



**UNIVERSIDAD DE CUENCA**  
**FACULTAD DE INGENIERÍA**  
Sep 2025 - Feb 2026

Base Datos II

Capítulo 2: SQL Procedural y Triggers (Funciones)

Juan Carlos Pesántez Ing., MgT., MBA., MTD.

# Notas

- 10 trabajos del futuro búsqueda en Internet, Chatgpt, Gemini, Claude,
- Ingresar en <https://www.awana.io/>



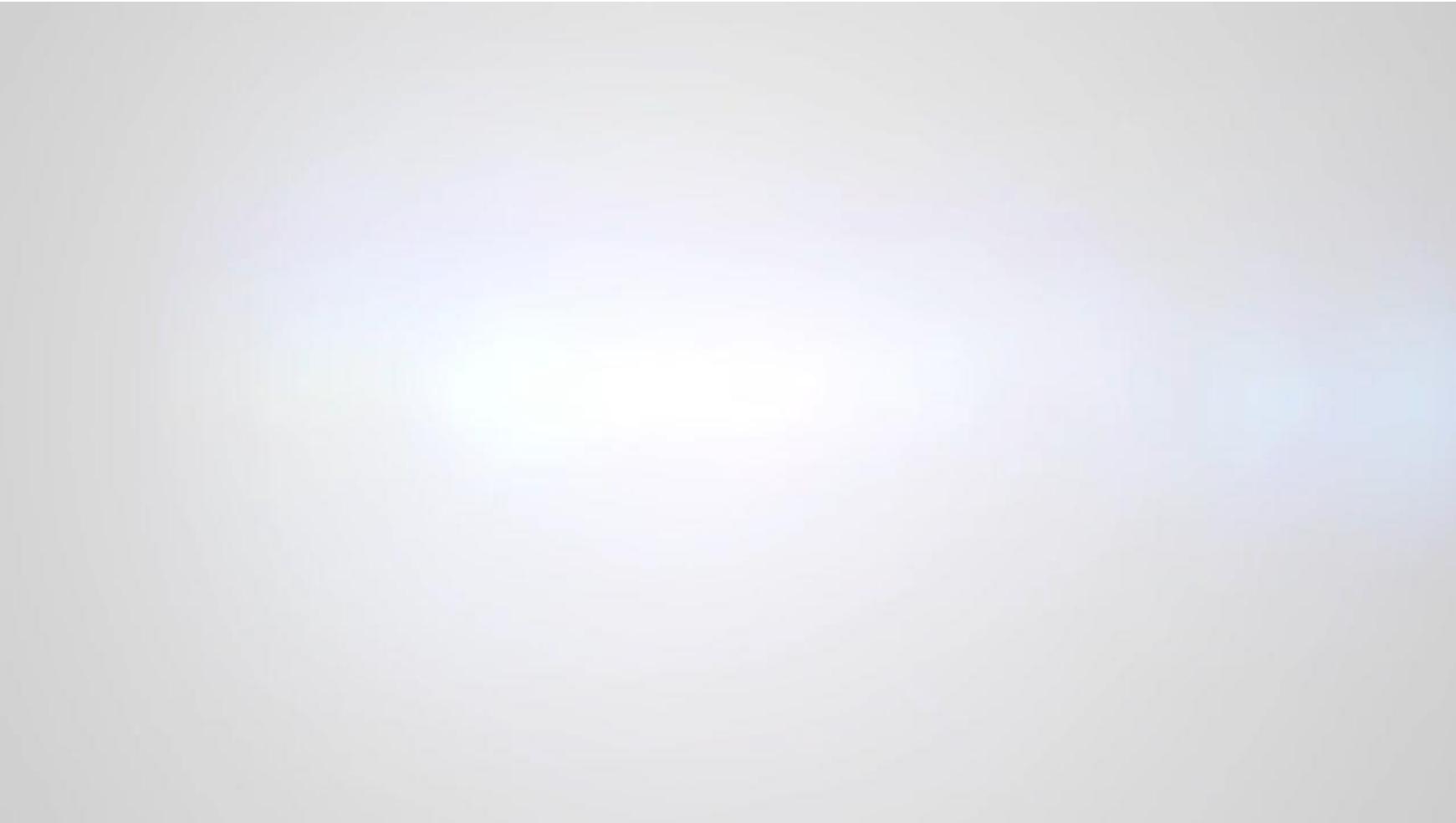
Miguel Ángel Durán ✅  
@midudev

Bootcamp FullStack JavaScript en Español  
Gratis y con Certificado final

- ✓ React + Redux
- ✓ Node con TypeScript
- ✓ APIs Express y GraphQL
- ✓ Desarrollo apps móviles
- ✓ Docker & GitHub Actions
- ✓ PostgreSQL y MongoDB

De la Universidad de Helsinki:  
→ [fullstackopen.com/es](https://fullstackopen.com/es)

# Video Introducción



# Objetivos

Al finalizar la sesión, el estudiante será capaz de diseñar y **crear funciones SQL personalizadas** que encapsulen lógica de negocio reutilizable.

Utilizando estructuras de control y parámetros de entrada, con el fin de **automatizar tareas, validar reglas** y optimizar consultas en sistemas reales de bases de datos.



# Qué es una función en SQL?

Una **función en SQL** es un bloque de código que **recibe parámetros de entrada**, ejecuta una **lógica específica**, y **retorna un valor (o una tabla)**, permitiendo **automatizar y reutilizar procesos** dentro del motor de base de datos.



# Qué es una función en SQL?

## Características clave:

- Devuelve **un solo valor** (función escalar) o **una tabla** (función de tabla).
- Se puede **invocar desde consultas SQL** como si fuera una columna.
- Encapsula lógica que se repite, como **cálculos, validaciones o transformaciones**.

## Ejemplo:

```
SELECT calcular_total_con_descuento(100, 0.10); -- Devuelve 90
```



# Para que sirven?

Permiten **automatizar decisiones, reducir errores y acelerar procesos** al ejecutar lógica directamente en el servidor de base de datos, sin depender del código de la aplicación.

CONTEXTOS	PROBLEMA QUE RESUELVE	FUNCIÓN EJEMPLO	VALOR PARA EL SISTEMA
 Salud	Calcular automáticamente la dosis según paciente	<code>calcular_dosis(peso, edad, tipo)</code>	Reduce errores médicos y estandariza tratamientos
 E-commerce	Determinar el nivel de cliente según historial de compras	<code>clasificar_cliente(cliente_id)</code>	Automatiza promociones personalizadas
 Finanzas	Evaluar riesgo crediticio en base a ingresos y deudas	<code>evaluar_riesgo(ingresos, deudas)</code>	Agiliza decisiones y reduce sesgos
 Educación	Generar comentarios automáticos en reportes académicos	<code>obtener_observacion(promedio)</code>	Ahorra tiempo docente y personaliza feedback



# Sintaxis Básica

```
CREATE OR REPLACE FUNCTION nombre_funcion(parámetros)
RETURNS tipo_dato AS $$

BEGIN
    -- Lógica de la función
    RETURN valor;
END;

$$ LANGUAGE plpgsql;
```

## Lógica de la función: calcular\_descuento()

- Primero verifica que ambos valores sean válidos (no nulos ni negativos).
- Luego calcula el monto con descuento multiplicando el precio por (1 – descuento).
- Finalmente devuelve el total resultante, redondeado a dos decimales.

CREATE OR REPLACE FUNCTION	Crea la función o la reemplaza si ya existe
nombre_funcion(parámetros)	Define el nombre y los parámetros de entrada
RETURNS tipo_dato	Especifica el tipo de valor que se devolverá
AS \$\$ ... \$\$	Delimita el bloque de código PL/pgSQL
BEGIN ... END;	Contiene la lógica de la función
RETURN valor;	Devuelve el resultado de la función
LANGUAGE plpgsql	Indica el lenguaje procedural usado (PostgreSQL requiere especificarlo)



# Sintaxis Básica

## Usa LANGUAGE SQL si:

- La función es **muy simple**
- Solo necesitas una **única sentencia SELECT**
- Quieres **máximo rendimiento** para tareas pequeñas
- No necesitas lógica de control ( `IF` , `LOOP` , etc.)

### Ventajas:

- Más rápida de ejecutar (menos sobrecarga)
- Más fácil de leer para funciones pequeñas
- Menor consumo de recursos

 *Ejemplo ideal:* funciones matemáticas o de formato.

## Usa LANGUAGE plpgsql si:

- Necesitas **estructuras de control** ( `IF` , `CASE` , `LOOP` , etc.)
- Vas a **manipular variables locales**
- Requieres **manejo de errores** ( `RAISE EXCEPTION` )
- La lógica es **compleja o multipaso**

### Ventajas:

- Mucho más poderoso y flexible
- Permite construir lógica de negocio robusta
- Ideal para funciones empresariales reales

 *Ejemplo ideal:* validaciones, auditoría, reglas de negocio.



# Sintaxis Básica

## 🧠 ¿Qué es IN, OUT e INOUT?

Son modos de los parámetros de una función:

Modo	¿Para qué sirve?
IN	👉 El valor se pasa a la función como entrada (es el más común).
OUT	👉 El valor se genera dentro de la función y se devuelve como resultado.
INOUT	👉 El valor entra y también puede modificarse y salir de la función.

## 📝 ¿Cuándo usar cada uno?

Caso real	Usa
Calcular algo con valores dados	IN
Devolver múltiples resultados	OUT
Modificar un parámetro recibido	INOUT

IN: una función recibe el precio y el descuento para calcular el total. (Solo usa los valores de entrada).

OUT: una función genera un código de cliente nuevo y lo devuelve. (El valor se crea dentro de la función).

INOUT: una función recibe un saldo, le suma intereses y devuelve el saldo actualizado. (El valor entra y sale modificado).



# Sintaxis Básica

## Versión simple con LANGUAGE SQL

```
CREATE OR REPLACE FUNCTION calcular_area_circulo(radio NUMERIC)
RETURNS NUMERIC
AS $$

    SELECT PI() * radio * radio;

$$ LANGUAGE SQL;
```

## Versión con LANGUAGE plpgsql

```
CREATE OR REPLACE FUNCTION calcular_area_circulo(radio NUMERIC)
RETURNS NUMERIC AS $$

BEGIN
    RETURN PI() * radio * radio;
END;

$$ LANGUAGE plpgsql;
```



# Sintaxis Básica

## 💡 Ejemplo con IN

```
sql

CREATE FUNCTION sumar(a INT, b INT)
RETURNS INT AS $$

BEGIN
    RETURN a + b;
END;
$$ LANGUAGE plpgsql;
```

- `a` y `b` son `IN` (entrada).
- Solo recibes un resultado: la suma.

```
sql

SELECT sumar(2, 3); -- Resultado: 5
```



# Sintaxis Básica

## Ejemplo con `OUT`

```
CREATE FUNCTION dividir(a NUMERIC, b NUMERIC, OUT cociente NUMERIC, OUT resto NUMERIC)
AS $$

BEGIN
    cociente := a / b;
    resto := MOD(a, b);
END;

$$ LANGUAGE plpgsql;
```

- No usas `RETURN`, porque los valores se devuelven por los `OUT`.
- Devuelve **una tabla** con dos columnas: `cociente` y `resto`.

sql

```
SELECT * FROM dividir(10, 3);
-- cociente | resto
--      3.33  |   1
```



# Sintaxis Básica

## Ejemplo con `INOUT`

sql

```
CREATE FUNCTION incrementar(INOUT valor INT)
AS $$

BEGIN
    valor := valor + 1;
END;

$$ LANGUAGE plpgsql;
```

- Entra un número, se modifica **dentro de la función** y sale con el nuevo valor.

sql

```
SELECT incrementar(9); -- Resultado: 10
```



# Ejercicio para clases

## 1 Función con IN

Tema: Calcular el área de un rectángulo

Descripción:

Crea una función que reciba el *ancho* y el *alto* como parámetros de entrada ( **IN** ) y devuelva el *área*.

Ejemplo: ancho = 5, alto = 3 → resultado = 15

---

## 2 Función con OUT

Tema: Calcular el doble y el triple de un número

Descripción:

Crea una función que reciba un número ( **IN** ) y devuelva dos valores ( **OUT** ): su *doble* y su *triple*.

Ejemplo: número = 4 → doble = 8, triple = 12

---



# Ejercicio para clases

## 3 Función con INOUT

Tema: Incrementar un número

Descripción:

Crea una función que reciba un número ( `INOUT` ), lo aumente en **5** dentro de la función y lo devuelva modificado.

Ejemplo: `valor = 10 → nuevo valor = 15`

---

## 4 Función con IN y OUT

Tema: Calcular perímetro y área de un cuadrado

Descripción:

Crea una función que reciba el *lado* de un cuadrado ( `IN` ) y devuelva como salidas ( `OUT` ) su *perímetro* y su *área*.

Ejemplo: `lado = 4 → perímetro = 16, área = 16`



# Resolución Ejercicios

```
CREATE OR REPLACE FUNCTION area_rectangulo(ancho NUMERIC, alto NUMERIC)
RETURNS NUMERIC
AS $$
    SELECT ancho * alto;
$$ LANGUAGE SQL;

-- Prueba
SELECT area_rectangulo(5, 3); -- 15
```

```
CREATE OR REPLACE FUNCTION doble_y_triple(
    valor NUMERIC,
    OUT doble NUMERIC,
    OUT triple NUMERIC
)
AS $$
BEGIN
    doble := valor * 2;
    triple := valor * 3;
END;
$$ LANGUAGE plpgsql;

-- Prueba
SELECT * FROM doble_y_triple(4); -- doble=8, triple=12
```



# Resolución Ejercicios

```
CREATE OR REPLACE FUNCTION incrementar_en_cinco(INOUT valor NUMERIC)
AS $$

BEGIN
    valor := valor + 5;
END;

$$ LANGUAGE plpgsql;

-- Prueba
SELECT incrementar_en_cinco(10); -- 15
```

```
CREATE OR REPLACE FUNCTION perimetro_y_area_cuadrado(
    lado NUMERIC,
    OUT perimetro NUMERIC,
    OUT area      NUMERIC
)
AS $$

BEGIN
    perimetro := 4 * lado;
    area      := lado * lado;
END;

$$ LANGUAGE plpgsql;

-- Prueba
SELECT * FROM perimetro_y_area_cuadrado(4); -- perimetro=16, area=16
```

