ACTIVIDAD 2 - ESTRUCTURAS DE CONTROL

Cristóbal Suárez Abad

ADMINISTRACIÓN DE SISTEMAS GESTORES DE BASES DE DATOS - 2º ASIR

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:

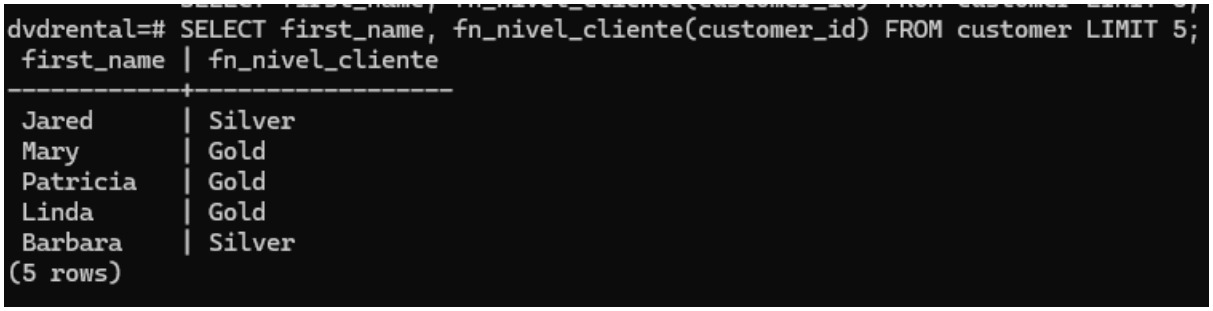
```
SQL
-- Prueba IF/CASE

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"



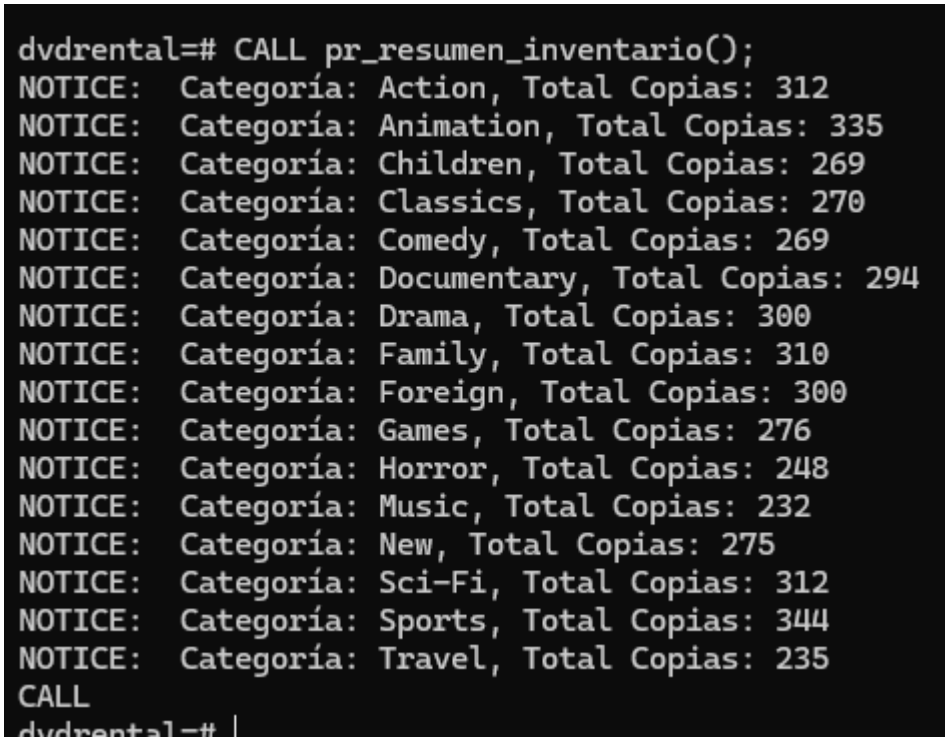
```
CREATE OR REPLACE FUNCTION fn_nivel_cliente(p_customer_id INT)
RETURNS VARCHAR AS
-- Crea una función llamada fn_nivel_cliente(p_customer_id INT) que devuelva un texto (VARCHAR) con la categoría del cliente:
$$
DECLARE
        -- Variables
   v_total NUMERIC(10,2);
   v_nivel VARCHAR(50);
BEGIN
   -- 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;
```

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]"*.



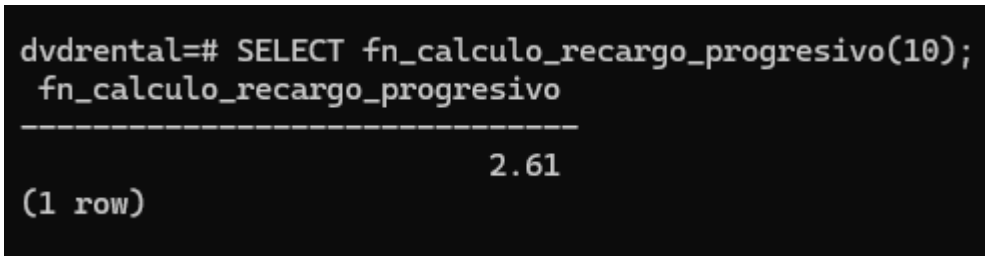
```
CREATE OR REPLACE PROCEDURE pr_resumen_inventario()
AS
$$
DECLARE
        -- RECORD: Tipo dinámico que puede almacenar una fila completa. Almacena (category_id y name).
   v_categoria RECORD;
   v_total_copias INT;
BEGIN
   -- FOR recorriendo todas las categorías
   FOR v_categoria IN SELECT category_id, 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;
```

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.



```
CREATE OR REPLACE FUNCTION fn_calculo_recargo_progresivo(p_dias_retraso INT)
RETURNS NUMERIC(10,2)
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;
```

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$.

```
dvdrental=# CALL pr_generar_cupones_campaña();
NOTICE:  Cupón asignado a cliente ID: 18. Total acumulado: 15.00
NOTICE:  Cupón asignado a cliente ID: 565. Total acumulado: 30.00
NOTICE:  Cupón asignado a cliente ID: 216. Total acumulado: 45.00
NOTICE:  Cupón asignado a cliente ID: 190. Total acumulado: 60.00
NOTICE:  Cupón asignado a cliente ID: 66. Total acumulado: 75.00
NOTICE:  Cupón asignado a cliente ID: 544. Total acumulado: 90.00
NOTICE:  Cupón asignado a cliente ID: 592. Total acumulado: 105.00
NOTICE:  Cupón asignado a cliente ID: 480. Total acumulado: 120.00
NOTICE:  Cupón asignado a cliente ID: 244. Total acumulado: 135.00
NOTICE:  Cupón asignado a cliente ID: 530. Total acumulado: 150.00
NOTICE:  Cupón asignado a cliente ID: 364. Total acumulado: 165.00
NOTICE:  Cupón asignado a cliente ID: 147. Total acumulado: 180.00
NOTICE:  Cupón asignado a cliente ID: 366. Total acumulado: 195.00
NOTICE:  Cupón asignado a cliente ID: 363. Total acumulado: 210.00
NOTICE:  Cupón asignado a cliente ID: 90. Total acumulado: 225.00
NOTICE:  Cupón asignado a cliente ID: 208. Total acumulado: 240.00
NOTICE:  Cupón asignado a cliente ID: 318. Total acumulado: 255.00
NOTICE:  Cupón asignado a cliente ID: 404. Total acumulado: 270.00
NOTICE:  Cupón asignado a cliente ID: 277. Total acumulado: 285.00
NOTICE:  Cupón asignado a cliente ID: 169. Total acumulado: 300.00
NOTICE:  Cupón asignado a cliente ID: 25. Total acumulado: 315.00
NOTICE:  Cupón asignado a cliente ID: 480. Total acumulado: 330.00
NOTICE:  Cupón asignado a cliente ID: 194. Total acumulado: 345.00
```

```
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;
```

SCRIPT

Archivo en “**actividad2_ud6.sql**”.

Ejecutar desde fuera de PostgreSQL:

psql -U postgres -h localhost -f actividad2_ud6.sql dvdrental