ACTIVIDAD 1 - BLOQUES, VARIABLES, FUNCIONES Y PROCEDIMIENTOS

Cristóbal Suárez Abad

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

Índice

Entregables

- Script `.sql` con todos los bloques, funciones y procedimientos.
- Captura de pantalla de la terminal o consola de mensajes mostrando la salida de los `RAISE NOTICE`.

Paso 0: Preparación del Entorno

Antes de empezar con la lógica, necesitamos datos reales. Vamos a utilizar la base de datos de ejemplo **dvdrental**.

1. Accede a PostgreSQL Sample Database.
2. Descarga el archivo `.tar` o el script SQL de la base de datos.

Nos descargamos el archivo .zip y de ahí obtenemos el .tar
https://neon.com/postgresqltutorial/dvdrental.zip

Lo subimos al servidor:
scp dvdrental.tar cristobal@10.2.7.101:/home/cristobal

```
root@ubuntumysqlsuarez:/home/cristobal# ls -l | grep dvd
-rw-rw-r-- 1 cristobal cristobal   2835456 feb  6 16:38 dvdrental.tar
root@ubuntumysqlsuarez:/home/cristobal#
```

3. Cárgalo en tu instancia de PostgreSQL (puedes usar `pgAdmin` o la terminal con `pg_restore`).

Siguiendo la guía: https://neon.com/postgresql/postgresql-getting-started/load-postgresql-sample-database

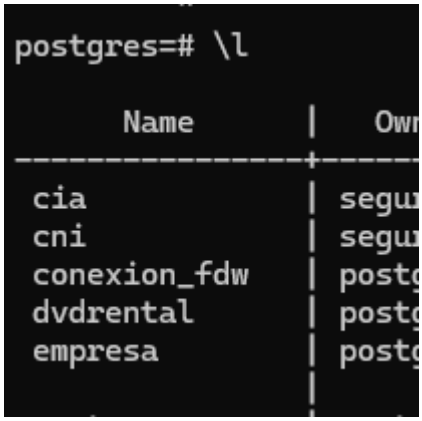
- Entramos en postgresql:

Desde cualquier usuario: sudo -i -u postgres psql
Desde el usuario UNIX “postgres”: psql -U postgres

- Creamos base de datos “dvdrental”:

CREATE DATABASE dvdrental;

```
postgres=# CREATE DATABASE dvdrental;
CREATE DATABASE
postgres=#
```



Ahora, desde fuera de postgresql:

- Si lo hacemos con cualquier usuario de UNIX que no sea postgres

pg_restore -U postgres -h localhost -d dvdrental dvdrental.tar

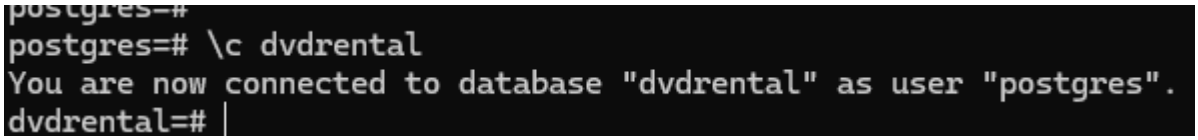
- Si lo queremos hacer con el usuario “postgres”:

sudo -u postgres pg_restore -d dvdrental dvdrental.tar

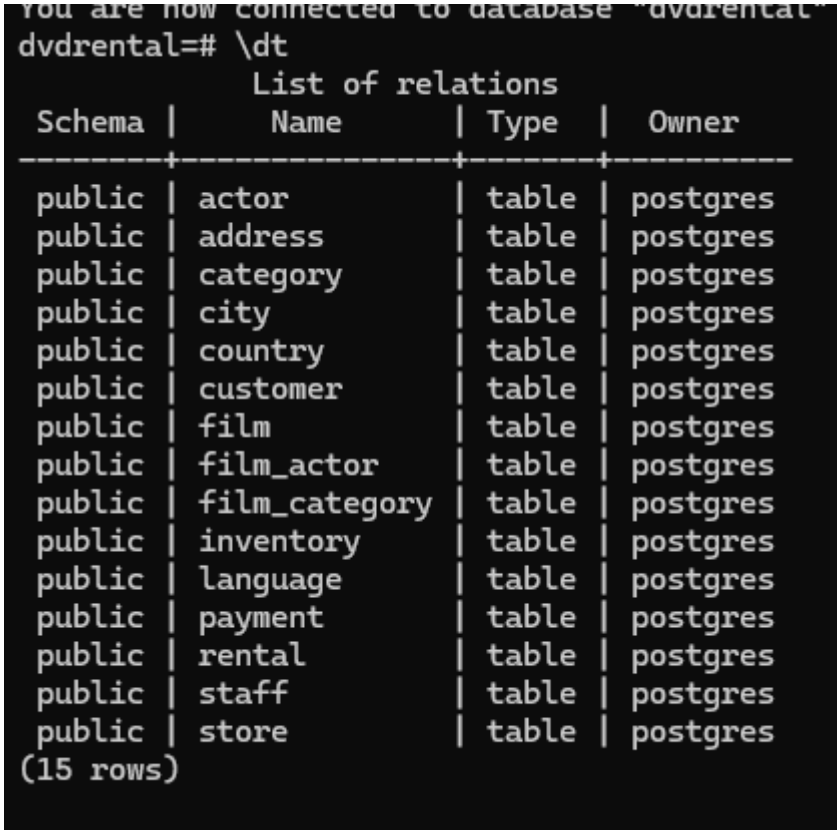
4. **Objetivo:** Debes tener disponibles tablas como `customer`, `film`, `payment` y `rental`.

Volvemos a entrar postgresql: **sudo -i -u postgres psql**

Nos conectamos a la base de datos: **\c dvdrental**



Mostramos todas las tablas: **\dt**



Fase 1: Bloques Anónimos y Ámbito (30 min)

PL/pgSQL organiza el código en bloques para facilitar el desarrollo y la lectura. Usaremos la instrucción `DO` para ejecutar código sin guardarlo en el servidor.

Tarea:

- Crea un bloque anónimo que declare una **constante** con el nombre de la academia y una variable para el nombre del alumno.
- Dentro, crea un **subbloque** que declare una variable con el mismo nombre que la anterior (aplicando *shadowing*).
- Usa `RAISE NOTICE` para imprimir los valores y comprobar cómo cambia el ámbito de las variables.

```
DO $$
DECLARE
   -- Bloque original
   v_academia CONSTANT TEXT := 'IES Delgado Hernández';
   v_alumno TEXT := 'Cristobal';
BEGIN
   RAISE NOTICE '--- Bloque original ---';
   RAISE NOTICE 'Academia: %, Alumno: %', v_academia, v_alumno;

   DECLARE
      -- El subbloque
      v_alumno TEXT := 'Andrés';
      v_academia TEXT := 'IES La Palma';
   BEGIN
      RAISE NOTICE '--- Subbloque ---';
      RAISE NOTICE 'Academia (Subbloque): %', v_academia;
      RAISE NOTICE 'Alumno (Subbloque): %', v_alumno;
   END;

   RAISE NOTICE '--- Otra vez el bloque original ---';
   RAISE NOTICE 'Academia: %, Alumno: %', v_academia, v_alumno;
END $$;
```

Desde fuera de postgresql creamos un script sql: fase1.sql

```
  GNU nano 7.2
DO $$
DECLARE
    -- Bloque original
    v_academia CONSTANT TEXT := 'IES Delgado Hernández';
    v_alumno TEXT := 'Cristobal';
BEGIN
    RAISE NOTICE '--- Bloque original ---';
    RAISE NOTICE 'Academia: %, Alumno: %', v_academia, v_alumno;

    DECLARE
        -- El subbloque
        v_alumno TEXT := 'Andrés';
        v_academia CONSTANT TEXT := 'IES La Palma';
    BEGIN
        RAISE NOTICE '--- Subbloque ---';
        RAISE NOTICE 'Academia (Subbloque): %', v_academia;
        RAISE NOTICE 'Alumno (Subbloque): %', v_alumno;
    END;

    RAISE NOTICE '--- Otra vez el bloque original ---';
    RAISE NOTICE 'Academia: %, Alumno: %', v_academia, v_alumno;
END $$;
```

Y luego, también desde fuera, lo ejecutamos así:

psql -U postgres -h localhost -f fase1.sql dvdrental

```
root@ubuntumysqlsuarez:/home/cristobal# psql -U postgres -h localhost -f fase1.sql dvdrental
Password for user postgres:
psql:fase1.sql:22: NOTICE:  --- Bloque original ---
psql:fase1.sql:22: NOTICE:  Academia: IES Delgado Hernández, Alumno: Cristobal
psql:fase1.sql:22: NOTICE:  --- Subbloque ---
psql:fase1.sql:22: NOTICE:  Academia (Subbloque): IES La Palma
psql:fase1.sql:22: NOTICE:  Alumno (Subbloque): Andrés
psql:fase1.sql:22: NOTICE:  --- Otra vez el bloque original ---
psql:fase1.sql:22: NOTICE:  Academia: IES Delgado Hernández, Alumno: Cristobal
DO
```

También podemos pegar el script directamente en el shell de postgresql:

```
dvdrental=# DO $$
DECLARE
    -- Bloque original
    v_academia CONSTANT TEXT := 'IES Delgado Hernández';
    v_alumno TEXT := 'Cristobal';
BEGIN
    RAISE NOTICE '--- Bloque original ---';
    RAISE NOTICE 'Academia: %, Alumno: %', v_academia, v_alumno;

    DECLARE
        -- El subbloque
        v_alumno TEXT := 'Andrés';
        v_academia CONSTANT TEXT := 'IES La Palma';
    BEGIN
        RAISE NOTICE '--- Subbloque ---';
        RAISE NOTICE 'Academia (Subbloque): %', v_academia;
        RAISE NOTICE 'Alumno (Subbloque): %', v_alumno;
    END;

    RAISE NOTICE '--- Otra vez el bloque original ---';
    RAISE NOTICE 'Academia: %, Alumno: %', v_academia, v_alumno;
END $$;
NOTICE:  --- Bloque original ---
NOTICE:  Academia: IES Delgado Hernández, Alumno: Cristobal
NOTICE:  --- Subbloque ---
NOTICE:  Academia (Subbloque): IES La Palma
NOTICE:  Alumno (Subbloque): Andrés
NOTICE:  --- Otra vez el bloque original ---
NOTICE:  Academia: IES Delgado Hernández, Alumno: Cristobal
DO
```

Fase 2: Recuperación de Datos con SELECT INTO

La sentencia `SELECT INTO` es fundamental para asignar resultados de una consulta a variables.

Tarea:

1. **Uso de %TYPE:** Declara dos variables para almacenar el título (`title`) y el precio de alquiler (`rental_rate`) de la tabla `film`, usando `%TYPE` para heredar sus tipos de datos automáticamente.
2. **Cálculo:** Recupera los datos de la película con `film_id = 50`.
3. **Salida:** Muestra un mensaje que diga: *"La película [titulo] tiene un coste de [precio]€"*.
4. **Cálculo complejo:** Realiza un bloque que calcule el total recaudado por el cliente con ID 10 sumando todos sus pagos en `payment` y guárdalo en una variable `DECIMAL`.

Para saber la estructura de cada tabla y así conocer el nombre de las columnas, usamos "**\d+ nombre_tabla**".

```
DO $$
DECLARE
   -- Variables con TYPE
   v_titulo film.title%TYPE;
   v_precio film.rental_rate%TYPE;
   -- Variable para cálculo complejo
   v_total_recaudado DECIMAL(10,2);
BEGIN
   -- Primer cálculo: película con ID 50
   SELECT title, rental_rate INTO v_titulo, v_precio
   FROM film
   WHERE film_id = 50;

   RAISE NOTICE 'La película % tiene un coste de %€', v_titulo, v_precio;

   -- Tarea 2: Cálculo para el cliente ID 10
   SELECT SUM(amount) INTO v_total_recaudado
   FROM payment
   WHERE customer_id = 10;

   RAISE NOTICE 'Total recaudado por el cliente 10: %€', v_total_recaudado;
END $$;
```

```
root@ubuntumysqlsuarez:/home/cristobal# psql -U postgres -h localhost -f fase2.sql dvdrental
Password for user postgres:
psql:fase2.sql:22: NOTICE:  La película Baked Cleopatra tiene un coste de 2.99€
psql:fase2.sql:22: NOTICE:  Total recaudado por el cliente 10: 94.76€
DO
root@ubuntumysqlsuarez:/home/cristobal#
```

Fase 3: Automatización con Funciones

Una función es código reutilizable que **siempre devuelve un valor**.

Tarea:

- Crea una función llamada `get_customer_status(p_id INT)` que devuelva un texto (`TEXT`).
- La función debe contar cuántos alquileres (`rental`) ha realizado el cliente.
- **Prueba:** Ejecuta la función con un `SELECT get_customer_status(7);`.

"returns text as $$": https://dba.stackexchange.com/questions/175192/writing-functions-in-postgres-that-returns-strings-with-dots-in-reverse-like-jav

```
CREATE OR REPLACE FUNCTION get_customer_status(p_id INT)
RETURNS TEXT AS $$
DECLARE
   v_conteo INT;
   v_resultado TEXT;
BEGIN
   SELECT COUNT(*) INTO v_conteo
   FROM rental
   WHERE customer_id = p_id;

   v_resultado := 'El cliente con ID ' || p_id || ' ha realizado ' || v_conteo || ' alquileres.';

   RETURN v_resultado;
END;
$$
LANGUAGE plpgsql;
```

```
dvdrental=# CREATE OR REPLACE FUNCTION get_customer_status(p_id INT)
RETURNS TEXT AS $$
DECLARE
    v_conteo INT;
    v_resultado TEXT;
BEGIN
    SELECT COUNT(*) INTO v_conteo
    FROM rental
    WHERE customer_id = p_id;

    v_resultado := 'El cliente con ID ' || p_id || ' ha realizado ' || v_conteo || ' alquileres.';

    RETURN v_resultado;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
```

```
dvdrental=# SELECT get_customer_status(7);
                get_customer_status
-------------------------------------------------
 El cliente con ID 7 ha realizado 33 alquileres.
(1 row)
```

Fase 4: Procedimientos y Mantenimiento

A diferencia de las funciones, los procedimientos se usan para realizar acciones (como actualizaciones) y no devuelven valores directos. Se ejecutan con la instrucción `CALL`.

Tarea:

- Crea un procedimiento llamado `apply_late_fee` que reciba un `p_rental_id` y un `p_amount`.
- El procedimiento debe actualizar la tabla `payment` sumando ese recargo al pago correspondiente a ese alquiler.
- **Prueba:** Ejecuta el procedimiento para un alquiler real de la base de datos.

```
CREATE OR REPLACE PROCEDURE apply_late_fee(p_rental_id INT, p_amount
DECIMAL)
AS
$$
BEGIN
   -- Actualizamos el pago asociado a ese alquiler
   UPDATE payment
   SET amount = amount + p_amount
   WHERE rental_id = p_rental_id;

   RAISE NOTICE 'Recargo de %€ aplicado al alquiler ID %', p_amount, p_rental_id;
END;
$$
LANGUAGE plpgsql;
```

- Elegimos un “rental_id” al azar.

select * from payment where rental_id = 1201;

```
dvdrental=# select * from payment where rental_id = 1201;
 payment_id | customer_id | staff_id | rental_id | amount |        payment_date
------------+-------------+----------+-----------+--------+----------------------------
      19289 |         209 |        2 |      1201 |   4.99 | 2007-02-15 00:34:54.996577
(1 row)
```

CALL apply_late_fee (1201, 117.69);

```
dvdrental=# CALL apply_late_fee (1201, 117.69);
NOTICE:  Recargo de 117.69€ aplicado al alquiler ID 1201
CALL
dvdrental=# select * from payment where rental_id = 1201;
 payment_id | customer_id | staff_id | rental_id | amount |        payment_date
------------+-------------+----------+-----------+--------+----------------------------
      19289 |         209 |        2 |      1201 | 122.68 | 2007-02-15 00:34:54.996577
(1 row)

dvdrental=#
```