables globales que se pueden declarar fuera de cualquier bloque de código y están disponibles en todo el programa.
   * Las variables globales pueden ser accedidas desde cualquier parte del programa, incluyendo procedimientos, funciones y bloques anónimos.
   * Se debe tener precaución al usar variables globales para evitar problemas de mantenimiento y riesgos de modificación no deseada.

EJEMPLO:

DECLARE

-- Variable global

total\_empleados INT := 0;

BEGIN

-- Alcance local dentro del bloque anónimo

DECLARE

-- Variable local

salario\_promedio NUMBER := 5000;

BEGIN

-- Mostrar el salario promedio

DBMS\_OUTPUT.PUT\_LINE('El salario promedio es: ' || salario\_promedio);

-- Modificar la variable global dentro del bloque anónimo

total\_empleados := 100;

END;

-- Mostrar el total de empleados modificado

DBMS\_OUTPUT.PUT\_LINE('El total de empleados es: ' || total\_empleados);

END;

/

Hemos declarado una variable global total\_empleados fuera del bloque anónimo principal.

Dentro del bloque anónimo, declaramos una variable local salario\_promedio.

La variable local salario\_promedio solo es visible dentro de su propio bloque.

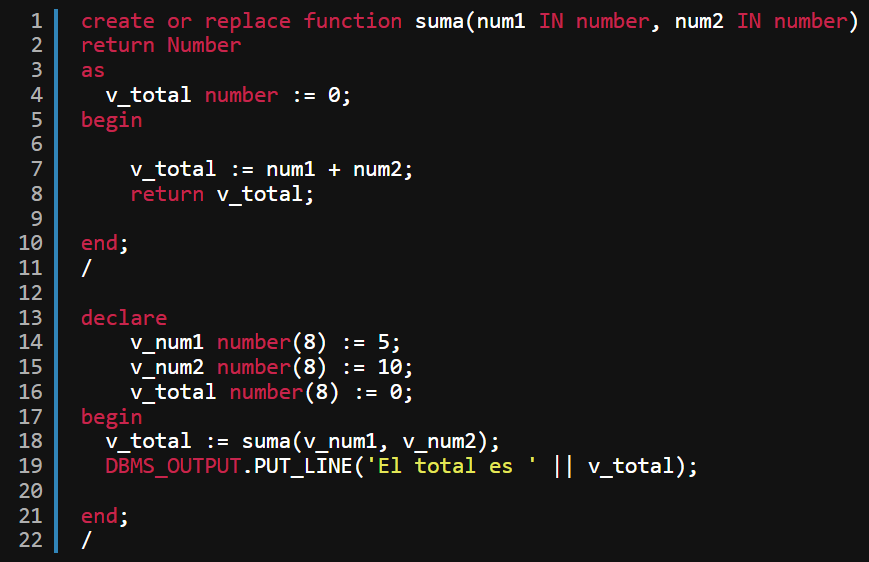
Modificamos la variable global total\_empleados dentro del bloque anónimo.

Fuera del bloque anónimo, podemos acceder a la variable global total\_empleados y mostrar su valor modificado.

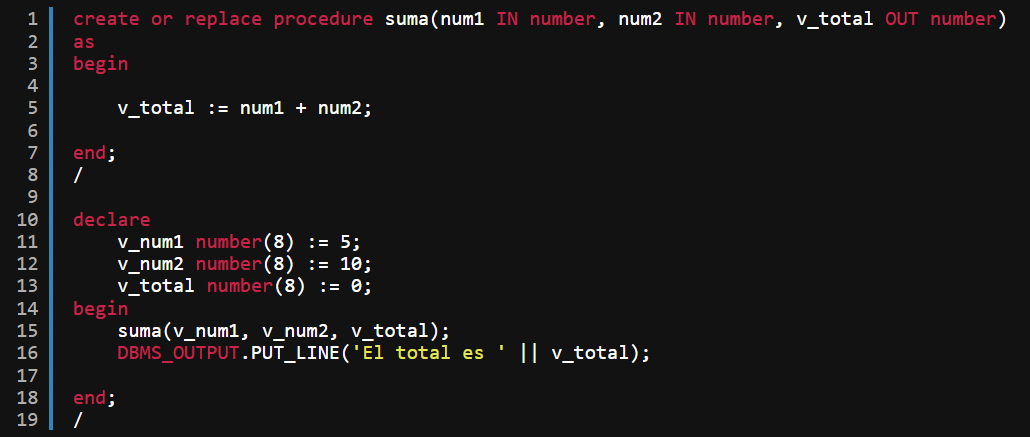
* **2.8 Entrada y Salida. Interacción con el Usuario: Mecanismos de entrada/salida en PL/SQL Y SQL/PSM:**
* **PL/SQL**

En PL/SQL los parámetros pueden ser de entrada o salida, vamos a explicar cómo se utilizan ambos:

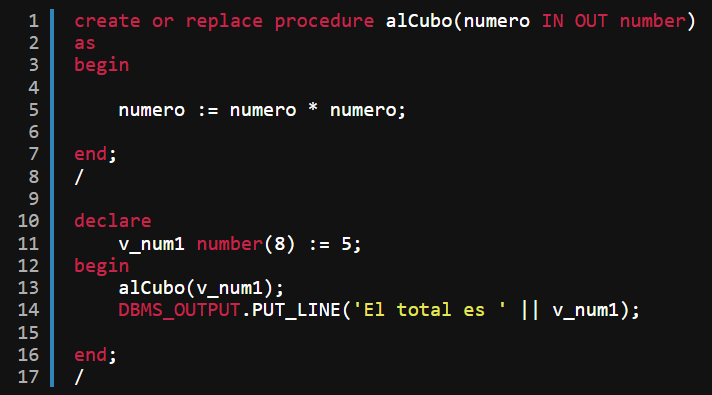
* Entrada (in) : Los parámetros de entrada son aquellos a los que se les debe asignar un valor al llamar a la función o procedimiento. Si no se especifica, por defecto, un parámetro será de entrada.



* Salida (out) : Los parámetros de salida son aquellos que, al finalizar la función o procedimiento, mantienen su valor fuera de este. Pueden utilizarse para almacenar múltiples valores simultáneamente o para devolver valores desde un procedimiento, similar a una función.



* Entrada/salida (in out) : Es la combinación de ambos tipos de parámetros, permitiendo tanto la asignación de valores al llamar a la función o procedimiento como la retención de valores al finalizar.



* **SQL/PSM**
* Entrada (in) : En SQL/PSM, los parámetros de entrada son aquellos que se utilizan para pasar valores a un procedimiento almacenado.

Estos valores pueden ser utilizados dentro del procedimiento, pero no pueden ser modificados. Por lo general, se utilizan para proporcionar datos de entrada al procedimiento.

CREATE PROCEDURE EjemploIN(IN nombre VARCHAR(50))

BEGIN

SELECT \* FROM tabla WHERE columna = nombre;

END;

* Salida (out) : Los parámetros de salida en SQL/PSM son utilizados para devolver valores desde un procedimiento almacenado.

Estos valores pueden modificarse dentro del procedimiento y se pueden utilizar fuera de él cuando el procedimiento finalice.

CREATE PROCEDURE EjemploOUT(OUT total INT)

BEGIN

SELECT COUNT(\*) INTO total FROM tabla;

END;

* Entrada/Salida (in/out) : Los parámetros de entrada/salida en SQL/PSM son aquellos que se utilizan tanto para pasar valores como para devolver valores desde un procedimiento almacenado.

Estos valores pueden modificarse dentro del procedimiento y se pueden utilizar fuera de él cuando el procedimiento finalice.

CREATE PROCEDURE EjemploINOUT(INOUT num INT)

BEGIN

SET num = num + 1;

END;

* **¿Comparación mecanismos de entrada/salida en PL/SQL Y SQL/PSM con ejemplos?**

|  |  |  |
| --- | --- | --- |
| **MECANISMO** | **PL/SQL** | **SQL/PSM** |
|  |  |  |
| **ENTRADA** | Utiliza principalmente variables ligadas a parámetros de procedimientos y funciones.  DECLARE  v\_nombre VARCHAR2(100);  BEGIN  v\_nombre := 'Juan';  END;  / | Utiliza principalmente parámetros en la declaración de procedimientos.  CREATE PROCEDURE nombre\_procedimiento (  IN p\_nombre VARCHAR(100),  IN p\_edad INT,  IN p\_genero CHAR(1),  OUT p\_mensaje VARCHAR(200)  )  BEGIN  SET p\_mensaje := 'Hola, ' || p\_nombre || ', tienes ' || p\_edad || ' años.';  END;  / |
|  |  |  |
| **SALIDA** | Principalmente mediante la asignación de valores a variables o envío de valores de retorno.  DECLARE  v\_mensaje VARCHAR2(200);  BEGIN  v\_mensaje := '¡Hola, mundo!';  DBMS\_OUTPUT.PUT\_LINE(v\_mensaje);  END;  / | Principalmente mediante la asignación de valores a variables o envío de valores de retorno.  CREATE PROCEDURE nombre\_procedimiento (  IN p\_numero INT,  OUT p\_resultado INT  )  BEGIN  SET p\_resultado := p\_numero \* 2;  END;  / |
|  |  |  |
| **ENTRADA/SALIDA** | Utiliza tanto parámetros de entrada como variables para salida, además de cursores para resultados de consultas.  CREATE PROCEDURE nombre\_procedimiento (  p\_id\_cliente IN INT,  p\_nombre\_cliente OUT VARCHAR2  )  AS  BEGIN  SELECT nombre INTO p\_nombre\_cliente  FROM clientes  WHERE id = p\_id\_cliente;  END;  / | Similar a PL/SQL, utilizando parámetros de entrada y salida.  CREATE PROCEDURE nombre\_procedimiento (  IN p\_numero1 INT,  IN p\_numero2 INT,  OUT p\_resultado INT  )  BEGIN  SET p\_resultado := p\_numero1 + p\_numero2;  END;  / |

* **2.9 Estructuras de Control: SQL-PSM Detalle sobre las estructuras de control como IF->THEN,CASE->WHEN->THEN,etc.**

se pueden utilizar estructuras de control como **IF**, **CASE**, **WHILE**, y **LOOP** para controlar el flujo de ejecución en bloques de código. Aquí tienemos más detalles sobre estas estructuras de control:

**IF -> THEN:**

La estructura **IF -> THEN** se utiliza para ejecutar un bloque de código si se cumple una condición especificada. Si la condición es verdadera, se ejecuta el bloque de código dentro del **THEN**.

IF condicion THEN

-- Bloque de código a ejecutar si la condición es verdadera

END IF;

**CASE -> WHEN -> THEN:**

La estructura **CASE -> WHEN -> THEN** se utiliza para evaluar múltiples condiciones y ejecutar bloques de código dependiendo de la condición que sea verdadera.

CASE

WHEN condicion1 THEN

-- Bloque de código a ejecutar si la condicion1 es verdadera

WHEN condicion2 THEN

-- Bloque de código a ejecutar si la condicion2 es verdadera

...

ELSE

-- Bloque de código a ejecutar si ninguna de las condiciones anteriores es verdadera

END CASE;

**WHILE:**

La estructura **WHILE** se utiliza para ejecutar repetidamente un bloque de código mientras una condición específica sea verdadera.

WHILE condicion DO

-- Bloque de código a ejecutar mientras la condición sea verdadera

END WHILE;

**LOOP:**

La estructura **LOOP** se utiliza para ejecutar un bloque de código repetidamente indefinidamente o hasta que se alcance una condición de salida.

LOOP

-- Bloque de código a ejecutar en cada iteración

IF condicion THEN

LEAVE; -- Sentencia para salir del bucle si se cumple la condición

END IF;

END LOOP;

* **2.9 Estructuras de Control: PL-SQL Detalle sobre las estructuras de control como IF->THEN,CASE->WHEN->THEN,etc**

Es lo mismo de arriba, quiere decir que se utilizan las mismas estructuras de control que en SQL-PSM que en PL-SQL

**PARTE II PL/SQL Y SQL/PSM AVANCES**

* **1.1 Tipos de Excepciones: Clasificación y ejemplos (PL-SQL)**

**Clasificación de Excepciones en PL/SQL:**

**Excepciones Predefinidas:** Estas son excepciones que están predefinidas en PL/SQL y se activan en situaciones específicas, como errores de división por cero, desbordamiento numérico, violaciones de restricciones de integridad, etc.

**Excepciones de Usuario:** Estas son excepciones definidas por el usuario para manejar situaciones específicas dentro de un programa, como errores de validación de datos, errores de lógica de negocio, etc.

**Ejemplos de Excepciones en PL/SQL:**

Excepciones Predefinidas:

**ZERO\_DIVIDE:** Se activa cuando se intenta dividir por cero.

DECLARE

resultado NUMBER;

BEGIN

resultado := 10 / 0; -- Esto activará la excepción ZERO\_DIVIDE

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Error: División por cero');

END;

**TOO\_MANY\_ROWS:** Se activa cuando una consulta devuelve más de una fila en un contexto donde se espera una sola fila.

DECLARE

v\_variable NUMBER;

BEGIN

SELECT columna INTO v\_variable FROM tabla WHERE condicion;

EXCEPTION

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Demasiadas filas devueltas por la consulta');

END;

Excepciones de Usuario:

* **ERROR\_VALIDACION\_DATOS:** Se activa cuando falla una validación de datos personalizada.

DECLARE

edad NUMBER := 15;

BEGIN

IF edad < 18 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Error: Edad debe ser mayor o igual a 18');

END IF;

END;

**ERROR\_LOGICA\_NEGOCIO:** Se activa cuando falla una condición de lógica de negocio.

DECLARE

saldo\_actual NUMBER := 1000;

monto\_retiro NUMBER := 1500;

BEGIN

IF monto\_retiro > saldo\_actual THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Error: Saldo insuficiente para realizar el retiro');

END IF;

END;

Estos son solo algunos ejemplos de excepciones en PL/SQL. Las excepciones son una parte importante de la gestión de errores en cualquier programa y permiten manejar situaciones inesperadas de manera adecuada.

* **1.2Propagación de Excepciones: Cómo se propagan las excepciones en SQL-PSM**

la propagación de excepciones se refiere a cómo una excepción que ocurre en un bloque de código se propaga hacia arriba a través de la jerarquía de bloques de código anidados hasta que se maneja o se propaga a la unidad de proceso de llamada.

Cuando se produce una excepción dentro de un bloque de código en SQL-PSM, el sistema busca un manejador de excepciones adecuado para la excepción en ese bloque. Si se encuentra un manejador de excepciones, el código dentro de ese manejador se ejecuta y la excepción se considera manejada.

Si no hay un manejador de excepciones en el bloque actual, la excepción se propaga hacia arriba por la jerarquía de bloques de código anidados, comenzando desde el bloque actual hasta el bloque de nivel superior donde está la unidad de proceso de llamada (un procedimiento almacenado o un bloque anónimo).

Si la excepción no se maneja en ningún lugar a lo largo de esta ruta de propagación, se considera no manejada y puede resultar en la terminación anormal del programa y la generación de un mensaje de error.

Aquí tienes un ejemplo que ilustra cómo se propaga una excepción en SQL-PSM:

BEGIN

-- Bloque de código principal

DECLARE

v\_numero INT;

BEGIN

-- Bloque de código anidado

SELECT columna INTO v\_numero FROM tabla WHERE condicion;

EXCEPTION

WHEN ERROR\_DATATYPE THEN

-- Manejar la excepción

DBMS\_OUTPUT.PUT\_LINE('Error de tipo de datos');

END;

END;

En este ejemplo:

* Se intenta realizar una operación que podría generar la excepción ERROR\_DATATYPE en el bloque anidado.
* Si se produce la excepción, se maneja en el bloque anidado.
* Si la excepción no se maneja en el bloque anidado, se propaga hacia arriba hasta el bloque principal.
* Si aún no se maneja la excepción en el bloque principal, podría propagarse a la unidad de proceso de llamada o, si no hay ningún manejador de excepciones adecuado, se considera una excepción no manejada.
* **1.2 Propagación de Excepciones: Cómo se propagan las excepciones en PL-SQL**

la propagación de excepciones sigue un proceso similar al de SQL-PSM. Cuando se produce una excepción dentro de un bloque de código en PL/SQL, el sistema busca un manejador de excepciones adecuado para esa excepción en el bloque actual.

Si se encuentra un manejador de excepciones, el código dentro de ese manejador se ejecuta y la excepción se considera manejada.

Si no se encuentra un manejador de excepciones en el bloque actual, la excepción se propaga hacia arriba por la jerarquía de bloques de código anidados, comenzando desde el bloque actual hasta el bloque de nivel superior donde está la unidad de proceso de llamada (un procedimiento almacenado, una función o un bloque anónimo).

Si la excepción no se maneja en ningún lugar a lo largo de esta ruta de propagación, se considera no manejada y puede resultar en la terminación anormal del programa y la generación de un mensaje de error.

Aquí tienes un ejemplo que ilustra cómo se propaga una excepción en PL/SQL:

BEGIN

-- Bloque de código principal

DECLARE

v\_numero NUMBER;

BEGIN

-- Bloque de código anidado

SELECT columna INTO v\_numero FROM tabla WHERE condicion;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- Manejar la excepción

DBMS\_OUTPUT.PUT\_LINE('No se encontraron datos');

END;

END;

En este ejemplo:

* Se intenta realizar una operación que podría generar la excepción NO\_DATA\_FOUND en el bloque anidado.
* Si se produce la excepción, se maneja en el bloque anidado.
* Si la excepción no se maneja en el bloque anidado, se propaga hacia arriba hasta el bloque principal.
* Si aún no se maneja la excepción en el bloque principal, podría propagarse a la unidad de proceso de llamada o, si no hay ningún manejador de excepciones adecuado, se considera una excepción no manejada.
* 1.3 Ámbito de las Excepciones: Dónde se capturan y se manejan las excepciones en PL/SQL & SQL/PSM:
* **PL/SQL**

En PL/SQL, las excepciones son eventos inusuales o errores que pueden ocurrir durante la ejecución de un bloque de código.

El manejo de excepciones permite controlar estos eventos inusuales, evitando que interrumpan el flujo normal del programa y permitiendo tomar acciones específicas para gestionarlos.

se capturan y manejan dentro de los bloques BEGIN EXCEPTION END.

**EJEMPLO**

DECLARE

mi\_excepcion EXCEPTION;

v\_numero number := 0;

BEGIN

v\_numero := 10 / 0;

EXCEPTION

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Error: División por cero');

WHEN mi\_excepcion THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Mi excepción personalizada');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Ocurrió una excepción no prevista');

END;

/

* **2.1 Procedimientos: Creación y uso de procedimientos en PL-SQL**

**Creación de Procedimientos en PL/SQL:**

Para crear un procedimiento en PL/SQL, se utiliza la declaración **CREATE PROCEDURE**. Aquí tienes un ejemplo de cómo crear un procedimiento simple que imprime un mensaje:

CREATE OR REPLACE PROCEDURE nombre\_procedimiento AS

BEGIN

-- Bloque de código del procedimiento

DBMS\_OUTPUT.PUT\_LINE('Este es un mensaje desde el procedimiento.');

END nombre\_procedimiento;

/

**Uso de Procedimientos en PL/SQL:**

Después de crear un procedimiento, puedes utilizarlo llamándolo desde otro bloque de código. Aquí tienes un ejemplo de cómo llamar al procedimiento **nombre\_procedimiento** creado anteriormente:

BEGIN

-- Llamada al procedimiento

nombre\_procedimiento;

END;

/

**Parámetros en Procedimientos:**

Los procedimientos pueden aceptar parámetros para permitir la reutilización y la personalización de la lógica del procedimiento. Aquí tienes un ejemplo de cómo crear un procedimiento con parámetros:

CREATE OR REPLACE PROCEDURE nombre\_procedimiento(parametro1 IN NUMBER, parametro2 IN VARCHAR2) AS

BEGIN

-- Bloque de código del procedimiento que utiliza los parámetros

DBMS\_OUTPUT.PUT\_LINE('Parámetro 1: ' || parametro1);

DBMS\_OUTPUT.PUT\_LINE('Parámetro 2: ' || parametro2);

END nombre\_procedimiento;

/

**Llamada a Procedimientos con Parámetros:**

Cuando llamas a un procedimiento con parámetros, debes proporcionar los valores para esos parámetros. Aquí tienes un ejemplo de cómo llamar al procedimiento **nombre\_procedimiento** con parámetros:

BEGIN

-- Llamada al procedimiento con parámetros

nombre\_procedimiento(123, 'ejemplo');

END;

/

**Procedimientos con Resultados:**

Los procedimientos también pueden devolver resultados, como conjuntos de filas o valores escalares, utilizando parámetros OUT o mediante el uso de cursores. Esto permite que los procedimientos realicen operaciones más complejas y devuelvan resultados que pueden ser utilizados por otros bloques de código.

* **2.2 Funciones: Diferencias con los procedimientos y ejemplos de uso en SQL-PSM**

**Diferencias entre Funciones y Procedimientos en SQL-PSM:**

1. **Estructura:**
   * Una función tiene una estructura similar a una sentencia SQL, donde se espera que devuelva un valor.
   * Un procedimiento no devuelve un valor específico y se usa para realizar acciones o tareas.
2. **Uso de Resultados:**
   * Una función debe devolver un valor, ya sea escalar o de conjunto de filas.
   * Un procedimiento puede realizar operaciones sin necesidad de devolver un valor específico.
3. **Invocación:**
   * Las funciones pueden ser invocadas en expresiones SQL y se pueden utilizar en cualquier lugar donde se permita una expresión SQL.
   * Los procedimientos no se pueden invocar directamente en expresiones SQL y se usan para acciones dentro de bloques de código.

**Ejemplo de Función en SQL-PSM:**

Supongamos que necesitamos una función en SQL-PSM que calcule el área de un círculo dado su radio.

CREATE FUNCTION calcular\_area\_circulo(radio FLOAT) RETURNS FLOAT

BEGIN

DECLARE area FLOAT;

SET area = 3.14159 \* radio \* radio;

RETURN area;

END;

Esta función toma un parámetro **radio** y devuelve el área del círculo correspondiente.

Para utilizar esta función:

SELECT calcular\_area\_circulo(5) AS area\_circulo;

Este ejemplo calculará el área de un círculo con un radio de 5 unidades utilizando la función **calcular\_area\_circulo** y mostrará el resultado en la columna **area\_circulo**.

**Ejemplo de Procedimiento en SQL-PSM:**

Por otro lado, supongamos que necesitamos un procedimiento en SQL-PSM que imprima un mensaje de saludo.

CREATE PROCEDURE saludar(nombre VARCHAR(100))

AS

BEGIN

DECLARE mensaje VARCHAR(100);

SET mensaje = 'Hola, ' || nombre || '. ¡Bienvenido!';

DBMS\_OUTPUT.PUT\_LINE(mensaje);

END;

Este procedimiento toma un parámetro **nombre** y simplemente imprime un mensaje de saludo utilizando **DBMS\_OUTPUT.PUT\_LINE**.

Para utilizar este procedimiento:

CALL saludar('Juan');

Este ejemplo llamará al procedimiento **saludar** con el argumento **'Juan'**, lo que imprimirá por pantalla el mensaje "Hola, Juan. ¡Bienvenido!".

* **2.2 Funciones: Diferencias con los procedimientos y ejemplos de uso en PL-SQL**

**Diferencias entre Funciones y Procedimientos en PL/SQL:**

1. **Estructura y Retorno de Valores:**
   * Una función siempre devuelve un valor a quien la llama.
   * Un procedimiento puede o no devolver valores.
2. **Llamada y Uso:**
   * Las funciones pueden ser llamadas desde SQL, dentro de expresiones o sentencias, así como desde otros bloques de PL/SQL.
   * Los procedimientos se utilizan principalmente para realizar acciones, como actualizar datos o realizar cálculos, pero no se pueden llamar directamente desde SQL.
3. **Invocación:**
   * Las funciones se invocan en expresiones SQL y se pueden utilizar en cualquier lugar donde se permita una expresión SQL.
   * Los procedimientos se invocan mediante la sentencia **CALL** o dentro de bloques PL/SQL.

**Ejemplo de Función en PL/SQL:**

Supongamos que necesitamos una función en PL/SQL que calcule el área de un círculo dado su radio.

CREATE OR REPLACE FUNCTION calcular\_area\_circulo(radio NUMBER) RETURN NUMBER IS

area NUMBER;

BEGIN

area := 3.14159 \* radio \* radio;

RETURN area;

END;

/

Esta función toma un parámetro **radio** y devuelve el área del círculo correspondiente.

Para utilizar esta función:

DECLARE

area\_circulo NUMBER;

BEGIN

area\_circulo := calcular\_area\_circulo(5);

DBMS\_OUTPUT.PUT\_LINE('El área del círculo es: ' || area\_circulo);

END;

/

Este ejemplo calculará el área de un círculo con un radio de 5 unidades utilizando la función **calcular\_area\_circulo** y mostrará el resultado utilizando **DBMS\_OUTPUT.PUT\_LINE**.

**Ejemplo de Procedimiento en PL/SQL:**

Por otro lado, supongamos que necesitamos un procedimiento en PL/SQL que imprima un mensaje de saludo.

CREATE OR REPLACE PROCEDURE saludar(nombre VARCHAR2) AS

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hola, ' || nombre || '. ¡Bienvenido!');

END;

/

Este procedimiento toma un parámetro **nombre** y simplemente imprime un mensaje de saludo utilizando **DBMS\_OUTPUT.PUT\_LINE**.

Para utilizar este procedimiento:

BEGIN

saludar('Juan');

END;

/

Este ejemplo llamará al procedimiento **saludar** con el argumento **'Juan'**, lo que imprimirá por pantalla el mensaje "Hola, Juan. ¡Bienvenido!".

* **2.3 Uso de Parámetros en los Subprogramas: Cómo se pasan los parámetros a funciones y procedimientos en PL/SQL & SQL/PSM?**
* **PL/SQL**

En PL/SQL, los subprogramas, es decir, las funciones y los procedimientos, pueden aceptar parámetros para realizar operaciones específicas.

Hay dos tipos principales de parámetros que se pueden utilizar: parámetros de entrada (IN) y parámetros de salida (OUT).

Los parámetros de entrada se utilizan para pasar valores al subprograma, mientras que los parámetros de salida se utilizan para devolver valores del subprograma.

**Parámetros de Entrada**

Los parámetros de entrada se utilizan para pasar valores al subprograma. Estos valores se pueden utilizar dentro del subprograma, pero no se pueden modificar.

Un ejemplo de cómo se usan los parámetros de entrada en un procedimiento en PL/SQL:

CREATE OR REPLACE PROCEDURE mostrar\_mensaje(p\_mensaje IN VARCHAR2) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(p\_mensaje);

END;

/

**Parámetros de Salida**

Los parámetros de salida se utilizan para devolver valores del subprograma. Estos valores pueden ser modificados dentro del subprograma y devueltos al programa principal.

Un ejemplo de cómo se usan los parámetros de salida en un procedimiento en PL/SQL:

CREATE OR REPLACE PROCEDURE suma(a IN NUMBER, b IN NUMBER, resultado OUT NUMBER) IS

BEGIN

resultado := a + b;

END;

/

* **SQL/PSM**

En SQL/PSM, los subprogramas, es decir, las funciones y los procedimientos, pueden aceptar parámetros para realizar operaciones específicas. Hay dos tipos principales de parámetros que se pueden utilizar: parámetros de entrada (IN) y parámetros de salida (OUT).

Los parámetros de entrada se utilizan para pasar valores al subprograma, mientras que los parámetros de salida se utilizan para devolver valores del subprograma.

**Parámetros de Entrada**

Los parámetros de entrada se utilizan para pasar valores al subprograma. Estos valores se pueden utilizar dentro del subprograma, pero no se pueden modificar.

Un ejemplo de cómo se usan los parámetros de entrada en un procedimiento en SQL/PSM:

CREATE PROCEDURE mostrar\_mensaje (IN p\_mensaje VARCHAR(255))

BEGIN

SELECT p\_mensaje;

END;

**Parámetros de Salida**

Los parámetros de salida se utilizan para devolver valores del subprograma. Estos valores pueden ser modificados dentro del subprograma y devueltos al programa principal.

Un ejemplo de cómo se usan los parámetros de salida en un procedimiento en SQL/PSM:

CREATE PROCEDURE suma (IN a INT, IN b INT, OUT resultado INT)

BEGIN

SET resultado = a + b; -- Asigna el resultado de la suma a la variable de salida

END;

* **2.4 Sobre carga de Subprogramas y Recursividad: Explicación y ejemplos en SQL-PSM**

**Sobrecarga de Subprogramas en SQL-PSM:**

La sobrecarga de subprogramas se refiere a la capacidad de tener múltiples subprogramas con el mismo nombre pero con diferentes listas de parámetros. En SQL-PSM, esta funcionalidad no está disponible directamente. Los procedimientos y funciones deben tener nombres únicos en una base de datos.

Sin embargo, puedes simular la sobrecarga de subprogramas utilizando diferentes nombres para tus subprogramas y documentando claramente su propósito y las diferencias en la lista de parámetros.

**Ejemplo de Sobrecarga de Subprogramas en SQL-PSM:**

CREATE PROCEDURE calcular\_area\_circulo(radio FLOAT) AS

BEGIN

DECLARE area FLOAT;

SET area = 3.14159 \* radio \* radio;

DBMS\_OUTPUT.PUT\_LINE('El área del círculo es: ' || area);

END;

CREATE PROCEDURE calcular\_area\_rectangulo(base FLOAT, altura FLOAT) AS

BEGIN

DECLARE area FLOAT;

SET area = base \* altura;

DBMS\_OUTPUT.PUT\_LINE('El área del rectángulo es: ' || area);

END;

**Recursividad en SQL-PSM:**

La recursividad es la capacidad de una función o procedimiento para llamarse a sí mismo. En SQL-PSM, esta característica está disponible, aunque debes tener cuidado con la recursividad infinita, ya que puede llevar a un desbordamiento de pila.

**Ejemplo de Recursividad en SQL-PSM:**

Supongamos que queremos calcular el factorial de un número utilizando un procedimiento recursivo en SQL-PSM:

CREATE PROCEDURE calcular\_factorial(numero INT, OUT resultado INT) AS

BEGIN

IF numero = 0 THEN

resultado := 1;

ELSE

CALL calcular\_factorial(numero - 1, resultado);

resultado := resultado \* numero;

END IF;

END;

Este procedimiento calcula el factorial de un número utilizando la recursividad. Cuando el número llega a 0, el resultado se establece en 1. De lo contrario, se llama al procedimiento con un número decrementado y el resultado se multiplica por el número actual.

DECLARE

factorial\_resultado INT;

BEGIN

CALL calcular\_factorial(5, factorial\_resultado);

DBMS\_OUTPUT.PUT\_LINE('El factorial de 5 es: ' || factorial\_resultado);

END;

Este ejemplo llama al procedimiento **calcular\_factorial** para calcular el factorial de 5 y luego imprime el resultado.

* **2.4 Sobre carga de Subprogramas y Recursividad: Explicación y ejemplos en PL-SQL**

**Sobrecarga de Subprogramas en PL/SQL:**

Para sobrecargar subprogramas en PL/SQL, simplemente defines múltiples subprogramas con el mismo nombre pero con diferentes listas de parámetros. El compilador distingue entre ellos según la cantidad o tipo de parámetros que se utilicen al llamar al subprograma.

Ejemplo de Sobrecarga de Subprogramas en PL/SQL:

-- Declaración de dos funciones con el mismo nombre pero diferentes listas de parámetros

CREATE OR REPLACE FUNCTION calcular\_area(lado NUMBER) RETURN NUMBER AS

BEGIN

RETURN lado \* lado; -- Área de un cuadrado

END;

/

CREATE OR REPLACE FUNCTION calcular\_area(base NUMBER, altura NUMBER) RETURN NUMBER AS

BEGIN

RETURN base \* altura; -- Área de un rectángulo

END;

/

**Recursividad en PL/SQL:**

La recursividad es la capacidad de una función o procedimiento para llamarse a sí mismo, lo que permite resolver problemas de manera más elegante y concisa. En PL/SQL, puedes implementar funciones y procedimientos recursivos para realizar tareas repetitivas o resolver problemas que se pueden descomponer en casos más pequeños del mismo tipo.

Ejemplo de Recursividad en PL/SQL:

Supongamos que queremos calcular el factorial de un número utilizando un procedimiento recursivo en PL/SQL:

CREATE OR REPLACE FUNCTION calcular\_factorial(numero INT) RETURN INT AS

BEGIN

IF numero <= 1 THEN

RETURN 1;

ELSE

RETURN numero \* calcular\_factorial(numero - 1); -- Llamada recursiva

END IF;

END;

/

Este ejemplo calcula el factorial de un número utilizando la recursividad. Cuando el número llega a 1 o menos, la función devuelve 1. De lo contrario, se llama a sí misma con un número decrementado y el resultado se multiplica por el número actual.

DECLARE

factorial\_resultado INT;

BEGIN

factorial\_resultado := calcular\_factorial(5);

DBMS\_OUTPUT.PUT\_LINE('El factorial de 5 es: ' || factorial\_resultado);

END;

/

Este bloque PL/SQL llama a la función **calcular\_factorial** para calcular el factorial de 5 y luego imprime el resultado. La función se llama recursivamente para calcular el factorial de números más pequeños hasta llegar a 1.

* **2.5 Procedimientos frente a Funciones: Cuándo usar cada uno. Paquetes (SQL-PSM)**

**Procedimientos vs Funciones en SQL-PSM:**

Procedimientos:

* Los procedimientos son subprogramas que realizan una tarea específica y pueden o no devolver valores.
* Se utilizan para realizar acciones como manipulación de datos, actualizaciones, inserciones, etc.
* No necesitan devolver un valor, pero pueden tener parámetros de entrada y salida.
* Se invocan usando la palabra clave **CALL**.
* Se utilizan cuando necesitas realizar acciones sin necesariamente devolver un valor específico.

Funciones:

* Las funciones son subprogramas que siempre devuelven un valor.
* Se utilizan para realizar cálculos o manipulaciones de datos que devuelven un resultado.
* Siempre deben devolver un valor y pueden tener parámetros de entrada.
* Se invocan dentro de expresiones SQL o en bloques de código PL/SQL.
* Se utilizan cuando necesitas calcular o manipular datos y devolver un resultado específico.

**Paquetes en SQL-PSM:**

Los paquetes en SQL-PSM son una forma de organizar y encapsular lógicamente procedimientos y funciones relacionadas, así como variables y constantes. Proporcionan un medio para agrupar de manera coherente la lógica de negocio y los datos asociados.

Creación de un Paquete en SQL-PSM:

CREATE PACKAGE nombre\_paquete AS

-- Declaraciones de procedimientos y funciones públicas

PROCEDURE nombre\_procedimiento;

FUNCTION nombre\_funcion RETURN NUMBER;

END nombre\_paquete;

/

CREATE PACKAGE BODY nombre\_paquete AS

-- Implementación de procedimientos y funciones

PROCEDURE nombre\_procedimiento AS

BEGIN

-- Código del procedimiento

END nombre\_procedimiento;

FUNCTION nombre\_funcion RETURN NUMBER AS

BEGIN

-- Código de la función

END nombre\_funcion;

END nombre\_paquete;

/

***Uso de un Paquete en SQL-PSM:***

DECLARE

resultado\_funcion NUMBER;

BEGIN

nombre\_paquete.nombre\_procedimiento; -- Llamada a un procedimiento del paquete

resultado\_funcion := nombre\_paquete.nombre\_funcion; -- Llamada a una función del paquete

-- Resto del código

END;

/

Los paquetes se utilizan para organizar y modularizar el código, facilitando su mantenimiento y reutilización. Puedes agrupar procedimientos y funciones relacionadas en un paquete y luego llamar a esos subprogramas según sea necesario en tu código.

* **2.5 Procedimientos frente a Funciones: Cuándo usar cada uno. Paquetes. (PL-SQL)**

**Procedimientos vs Funciones en PL/SQL:**

Procedimientos:

* Los procedimientos son subprogramas que realizan una tarea específica y pueden o no devolver valores.
* Se utilizan principalmente para realizar acciones como manipulación de datos, actualizaciones, inserciones, etc.
* No necesitan devolver un valor, pero pueden tener parámetros de entrada y salida.
* Se invocan usando la palabra clave **CALL** o simplemente llamándolos dentro de bloques de código PL/SQL.
* Se utilizan cuando necesitas realizar acciones sin necesariamente devolver un valor específico.

Funciones:

* Las funciones son subprogramas que siempre devuelven un valor.
* Se usan para realizar cálculos o manipulaciones de datos que devuelven un resultado.
* Siempre deben devolver un valor y pueden tener parámetros de entrada.
* Se invocan dentro de expresiones SQL, en bloques de código PL/SQL o en otras funciones y procedimientos.
* Se utilizan cuando necesitas calcular o manipular datos y devolver un resultado específico.

**Paquetes en PL/SQL:**

Los paquetes en PL/SQL son objetos que contienen procedimientos, funciones, variables, constantes y excepciones relacionadas. Proporcionan un mecanismo para modularizar y organizar el código, lo que facilita su mantenimiento y reutilización.

Creación de un Paquete en PL/SQL:

CREATE OR REPLACE PACKAGE nombre\_paquete AS

-- Declaraciones de procedimientos y funciones públicas

PROCEDURE nombre\_procedimiento;

FUNCTION nombre\_funcion RETURN NUMBER;

END nombre\_paquete;

/

CREATE OR REPLACE PACKAGE BODY nombre\_paquete AS

-- Implementación de procedimientos y funciones

PROCEDURE nombre\_procedimiento AS

BEGIN

-- Código del procedimiento

END nombre\_procedimiento;

FUNCTION nombre\_funcion RETURN NUMBER AS

BEGIN

-- Código de la función

END nombre\_funcion;

END nombre\_paquete;

/

***Uso de un Paquete en PL/SQL:***

DECLARE

resultado\_funcion NUMBER;

BEGIN

nombre\_paquete.nombre\_procedimiento; -- Llamada a un procedimiento del paquete

resultado\_funcion := nombre\_paquete.nombre\_funcion; -- Llamada a una función del paquete

-- Resto del código

END;

/

Los paquetes se utilizan para organizar y modularizar el código, facilitando su mantenimiento y reutilización.

Puedes agrupar procedimientos y funciones relacionadas en un paquete y luego llamar a esos subprogramas según sea necesario en tu código.

**3.1 Creación de Paquetes: Cómo y por qué crear paquetes en PL/SQL & SQL/PSM**

Los paquetes en PL/SQL y SQL/PSM son como cajas que contienen funciones, procedimientos, variables y constantes relacionadas.

La creación de paquetes ayuda a organizar y modularizar el código, lo que facilita el mantenimiento y la reutilización de este.

Principalmente se deben crear paquetes por estos motivos:

* Organización del Código: Los paquetes ayudan a organizar el código en módulos lógicos, lo que facilita su comprensión y mantenimiento.
* Reutilización del Código: El código dentro de un paquete puede ser reutilizado en diferentes partes del programa. Esto promueve la cohesión y reduce la duplicación de código.

**EJEMPLO**

CREATE OR REPLACE PACKAGE GestionEmpleado AS

-- Declaración de procedimientos

PROCEDURE obtener\_empleado(p\_id IN NUMBER, p\_nombre OUT VARCHAR2, p\_salario OUT NUMBER);

PROCEDURE aumentar\_salario(p\_id IN NUMBER, p\_aumento IN NUMBER);

END GestionEmpleado;

/

**Diferencias en el uso de paquetes entre PL/SQL y SQL/PSM**

|  |  |
| --- | --- |
| **PL/SQL** | **SQL/PSM** |
| **Definición de paquetes**: En PL/SQL, los paquetes se definen con dos partes: la especificación y el cuerpo.  La especificación contiene los encabezados de los procedimientos y funciones, así como las declaraciones de los tipos de datos y variables globales.  El cuerpo contiene la implementación real de los procedimientos y funciones. | **Definición de paquetes**: En SQL/PSM, no existe una división formal entre la especificación y el cuerpo del paquete.  Se pueden crear procedimientos directamente sin una especificación formal, lo que simplifica la sintaxis en comparación con PL/SQL. |
| **Sintaxis de Creación y Uso**: En PL/SQL, para llamar a un procedimiento de un paquete, se utiliza el nombre del paquete seguido del nombre del procedimiento. | **Sintaxis de Creación y Uso**: En SQL/PSM, los procedimientos se definen y llaman directamente, sin especificar un paquete. |
| **Conclusión**  PL/SQL: Ofrece una estructura más formal con especificaciones y cuerpos separados. | **Conclusión**  SQL/PSM: Ofrece una sintaxis más simple y directa para la creación de procedimientos. |
|  |  |

**BIBLIOTECA**

**https://www.unir.net/ingenieria/revista/que-es-plsql/**

**https://elbauldelprogramador.com/fundamentos-de-plsql/**

**https://www3.uji.es/~mmarques/apuntes\_bbdd/apuntes.pdf**

**https://es.scribd.com/document/391249719/Que-es-PSM**

**https://plsql.online/basico-pl-sql/operadores-en-pl-sql/#Operadores\_de\_comparacion**

**https://www.ibm.com/docs/es/i/7.4?topic=p-sql**

**https://www.discoduroderoer.es/parametros-de-salida-y-entrada-en-pl-sql/**

**http://basededatossql.blogspot.com/2013/07/cursores-implicitos-y-cursores.html**

**https://www.plsql.biz/2007/03/procedimientos-y-funciones-en-plsql.html**

**https://administracionbasedatosiutllano.blogspot.com/2012/06/grupo-1-psm-sql.html**