Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica

Escuela de Computación

Base de datos II
IC-4302

Tema: Proyecto Replicación

Grupo 20 - Integrantes:
Araya Nash Hengerlyn D Angiely

Prof.:
Alberto Shum Chan

November 7, 2024

Contents

1. Base de datos para alquiler de películas

1.1 Funciones y procedimientos del sistema transaccional:

- **Insertar un nuevo cliente:**

Para la funcionalidad se trata de insertar un nuevo cliente en la base de datos, para ello se desarrolló un procedimiento almacenado en donde se realiza la acción mencionada anteriormente.

Atributos: Son todos aquellos datos de entrada o salida del procedimiento almacenado. En este caso solo son datos de entrada.

Atributo	Explicación
in_first_name	Entrada de primer nombre
in_last_name	Entrada de último nombre
in_email	Entrada de correo electrónico
in_address_id	Identificador de dirección
in_store_id	Identificador de tienda
in_active_bool	Booleano de activo
in_create_date	Fecha de creación
in_last_update	Fecha de última modificación

in_active	Entero de activo

El procedimiento almacenado consiste en ingresar los respectivos atributos que fueron mencionados anteriormente. Se debe tomar en cuenta lo siguiente: Durante la inserción se asume que el identificador de la tienda y la dirección es de datos que ya existen en la base de datos, de lo contrario no se realizará la inserción. Como cualquier inserción se utiliza la el elemento de la línea 17 a la línea 24, si se realiza de manera correcta, levanta una noticia avisando con el nombre del nuevo cliente insertado. A continuación adjunta el código correspondiente:

```
1  CREATE OR REPLACE PROCEDURE insertar nuevo cliente(
2      in_first_name VARCHAR,
3      in_last_name VARCHAR,
4      in_email VARCHAR,
5      in_address_id INT,
6      in_store_id INT,
7      in_active_bool BOOLEAN,
8      in_create_date DATE,
9      in_last_update TIMESTAMP WITHOUT TIME ZONE,
10     in_active INT
11 )
12 LANGUAGE plpgsql
13 SECURITY DEFINER
14 AS $$
15 BEGIN
16     -- Insertar cliente
17     INSERT INTO public.customer(
18         store_id, first_name, last_name, email, address_id, activebool, create_date,
19 last_update, active
20     )
21     VALUES (
22         in_store_id, in_first_name, in_last_name, in_email,
23         in_address_id, in_active_bool, in_create_date, in_last_update, in_active
24     );
25     RAISE NOTICE 'Nuevo cliente registrado correctamente, nombre: % %', in first_name,
26     in_last_name;
27
28 EXCEPTION
29     WHEN unique_violation THEN
30         RAISE NOTICE 'El cliente ya existe con el email %.', un email;
31     WHEN foreign_key_violation THEN
32         RAISE NOTICE 'El store_id o address_id no son válidos.';
33     WHEN others THEN
34         RAISE NOTICE 'Ocurrió un error: %', SQLERRM;
35 END;
```

- **Registrar un alquiler:**

Para el caso del registro de un alquiler, se desarrolló un procedimiento almacenado en la base de datos en donde se realiza la acción mencionada anteriormente. Note que se asume que el registro de inventario y de staff ya existen en sus respectivas tablas de la base de datos.

Atributos: Son todos aquellos datos de entrada o salida del procedimiento almacenado. En este caso solo son datos de entrada.

Atributo	Explicación
in_rental_date	Fecha de renta de película
in_inventory_id	Identificador de inventario
in_customer_id	Identificador de cliente
in_return_date	Fecha de regreso de película
in_staff_id	Identificador de empleado
in_last_update	Fecha de última modificación

Parecido al caso del procedimiento almacenado anteriormente, primero mediante un comando SQL representado en la línea 14 a la línea 20 se insertan los datos correspondientes, en caso que ocurra un error al momento de hacer la acción, salta una noticia del error presentado y su respectiva causa.

```
 1 CREATE OR REPLACE PROCEDURE registrar_alquiler(
 2     in_rental_date TIMESTAMP WITHOUT TIME ZONE,
 3     in_inventory_id INT,
 4     in_customer_id SMALLINT,
 5     in_return_date TIMESTAMP WITHOUT TIME ZONE,
 6     in_staff_id SMALLINT,
 7     in_last_update TIMESTAMP WITHOUT TIME ZONE
 8 )
 9 LANGUAGE plpgsql
10 SECURITY DEFINER
11 AS $$
12 BEGIN
13     -- Registra un alquiler en la tabla de alquileres
14     INSERT INTO rental (
```

```
        rental_date, inventory_id, customer_id, return_date, staff_id,
last_update
    )
    VALUES (
        in_rental_date, in_inventory_id, in_customer_id,
        in_return_date, in_staff_id, in_last_update
    );

    RAISE NOTICE 'Alquiler registrado correctamente';


EXCEPTION
    WHEN unique_violation THEN
        RAISE NOTICE 'El ID de alquiler ya existe.';

    WHEN foreign_key_violation THEN
        RAISE NOTICE 'El inventory_id, customer_id o staff_id no son válidos.';

    WHEN others THEN
        RAISE NOTICE 'Ocurrió un error: %', SQLERRM;

END;
$$;
```

- **Registrar una devolución**

Para el caso de la funcionalidad de registrar una devolución, se debe primero entender qué es la devolución en el contexto de la base de datos. En este caso hace referencia a la acción de registrar la devolución de una película por parte del cliente.

Atributos: Son todos aquellos datos de entrada o salida del procedimiento almacenado. En este caso solo son datos de entrada. A continuación se presentan todos los datos de entrada que utiliza este procedimiento almacenado:

Atributo	Explicación
InRental_id	Identificador de renta realizada
InReturn_date	Fecha de devolución.

```
CREATE OR REPLACE PROCEDURE registrar_devolucion(InRental_id INT, InReturn_date
DATE)
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
DECLARE
```

```
    IdInventory INT;
BEGIN
    -- Actualizar la fecha de devolución en la tabla rental
    UPDATE rental
    SET return_date = InReturn_date
    WHERE rental_id = InRental_id;

    -- Obtener el inventory id asociado con el rental id
    SELECT inventory_id INTO IdInventory
    FROM rental
    WHERE rental_id = InRental_id;

    -- Actualizar la disponibilidad en la tabla inventory
    UPDATE inventory
    SET last_update= NOW()
    WHERE inventory_id = IdInventory;

    -- Confirmar la transacción
    RAISE NOTICE 'Devolución registrada exitosamente para rental id: %',InTernal
id;

EXCEPTION
    WHEN OTHERS THEN
     -- Revertir la transacción en caso de error

         RAISE NOTICE 'Error al registrar la devolución: %', SQLERRM;

END;
$$;
```

Primero se declara una variable de tipo entera para poder guardar el identificador del inventario, a partir de esto después se actualiza la fecha de devolución con la fecha de devolución de entrada, aquellos registros donde el identificador de la renta concuerde con el identificador de entrada. Después se busca el identificador de inventario que esté asociado al identificador de la renta, adicionalmente de actualiza la disponibilidad de la tabla de inventario a partir del identificador de este mismo.

Finalmente se confirma la transacción con una noticia.

1.2 Seguridad y manejo de roles:

- **Creación de los roles**

EMP: solo tiene el derecho de ejecutar los siguientes procedimientos almacenados; no puede leer ni actualizar ningún objeto de la base de datos .

▪ Registrar un alquiler

▪ Registrar una devolución

▪ Buscar una película

Para la creación del rol "**emp**" el cual solo puede ejecutar procedimientos especificados, y estos estén ligados al rol. Para lo anterior se utiliza el siguiente código, el cual lo que realiza es la creación de un rol de nombre "**emp**".

Código de creación del rol de "EMP":

```
-> CREATE ROLE emp;
```

A continuación, se muestran las funciones las cuales fueron mencionadas y explicadas ampliamente en el apartado pasado.

- *Registrar un alquiler*
- *Registrar devolucion:*
- *Buscar una película*

Estos procedimientos creados dentro de la base de datos hay que otorgarles a los procedimientos la opción para que el rol "**emp"** pueda usarlos correctamente.

Para realizar esto escribimos las siguientes líneas de código, estas lo que hacen es que al usuario "**emp**" se le otorga permisos para poder utilizar los procedimientos.

Código para otorgar los permisos:

- > GRANT EXECUTE ON PROCEDURE registrar_alquiler(TIMESTAMP WITHOUT TIME ZONE, INT, SMALLINT, TIMESTAMP

-> WITHOUT TIME ZONE, SMALLINT, TIMESTAMP WITHOUT TIME ZONE) TO emp;

-> GRANT EXECUTE ON PROCEDURE registrar_devolucion(INT, DATE) TO emp;

-> GRANT EXECUTE ON PROCEDURE buscar_pelicula(VARCHAR) TO emp;

Como explicación de las líneas del código mostrado anteriormente, se tiene las siguientes explicaciones:

GRANT EXECUTE: Otorga permisos de ejecución.

ON FUNCTION NombreFuncion(): Indica que se otorga permisos de ejecución para la función llamada NombreFuncion().

TO emp: Especifica el rol al cual se le otorgan los permisos.

ADMIN: Tiene el derecho de un empleado más el derecho de ejecutar los siguientes procedimientos almacenados; no puede leer ni actualizar ningún objeto de la base de datos

▪ Insertar un nuevo cliente

Al igual que el rol **"emp"**, el rol "**admin**" solo puede ejecutar procedimientos especificados, y estos estén ligados al rol. Aparte de los procedimientos ligados al rol "**admin**" este podrá ejecutar los procedimientos que estén ligados al rol **"emp"**-

Para la creación del rol “**admin**” el cual solo puede ejecutar procedimientos especificados y estén ligados al rol se utiliza el siguiente código.

```
-> CREATE ROLE admin;

-> GRANT emp TO admin;
```

Lo que realiza el “GRANT emp TO admin;” es asignar que el rol **admin** se cómo una extensión o hereda del rol **emp**. Esto lo que hace es que el rol **admin** tiene ahora todos los privilegios que se le han otorgado al rol **emp**.

Como se menciona el rol “**admin**” tiene una serie de procedimientos que solo él puede utilizar. A continuación, se muestra las funciones las cuales el rol puede utilizar.

Los procedimientos que corresponden al rol “**admin**” es el siguiente:

- Insertar un nuevo cliente

Recordemos que el rol **admin** es una extensión del rol **emp**, es por ello que el rol admin puede hacer uno de los procedimientos ligados al rol **emp**. Que son lo procedimientos de:

▪ Registrar un alquiler
▪ Registrar una devolución
▪ Buscar una película

Ya con el procedimiento creado lo que tenemos que realizar es la otorgación de permisos al rol **admin** para que pueda realizar uso del procedimiento. Para poder realizar esto lo que tenemos que hacer es utilizar la siguiente línea de código.

```
->GRANT EXECUTE ON PROCEDURE insertar_nuevo_cliente(VARCHAR, VARCHAR, VARCHAR, INT, INT, BOOLEAN, DATE, TIMESTAMP WITHOUT TIME ZONE, INT) TO admin;
```

Las líneas anteriores lo que realizan lo siguiente:

GRANT EXECUTE: Otorga permisos de ejecución.

ON FUNCTION NombreFuncion(): Indica que se otorga permisos de ejecución para la función llamada NombreFuncion().

TO emp: Especifica el rol al cual se le otorgan los permisos.

Video: no login, dueño de todas las tablas y de todos los procedimientos creados.

Para realizar esto simplemente se creará un rol de nombre **video** el cual tiene permisos a todas las tablas y procedimientos sin necesidad de permisos de login, ósea el rol **NO** podrá iniciar sesión a la base.

Para ello se escribe la siguiente línea de comando.

```
-> CREATE ROLE video NOLOGIN;
```

Acá se crea un rol llamado **video** y el **NOLOGIN** indica que este rol no tiene permiso para iniciar sesión en la base de datos.

Como se mencionó este tiene acceso sobre las tablas; para poder darle acceso a todas las tablas en la base de datos al rol "**video**" se utiliza el siguiente comando:

-> GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO video;

-> ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO video;

Para poder utilizar los procedimientos desde del rol de **video** se utiliza los siguientes comandos, estos lo que realizan es que convierten al rol **video** como el dueño de los procesos.

-> ALTER PROCEDURE registrar_alquiler(TIMESTAMP WITHOUT TIME ZONE, INT, SMALLINT, TIMESTAMP WITHOUT TIME ZONE, SMALLINT, TIMESTAMP WITHOUT TIME ZONE) OWNER TO video;

-> ALTER PROCEDURE registrar_devolucion(INT, DATE) OWNER TO video;

-> ALTER PROCEDURE buscar_pelicula(VARCHAR) OWNER TO video;

-> ALTER PROCEDURE insertar_nuevo_cliente(VARCHAR, VARCHAR, VARCHAR, INT, INT, BOOLEAN, DATE, TIMESTAMP WITHOUT TIME ZONE, INT) OWNER TO video;

Para poder limitar el uso del procedimiento y que solo los roles que tengan acceso a los procedimientos se revocaron los permisos del rol **PUBLIC**, el rol **PUBLIC** es un rol el cual automáticamente se asocia a los procesos. Para revocar el permiso se utiliza los siguientes códigos:

```
-> REVOKE ALL ON PROCEDURE registrar_alquiler FROM PUBLIC;
-> REVOKE ALL ON PROCEDURE registrar_devolucion FROM PUBLIC;
-> REVOKE ALL ON PROCEDURE buscar_pelicula FROM PUBLIC;
-> REVOKE ALL ON PROCEDURE insertar_nuevo_cliente FROM PUBLIC;
```

El rol de **video** no cuenta con permisos para realizar secuencias en algunas tablas utilizadas en los procesos es por ello por lo que se tienen que dar permisos al rol de **video** para que al hacer uso del proceso todo funcione de forma correcta. Para ello se utiliza los siguientes códigos:

```
-> GRANT USAGE, SELECT ON SEQUENCE public.customer_customer_id_seq TO video;
-> GRANT USAGE, SELECT ON SEQUENCE public.rental_rental_id_seq TO video;
```

El propósito de estas líneas es asegurarse que las funciones se ejecuten bajo las credenciales del rol **video**, lo que es útil si se utilizan los mecanismos de seguridad definidos para limitar el acceso a los datos.

- **Creación de usuarios**

Empleado1: Un usuario con rol **EMP**. Para crear un nuevo empleado de ron "**emp**" se utiliza las siguientes líneas:

```
-> CREATE USER empleado1 WITH PASSWORD ' Jeffer123 ';
-> GRANT emp TO empleado1;
```

Estas líneas crean un nuevo usuario **empleado1** con una contraseña y le asignan el rol **emp**, lo que le otorga ciertos permisos específicos sobre la base de datos.

Este usuario tendrá acceso a los procedimientos asociados con el rol **emp**. Aparte también se podrá realizar login utilizando el usuario y la contraseña asignada.

Administrador1: Un usuario con rol **ADMIN.** Para poder crear un empleado el cual tenga privilegio de administrador se utiliza el siguiente código.

```
-> CREATE USER administrador1 WITH PASSWORD 'Pass123';

-> GRANT admin TO administrador1;
```

Estas líneas crean un nuevo usuario **administrador1** con una contraseña y le asignan el rol **admin**. Esto le da a **administrador1** todos los privilegios que tiene el rol **admin.** Al igual que empleado1 el **administrador1** tiene la posibilidad de realizar un login utilizando el usuario y la contraseña asignada.

- **Seguridad en procedimientos almacenados:**

Los procedimientos almacenados deben correr usando las credenciales de su dueño, **video**.

Para hacer que los procedimientos almacenados se ejecuten utilizando las credenciales de su propietario, en este caso, **video.**

Lo que se hace es tener un rol llamado **video** y hay que agregarle a los procedimientos almacenados la cláusula **SECURITY DEFINER**. Esto asegura que los procedimientos se ejecuten con los privilegios del rol del creado, en este caso es **video**.

También hay que asegurar que el propietario del proceso sea asignado.

Crear el Rol video

- > CREATE ROLE video NOLOGIN;

Definir los Procedimientos con SECURITY DEFINER

```
CREATE OR REPLACE PROCEDURE inserter_nuevo_cliente(
    in_first_name VARCHAR,
    in_last_name VARCHAR,
    in_email VARCHAR,
    in_address_id INT,
    in_store_id INT,
    in_active_bool BOOLEAN,
    in_create_date DATE,
    in_last_update TIMESTAMP WITHOUT TIME ZONE,
    in_active INT
)
LANGUAGE plpgsql
SECURITY DEFINER <----------------------
AS $$
BEGIN
    -- Insertar cliente
    INSERT INTO public.customer(
        store_id, first_name, last_name, email, address_id, activebool, create_date, last_update, active
    )
    VALUES (
        in_store_id, in_first_name, in_last_name, in_email,
        in_address_id, in_active_bool, in_create_date, in_last_update, in_active
    );


    RAISE NOTICE 'El cliente se creo correctamente';
EXCEPTION
    WHEN unique_violation THEN
        RAISE NOTICE 'El cliente ya existe con el email %.', in_email;
    WHEN foreign_key_violation THEN
        RAISE NOTICE 'El store_id o address_id no son válidos.';
    WHEN others THEN
        RAISE NOTICE 'Ocurrió un error: %', SQLERRM;
END;
```

;

SECURITY DEFINER:

El **SECURITY DEFINER** permite que los procedimientos que han sido creados como procedimiento con la cláusula SECURITY DEFINER permite que se ejecuten con los privilegios del creador, en este caso el usuario **video.**

Asignar el Propietario del Procedimiento:

-> ALTER PROCEDURE insertar_nuevo_cliente(VARCHAR, VARCHAR, VARCHAR, INT, INT, BOOLEAN, DATE, TIMESTAMP WITHOUT TIME ZONE, INT) OWNER TO video;

Permisos a tablas:

Para poder darle acceso a todas las tablas en la base de datos al rol "**video**" se utiliza el siguiente comando:

-> GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO video;

-> ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO video;

Nota: Este es el único rol el cual tiene permisos a todas las tablas.

1 Introducción

Esta guía cubre la instalación de dos instancias de PostgreSQL en **Windows**:

- La primera instancia se instala mediante el instalador oficial de PostgreSQL y corre en el puerto **5432**.
- La segunda instancia se crea desde el **archivo binario ZIP** y corre en el puerto **5433**.

Incluye los pasos para registrar la réplica como un servicio de Windows y cómo iniciar manualmente la réplica en caso de problemas con el servicio.

2 Paso 1: Instalación de la Primera Instancia

2.1 1.1 Descargar e Instalar PostgreSQL

1. Descarga el instalador desde: https://www.postgresql.org/download/windows/.
2. Durante la instalación, selecciona las opciones por defecto y habilita los servicios en el puerto **5432**.

2.2 1.2 Verificar la Instalación

Abre CMD y ejecuta:

```
psql --version
```

3 Paso 2: Crear la Segunda Instancia desde ZIP

3.1 2.1 Descargar y Extraer PostgreSQL

1. Descarga el archivo ZIP binario desde: https://www.enterprisedb.com/download-postgresql-
2. Extrae los archivos en una ruta, por ejemplo: `C:\replica\`.

3.2 2.2 Inicializar la Segunda Instancia

Abre CMD y navega al directorio binario:

```
cd C:\Replica\bin
initdb -D C:\postgresql\data\replica
```

3.3 2.3 Editar el Archivo `postgresql.conf`

Modifica las siguientes líneas en:

```
C:\postgresql\data\replica\postgresql.conf
```

```
port = 5433
hot_standby = on
listen_addresses = 'localhost'
```

3.4 2.4 Crear el Archivo standby.signal

```
echo > C:\postgresql\data\replica\standby.signal
```

4 Paso 3: Configurar el Servidor Primario

4.1 3.1 Editar postgresql.conf

```
C:\Program Files\PostgreSQL\version\data\postgresql.conf
```

```
listen_addresses = 'localhost'
wal_level = replica
archive_mode = on
max_wal_senders = 10
wal_keep_size = 256MB
```

4.2 3.2 Editar pg_hba.conf

```
C:\Program Files\PostgreSQL\version\data\pg_hba.conf
```

```
host    replication     replicator      127.0.0.1/32            md5
```

4.3 3.3 Crear el Usuario Replicator

Abre PostgreSQL desde CMD:

```
psql -U postgres
```

```
CREATE USER Replicator REPLICATION LOGIN ENCRYPTED PASSWORD '
   ↪ replica';
\q
```

5 Paso 4: Clonar los Datos con pg_basebackup

```
pg_basebackup -h localhost -D C:\PostgreSQL\Replica\Data -U
   ↪ Replicator -Fp -Xs -P -R -p 5432
```

6 Paso 5: Registrar la Réplica como Servicio en Windows

6.1 5.1 Crear el Servicio

Ejecuta este comando en CMD como administrador:

```
sc create PostgreSQL-Replica binPath= "\"C:\Replica\bin\pg_ctl.
   ↪ exe\"␣run␣-N␣PostgreSQL-Replica␣-D␣\"C:\postgresql\data\
   ↪ replica\"␣-w" start= auto
```

Este comando:

- Crea un servicio llamado `PostgreSQL-Replica`.
- Configura el arranque automático del servicio al iniciar Windows.

6.2 5.2 Iniciar y Verificar el Servicio

Inicia el servicio con:

```
net start PostgreSQL-Replica
```

Verifica el estado del servicio:

```
sc query PostgreSQL-Replica
```

6.3 5.3 Solucionar Problemas del Servicio

Si el servicio no inicia, revisa los logs del sistema con:

```
eventvwr
```

7 Paso 6: Iniciar la Instancia Manualmente

7.1 6.1 Comando para Iniciar Manualmente la Réplica

Si el servicio no funciona, inicia la réplica manualmente con:

```
pg_ctl -D C:\postgresql\data\replica -l logfile start
```

7.2 6.2 Verificar que el Servidor Está Corriendo

```
netstat -an | findstr 5433
```

7.3 6.3 Detener la Instancia Manualmente

```
pg_ctl -D C:\postgresql\data\replica stop
```

8 Paso 7: Verificar la Replicación

8.1 7.1 Verificar el Estado en la Instancia Primaria

```
psql -U postgres -c "SELECT␣*␣FROM␣pg_stat_replication;"
```

8.2 7.2 Verificar el Estado en la Réplica

```
psql -U postgres -c "SELECT␣*␣FROM␣pg_stat_wal_receiver;"
```

9 Fuente de Datos y Configuración en Tableau

Para el análisis y la generación de visualizaciones, se utilizó la base de datos `dvdrental`. Esta base de datos contiene información detallada sobre alquileres de películas, incluyendo clientes, películas, sucursales y datos temporales. En esta sección se explica la conexión entre Tableau y PostgreSQL, junto con las tablas y vistas utilizadas en el modelo multidimensional.

9.1 Script SQL para la Creación de las Tablas y la Vista

El siguiente script muestra cómo se crearon las tablas y la vista en PostgreSQL.

```
CREATE TABLE rental_fact (
    rental_id BIGINT PRIMARY KEY,
    film_id INT,
    customer_id INT,
    store_id INT,
    address_id INT,
    rental_date DATE,
    payment_amount NUMERIC(10, 2)
);

CREATE TABLE dim_film (
    film_id INT PRIMARY KEY,
    title VARCHAR(255),
    category_name VARCHAR(100),
    release_year INT,
    actor_names TEXT
);

CREATE TABLE dim_address (
    address_id INT PRIMARY KEY,
    address VARCHAR(255),
    city VARCHAR(100),
    country VARCHAR(100)
);

CREATE TABLE dim_date (
    rental_date DATE PRIMARY KEY,
    year INT,
    month INT,
    day INT
);

CREATE TABLE dim_store (
    store_id INT PRIMARY KEY,
    manager_staff_id INT,
    address_id INT
);
```

9.2 Vista `film_actor_detail`

Esta vista se utilizó para mostrar los detalles de los actores en cada película. A continuación se muestra el código SQL que define la vista:

```
CREATE VIEW film_actor_detail AS
SELECT
    f.film_id,
    f.title,
    c.name AS category_name,
    f.release_year,
    a.actor_id,
    a.first_name || ' ' || a.last_name AS actor_name
FROM film f
JOIN film_actor fa ON f.film_id = fa.film_id
JOIN actor a ON fa.actor_id = a.actor_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id;
```

9.3 Procedimientos Almacenados para Población de Datos

Los siguientes procedimientos almacenados fueron implementados para poblar las tablas del modelo estrella con datos provenientes de la base de datos transaccional:

```
CREATE OR REPLACE PROCEDURE populate_dim_film()
LANGUAGE plpgsql AS $$
BEGIN
    INSERT INTO dim_film (film_id, title, category_name, release_year, actor_names)
    SELECT f.film_id, f.title, c.name AS category_name, f.release_year,
           STRING_AGG(DISTINCT a.first_name || ' ' || a.last_name, ', ') AS actor_nam
    FROM film f
    JOIN film_category fc ON f.film_id = fc.film_id
    JOIN category c ON fc.category_id = c.category_id
    JOIN film_actor fa ON f.film_id = fa.film_id
    JOIN actor a ON fa.actor_id = a.actor_id
    GROUP BY f.film_id, f.title, c.name, f.release_year;
END;
$$;

CREATE OR REPLACE PROCEDURE populate_dim_address()
LANGUAGE plpgsql AS $$
BEGIN
    INSERT INTO dim_address (address_id, address, city, country)
    SELECT a.address_id, a.address, ci.city, co.country
    FROM address a
    JOIN city ci ON a.city_id = ci.city_id
    JOIN country co ON ci.country_id = co.country_id;
END;
$$;
```

```
CREATE OR REPLACE PROCEDURE populate_dim_date()
LANGUAGE plpgsql AS $$
BEGIN
    INSERT INTO dim_date (rental_date, year, month, day)
    SELECT DISTINCT DATE_TRUNC('day', r.rental_date) AS rental_date,
           EXTRACT(YEAR FROM r.rental_date) AS year,
           EXTRACT(MONTH FROM r.rental_date) AS month,
           EXTRACT(DAY FROM r.rental_date) AS day
    FROM rental r
    WHERE DATE_TRUNC('day', r.rental_date) NOT IN (SELECT rental_date FROM dim_date);
END;
$$;

CREATE OR REPLACE PROCEDURE populate_dim_store()
LANGUAGE plpgsql AS $$
BEGIN
    INSERT INTO dim_store (store_id, manager_staff_id, address_id)
    SELECT s.store_id, s.manager_staff_id, s.address_id
    FROM store s;
END;
$$;

CREATE OR REPLACE PROCEDURE populate_rental_fact()
LANGUAGE plpgsql AS $$
BEGIN
    INSERT INTO rental_fact (rental_id, film_id, customer_id, store_id, address_id, r
    SELECT r.rental_id, i.film_id, r.customer_id, c.store_id, c.address_id, r.rental_
    FROM rental r
    JOIN inventory i ON r.inventory_id = i.inventory_id
    JOIN payment p ON r.rental_id = p.rental_id
    JOIN customer c ON r.customer_id = c.customer_id
    ON CONFLICT (rental_id) DO NOTHING;
END;
```

9.4 Conexión entre PostgreSQL y Tableau

Para conectar Tableau con la base de datos PostgreSQL, se siguieron los siguientes pasos:

1. En Tableau, se seleccionó la opción **PostgreSQL** desde el menú de conexiones.
2. Se ingresaron los siguientes parámetros de conexión:
    - **Host:** localhost
    - **Base de datos:** dvdrental
    - **Puerto:** 5432
    - **Usuario:** video (propietario de las tablas)

- **Contraseña:** [Contraseña del usuario configurado]

3. Se probó la conexión para asegurar que Tableau pudiera acceder a la base de datos.

9.5 Tablas y Vista Utilizadas

El modelo multidimensional se basa en las siguientes tablas y vista:

- **rental_fact**: Tabla de hechos que almacena los alquileres realizados con detalles como la película, cliente, sucursal, fecha y monto pagado.
- **dim_film**: Tabla de dimensión que contiene información de las películas, categorías, año de lanzamiento y actores involucrados.
- **dim_address**: Almacena las direcciones de los clientes y tiendas, incluyendo ciudad y país.
- **dim_date**: Tabla de dimensión temporal que permite el análisis por año, mes y día.
- **dim_store**: Contiene la información de cada tienda, como su manager y ubicación.
- **film_actor_detail**: Vista que muestra los detalles de cada película con sus actores y categoría.

9.6 Definición del Esquema Estrella en Tableau

En Tableau, las tablas se relacionaron como sigue:

- **rental_fact** es la tabla central de hechos y se conectó con las dimensiones a través de sus claves foráneas:
  - `film_id` → `dim_film`
  - `store_id` → `dim_store`
  - `address_id` → `dim_address`
  - `rental_date` → `dim_date`
- La vista `film_actor_detail` se utilizó para enriquecer los datos con los nombres de los actores en cada película.

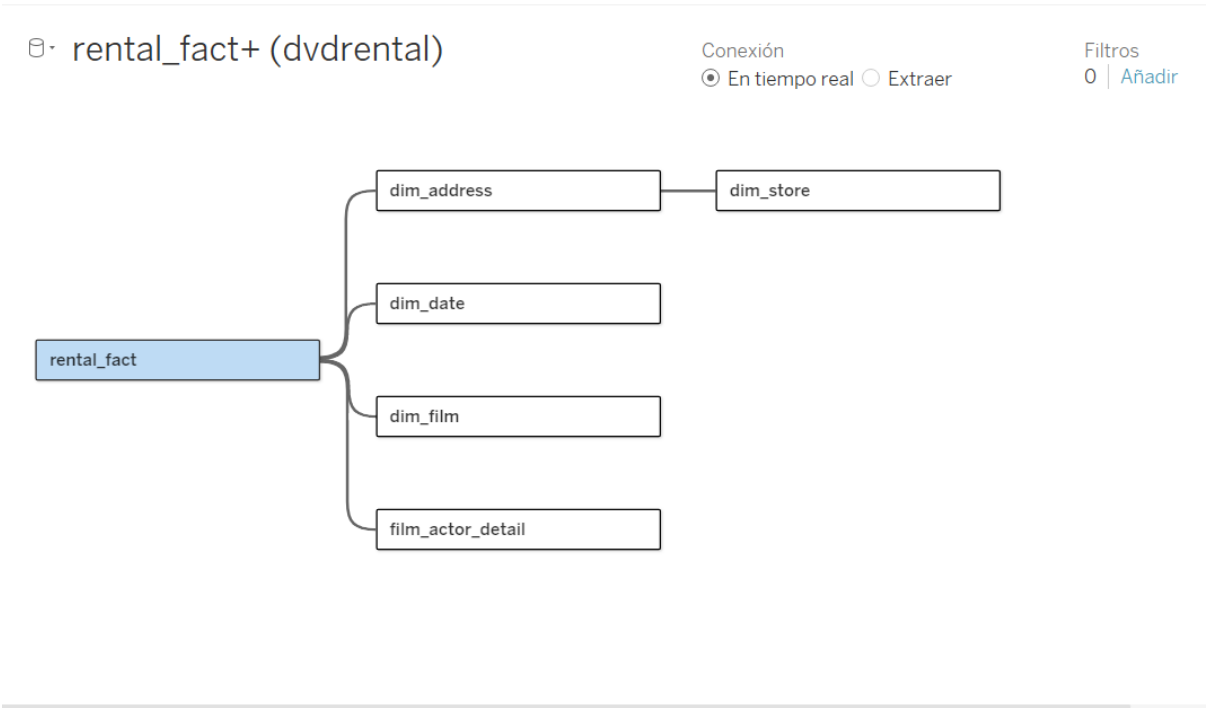

Figure 1: Relación entre las Tablas del Modelo Estrella en Tableau.

9.7 Descripción de las Tablas y Vista Utilizadas

A continuación, se describe cada tabla y vista utilizada en el modelo multidimensional:

- **rental_fact:** Es la tabla de hechos que contiene información sobre cada alquiler realizado. Incluye el identificador de película, cliente, tienda, dirección, fecha y monto pagado.
- **dim_film:** Tabla de dimensión que almacena datos sobre las películas, incluyendo el título, la categoría, el año de lanzamiento y los nombres de los actores.
- **dim_address:** Almacena las direcciones relacionadas con los clientes y las tiendas, incluyendo ciudad y país.
- **dim_date:** Tabla de dimensión que permite el análisis por jerarquía temporal (año, mes y día).
- **dim_store:** Contiene la información sobre cada tienda, como el manager y la ubicación.
- **film_actor_detail:** Vista que detalla las películas junto con sus actores y categorías.

November 7, 2024

10 Interpretación de los Datos en Tableau

Las visualizaciones en Tableau se basaron en las tablas y vistas mencionadas, garantizando flexibilidad para personalizar los reportes según las necesidades del usuario final. A continuación, se explican los gráficos más relevantes.

10.1 Número de Alquileres y Monto Total por Categoría

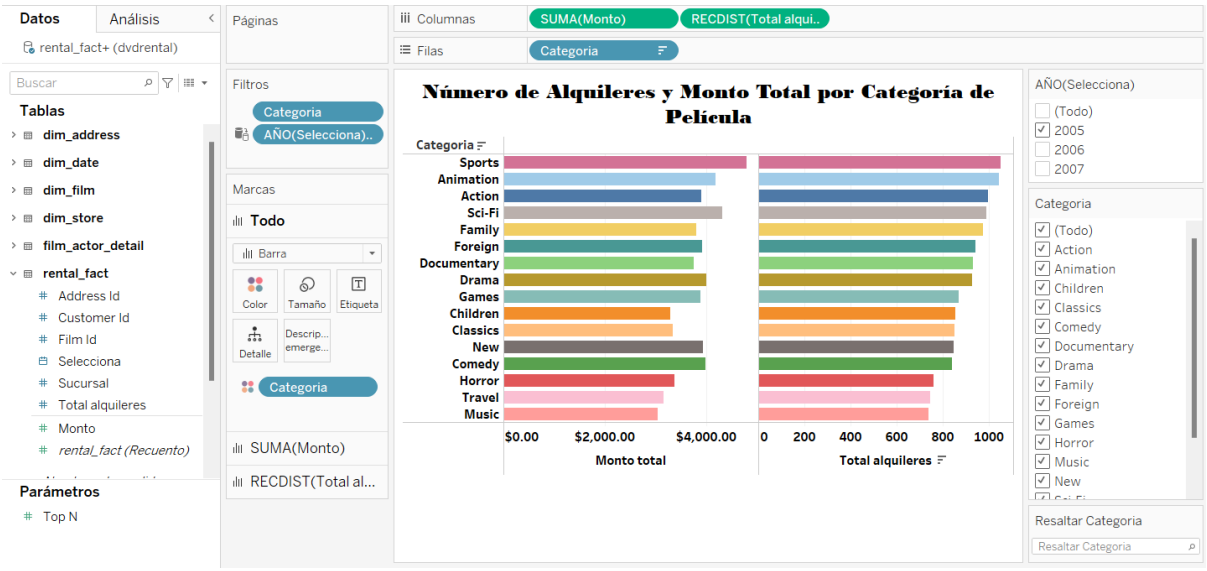

Figure 2: Número de Alquileres y Monto Total por Categoría de Película

Parámetros utilizados:

- **Dimensión:** Categoría (desde `dim_film`).
- **Medidas:** SUMA(Monto), REC(Conteo Total de Alquileres).
- **Filtro:** Año de alquiler (`dim_date`).

Este gráfico permite identificar las categorías más populares y rentables, facilitando el análisis del desempeño por tipo de película.

10.2 Monto Total por Alquileres para un Año Seleccionado (por Mes)

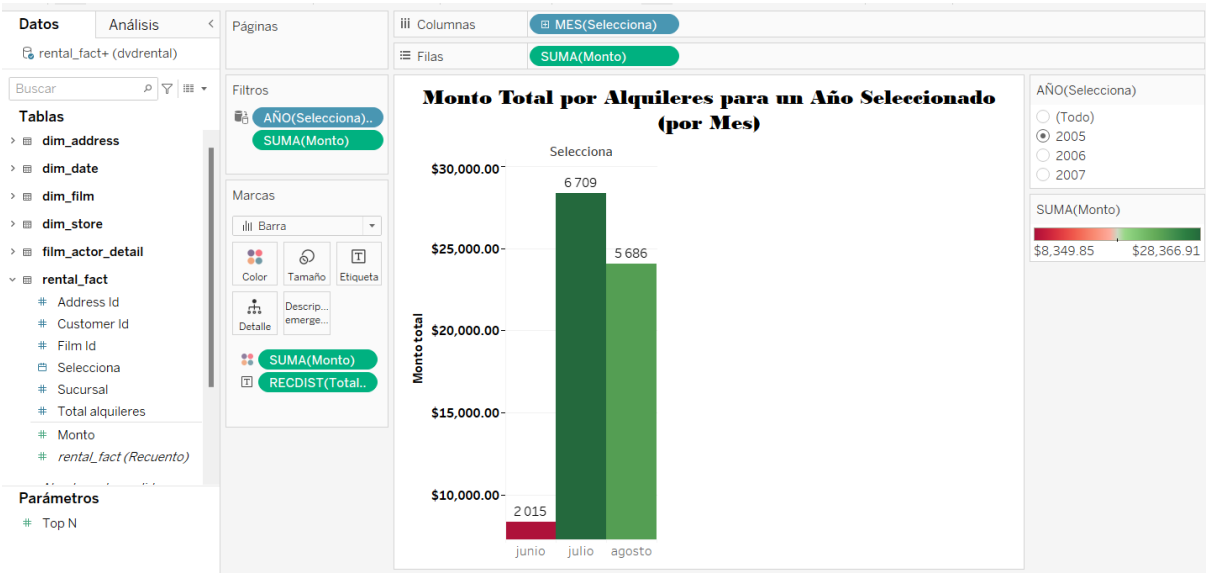

Figure 3: Monto Total por Alquileres por Mes en 2005

Parámetros utilizados:

- **Dimensión:** Mes (`dim_date`).
- **Medida:** SUMA(Monto).
- **Filtro:** Año seleccionado (2005).

Este gráfico revela patrones estacionales en los ingresos mensuales de alquileres, lo que permite identificar meses de mayor rentabilidad.

10.3 Top 10 Actores con Más Alquileres

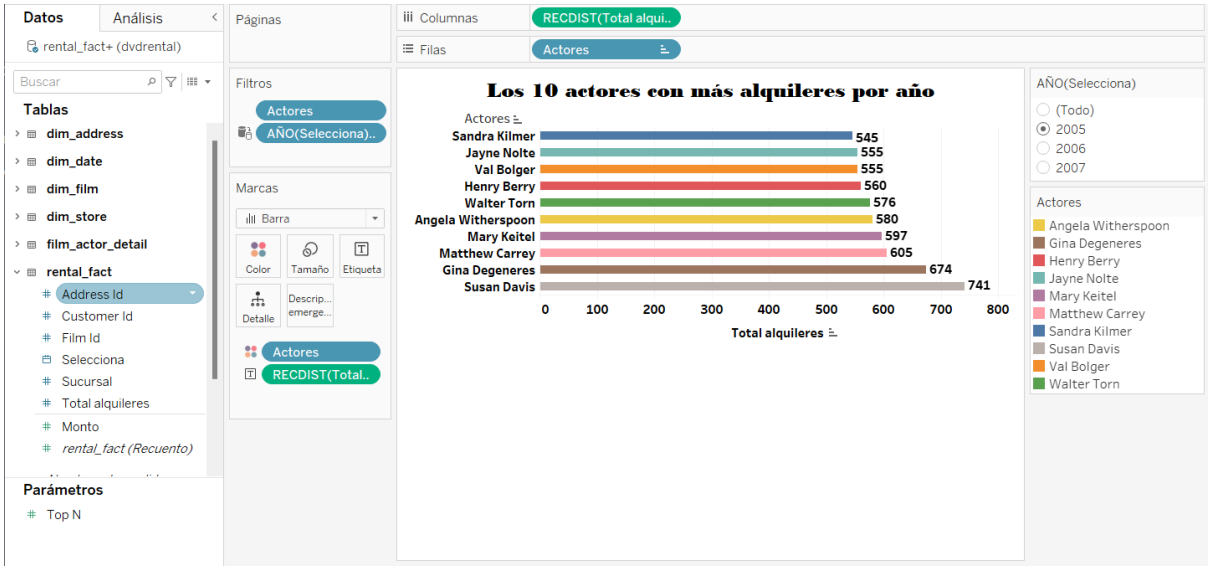

Figure 4: Top 10 Actores con Más Alquileres

Parámetros utilizados:

- **Dimensión:** Actores (desde `film_actor_detail`).
- **Medida:** REC(Conteo de Alquileres).
- **Filtro:** Año de alquiler.

Este gráfico destaca a los actores más populares en función del número de alquileres, permitiendo comparaciones rápidas entre ellos.

10.4 Mapa de Ciudades con el Monto Total de Alquiler

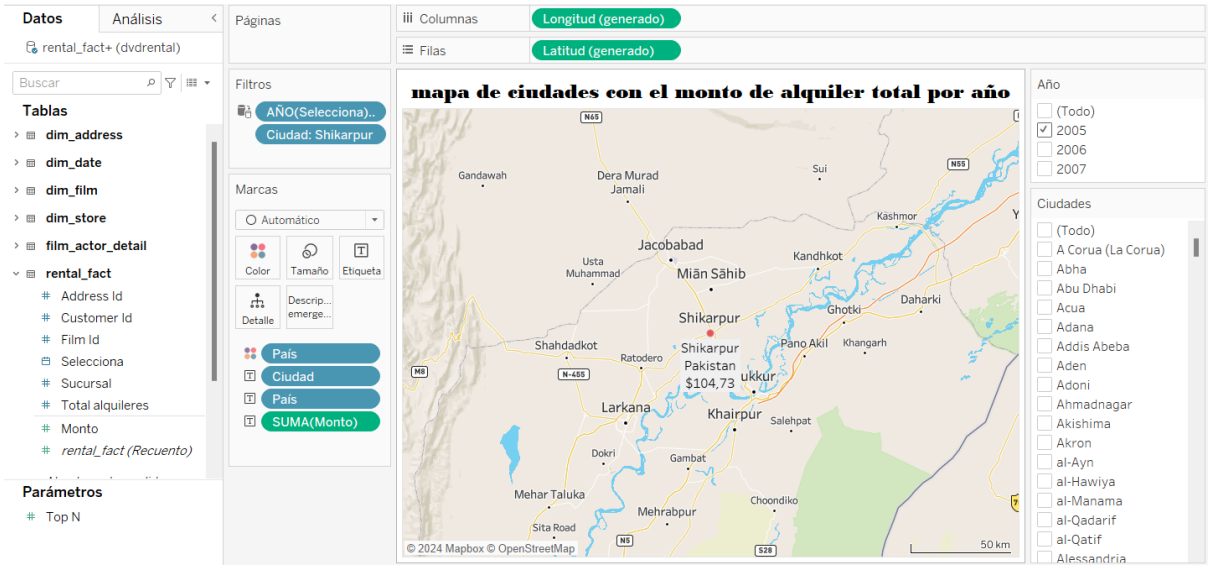

Figure 5: Mapa de Ciudades con el Monto Total de Alquiler

Parámetros utilizados:

- **Dimensiones:** Ciudad y País (`dim_address`).
- **Medida:** SUMA(Monto).
- **Filtro:** Año seleccionado.
- **Mapa:** OpenStreetMap integrado en Tableau.

Este mapa proporciona una vista geográfica del desempeño por ciudad, facilitando la identificación de regiones más rentables.

10.5 Alquileres y Monto Total por Sucursal y Mes

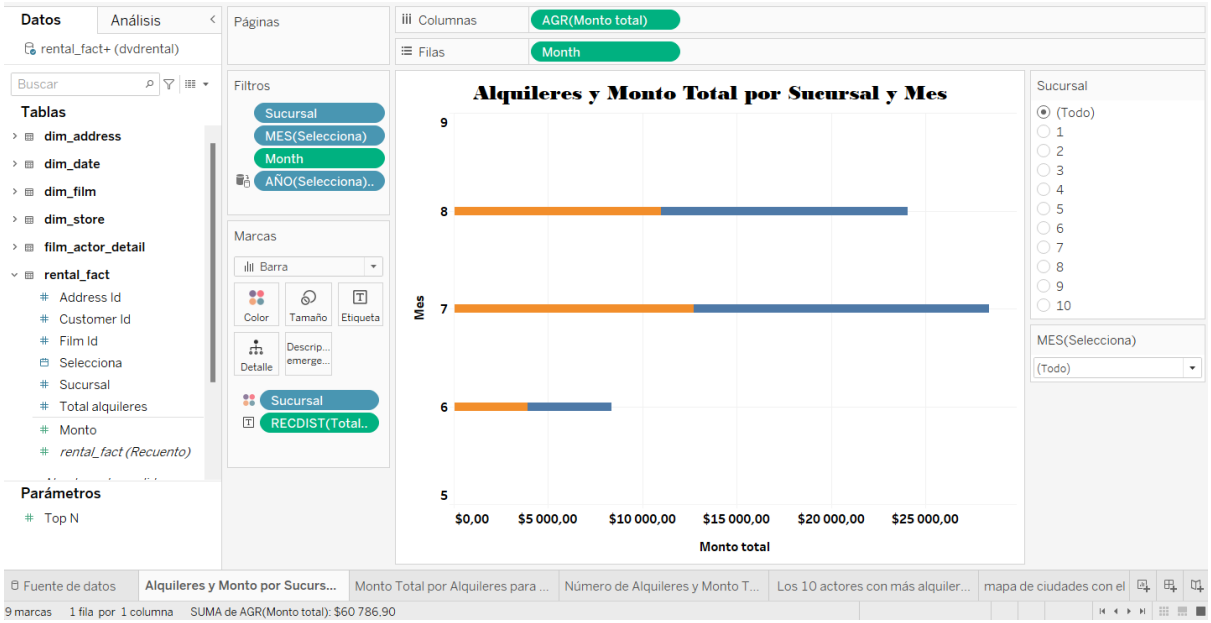

Figure 6: Alquileres y Monto Total por Sucursal y Mes

Parámetros utilizados:

- **Dimensión:** Sucursal (`dim_store`), Mes (`dim_date`).
- **Medidas:** SUMA(Monto), REC(Total de Alquileres).
- **Filtro:** Año.

Este gráfico permite comparar el rendimiento de diferentes sucursales a lo largo de los meses, ayudando a detectar tendencias de desempeño.

10.6 Dashboard

El dashboard completo fue desarrollado utilizando todas las tablas y vistas mencionadas, combinando varias dimensiones y medidas para ofrecer una visión integral del desempeño. La flexibilidad de Tableau permite explorar los datos desde múltiples perspectivas, facilitando análisis personalizados según los intereses del usuario final.

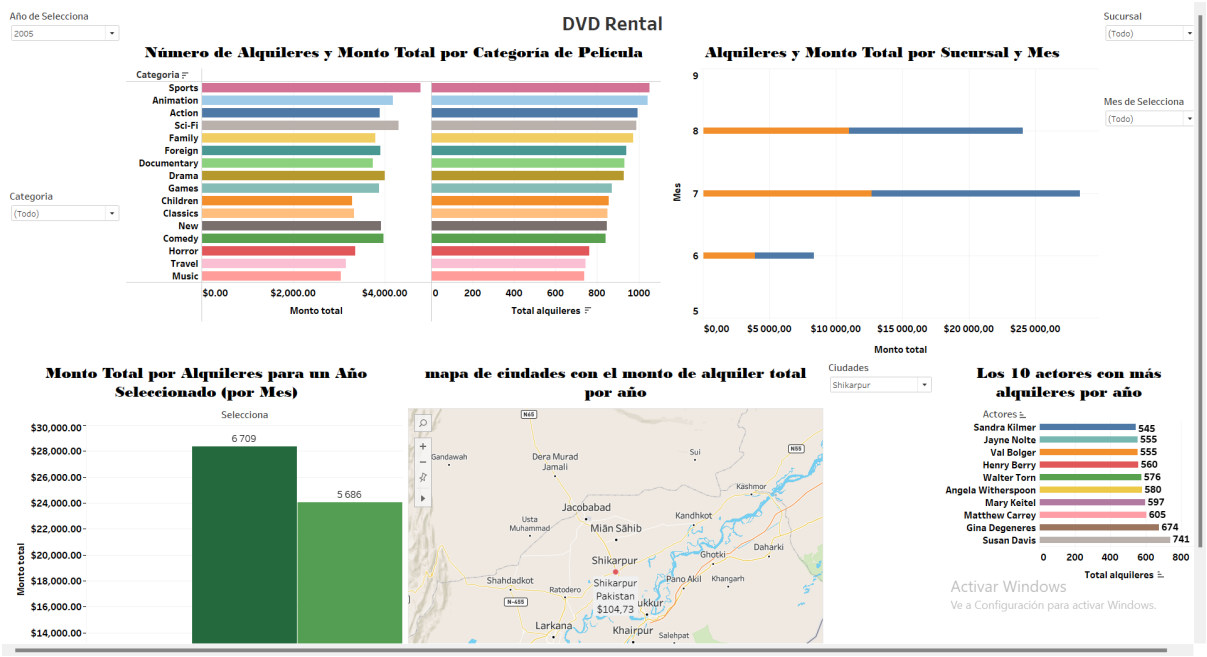

Figure 7: Dashboard