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.

Sintaxis Básica

🧠 ¿Qué es `VARIADIC`?

Es una forma de decir:

> "Esta función puede recibir 0, 1 o muchos valores **del mismo tipo** como un arreglo."

🔷 Sin `VARIADIC`:

Debes definir cada argumento uno por uno:

```sql
CREATE FUNCTION sumar3(a INT, b INT, c INT)
RETURNS INT AS $$
BEGIN
    RETURN a + b + c;
END;
$$ LANGUAGE plpgsql;
```

Solo sirve para 3 números. Si pasas más, falla.

🔷 Con `VARIADIC`:

Recibes todos los argumentos como **un arreglo**.

```sql
CREATE FUNCTION sumar_varios(VARIADIC numeros INT[])
RETURNS INT AS $$
DECLARE
    total INT := 0;
    n INT;
BEGIN
    FOREACH n IN ARRAY numeros LOOP
        total := total + n;
    END LOOP;
    RETURN total;
END;
$$ LANGUAGE plpgsql;
```

Sintaxis Básica

```
SELECT sumar_varios(1, 2, 3, 4);   -- Resultado: 10
SELECT sumar_varios();             -- Resultado: 0
SELECT sumar_varios(100, -50);     -- Resultado: 50
```

🛠️ Importante

- El tipo `VARIADIC` **debe ser un arreglo**, como `INT[]`, `TEXT[]`, etc.
- Solo puedes tener **un parámetro VARIADIC** por función y debe estar al final.
- Es útil cuando no sabes cuántos argumentos vas a pasar.

💡 Cuándo usarlo

Caso	¿Usar `VARIADIC`?
Sumar una cantidad flexible de valores	✅ Sí
Recibir una lista de palabras para buscar	✅ Sí
Parámetros fijos siempre	❌ No

Ejercicio

Ejemplo (IN + OUT + VARIADIC)

Suma una lista variable de números, primero multiplicados por un factor de entrada. Devuelve (por OUT) la suma y el conteo de elementos.

```sql
CREATE OR REPLACE FUNCTION resumen_con_factor(
    factor NUMERIC,                    -- IN
    OUT suma NUMERIC,                  -- OUT
    OUT conteo INT,                    -- OUT
    VARIADIC numeros NUMERIC[]         -- VARIADIC (debe ir al final)
)
AS $$
DECLARE
    n NUMERIC;
BEGIN
    suma := 0;
    conteo := 0;

    FOREACH n IN ARRAY numeros LOOP
        suma := suma + (n * factor);
        conteo := conteo + 1;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

```sql
-- Pasando valores sueltos
SELECT * FROM resumen_con_factor(2, 1, 2, 3);
-- → suma = 12, conteo = 3   ( (1+2+3)*2 = 12 )

-- Pasando un arreglo
SELECT * FROM resumen_con_factor(1.5, VARIADIC ARRAY[4, 6]);
-- → suma = 15, conteo = 2   ( (4+6)*1.5 = 15 )
```

Ejercicio para clases

Los alumnos deben realizar un ejercicio utilizando

(IN + OUT + INOUT + VARIADIC)

Ejercicio para clases

💡 Ejercicio: Calcular estadísticas de ventas con ganancia (ajuste)

Descripción: Crea una función llamada resumen_ventas que:

- Reciba como entrada (IN) un porcentaje de ajuste que se aplicará a cada venta (por ejemplo, 0.10 para sumar 10%).
- Tenga un parámetro de entrada/salida (INOUT) llamado total_base, que inicia con un valor y se incrementa con las ventas procesadas.
- Devuelva como salidas (OUT):
  - promedio → el promedio de las ventas ajustadas.
  - mayor → la venta más alta después del ajuste.
- Y reciba un número variable de ventas (VARIADIC).

Ejercicio para clases

```sql
CREATE OR REPLACE FUNCTION resumen_ventas(
    ajuste NUMERIC,                 -- IN: porcentaje de ajuste (p.ej., 0.10)
    INOUT total_base NUMERIC,       -- INOUT: entra con un total y sale actualizad
    OUT promedio NUMERIC,           -- OUT: promedio de ventas ajustadas
    OUT mayor NUMERIC,              -- OUT: mayor venta ajustada
    VARIADIC ventas NUMERIC[]       -- VARIADIC: lista variable de ventas
)
AS $$
DECLARE
    v        NUMERIC;
    suma     NUMERIC := 0;
    cantidad INT := 0;
BEGIN
    mayor := NULL;

    FOREACH v IN ARRAY ventas LOOP
        v := v * (1 + ajuste);      -- aplicar ajuste
        suma := suma + v;           -- acumular
        cantidad := cantidad + 1;   -- contar
        IF mayor IS NULL OR v > mayor THEN
            mayor := v;             -- trackear máximo
        END IF;
    END LOOP;

    IF cantidad > 0 THEN
        promedio := suma / cantidad;
    ELSE
        promedio := NULL;           -- sin datos
        mayor := NULL;
    END IF;

    total_base := total_base + suma;    -- actualizar INOUT
END;
$$ LANGUAGE plpgsql;
```

```sql
-- Ejemplo 1
SELECT * FROM resumen_ventas(0.10, 1000, 200, 300, 500, 100);
-- total_base = 2210, promedio = 302.5, mayor = 550

-- Ejemplo 2: pasando un arreglo
SELECT * FROM resumen_ventas(0.05, 0, VARIADIC ARRAY[120, 80, 100]);
-- total_base = 315, promedio = 105, mayor = 126
```

Ejercicio para clases

¿Quién es quién?

- `0.10` → **ajuste** (IN)
- `1000` → **total_base** (INOUT)
- `200, 300, 500, 100` → **ventas** (son el **VARIADIC**; PostgreSQL los empaqueta como `ARRAY[200,300,500,100]`)

> O sea: **1000 NO es variadic**; es el `total_base`.
> Los variadic son **200, 300, 500 y 100**.

Cálculo paso a paso

- Factor de ajuste: $1 + 0.10 = 1.10$
- Ventas ajustadas:
  - 200 → 220
  - 300 → 330
  - 500 → 550
  - 100 → 110
- **Suma** ajustada: $220 + 330 + 550 + 110 = 1210$
- **Conteo**: 4
- **promedio**: $1210/4 = 302.5$
- **mayor**: 550
- **total_base (INOUT)**: $1000 + 1210 = 2210$

Resultado que retorna

Como devuelve `INOUT + OUT + OUT`, verás una fila con:

```
diff

total_base | promedio | mayor
-----------+----------+------
     2210  |   302.5  |  550
```

Equivalente usando arreglo explícito:

```sql
SELECT * FROM resumen_ventas(0.10, 1000, VARIADIC ARRAY[200,300,500,100]);
```

Ejemplos SQL

✅ Ejemplo 1: Cálculo matemático simple

```sql
CREATE OR REPLACE FUNCTION convertir_a_fahrenheit(celsius NUMERIC)
RETURNS NUMERIC AS $$
   SELECT (celsius * 9/5) + 32;
$$ LANGUAGE SQL;
```

📌 *Por qué SQL:* Es solo una fórmula directa → no necesita control de flujo.

✅ Ejemplo 2: Concatenar texto

```sql
CREATE OR REPLACE FUNCTION nombre_completo(nombre TEXT, apellido TEXT)
RETURNS TEXT AS $$
   SELECT nombre || ' ' || apellido;
$$ LANGUAGE SQL;
```

📌 *Por qué SQL:* Sencilla operación de cadena → rápida y clara.

✅ Ejemplo 3: Consultar un dato puntual

```sql
CREATE OR REPLACE FUNCTION obtener_precio(producto_id INT)
RETURNS NUMERIC AS $$
   SELECT precio FROM productos WHERE id = producto_id;
$$ LANGUAGE SQL;
```

📌 *Por qué SQL:* Esencialmente un `SELECT` directo.

Ejemplos PLPGSQL

✅ Ejemplo 4: Validación antes de cálculo

```sql
CREATE OR REPLACE FUNCTION calcular_area_seguro(radio NUMERIC)
RETURNS NUMERIC AS $$
BEGIN
   IF radio <= 0 THEN
      RAISE EXCEPTION 'El radio debe ser mayor que cero';
   END IF;
   RETURN PI() * radio * radio;
END;
$$ LANGUAGE plpgsql;
```

📌 *Por qué* `plpgsql` : Requiere validación condicional (`IF`).

Ejemplos PLPGSQL

✅ Ejemplo 5: Uso de variables y múltiples pasos

sql

```sql
CREATE OR REPLACE FUNCTION total_con_impuesto(precio NUMERIC, iva NUMERIC)
RETURNS NUMERIC AS $$
DECLARE
   total NUMERIC;
BEGIN
   total := precio + (precio * iva / 100);
   RETURN total;
END;
$$ LANGUAGE plpgsql;
```

📌 *Por qué* `plpgsql` : Usa variable interna y cálculo compuesto.

Ejemplos PLPGSQL

✅ Ejemplo 6: Validación + lógica de negocio

```sql
CREATE OR REPLACE FUNCTION clasificar_cliente(puntos INT)
RETURNS TEXT AS $$
BEGIN
   IF puntos < 100 THEN
      RETURN 'Bronce';
   ELSIF puntos < 500 THEN
      RETURN 'Plata';
   ELSE
      RETURN 'Oro';
   END IF;
END;
$$ LANGUAGE plpgsql;
```

📌 *Por qué* `plpgsql` : Flujo condicional que no puede hacerse en SQL puro.