## **PL/pgSQL**

#### ***Declaraciones y Tipado***

* DECLARE var tipo;: Declara una variable local con un tipo.
* var := expresión;: Asignación de valores a variables declaradas.

#### ***Tipos de Variables***

| **Tipo** | **Descripción** | **Ejemplos** |
| --- | --- | --- |
| **INTEGER** | Números enteros | 5, 3, 24, -1 |
| **DECIMAL** | Números decimales | 3.14, -2.4, 0.2 |
| **VARCHAR(n)** | Strings de texto, van entre comillas dobles | “Hola mundo”, “Texto” |
| **CHAR** | Carácter entre comillas simples | ‘c’, ‘a’, ‘x’ |
| **BOOLEAN** | Booleano que representa verdadero o falso | true, false |
| **ARRAY** | Arreglo de elementos, es posible hacer ARRAY de cualquier otro tipo excepto del mismo ARRAY | [10,20,30] |
| **DATE** | Fechas sin tiempo | 2025-06-23 |

#### ***Operaciones Aritméticas***

Es posible la realización de operaciones matemáticas simples definidas por los siguientes operadores.

| **Símbolo** | **Descripción** | **Precedencia (de menor a mayor)** |
| --- | --- | --- |
| **+** | Suma | 1 |
| **-** | Resta | 1 |
| **\*** | Multiplicación | 2 |
| **/** | División | 2 |
| ^ | Potenciación | 3 |

#### ***Operaciones Relacionales***

Es posible realizar operaciones relacionales que conviertan condiciones en booleanos.

| **Símbolo** | **Descripción** | **Precedencia (todas 4)** |
| --- | --- | --- |
| **>** | Mayor que |  |
| **<** | Menor que |  |
| **>=** | Mayor o igual que |  |
| **<=** | Menor o igual que |  |
| **=** | Igual |  |
| **<>** | Diferente |  |

#### ***Operaciones Booleanas***

Es posible realizar operaciones booleanas para alterar un resultado. Estas se pueden aplicar fuera de condiciones.

| **Símbolo** | **Descripción** | **Precedencia (de menor a mayor)** |
| --- | --- | --- |
| **OR** | OR | 1 |
| **AND** | AND | 2 |
| **NOT** | Negación | 3 |

#### ***Conversiones de Tipos***

Es posible convertir ciertos tipos a otros a través de las diferentes operaciones especificadas.

No todas las operaciones están disponibles para todos los tipos.

Las **únicas** operaciones disponibles entre tipos están especificadas en la siguiente tabla:

| **Tipo 1** | **Tipo 2** | **Operaciones** | **Tipo Resultado** |
| --- | --- | --- | --- |
| **INTEGER** | **DECIMAL** | **Todas Aritméticas** | **DECIMAL** |
| **INTEGER** | **VARCHAR** | **+** | **VARCHAR** |
| **INTEGER** | **BOOLEAN** | **Todas Aritméticas (TRUE es 1, FALSE es 0)** | **INTEGER** |
| **VARCHAR** | **VARCHAR** | **+** | **VARCHAR** |
| **VARCHAR** | **CHAR** | **+** | **VARCHAR** |
| **VARCHAR** | **BOOLEAN** | **+ (TRUE es ‘TRUE’, FALSE es ‘FALSE’)** | **VARCHAR** |
| **CHAR** | **INTEGER** | **+** | **VARCHAR** |
| **BOOLEAN** | **VARCHAR** | **+** | **VARCHAR** |
| **BOOLEAN** | **BOOLEAN** | **Todas Aritméticas (True es 1, False es 0)** | **INTEGER** |

Si se intenta realizar operaciones fuera de las especificadas debe de emitirse un error.

#### ***Control de Flujo***

* IF condición THEN ... END IF;
* IF condición THEN ... ELSE ... END IF;
* CASE WHEN ... THEN ... ELSE ... END CASE;

#### ***Ciclos***

* FOR i IN [start]..[end] LOOP ... END LOOP;

| BEGIN FOR i IN 1..5 LOOP RAISE NOTICE 'Iteración número: %', i; END LOOP;END; |
| --- |

* FOREACH var IN ARRAY array LOOP ... END LOOP;

| DECLARE num INTEGER; numeros INTEGER[] := ARRAY[10, 20, 30, 40];BEGIN FOREACH num IN numeros LOOP RAISE NOTICE 'Número: %', num; END LOOP;END; |
| --- |

* FOREACH row IN result LOOP ... END LOOP;

| BEGIN FOREACH fila IN SELECT id, nombre FROM usuarios LOOP RAISE NOTICE 'Usuario: %', fila; END LOOP;END; |
| --- |

#### ***Sentencias de Control***

* EXIT;: Finaliza un ciclo.
* CONTINUE;: Salta a la siguiente iteración.

#### ***Impresión de Mensajes y Excepciones***

* RAISE;: Imprime un mensaje.
* RAISE NOTICE;: Imprime un mensaje informativo.
* RAISE WARNING;: Imprime un mensaje de alerta.
* RAISE EXCEPTION;: Imprime un error que detiene la ejecución del procedimiento.

Para:

| BEGIN RAISE NOTICE 'Proceso iniciado a las %', NOW();END; |
| --- |

| BEGIN RAISE NOTICE 'Proceso finalizado correctamente';END; |
| --- |

La salida sería:

NOTICE: Proceso iniciado a las 2025-08-07 10:15:23

NOTICE: Proceso finalizado correctamente

Para:

| DECLARE saldo NUMERIC := -50;BEGIN IF saldo < 0 THEN RAISE WARNING 'Saldo negativo detectado: %', saldo; END IF; RAISE NOTICE 'Proceso continúa a pesar de la advertencia';END; |
| --- |

La salida sería:

WARNING: Saldo negativo detectado: -50

NOTICE: Proceso continua a pesar de la advertencia

Para:

| DECLARE saldo NUMERIC := -100;BEGIN IF saldo < 0 THEN RAISE EXCEPTION 'Error: saldo negativo no permitido (%).', saldo; END IF; RAISE NOTICE 'Este mensaje no se mostrará porque la ejecución se detuvo';END; |
| --- |

La salida sería:

ERROR: Error: saldo negativo no permitido(-100).

#### ***Transaccionalidad***

* BEGIN ... COMMIT;: Cambios se aplican solo al confirmar.
* BEGIN ... ROLLBACK;: Reversión del estado previo.

#### ***Funciones Built-In***

El lenguaje PL/pgSQL cuenta con algunas funciones integradas en el propio lenguaje:

| **Nombre y Parámetros** | **Descripción** | **Tipo de Retorno** |
| --- | --- | --- |
| **NOW()** | Devuelve la fecha actual | **DATE** |
| **CONCAT(STRING, STRING…)** | Concatena varios STRING y devuelve el resultante | **STRING** |
| **MODIFY\_DATE(DATE, day: INTEGER, month: INTEGER, year: INTEGER)** | Modifica una fecha ingresada en una cantidad de días, meses o años. Si el valor es positivo aumenta, si es negativo disminuye. | **DATE** |
| **CAST(any, target)** | Convierte una variable de cualquier tipo en otro tipo (revisar tabla de conversiones) si los tipos son incompatibles se emite un error. | **ANY** |
| **RANDOM()** | Devuelve un valor aleatorio entre 0 y 1 | **DECIMAL** |
| **SUBSTRING(text: STRING, init: INTEGER, len: INTEGER)** | Devuelve una subcadena cortada desde el punto *init* con longitud *len.* Si *init* o *len* fueran inválidos ya que la cadena no es lo suficientemente larga, o contienen valores negativos, debe de emitir un error. | **STRING** |
| **ARRAY\_PUSH(target: ARRAY, value: ANY)** | Agrega un valor al final de un array especificado. Devuelve el array con el elemento agregado | **ARRAY** |
| **LEN(target: STRING|ARRAY)** | Devuelve la longitud de un STRING o un ARRAY | **INTEGER** |

#### ***Funciones***

Son bloques de código reutilizables que sirven para ejecutar alguna lógica sobre la base de datos, debe retornar un resultado y no admite transacciones, si la función falla y hay algún error, la transacción debe hacer ROLLBACK automáticamente.

* CREATE FUNCTION nombre(parámetros) RETURNS tipo AS $$ ... $$ LANGUAGE plpgsql;
* RETURN valor;: Valor devuelto desde una función.
* Ejemplo de uso SELECT nombre(parametros);.
* Las funciones solo pueden ser invocadas a través de SELECT y no pueden ser anidadas.
* Es posible renombrar funciones usando ALTER FUNCTION nombre(parámetros) RENAME TO nuevo\_nombre;
* Es posible cambiar el cuerpo de una función usando usando REPLACE FUNCTION nombre(parámetros) RETURNS tipo AS $$ ... $$ LANGUAGE plpgsql;

| CREATE FUNCTION calcular\_iva(precio DECIMAL, porcentaje DECIMAL)RETURNS DECIMAL AS $$DECLARE iva DECIMAL;BEGIN *-- Calcula el IVA* iva := precio \* (porcentaje / 100); *-- Devuelve el valor del IVA* RETURN iva;END;$$ LANGUAGE plpgsql; |
| --- |

#### ***Procedimientos***

Son bloques de código reutilizables que sirven para aplicar alguna lógica sobre la base de datos, estos no pueden retornar nada, pero admiten transacciones.

* CREATE PROCEDURE nombre(parámetros) LANGUAGE plpgsql AS $$ ... $$;
* Ejemplo de uso CALL nombre(parametros);.
* Los procedimientos sólo pueden ser invocados a través de CALL y no pueden ser anidados.
* Es posible renombrar funciones usando ALTER PROCEDURE nombre(parámetros) RENAME TO nuevo\_nombre;
* Es posible cambiar el cuerpo de una función usando usando REPLACE PROCEDURE nombre(parámetros) RETURNS tipo AS $$ ... $$ LANGUAGE plpgsql;

| CREATE PROCEDURE registrar\_usuario(nombre STRING, edad INTEGER)LANGUAGE plpgsqlAS $$BEGIN *-- Inserta un nuevo usuario en la tabla* INSERT INTO usuarios(nombre, edad) VALUES (nombre, edad); *-- Mensaje de confirmación* RAISE NOTICE 'Usuario % registrado con edad %', nombre, edad;END;$$; |
| --- |

Estas estructuras se validan con árbol de sintaxis, tipos correctos y contexto semántico apropiado.

#### ***Reglas adicionales***

* El lenguaje es case-insensitive.
* Se permiten comentarios tipo -- comentario de una línea y bloque /\* comentario \*/.
* Se requiere el uso del punto y coma ; como finalizador de instrucciones.

## **Manejo de Errores**

Los errores de un procedimiento serán mostrados en el entorno al momento de ejecutarlo, este botón efectuará las comprobaciones sobre el código y se mostrará en la barra inferior cualquier error presente en el código. No se podrán guardar cambios efectuados por un procedimiento sobre tablas si existen errores dentro del mismo.