

Unknown Title



Tú dijiste:

Actividad 1 - Bloques, variables, funciones y procedimientos.pdf

PDF

Ya hemos hecho la Actividad 1. Ahora vamos a hacer la 2: Actividad 2 - Estructuras de control Objetivo Desarrollar un "Toolkit de Administración" mediante funciones y procedimientos almacenados que optimicen la gestión de clientes, inventario y fidelización, utilizando todas las estructuras de control de flujo. Entregables 1. Script SQL (.sql): Con el código de las 4 funciones/procedimientos. 2. Capturas de pantalla: Ejecutando cada función con los siguientes comandos de prueba: SQL -- Prueba IF/CASE
SELECT first_name, fn_nivel_cliente(customer_id) FROM customer LIMIT 5; -- Prueba FOR CALL pr_resumen_inventario(); -- Prueba WHILE (Recargo por 10 días de retraso) SELECT fn_calculo_recargo_progresivo(10); -- Prueba LOOP CALL pr_generar_cupones_campaña(); Tarea 1: Clasificación Dinámica de Clientes (IF y CASE) El gerente del videoclub quiere premiar a los clientes según el gasto total que hayan realizado en alquileres. Requerimiento: Crea una función llamada fn_nivel_cliente(p_customer_id INT) que devuelva un texto (VARCHAR) con la categoría del cliente: 1. Calcula el total gastado por el cliente sumando la columna amount de la tabla payment. 2. Usa una estructura IF para verificar si el cliente existe (si el total es NULL, lanzar un aviso). 3. Usa una estructura CASE para asignar el nivel: ○ Más de 150\$: "VIP Platinum" ○ Entre 100\$ y 150\$: "Gold" ○ Entre 50\$ y 99.99\$: "Silver" ○ Menos de 50\$: "Bronze" Tarea 2: Informe de Disponibilidad por Categoría (FOR)

Necesitamos saber cuántas copias tenemos de cada categoría de película (Acción, Comedia, etc.) para decidir qué secciones ampliar. Requerimiento: Crea un procedimiento llamado pr_resumen_inventario() que: 1. Utilice un bucle FOR que recorra los registros de la tabla category. 2. Por cada categoría, cuente cuántas películas hay asociadas en la tabla inventory (necesitarás hacer joins entre category, film_category e inventory). 3. Muestre por pantalla (usando RAISE NOTICE) un mensaje tipo: "Categoría: [Nombre], Total Copias: [Número]". Tarea 3: Simulador de Recargos por Demora (WHILE) El videoclub quiere implementar un sistema donde el recargo por no devolver una película aumenta exponencialmente cada día que pasa después de la fecha límite. Requerimiento: Crea una función fn_calculo_recargo_progresivo(p_dias_retraso INT) que: 1. Inicie una variable v_total_recargo en 1.0 (dólar). 2. Inicie un contador v_dia en 1. 3. Utilice un bucle WHILE que se ejecute mientras v_dia sea menor o igual a p_dias_retraso. 4. En cada iteración, el recargo aumenta un 10% respecto al día anterior. 5. Devuelva el recargo final redondeado a dos decimales. Tarea 4: Generador de Cupones de Descuento (LOOP + EXIT) Queremos generar códigos de descuento únicos para una campaña de marketing hasta alcanzar un presupuesto máximo de 500\$ en descuentos regalados. Requerimiento: Crea un procedimiento pr_generar_cupones_campaña() que: 1. Declare una variable v_acumulado iniciada en 0. 2. Utilice un bucle LOOP básico. 3. En cada vuelta del bucle:

- Elija un cliente al azar (puedes usar ORDER BY random() LIMIT 1).
- "Asigne" un cupón simbólico de 15\$ (sumándolo a v_acumulado).
- Muestre el mensaje: "Cupón asignado a cliente ID: [id]. Total acumulado: [v_acumulado]".

4. Utilice una cláusula EXIT WHEN para salir del bucle cuando v_acumulado supere los 500\$.

ChatGPT Plus

Perfecto, Cristóbal 🌟 Vamos a hacer la **Actividad 2 – Estructuras de control** adaptada a PostgreSQL (dvrental), siguiendo exactamente lo que te piden: IF, CASE, FOR, WHILE y LOOP + EXIT.

Te dejo el **script completo** listo para guardar como actividad2_toolkit.sql.

◆ TAREA 1 – IF + CASE

Función: fn_nivel_cliente

sql

```
CREATE OR REPLACE FUNCTION fn_nivel_cliente(p_customer_id INT)
RETURNS VARCHAR AS
$ 
DECLARE
    v_total NUMERIC(10,2);
    v_nivel VARCHAR(50);
BEGIN
    -- Calculamos total gastado
    SELECT SUM(amount)
```

```

INTO v_total
FROM payment
WHERE customer_id = p_customer_id;

-- IF → verificar si existe cliente (sin pagos)
IF v_total IS NULL THEN
    RAISE NOTICE 'El cliente con ID % no existe o no tiene pagos.', p_customer_id;
    RETURN 'Sin registros';
END IF;

-- CASE → clasificación
CASE
    WHEN v_total > 150 THEN
        v_nivel := 'VIP Platinum';
    WHEN v_total BETWEEN 100 AND 150 THEN
        v_nivel := 'Gold';
    WHEN v_total BETWEEN 50 AND 99.99 THEN
        v_nivel := 'Silver';
    ELSE
        v_nivel := 'Bronze';
END CASE;

RETURN v_nivel;
END;
$#
LANGUAGE plpgsql;

```

Prueba:

sql

```

SELECT first_name, fn_nivel_cliente(customer_id)
FROM customer
LIMIT 5;

```

◆ TAREA 2 – FOR

Procedimiento: pr_resumen_inventario

sql

```

CREATE OR REPLACE PROCEDURE pr_resumen_inventario()
AS

```

```

$ 
DECLARE
    v_categoria RECORD;
    v_total_copias INT;
BEGIN
    -- FOR recorriendo todas las categorías
    FOR v_categoria IN SELECT category_id, name FROM category
    LOOP
        -- Contamos copias por categoría
        SELECT COUNT(i.inventory_id)
        INTO v_total_copias
        FROM inventory i
        JOIN film f ON i.film_id = f.film_id
        JOIN film_category fc ON f.film_id = fc.film_id
        WHERE fc.category_id = v_categoria.category_id;

        RAISE NOTICE 'Categoría: %, Total Copias: %',
                      v_categoria.name, v_total_copias;
    END LOOP;
END;
$ 
LANGUAGE plpgsql;

```

Prueba:

sql

```
CALL pr_resumen_inventario();
```

◆ TAREA 3 – WHILE

Función: fn_calculo_recargo_progresivo

sql

```

CREATE OR REPLACE FUNCTION fn_calculo_recargo_progresivo(p_dias_retraso
INT)
RETURNS NUMERIC(10,2)
AS
$
DECLARE
    v_total_recargo NUMERIC(10,2) := 1.0;
    v_dia INT := 1;
BEGIN

```

```

    WHILE v_dia <= p_dias_retraso LOOP
        v_total_recargo := v_total_recargo * 1.10;
        v_dia := v_dia + 1;
    END LOOP;

    RETURN ROUND(v_total_recargo, 2);
END;
$-
LANGUAGE plpgsql;

```

 **Prueba:**

sql

```
SELECT fn_calculo_recargo_progresivo(10);
```

◆ **TAREA 4 – LOOP + EXIT**

Procedimiento: pr_generar_cupones_campaña

sql

```

CREATE OR REPLACE PROCEDURE pr_generar_cupones_campaña()
AS
$-
DECLARE
    v_acumulado NUMERIC(10,2) := 0;
    v_cliente RECORD;
BEGIN
    LOOP
        -- Elegimos cliente aleatorio
        SELECT customer_id
        INTO v_cliente
        FROM customer
        ORDER BY random()
        LIMIT 1;

        -- Sumamos cupón simbólico de 15$
        v_acumulado := v_acumulado + 15;

        RAISE NOTICE 'Cupón asignado a cliente ID: %. Total acumulado: %',
                      v_cliente.customer_id, v_acumulado;

        -- Salimos cuando superemos 500$
    END LOOP;
END;
$-
LANGUAGE plpgsql;

```

```
    EXIT WHEN v_acumulado > 500;
END LOOP;
END;
$-
LANGUAGE plpgsql;
```

Prueba:

sql

```
CALL pr_generar_cupones_campaña();
```

Entregable Final

Debes entregar:

1.  Archivo actividad2_toolkit.sql con las 4 estructuras.
2.  Capturas ejecutando:

- SELECT first_name, fn_nivel_cliente(customer_id) FROM customer LIMIT 5;
- CALL pr_resumen_inventario();
- SELECT fn_calculo_recargo_progresivo(10);
- CALL pr_generar_cupones_campaña();

Si quieres, ahora puedo prepararte una versión **comentada y más “profesional” para sacar mejor nota**, con encabezado tipo documentación técnica como hiciste en la Actividad 1 😊

Tú dijiste:

hace falta lo de "return 'sin ingresos'? CREATE OR REPLACE FUNCTION
fn_nivel_cliente(p_customer_id INT) RETURNS VARCHAR AS -- Crea una función llamada
fn_nivel_cliente(p_customer_id INT) que devuelva un texto (VARCHAR) con la categoría del cliente: \$
DECLARE -- Variables v_total NUMERIC(10,2); v_nivel VARCHAR(50); BEGIN -- 1. Calcula el total
gastado por el cliente sumando la columna amount de la tabla payment. Lo guardamos en v_total.
p_customer_id viene cuando llamamos a la función SELECT SUM(amount) INTO v_total FROM payment
WHERE customer_id = p_customer_id; -- 2. Usa una estructura IF para verificar si el cliente existe (si el
total es NULL, lanzar un aviso). IF v_total IS NULL THEN RAISE NOTICE 'El cliente con ID % no existe o
no tiene pagos.', p_customer_id; RETURN 'Sin registros'; END IF; -- 3. La estructura CASE
WHEN v_total > 150 THEN v_nivel := 'VIP Platinum'; WHEN v_total BETWEEN 100 AND 150 THEN

```
v_nivel := 'Gold'; WHEN v_total BETWEEN 50 AND 99.99 THEN v_nivel := 'Silver'; ELSE v_nivel := 'Bronze'; END CASE; RETURN v_nivel; END; $ LANGUAGE plpgsql;
```

ChatGPT Plus

Buena pregunta 🌟

🔍 ¿Hace falta el RETURN 'Sin registros'?

Sí, en este caso es necesario devolver algo.

Una función en PostgreSQL que tiene RETURNS VARCHAR siempre debe devolver un valor en todos los caminos posibles.

Si quitas:

sql

```
RETURN 'Sin registros';
```

y dejas solo el RAISE NOTICE, la función podría terminar sin hacer RETURN, y te daría este error:

sql

```
ERROR: control reached end of function without RETURN
```

🎯 ¿Es obligatorio que sea "Sin registros"?

No.

Podrías devolver:

- 'Cliente inexistente'
- 'Sin pagos'
- NULL
- O incluso lanzar una excepción (RAISE EXCEPTION)

Pero algo tienes que devolver.

◆ Alternativa más limpia (recomendada para mejor nota)

Podrías hacerlo así:

sql

```
IF v_total IS NULL THEN
    RAISE NOTICE 'El cliente con ID % no existe o no tiene pagos.', p_customer_id;
    RETURN NULL;
END IF;
```

Eso sería más “formal” desde el punto de vista técnico.

- ◆ **¿Y si realmentequieres comprobar que el cliente existe?**

Ahora mismo estás comprobando si **no tiene pagos**, no si no existe.

Si quisieras hacerlo perfecto, sería:

sql

```
IF NOT EXISTS (SELECT 1 FROM customer WHERE customer_id = p_customer_id)
THEN
    RAISE NOTICE 'El cliente no existe.';
    RETURN NULL;
END IF;
```

Y después ya calculas el total.

Conclusión

- ✓ Sí necesitas un RETURN en ese IF.
 - ✓ El texto que devuelvas es decisión tuya.
 - ✓ Técnicamente más correcto: RETURN NULL;

Siquieres, te dejo la versión “de matrícula” lista para entregar 😊

Tú dijiste:

quiero que expliques detenidamente cada una de ellas. Es muy importante que seas detallista en cada explicación. Todo depende de ello. No la cagues: \\\\\\ CREATE OR REPLACE FUNCTION fn_nivel_cliente(p_customer_id INT) RETURNS VARCHAR AS -- Crea una función llamada fn_nivel_cliente(p_customer_id INT) que devuelva un texto (VARCHAR) con la categoría del cliente: \$ DECLARE -- Variables v_total NUMERIC(10,2); v_nivel VARCHAR(50); BEGIN -- 1. Calcula el total gastado por el cliente sumando la columna amount de la tabla payment. Lo guardamos en v_total. p_customer_id viene cuando llamamos a la función SELECT SUM(amount) INTO v_total FROM payment WHERE customer_id = p_customer_id; -- 2. Usa una estructura IF para verificar si el cliente existe (si el

```

total es NULL, lanzar un aviso). IF v_total IS NULL THEN RAISE NOTICE 'El cliente con ID % no existe.';
p_customer_id; RETURN NULL; END IF; -- 3. La estructura CASE CASE WHEN v_total > 150 THEN
v_nivel := 'VIP Platinum'; WHEN v_total BETWEEN 100 AND 150 THEN v_nivel := 'Gold'; WHEN v_total
BETWEEN 50 AND 99.99 THEN v_nivel := 'Silver'; ELSE v_nivel := 'Bronze'; END CASE; RETURN
v_nivel; END; $ LANGUAGE plpgsql; \||||||| CREATE OR REPLACE PROCEDURE
pr_resumen_inventario() AS $ DECLARE v_categoria RECORD; v_total_copias INT; BEGIN -- FOR
recorriendo todas las categorías FOR v_categoria IN SELECT category_id, name FROM category LOOP
-- Contamos copias por categoría SELECT COUNT(i.inventory_id) INTO v_total_copias FROM inventory
i JOIN film f ON i.film_id = f.film_id JOIN film_category fc ON f.film_id = fc.film_id WHERE fc.category_id
= v_categoria.category_id; RAISE NOTICE 'Categoría: %, Total Copias: %', v_categoria.name,
v_total_copias; END LOOP; END; $ LANGUAGE plpgsql; \||||||| CREATE OR REPLACE FUNCTION
fn_calculo_recargo_progresivo(p_dias_retraso INT) RETURNS NUMERIC(10,2) AS $ DECLARE
v_total_recargo NUMERIC(10,2) := 1.0; v_dia INT := 1; BEGIN WHILE v_dia <= p_dias_retraso LOOP
v_total_recargo := v_total_recargo * 1.10; v_dia := v_dia + 1; END LOOP; RETURN
ROUND(v_total_recargo, 2); END; $ LANGUAGE plpgsql; \||||||| CREATE OR REPLACE
PROCEDURE pr_generar_cupones_campaña() AS $ DECLARE v_acumulado NUMERIC(10,2) := 0;
v_cliente RECORD; BEGIN LOOP -- Elegimos cliente aleatorio SELECT customer_id INTO v_cliente
FROM customer ORDER BY random() LIMIT 1; -- Sumamos cupón simbólico de 15$ v_acumulado :=
v_acumulado + 15; RAISE NOTICE 'Cupón asignado a cliente ID: %. Total acumulado: %',
v_cliente.customer_id, v_acumulado; -- Salimos cuando superemos 500$ EXIT WHEN v_acumulado >
500; END LOOP; END; $ LANGUAGE plpgsql;

```

ChatGPT Plus

Perfecto, Cristóbal.

Voy a explicarte **cada función y procedimiento línea por línea**, detallando:

- Qué hace técnicamente
- Qué estructura de control se está utilizando
- Qué ocurre internamente en PostgreSQL
- Qué pasaría si algo cambiara
- Por qué está bien implementado

Voy por partes.

1 **FUNCTION: fn_nivel_cliente**

(Estructuras usadas: IF + CASE)

◆ Cabecera

sql

```
CREATE OR REPLACE FUNCTION fn_nivel_cliente(p_customer_id INT)
RETURNS VARCHAR AS
```

¿Qué significa?

- CREATE OR REPLACE

Si la función ya existe, la reemplaza.

Esto evita tener que borrarla antes.

- FUNCTION fn_nivel_cliente(p_customer_id INT)

Declara una función llamada fn_nivel_cliente que recibe:

- p_customer_id → parámetro de entrada
- Tipo INT

La p_ indica que es un **parámetro**, buena práctica de nomenclatura.

- RETURNS VARCHAR

Es obligatorio indicar el tipo de dato que devuelve.

En este caso devuelve texto con la categoría del cliente.

◆ Bloque PL/pgSQL

sql

```
$
DECLARE
```

\$ delimita el cuerpo de la función.

DECLARE es donde definimos variables locales.

◆ Declaración de variables

sql

```
v_total NUMERIC(10,2);
v_nivel VARCHAR(50);
```

◆ v_total NUMERIC(10,2);

- Guardará el total gastado.
- 10 → número total de dígitos.

- 2 → decimales.
 - Es apropiado para dinero.
- ◆ **v_nivel VARCHAR(50);**
- Guardará la categoría final.
 - 50 caracteres es más que suficiente.
-

◆ **BEGIN**

sql

```
BEGIN
```

Aquí empieza la lógica ejecutable.

◆ **SELECT INTO**

sql

```
SELECT SUM(amount)
INTO v_total
FROM payment
WHERE customer_id = p_customer_id;
```

¿Qué ocurre aquí?

1. Se filtran los pagos del cliente.
2. SUM(amount) calcula el total.
3. El resultado se guarda en v_total.

 **Detalle importante**

Si el cliente:

- Existe pero no tiene pagos → SUM devuelve NULL.
- No existe → también devuelve NULL.

PostgreSQL no lanza error si no hay filas.

Simplemente asigna NULL.

◆ **IF (estructura condicional)**

sql

```
IF v_total IS NULL THEN
```

Aquí usamos la estructura de control IF.

¿Por qué IS NULL y no = NULL?

Porque en SQL:

ini

```
NULL = NULL → no es TRUE
```

Debe usarse IS NULL.

sql

```
RAISE NOTICE 'El cliente con ID % no existe.', p_customer_id;
```

- RAISE NOTICE muestra mensaje en consola.
 - % funciona como placeholder.
 - No detiene ejecución.
-

sql

```
RETURN NULL;
```

Esto es **fundamental**.

Una función que declara RETURNS VARCHAR:

- Debe devolver algo en todos los caminos posibles.

Si no pusieras RETURN aquí:

→ ERROR: control reached end of function without RETURN.

◆ CASE (estructura de decisión múltiple)

sql

```
CASE
```

Aquí empieza la estructura CASE.

Se usa cuando hay múltiples condiciones excluyentes.

sql

```
WHEN v_total > 150 THEN  
    v_nivel := 'VIP Platinum';
```

Si el total supera 150 → nivel máximo.

sql

```
WHEN v_total BETWEEN 100 AND 150 THEN
```

BETWEEN incluye ambos extremos:

- 100
- 150

⚠ Importante:

Como el primer WHEN ya capturó >150,
aquí realmente cubre 100–150 exactos.

sql

```
WHEN v_total BETWEEN 50 AND 99.99 THEN
```

Rango medio.

sql

```
ELSE  
    v_nivel := 'Bronze';
```

Todo lo inferior a 50 entra aquí.

El ELSE es importante para evitar casos no cubiertos.

◆ RETURN final

sql

```
RETURN v_nivel;
```

Devuelve la categoría calculada.

Si no existiera este RETURN:

→ ERROR de compilación.

2 PROCEDIMIENTO: pr_resumen_inventario

(Estructura usada: FOR)

◆ Cabecera

sql

```
CREATE OR REPLACE PROCEDURE pr_resumen_inventario()
```

- Es procedimiento (no función).
 - No devuelve valor.
 - Se ejecuta con CALL.
-

◆ Variables

sql

```
v_categoria RECORD;  
v_total_copias INT;
```

◆ RECORD

Tipo dinámico que puede almacenar una fila completa.

Aquí almacenará:

- category_id
 - name
-

◆ FOR automático en PostgreSQL

sql

```
FOR v_categoria IN SELECT category_id, name FROM category  
LOOP
```

Este FOR es especial:

- Ejecuta el SELECT.
- Recorre cada fila.
- En cada iteración, v_categoria contiene una fila.

Es un FOR implícito sobre un cursor.

◆ **Conteo de copias**

sql

```
SELECT COUNT(i.inventory_id)
INTO v_total_copias
```

Cuenta copias físicas, no películas.

◆ **JOINS explicados**

sql

```
FROM inventory i
JOIN film f ON i.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
```

Relación:

category
→ film_category
→ film
→ inventory

Estamos contando inventario por categoría.

◆ **WHERE**

sql

```
WHERE fc.category_id = v_categoria.category_id;
```

Filtrá según la categoría actual del bucle.

◆ **RAISE NOTICE**

Muestra algo como:

css

Categoría: Action, Total Copias: 150

◆ END LOOP

Finaliza el FOR.

◆ 3 FUNCIÓN: `fn_calculo_recargo_progresivo`

(Estructura usada: WHILE)

◆ Variables inicializadas

sql

```
v_total_recargo NUMERIC(10,2) := 1.0;  
v_dia INT := 1;
```

- Empieza en 1 dólar.
- Día 1.

◆ WHILE

sql

```
WHILE v_dia <= p_dias_retraso LOOP
```

Se ejecuta mientras la condición sea verdadera.

Si `p_dias_retraso = 10`:

→ 10 iteraciones.

◆ Incremento exponencial

sql

```
v_total_recargo := v_total_recargo * 1.10;
```

Cada día aumenta un 10%.

Matemáticamente:

scss

```
1 * (1.10)^n
```

◆ Incremento contador

sql

```
v_dia := v_dia + 1;
```

Sin esto:

→ bucle infinito.

◆ RETURN

sql

```
RETURN ROUND(v_total_recargo, 2);
```

Redondea a 2 decimales.

◆ 4 PROCEDIMIENTO: pr_generar_cupones_campaña

(Estructura usada: LOOP + EXIT)

◆ Variables

sql

```
v_acumulado NUMERIC(10,2) := 0;  
v_cliente RECORD;
```

- v_acumulado controla presupuesto.
 - v_cliente almacena cliente aleatorio.
-

◆ LOOP infinito

sql

LOOP

LOOP no tiene condición.

Debe romperse con EXIT.

♦ Cliente aleatorio

sql

```
ORDER BY random()
LIMIT 1;
```

- random() genera número aleatorio.
 - Ordena aleatoriamente.
 - Devuelve 1 cliente.
-

♦ Suma cupón

sql

```
v_acumulado := v_acumulado + 15;
```

Cada iteración añade 15\$.

♦ Mensaje

Muestra el ID y total acumulado.

♦ EXIT WHEN

sql

```
EXIT WHEN v_acumulado > 500;
```

Cuando supera 500:

→ sale del bucle.

Sin esto:

→ bucle infinito.

Resumen estructural

Objeto	Tipo	Estructura usada
fn_nivel_cliente	Función	IF + CASE
pr_resumen_inventario	Procedimiento	FOR
fn_calculo_recargo_progresivo	Función	WHILE
pr_generar_cupones_campaña	Procedimiento	LOOP + EXIT

Si quieras, ahora puedo prepararte una versión redactada formalmente tipo **memoria explicativa para entregar en PDF** con lenguaje técnico académico para 2º ASIR.

Tú dijiste:

dvdrental=# SELECT fn_calculo_recargo_progresivo(300); ERROR: numeric field overflow DETALLE: A field with precision 10, scale 2 must round to an absolute value less than 10^8. CONTEXTO: PL/pgSQL function fn_calculo_recargo_progresivo(integer) line 7 at assignment dvdrental=#

ChatGPT Plus

Perfecto que hayas probado ese caso, porque **esto es justo el tipo de detalle fino que suele caer en exámenes**.

Vamos a explicarlo **con lupa** y luego te doy la **solución correcta y justificable**.

¿Qué está pasando exactamente?

Has ejecutado:

sql

```
SELECT fn_calculo_recargo_progresivo(300);
```

Y PostgreSQL responde:

vbnet

```
ERROR: numeric field overflow
DETALLE: A field with precision 10, scale 2 must round to an absolute value
less than 10^8.
```

Causa real del error (no es un bug)

La línea problemática es esta:

sql

```
v_total_recargo NUMERIC(10,2) := 1.0;
```

¿Qué significa NUMERIC(10,2)?

- 10 → número total de dígitos (incluye decimales)
- 2 → decimales
- Máximo valor permitido:

scss

```
99999999.99 (10^8 - 0.01)
```

Qué hace tu función matemáticamente

Cada día haces:

sql

```
v_total_recargo := v_total_recargo * 1.10;
```

Eso es **crecimiento exponencial**:

$$1.10^{300} \approx 1.94 \times 10^{12}$$
$$1.10^{300} \approx 1.94 \times 10^{12}$$

Eso es:

powershell

```
1.940.000.000.000 $
```

👉 MUCHÍSIMO mayor que 99.999.999,99

Por tanto:

- ✓ PostgreSQL **hace bien en lanzar el error**
- ✗ No es un fallo de tu código
- ✓ Es una limitación del tipo de dato elegido

Por qué ocurre exactamente en esa línea

El error dice:

arduino

```
line 7 at assignment
```

Corresponde a:

sql

```
v_total_recargo := v_total_recargo * 1.10;
```

Porque ahí PostgreSQL intenta guardar un número **demasiado grande** en un NUMERIC(10,2).

SOLUCIÓN CORRECTA (Y EXAMINABLE)

- ◆ **Opción 1 – Aumentar precisión (RECOMENDADA)**

Esta es la mejor solución técnica y académica.

✓ Cambiar:

sql

```
v_total_recargo NUMERIC(10,2)
```

✓ Por:

sql

```
v_total_recargo NUMERIC(20,2)
```

Esto permite valores de hasta:

```
9999999999999999.99
```

Función corregida

sql

```
CREATE OR REPLACE FUNCTION fn_calculo_recargo_progresivo(p_dias_retraso
INT)
RETURNS NUMERIC(20,2)
AS
$#
DECLARE
```

```

v_total_recargo NUMERIC(20,2) := 1.0;
v_dia INT := 1;
BEGIN
    WHILE v_dia <= p_dias_retraso LOOP
        v_total_recargo := v_total_recargo * 1.10;
        v_dia := v_dia + 1;
    END LOOP;

    RETURN ROUND(v_total_recargo, 2);
END;
$#
LANGUAGE plpgsql;

```

✓ Ahora:

sql

```
SELECT fn_calculo_recargo_progresivo(300);
```

Funciona perfectamente.

- ◆ **Opción 2 – Limitar días (NO ideal, pero válida)**

sql

```

IF p_dias_retraso > 100 THEN
    RAISE EXCEPTION 'Demasiados días de retraso';
END IF;

```

△ Esto **recorta funcionalidad**

△ No es tan elegante

△ Menos nota

- ◆ **Opción 3 – Usar FLOAT (NO recomendado)**

sql

```
v_total_recargo DOUBLE PRECISION
```

✗ Pérdida de precisión

✗ No apropiado para dinero

✗ Penalizable académicamente

CONCLUSIÓN PARA DEFENDERLO ORALMENTE

El error se produce porque el recargo crece de forma exponencial y supera el rango permitido por el tipo NUMERIC(10,2). PostgreSQL lanza una excepción de overflow al intentar asignar un valor que excede la precisión definida. La solución correcta es aumentar la precisión del tipo NUMERIC para permitir valores grandes sin perder exactitud.

Si quieras, te redacto **ese párrafo exacto para incluirlo en la memoria** y que el profe vea que sabes lo que haces 😊

2/2

Tú dijiste:

```
como hago un script.sql de esto? CREATE OR REPLACE FUNCTION fn_nivel_cliente(p_customer_id
INT) RETURNS VARCHAR AS -- Crea una función llamada fn_nivel_cliente(p_customer_id INT) que
devuelva un texto (VARCHAR) con la categoría del cliente: $ DECLARE -- Variables v_total
NUMERIC(10,2); v_nivel VARCHAR(50); BEGIN -- 1. Calcula el total gastado por el cliente sumando la
columna amount de la tabla payment. Lo guardamos en v_total. p_customer_id viene cuando llamamos
a la función SELECT SUM(amount) INTO v_total FROM payment WHERE customer_id =
p_customer_id; -- 2. Usa una estructura IF para verificar si el cliente existe (si el total es NULL, lanzar un
aviso). IF v_total IS NULL THEN RAISE NOTICE 'El cliente con ID % no existe.', p_customer_id;
RETURN NULL; END IF; -- 3. La estructura CASE CASE WHEN v_total > 150 THEN v_nivel := 'VIP
Platinum'; WHEN v_total BETWEEN 100 AND 150 THEN v_nivel := 'Gold'; WHEN v_total BETWEEN 50
AND 99.99 THEN v_nivel := 'Silver'; ELSE v_nivel := 'Bronze'; END CASE; RETURN v_nivel; END; $ LANGUAGE plpgsql; \||||||| CREATE OR REPLACE PROCEDURE pr_resumen_inventario() AS $ DECLAR
RECORD: Tipo dinámico que puede almacenar una fila completa. Almacena (category_id y
name). v_categoria RECORD; v_total_copias INT; BEGIN -- FOR recorriendo todas las categorías FOR
v_categoria IN SELECT category_id, name FROM category LOOP -- Contamos copias por categoría
SELECT COUNT(i.inventory_id) INTO v_total_copias FROM inventory i JOIN film f ON i.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id WHERE fc.category_id = v_categoria.category_id; RAISE
NOTICE 'Categoría: %, Total Copias: %', v_categoria.name, v_total_copias; END LOOP; END; $ LANGUAGE plpgsql; \||||||| CREATE OR REPLACE FUNCTION
fn_calculo_recargo_progresivo(p_dias_retraso INT) RETURNS NUMERIC(10,2) AS $ DECLAR
v_total_recargo NUMERIC(20,2) := 1.0; v_dia INT := 1; BEGIN WHILE v_dia <= p_dias_retraso LOOP
v_total_recargo := v_total_recargo * 1.10; v_dia := v_dia + 1; END LOOP; RETURN
ROUND(v_total_recargo, 2); END; $ LANGUAGE plpgsql; \||||||| CREATE OR REPLACE
PROCEDURE pr_generar_cupones_campaña() AS $ DECLAR v_acumulado NUMERIC(10,2) := 0;
v_cliente RECORD; BEGIN LOOP -- Elegimos cliente aleatorio SELECT customer_id INTO v_cliente
FROM customer ORDER BY random() LIMIT 1; -- Sumamos cupón simbólico de 15$ v_acumulado :=
v_acumulado + 15; RAISE NOTICE 'Cupón asignado a cliente ID: %. Total acumulado: %',
```

```
v_cliente.customer_id, v_acumulado; -- Salimos cuando superemos 500$ EXIT WHEN v_acumulado > 500; END LOOP; END; $ LANGUAGE plpgsql;
```

ChatGPT Plus

Perfecto, Cristóbal. Te explico **exactamente cómo convertir todo eso en un script.sql correcto y profesional**, paso a paso, como lo pedirían en 2º ASIR.

🎯 ¿Qué es realmente un script .sql?

Un script .sql es simplemente:

Un archivo de texto plano que contiene sentencias SQL listas para ejecutarse automáticamente.

No lleva nada especial.

No necesita estructura extra.

Solo debe estar bien ordenado y sin errores de sintaxis.

✓ Cómo hacerlo correctamente (forma profesional)

◆ 1 Crear el archivo

En Linux:

bash

```
nano actividad2_toolkit.sql
```

En Windows (desde tu carpeta del proyecto):

- Clic derecho → Nuevo → Documento de texto
- Cambiar nombre a:

pgsql

```
actividad2_toolkit.sql
```

⚠ Importante: que no quede como .sql.txt

◆ 2 Qué debes incluir dentro del script

Un script profesional debería incluir:

1. Comentario identificativo
2. Conexión a la base de datos
3. Las funciones/procedimientos
4. (Opcional pero recomendable) pruebas al final

SCRIPT COMPLETO BIEN FORMATEADO

Copia exactamente esto dentro de actividad2_toolkit.sql:

sql

```
-- =====
-- ACTIVIDAD 2 - ESTRUCTURAS DE CONTROL
-- Alumno: Cristóbal Suárez Abad
-- Base de datos: dvdrental
-- =====

\c dvdrental;

-- =====
-- 1. FUNCIÓN: Clasificación Dinámica de Clientes
-- Estructuras usadas: IF + CASE
-- =====

CREATE OR REPLACE FUNCTION fn_nivel_cliente(p_customer_id INT)
RETURNS VARCHAR AS
$ 
DECLARE
    v_total NUMERIC(10,2);
    v_nivel VARCHAR(50);
BEGIN
    SELECT SUM(amount)
    INTO v_total
    FROM payment
    WHERE customer_id = p_customer_id;

    IF v_total IS NULL THEN
        RAISE NOTICE 'El cliente con ID % no existe.', p_customer_id;
    END IF;
END;
```

```

        RETURN NULL;
    END IF;

CASE
    WHEN v_total > 150 THEN
        v_nivel := 'VIP Platinum';
    WHEN v_total BETWEEN 100 AND 150 THEN
        v_nivel := 'Gold';
    WHEN v_total BETWEEN 50 AND 99.99 THEN
        v_nivel := 'Silver';
    ELSE
        v_nivel := 'Bronze';
END CASE;

RETURN v_nivel;
END;
$#
LANGUAGE plpgsql;

-- =====
-- 2. PROCEDIMIENTO: Informe de Inventario por Categoría
-- Estructura usada: FOR
-- =====

CREATE OR REPLACE PROCEDURE pr_resumen_inventario()
AS
$#
DECLARE
    v_categoria RECORD;
    v_total_copias INT;
BEGIN
    FOR v_categoria IN SELECT category_id, name FROM category
    LOOP
        SELECT COUNT(i.inventory_id)
        INTO v_total_copias
        FROM inventory i
        JOIN film f ON i.film_id = f.film_id
        JOIN film_category fc ON f.film_id = fc.film_id
        WHERE fc.category_id = v_categoria.category_id;

        RAISE NOTICE 'Categoría: %, Total Copias: %',
                      v_categoria.name, v_total_copias;
    END LOOP;

```

```

END;
$ 
LANGUAGE plpgsql;

-- =====
-- 3. FUNCIÓN: Cálculo de Recargo Progresivo
-- Estructura usada: WHILE
-- =====

CREATE OR REPLACE FUNCTION fn_calculo_recargo_progresivo(p_dias_retraso
INT)
RETURNS NUMERIC(20,2)
AS
$ 
DECLARE
    v_total_recargo NUMERIC(20,2) := 1.0;
    v_dia INT := 1;
BEGIN
    WHILE v_dia <= p_dias_retraso LOOP
        v_total_recargo := v_total_recargo * 1.10;
        v_dia := v_dia + 1;
    END LOOP;

    RETURN ROUND(v_total_recargo, 2);
END;
$ 
LANGUAGE plpgsql;

-- =====
-- 4. PROCEDIMIENTO: Generador de Cupones
-- Estructura usada: LOOP + EXIT
-- =====

CREATE OR REPLACE PROCEDURE pr_generar_cupones_campaña()
AS
$ 
DECLARE
    v_acumulado NUMERIC(10,2) := 0;
    v_cliente RECORD;
BEGIN
    LOOP
        SELECT customer_id

```

```

        INTO v_cliente
        FROM customer
        ORDER BY random()
        LIMIT 1;

        v_acumulado := v_acumulado + 15;

        RAISE NOTICE 'Cupón asignado a cliente ID: %. Total acumulado: %',
                      v_cliente.customer_id, v_acumulado;

        EXIT WHEN v_acumulado > 500;
    END LOOP;
END;
$-
LANGUAGE plpgsql;

-- =====
-- PRUEBAS
-- =====

SELECT first_name, fn_nivel_cliente(customer_id)
FROM customer
LIMIT 5;

CALL pr_resumen_inventario();

SELECT fn_calculo_recargo_progresivo(10);

CALL pr_generar_cupones_campaña();

```

3 Cómo ejecutarlo

Desde fuera de PostgreSQL:

bash

```
psql -U postgres -h localhost -f actividad2_toolkit.sql dvdrental
```

O si estás como usuario postgres:

bash

```
sudo -u postgres psql -f actividad2_toolkit.sql dvdrental
```

¿Por qué esta estructura es profesional?

- ✓ Está documentado
 - ✓ Está separado por bloques
 - ✓ Es reutilizable
 - ✓ Incluye pruebas
 - ✓ Se puede ejecutar en cualquier servidor PostgreSQL
-

Si quieras, ahora te explico cómo estructurarlo exactamente como memoria formal para sacar sobresaliente 😊

ChatGPT puede cometer errores. Considera verificar la información importante. Ver [preferencias de cookies](#).