

Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
IC4302 Bases de Datos II
Grupo 20

Proyecto 2
Replicación y Análisis de Datos

Profesor:
Alberto Shum Chan

Estudiantes:

Andrés Felipe Arias Corrales
2015028947

Diego Esteban Castro Chaves
200419896

Fabián José Fernández Fernández
2022144383

Fecha de Entrega: 21 de Octubre del 2024

II semestre, 2024

Contents

| | | |
|----------|--|-----------|
| 1 | Introducción | 3 |
| 2 | Descripción del Dataset | 4 |
| 3 | Funciones y Procedimientos Almacenados | 6 |
| 3.1 | Procedimiento: <code>insert_customer</code> | 6 |
| 3.2 | Procedimiento: <code>register_rent</code> | 7 |
| 3.3 | Procedimiento: <code>return_movie</code> | 8 |
| 3.4 | Función: <code>search_movie</code> | 8 |
| 4 | Seguridad | 10 |
| 4.1 | Roles de Seguridad | 10 |
| 4.2 | Usuarios | 11 |
| 4.3 | Asignacion de Permisos para la Replicacion | 12 |
| 5 | Réplica de la Base de Datos | 13 |
| 5.1 | Creación de la Segunda Instancia de PostgreSQL y Configuración del Modelo de Replicación | 13 |
| 5.2 | Configuración en la Instancia Principal | 15 |
| 5.3 | Backup de la Base de Datos Principal en la Segunda Instancia | 16 |
| 5.4 | Inicialización del Servicio de la Segunda Instancia | 16 |
| 5.5 | Registro de la Réplica en PGAdmin | 18 |
| 6 | Modelo Multidimensional | 20 |
| 6.1 | Medidas de Interés | 20 |
| 6.2 | Dimensiones de Interés | 20 |
| 6.3 | Diseño del Esquema Estrella | 21 |
| 6.3.1 | Tabla de Hechos: <code>fact_rentals</code> | 21 |
| 6.3.2 | Tabla de Dimensiones: <code>dim_film</code> | 21 |
| 6.3.3 | Tabla de Dimensiones: <code>dim_address</code> | 21 |

| | | |
|-------|--|-----------|
| 6.3.4 | Tabla de Dimensiones: <code>dim_rental_date</code> | 22 |
| 6.3.5 | Tabla de Dimensiones: <code>dim_store</code> | 22 |
| 6.4 | Procedimientos Almacenados para la Carga de Datos | 23 |
| 6.4.1 | Procedimiento: <code>load_fact_rentals</code> | 23 |
| 6.4.2 | Procedimiento: <code>load_dim_film</code> | 24 |
| 6.4.3 | Procedimiento: <code>load_dim_address</code> | 24 |
| 6.4.4 | Procedimiento: <code>load_dim_rental_date</code> | 25 |
| 6.4.5 | Procedimiento: <code>load_dim_store</code> | 25 |
| 6.4.6 | Procedimiento: <code>load_star_schema</code> | 25 |
| 6.5 | Pruebas y Validación | 26 |
| 7 | Visualización y acceso por medio de interfaz gráfica al modelo estrella | 27 |

1 Introducción

El presente proyecto tiene como objetivo desarrollar un sistema completo de replicación y análisis de datos, que permita soportar la toma de decisiones mediante un modelo de inteligencia de negocios. El proyecto se enfoca en la configuración de una réplica de la base de datos, la implementación de un modelo multidimensional y la visualización de los resultados mediante dashboards interactivos creados con el software Tableau.

Para llevar a cabo este proyecto, se utilizó una base de datos transaccional del sistema de alquiler de películas, la cual fue replicada en una segunda instancia para separar el procesamiento transaccional del análisis de datos. Posteriormente, se diseñó un modelo multidimensional (estrella) basado en las medidas y dimensiones relevantes para el análisis de los alquileres de películas.

Finalmente, se creó un dashboard que permite visualizar la información relevante y obtener insights sobre los alquileres, ventas y el desempeño de la tienda. El uso de Tableau como herramienta de visualización proporciona una forma intuitiva de explorar y analizar los datos.

A lo largo de este documento, se detallarán todos los pasos realizados para la implementación del proyecto, incluyendo la descripción del dataset, las funciones y procedimientos almacenados, la seguridad, la replicación de la base de datos, el diseño del modelo multidimensional y la visualización del modelo estrella mediante una interfaz gráfica.

2 Descripción del Dataset

Para este proyecto, se ha utilizado la base de datos de ejemplo *DVD Rental*, proporcionada por PostgreSQL. Esta base de datos simula los procesos de negocio de una tienda de alquiler de DVDs y contiene diversos objetos, tales como tablas, vistas, funciones y secuencias. En total, la base de datos contiene:

- 15 tablas.
- 1 trigger.
- 7 vistas.
- 8 funciones almacenadas.
- 1 dominio.
- 13 secuencias.

Las tablas de la base de datos incluyen:

- **actor**: Almacena datos de los actores, incluyendo el nombre y apellido.
- **film**: Contiene información de las películas, como título, año de lanzamiento, duración, clasificación, etc.
- **film_actor**: Relaciona las películas con los actores.
- **category**: Almacena los géneros de las películas.
- **film_category**: Relaciona las películas con sus categorías.
- **store**: Almacena información sobre las tiendas, incluyendo los datos del gerente y la dirección.
- **inventory**: Almacena los datos del inventario de cada tienda.
- **rental**: Registra cada transacción de alquiler, incluyendo información sobre el cliente, la película, y las fechas de alquiler y devolución.
- **payment**: Almacena los pagos de los clientes.
- **customer**: Almacena los datos de los clientes, como nombre, dirección y estado de actividad.
- **address**: Almacena las direcciones de los empleados y los clientes.
- **city**: Almacena los nombres de las ciudades.
- **country**: Almacena los nombres de los países.
- **staff**: Almacena información sobre los empleados de la tienda, incluyendo su nombre, dirección y estado de actividad.
- **language**: Almacena los idiomas en los que están disponibles las películas.

Esta base de datos contiene todas las estructuras necesarias para realizar análisis de datos sobre los alquileres de películas, facilitando la construcción de un modelo multidimensional para analizar métricas clave como el número de alquileres y el total de ingresos.

A continuación, se presenta un diagrama entidad-relación (ER) que muestra las relaciones entre las principales tablas de la base de datos:

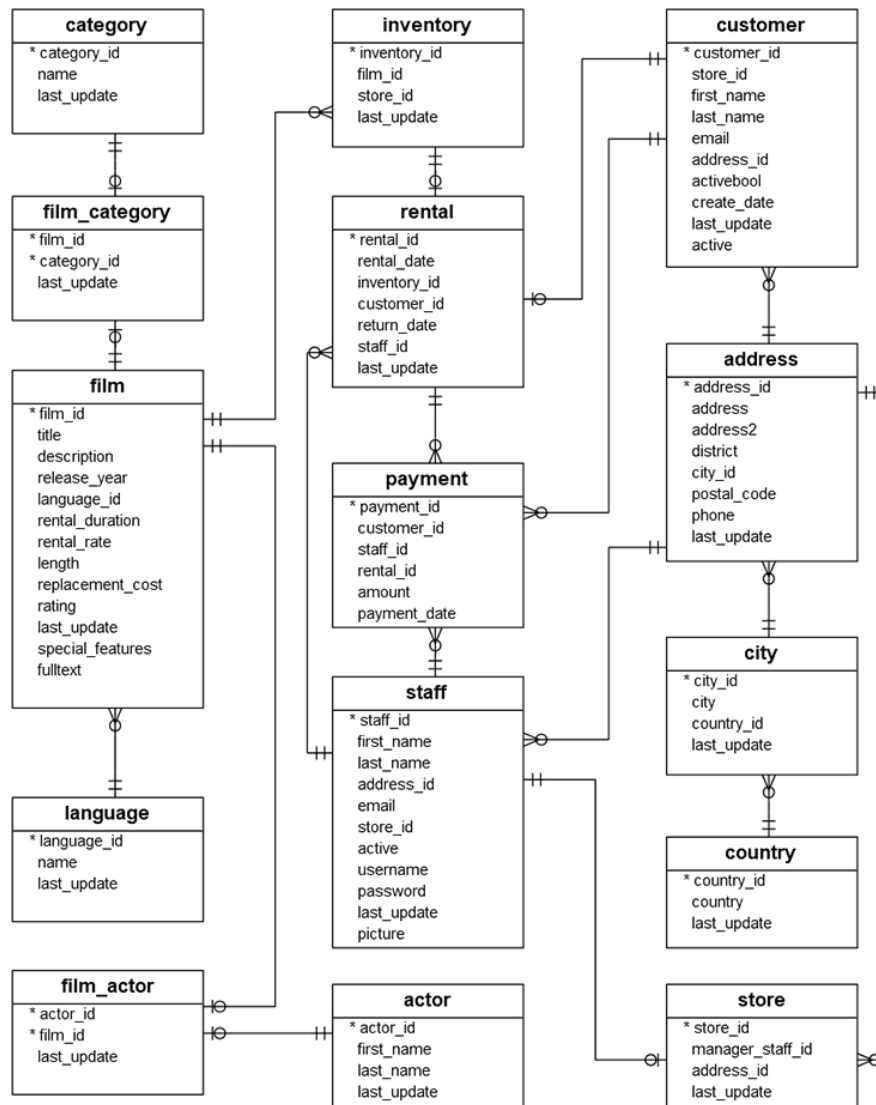


Figure 1: Diagrama ER de la base de datos DVD Rental

Esta base de datos fue descargada del siguiente enlace: [base de datos de ejemplo de PostgreSQL](#). La estructura de esta base de datos es fundamental para implementar el modelo en estrella que permite realizar consultas agregadas y generar reportes avanzados basados en los datos históricos de los alquileres.

3 Funciones y Procedimientos Almacenados

En esta sección se describen las funciones y procedimientos almacenados que afectan directamente el sistema transaccional de la base de datos. Cada uno de estos procedimientos incluye descripciones detalladas de sus parámetros, salidas y bloques relevantes.

3.1 Procedimiento: insert_customer

Descripción: Este procedimiento inserta un nuevo cliente en la tabla `customer`. Está diseñado para aceptar los datos principales de un cliente (nombre, apellidos, dirección, email, estado activo) y asociarlo a una tienda específica (`store_id`). Además, el procedimiento genera automáticamente un identificador único para el cliente mediante una secuencia (`customer_customer_id_seq`). La fecha de creación se establece automáticamente en la fecha actual.

Parámetros:

- `p_store_id` (INTEGER): ID de la tienda donde se registra el cliente.
- `p_first_name` (VARCHAR): Nombre del cliente.
- `p_last_name` (VARCHAR): Apellido del cliente.
- `p_email` (VARCHAR): Email del cliente.
- `p_address_id` (INTEGER): ID de la dirección del cliente.
- `p_active_bool` (BOOLEAN): Estado activo del cliente (True/False).

Uso: Para insertar un nuevo cliente, se puede utilizar la siguiente llamada:

```
CALL insert_customer(1, 'Juan', 'Perez', 'juan.perez@testuserbd2.com', 101, TRUE);
```

Código:

```
CREATE OR REPLACE PROCEDURE insert_customer(
    IN p_store_id INTEGER,
    IN p_first_name VARCHAR,
    IN p_last_name VARCHAR,
    IN p_email VARCHAR,
    IN p_address_id INTEGER,
    IN p_active_bool BOOLEAN
)
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
BEGIN
    INSERT INTO customer(
        customer_id, store_id, first_name, last_name, email, address_id,
        activebool,
        create_date, last_update, active
    )
    VALUES (
        nextval('customer_customer_id_seq'),
        p_store_id, p_first_name, p_last_name, p_email, p_address_id,
```

```
        true, CURRENT_DATE, CURRENT_TIMESTAMP AT TIME ZONE 'UTC', 1
    );
END;
$$;
```

3.2 Procedimiento: register_rent

Descripción: Este procedimiento permite registrar un alquiler de película en la tabla **rental**. Al ejecutarse, crea un nuevo registro en **rental** asociado a un cliente y a un ítem del inventario. También guarda la fecha del alquiler y el empleado que gestionó la transacción. El **return_date** se deja en blanco, ya que se actualiza al momento de devolver la película.

Parámetros:

- **p_inventory_id** (INTEGER): ID del ítem de inventario alquilado.
- **p_customer_id** (INTEGER): ID del cliente que realiza el alquiler.
- **p_staff_id** (INTEGER): ID del empleado que gestiona el alquiler.

Uso: Para registrar un alquiler, se puede utilizar la siguiente llamada:

```
CALL register_rent(1, 1, 1);
```

Código:

```
CREATE OR REPLACE PROCEDURE register_rent(
    p_inventory_id INTEGER,
    p_customer_id INTEGER,
    p_staff_id INTEGER
)
SECURITY DEFINER
AS $$
BEGIN
    INSERT INTO rental (
        rental_id, rental_date, inventory_id, customer_id, return_date, staff_id,
        last_update
    )
    VALUES (
        nextval('rental_rental_id_seq'),
        NOW()::TIMESTAMP(0),
        p_inventory_id, p_customer_id, NULL,
        p_staff_id, NOW()::TIMESTAMP(0)
    );
END;
$$;
```


3.3 Procedimiento: return_movie

Descripción: Este procedimiento actualiza el registro de un alquiler cuando una película es devuelta, asignando la fecha y hora de la devolución al campo `return_date` del alquiler.

Parámetro:

- `p_rental_id` (INTEGER): ID del alquiler que se va a actualizar.

Uso: Para devolver una película, se puede utilizar la siguiente llamada:

```
CALL return_movie(1);
```

Código:

```
CREATE OR REPLACE PROCEDURE return_movie(  
    p_rental_id INTEGER  
)  
SECURITY DEFINER  
AS $$  
BEGIN  
    UPDATE rental  
    SET return_date = NOW()::TIMESTAMP(0)  
    WHERE rental_id = p_rental_id;  
END;  
$$;
```

3.4 Función: search_movie

Descripción: Esta función permite buscar películas por título en la tabla `film`. El usuario puede ingresar un término de búsqueda y la función devolverá todas las películas que contengan ese término en su título, ignorando mayúsculas o minúsculas.

Parámetro:

- `p_titulo` (VARCHAR): Término de búsqueda que se usará para encontrar coincidencias en los títulos de películas.

Retorno: La función devuelve una tabla con las siguientes columnas:

- `return_film_id` (INTEGER): ID de la película.
- `return_title` (VARCHAR): Título de la película.
- `r_year` (YEAR): Año de lanzamiento de la película.

Uso: Para buscar películas, se puede utilizar la siguiente consulta:

```
SELECT * FROM search_movie('Inception');
```

Código:

```
CREATE OR REPLACE FUNCTION search_movie(p_titulo VARCHAR)
RETURNS TABLE(
    return_film_id INTEGER,
    return_title VARCHAR,
    r_year YEAR
)
SECURITY DEFINER
AS $$
BEGIN
    RETURN QUERY
    SELECT film_id, title, release_year
    FROM film
    WHERE film.title ILIKE '%' || p_titulo || '%';
END;
$$ LANGUAGE plpgsql;
```

4 Seguridad

Para proteger los datos y asegurar que solo los usuarios autorizados puedan ejecutar ciertos procedimientos, se implementaron varios roles y usuarios con diferentes niveles de acceso. Esta seccion detalla los roles creados, los privilegios otorgados y los usuarios asociados.

4.1 Roles de Seguridad

Se crearon los siguientes roles para controlar el acceso a las funciones y procedimientos almacenados:

- **EMP:** Este rol tiene el derecho de ejecutar ciertos procedimientos almacenados relacionados con el sistema de alquiler de peliculas, pero no puede leer ni actualizar directamente ninguna tabla de la base de datos. Sus privilegios incluyen:
 - `register_rent`: Registrar un alquiler.
 - `return_movie`: Registrar una devolucion.
 - `search_movie`: Buscar una pelicula.
- **ADMIN:** Este rol tiene los mismos derechos que **EMP**, pero con privilegios adicionales para ejecutar el procedimiento que permite insertar nuevos clientes:
 - `insert_customer`: Insertar un nuevo cliente.

El siguiente codigo muestra la creacion y configuracion de estos roles:

```
-- Creacion del rol EMP con privilegios de login
CREATE ROLE emp;
ALTER ROLE emp WITH LOGIN PASSWORD 'password';

-- Otorgar permisos de ejecucion de funciones y procedimientos al rol EMP
GRANT EXECUTE ON FUNCTION search_movie(varchar) TO emp;
GRANT EXECUTE ON PROCEDURE register_rent(p_inventory_id INTEGER, p_customer_id
    INTEGER, p_staff_id INTEGER) TO emp;
GRANT EXECUTE ON PROCEDURE return_movie(p_rental_id INTEGER) TO emp;

-- Creacion del rol ADMIN con los mismos derechos que EMP mas la habilidad de
    insertar clientes
CREATE ROLE admin WITH INHERIT LOGIN PASSWORD 'password';
GRANT emp TO admin; -- Heredar los privilegios de EMP

-- Otorgar permisos adicionales al rol ADMIN
GRANT EXECUTE ON PROCEDURE insert_customer(p_store_id INTEGER, p_first_name
    VARCHAR, p_last_name VARCHAR, p_email VARCHAR, p_address_id INTEGER,
    p_active_bool BOOLEAN) TO admin;
```

4.2 Usuarios

Se crearon usuarios especificos con roles asignados para que pudieran realizar las acciones necesarias en el sistema. Estos usuarios no tienen acceso directo a las tablas, solo pueden ejecutar los procedimientos que se les han asignado mediante sus roles.

- **video:** Este usuario es el dueño de todas las tablas y procedimientos de la base de datos, pero no tiene privilegios de login para evitar accesos no autorizados.
- **empleado1:** Usuario con el rol EMP, que tiene permiso para registrar alquileres, devoluciones y buscar películas.
- **administrador1:** Usuario con el rol ADMIN, que además de las funciones de EMP, puede insertar nuevos clientes.

El siguiente código muestra la creación y configuración de estos usuarios:

```
-- Crear el usuario sin login "video"
CREATE USER video;

-- Asignar la propiedad de la base de datos y todas las tablas a "video"
ALTER DATABASE dvdrental OWNER TO video;

-- Asignar la propiedad de todas las tablas a "video"
DO $$
DECLARE
    r RECORD;
BEGIN
    FOR r IN
        SELECT tablename FROM pg_tables WHERE schemaname = 'public'
    LOOP
        EXECUTE 'ALTER TABLE public.' || r.tablename || ' OWNER TO video;';
    END LOOP;
END $$;

-- Crear el usuario empleado1 con el rol EMP
CREATE USER empleado1;
ALTER USER empleado1 PASSWORD 'pass';
GRANT emp TO empleado1;

-- Crear el usuario administrador1 con el rol ADMIN
CREATE USER administrador1;
ALTER USER administrador1 PASSWORD 'pass';
GRANT admin TO administrador1;
```

4.3 Asignacion de Permisos para la Replicacion

Ademas, se creo un rol especial para manejar la replicacion de la base de datos. Este rol tiene privilegios para replicar los datos entre la instancia principal y la replica.

```
-- Creacion del rol replicator con privilegios de replicacion
CREATE ROLE replicator WITH REPLICATION PASSWORD 'replica' LOGIN;

-- Otorgar permisos de seleccion sobre todas las tablas al rol replicator
DO $$
DECLARE
    r RECORD;
BEGIN
    FOR r IN
        SELECT tablename FROM pg_tables WHERE schemaname = 'public'
    LOOP
        EXECUTE 'GRANT SELECT ON TABLE public.' || r.tablename || ' TO replicator;';
    END LOOP;
END $$;
```

5 Réplica de la Base de Datos

Después de revisar los distintos tipos de replicación vistos en clase e investigando un poco más a fondo, tomamos la decisión de configurar un sistema de replicación **Hot Standby / Streaming**.

Más específicamente hicimos la configuración **Hot Standby**, que permite que la instancia de réplica tenga acceso para operaciones de lectura, de forma que se pueda conectar con la aplicación de Tableau para hacer las operaciones de análisis de datos.

Esta configuración trabaja de modo sincrónico, de manera que la réplica tiene los datos actualizados en todo momento. Si bien esto recarga el procesamiento de las transacciones en el motor de la base de datos, no afecta de manera considerable el tiempo de ejecución desde que estamos trabajando sobre la misma máquina y el resultado es casi imperceptible para el usuario.

El sistema lo implementamos en una sola máquina instalada con **Windows 11**, utilizamos la versión **17 de PostgreSQL** y **Tableau Profesional Edition 2024.2.3** (usando el Trial de 14 días).

Seguidamente describimos el proceso de instalación del sistema.

5.1 Creación de la Segunda Instancia de PostgreSQL y Configuración del Modelo de Replicación

La creación de una segunda instancia con réplica en modo **Hot Stream** en PostgreSQL la configuramos de la siguiente manera.

Debemos considerar varias cosas antes de iniciar la configuración:

- Conocer la versión de **PostgreSQL** (en nuestro caso era la 17).
- Tener a mano la ubicación de instalación del motor de bases de datos (`C:/Program Files/PostgreSQL/17/`).
- Tener un backup de la carpeta de datos por si ocurre algo inesperado, para no perder los datos que tengamos dentro de nuestra instancia.

Seguidamente se crea un directorio en donde se alojará la información de la segunda instancia. Nosotros usamos `C:/PostgreSQL/Replica/Data`.

Luego vamos a inicializar el directorio de datos. Desde **PowerShell** nos dirigimos al directorio de instalación de PostgreSQL y ahí nos ubicamos en la carpeta `bin`.

Ahí ejecutamos el siguiente comando:

```
./initdb -D "C:/PostgreSQL/Replica/Data" -U postgres -W
```

Luego se debe configurar el archivo `postgresql.conf` de la segunda instancia. Modificamos algunas líneas utilizando un editor de texto que reconozca código, como **Visual Studio Code**.

Este archivo lo encontraremos en la carpeta que creamos para alojar la segunda instancia. Aquí, dentro de la carpeta `data`, se encuentran los archivos de configuración.

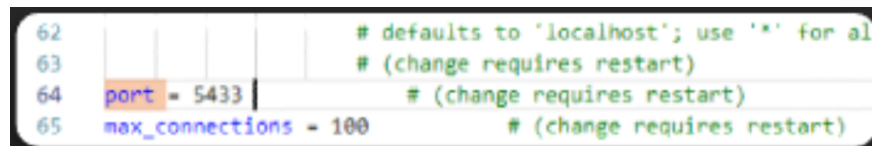
Primero, cambiamos la línea de la dirección de escucha con la siguiente configuración:



```
58 # - Connection Settings -
59
60 listen_addresses = 'localhost' # what IP address(es) to listen on;
61                                # comma-separated list of addresses;
```

Figure 2: Configuración de la dirección de escucha.

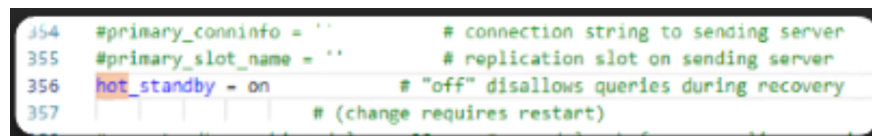
Segundo, modificamos el puerto al que se conecta la instancia para evitar conflicto con el puerto principal. Escogimos el puerto **5433**.



```
62                                # defaults to 'localhost'; use '*' for all
63                                # (change requires restart)
64 port = 5433                    # (change requires restart)
65 max_connections = 100         # (change requires restart)
```

Figure 3: Modificación del puerto de conexión.

Por último, activamos el modo *Hot Standby*.



```
354 #primary_conninfo = ''        # connection string to sending server
355 #primary_slot_name = ''       # replication slot on sending server
356 hot_standby = on              # "off" disallows queries during recovery
357                                # (change requires restart)
```

Figure 4: Activación del modo Hot Standby.

Guardamos el archivo y continuamos con la configuración del archivo `pg_hba.conf` de la segunda instancia. Este archivo también se encuentra en la carpeta `data` de la instancia de réplica.

Abrimos el archivo y añadimos la siguiente línea al final:

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

Luego guardamos y cerramos el archivo.

5.2 Configuración en la Instancia Principal

Abrimos **PGAdmin** para configurar la replicación en nuestro servidor. Creamos un usuario para la replicación en la instancia primaria con el siguiente script:

```
CREATE ROLE replicator WITH REPLICATION PASSWORD 'replica' LOGIN;
```

Ahora editamos los archivos de configuración de la base de datos de la instancia principal, ubicada en C:/Program Files/PostgreSQL/17/data. Primero, abrimos `postgresql.conf` y modificamos las siguientes líneas:

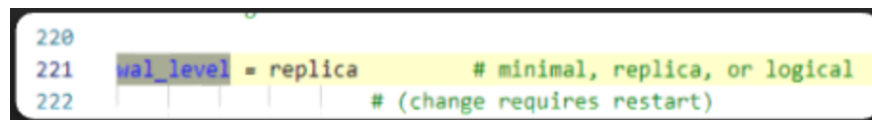
A screenshot of a text editor showing the configuration file postgresql.conf. Line 221 is highlighted in yellow and contains the text 'wal_level = replica # minimal, replica, or logical'. Line 222 is also highlighted in yellow and contains the text '# (change requires restart)'. The line numbers 220, 221, and 222 are visible on the left margin.

Figure 5: Configuración en `postgresql.conf`.


A screenshot of a text editor showing the configuration file postgresql.conf. Line 333 is highlighted in yellow and contains the text 'wal_keep_size = 256MB # in megabytes; 0 disables'. The line number 333 is visible on the left margin.

Figure 6: Configuración de los parámetros de replicación.

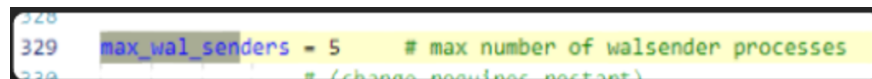
A screenshot of a text editor showing the configuration file postgresql.conf. Line 329 is highlighted in yellow and contains the text 'max_wal_senders = 5 # max number of walsender processes'. The line number 329 is visible on the left margin.

Figure 7: Configuración final en `postgresql.conf`.

Luego abrimos `pg_hba.conf` y añadimos la misma línea que agregamos en la segunda instancia:

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

Guardamos el archivo y reiniciamos la instancia principal para que los cambios tengan efecto, utilizando el administrador de servicios de **Windows** (`services.msc`).

5.3 Backup de la Base de Datos Principal en la Segunda Instancia

Para hacer un respaldo de la base de datos principal en la segunda instancia, nos ubicamos en la carpeta `bin` de la instancia principal y ejecutamos en **PowerShell** el siguiente comando:

```
./pg_basebackup -h localhost -p 5432 -D "C:/PostgreSQL/Replica/Data" -U replicator -Fp -Xs -P -R
```

Este comando realiza un backup y lo coloca en la carpeta `data` de la segunda instancia. Luego verificamos que el archivo `standby.signal` exista. Además, revisamos que en `postgresql.auto.conf` o `postgresql.conf` esté la siguiente línea:

```
primary_conninfo = 'host=localhost port=5432 user=replicator password=replica'
```

5.4 Inicialización del Servicio de la Segunda Instancia

Este procedimiento nos dio bastantes problemas, pero al final lo logramos ejecutar. A diferencia de los demás, tuvimos que hacerlo desde la línea de comandos de Windows y no desde PowerShell. Adicionalmente, tuvimos que crear un archivo **.BAT** con los parámetros correspondientes de forma que el comando se procese correctamente. En el **.BAT** tenemos el siguiente código:

```
@echo off
"C:/Program Files/PostgreSQL/17/bin/pg_ctl.exe" runservice -N postgresql-replica -D
"C:/PostgreSQL/Replica/Data" -o "-p 5433"
```

Y utilizamos el siguiente comando para crear el servicio:

```
sc create postgresql-replica-server binPath= "C:/PostgreSQL/start_postgres_replica.bat"
DisplayName= "PostgreSQL Replica Server" start= auto
```

Luego verificamos la ejecución de los servicios desde el monitor de servicios de **Windows**.

La explicación que obtuvimos para este comportamiento inesperado es que el comando no estaba procesando correctamente todos los parámetros. Algunos estaban destinados para `pg_ctl.exe` y otros para `sc`, lo que generaba conflictos cuando se incluían todos dentro de la misma línea. Al separar el `binpath` en un archivo `.BAT` con los parámetros correspondientes para `pg_ctl.exe` y dejando los parámetros del programa `sc` en la línea de comandos, logramos resolver el problema.

En este punto, ya podemos verificar la ejecución de ambos servicios desde el monitor de servicios de **Windows**. Podemos observar que el servicio **PostgreSQL Replica Server** está funcionando correctamente (es posible ignorar el servicio `postgresql-replica`, ya que fue un intento fallido durante este proceso de aprendizaje).

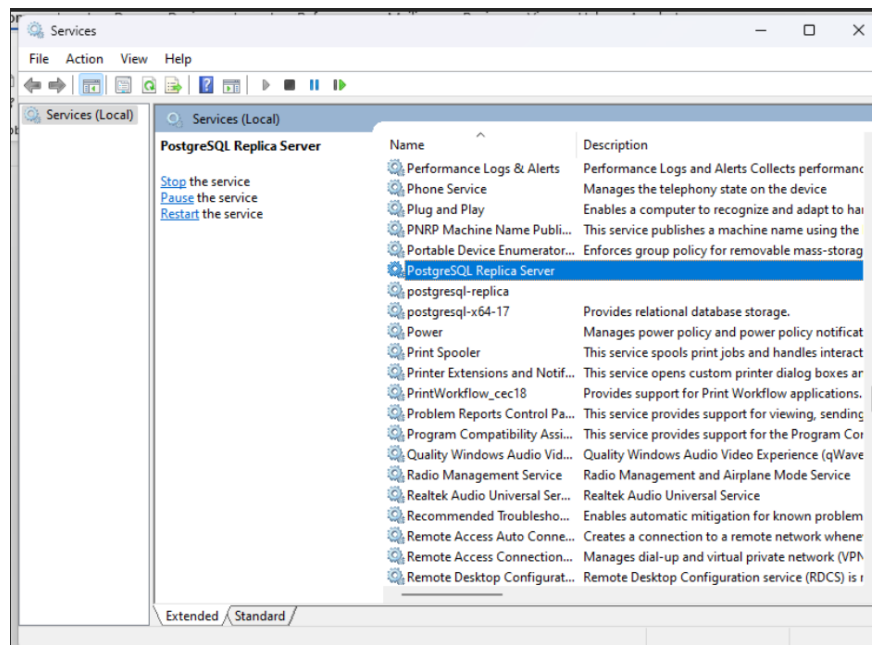


Figure 8: Verificación de PostgreSQL Replica Server.

5.5 Registro de la Réplica en PGAdmin

Finalmente, registramos la réplica en **PGAdmin**. Hacemos clic derecho en **SERVERS/REGISTER/SERVER...** y completamos la configuración:

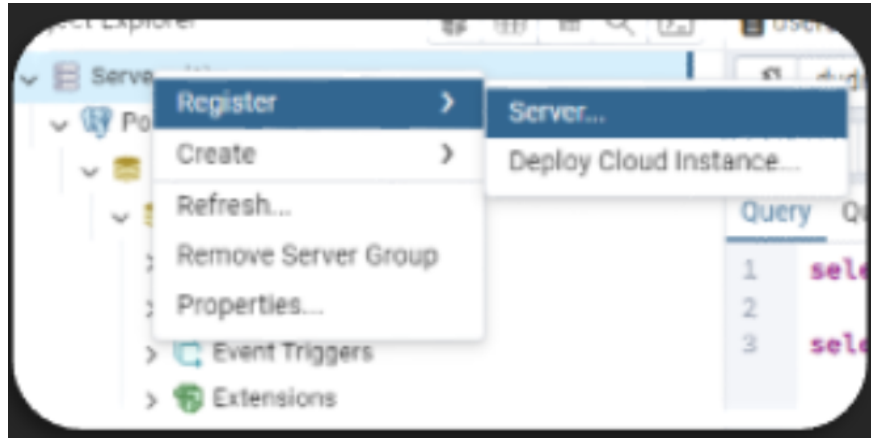


Figure 9: Registro de la réplica en PGAdmin.

Le damos un nombre al servidor, en nuestro caso *Replica*.

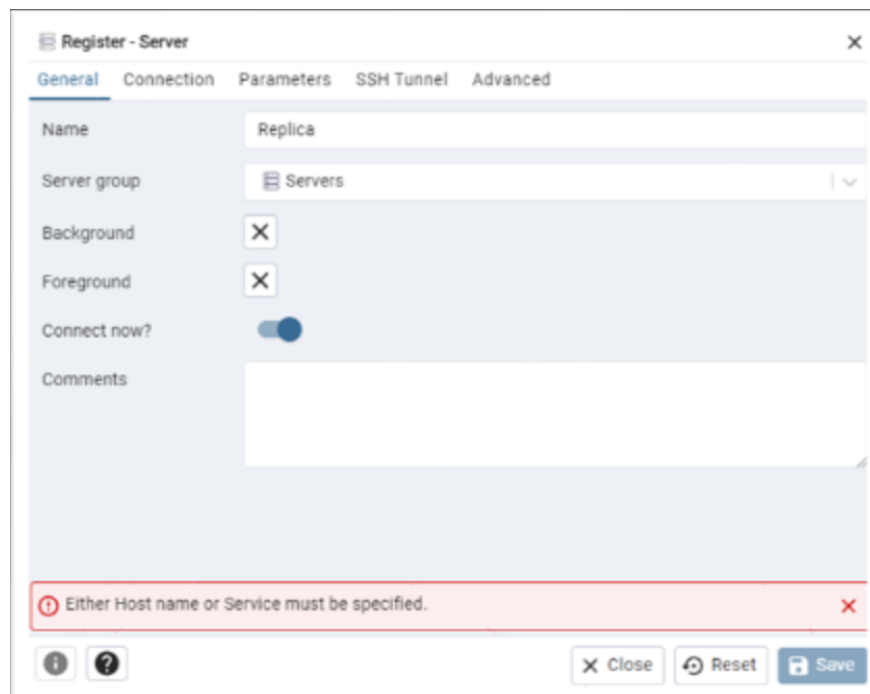
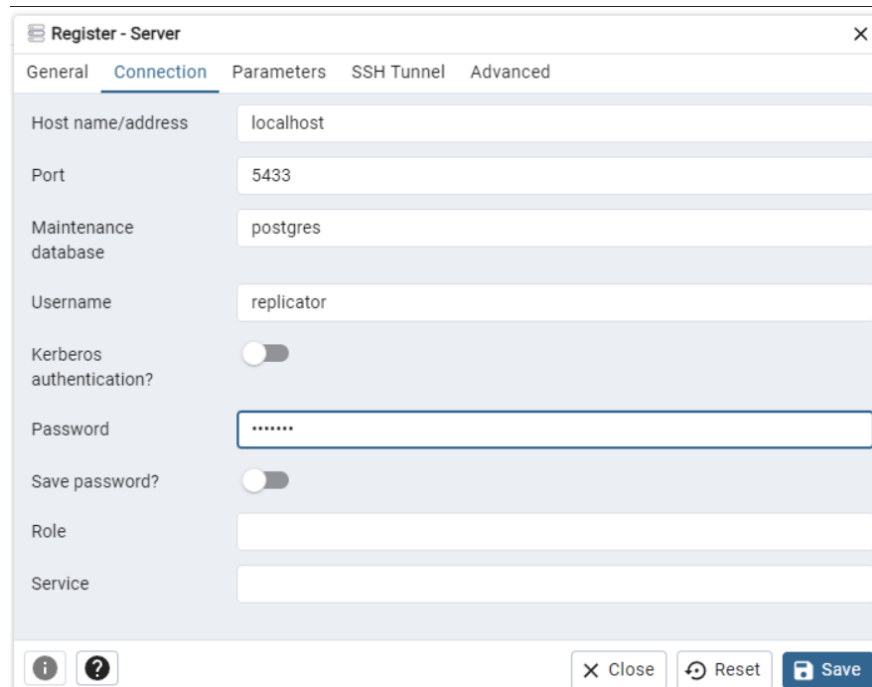


Figure 10: Configuración de la réplica en PGAdmin.

Luego, en la pestaña de *Connection*, introducimos los parámetros del `hostname` (localhost), el puerto (5433), el usuario (`replicator`) y la contraseña (`replica`).



The screenshot shows the 'Register - Server' dialog box in PGAdmin, specifically the 'Connection' tab. The fields are filled with the following values:

- Host name/address: localhost
- Port: 5433
- Maintenance database: postgres
- Username: replicator
- Kerberos authentication?: ☐
- Password: [masked with dots]
- Save password?: ☐
- Role: [empty]
- Service: [empty]

At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

Figure 11: Configuración de conexión en PGAdmin.

Finalmente, el árbol de servidores se debería ver algo así:

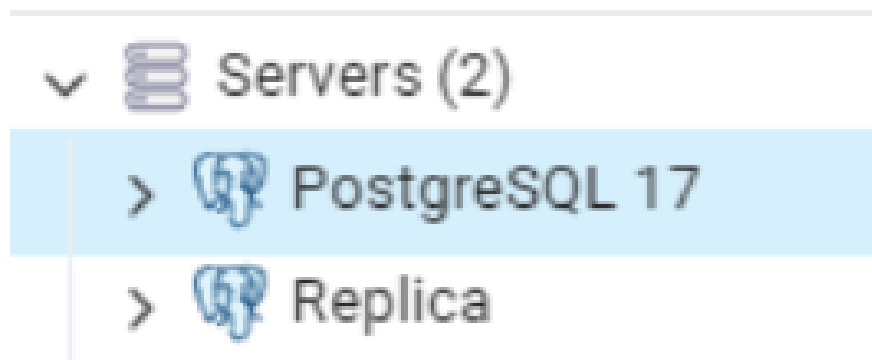


Figure 12: Verificación de la réplica funcionando correctamente.

Con esto, verificamos que todo esté funcionando. Creamos un actor nuevo en la tabla `actor` con el usuario principal (`postgres`), y luego, al conectarnos a la réplica, hicimos un `SELECT * FROM actor` para verificar la existencia del nuevo registro.

6 Modelo Multidimensional

Con el fin de analizar la información acumulada de los alquileres, se ha implementado un modelo multidimensional basado en un esquema en estrella utilizando PostgreSQL. El objetivo es crear un modelo eficiente para la consulta y análisis de las transacciones de alquileres. Este modelo está diseñado para permitir la generación de informes y análisis dinámicos sobre métricas clave como el número de alquileres y el monto total cobrado por alquileres.

6.1 Medidas de Interés

Las medidas de interés para este análisis son las siguientes:

- **Número de alquileres:** El número total de transacciones de alquiler registradas.
- **Monto total cobrado:** La cantidad total de ingresos generados por los alquileres.

6.2 Dimensiones de Interés

El análisis se realizará considerando las siguientes dimensiones:

- **Película (Film):** Jerarquía que incluye categoría, película y actores.
- **Lugar (Address):** Jerarquía que incluye país y ciudad.
- **Fecha (Rental):** Jerarquía que incluye año, mes y día.
- **Sucursal (Store):** Información sobre las tiendas donde se realizaron los alquileres.

A partir de esta información, se ha diseñado y construido un esquema en estrella compuesto por una tabla de hechos (`fact_rentals`) y varias tablas de dimensiones (`dim_film`, `dim_address`, `dim_rental_date`, `dim_store`). Este modelo será la base para futuras visualizaciones y análisis de los datos.

6.3 Diseño del Esquema Estrella

El esquema en estrella incluye las siguientes tablas:

6.3.1 Tabla de Hechos: fact_rentals

La tabla de hechos `fact_rentals` almacena las métricas clave de los alquileres, como el número de transacciones y los ingresos totales generados. Las relaciones con las tablas de dimensiones se realizan a través de claves foráneas.

```
CREATE TABLE fact_rentals (  
    rental_id SERIAL PRIMARY KEY,  
    film_id INT REFERENCES dim_film(film_id),  
    address_id INT REFERENCES dim_address(address_id),  
    store_id INT REFERENCES dim_store(store_id),  
    rental_date DATE REFERENCES dim_rental_date(rental_date),  
    rental_count INT, -- Medida: Numero de alquileres  
    total_amount NUMERIC(10, 2) -- Medida: Ingresos totales por alquiler  
);
```

6.3.2 Tabla de Dimensiones: dim_film

Esta tabla contiene información sobre las películas, incluidas su categoría, título y los actores asociados. Los datos son extraídos de las tablas originales del sistema transaccional y se almacenan en esta dimensión para facilitar el análisis.

```
CREATE TABLE dim_film (  
    film_id INT PRIMARY KEY,  
    title VARCHAR(255),  
    category VARCHAR(100),  
    actor_name VARCHAR(255)  
);
```

6.3.3 Tabla de Dimensiones: dim_address

La tabla `dim_address` almacena información jerárquica sobre las ubicaciones, incluidos el país y la ciudad. Esta tabla permite analizar los alquileres en función de la ubicación geográfica.

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

6.3.4 Tabla de Dimensiones: dim_rental_date

Esta tabla contiene la jerarquía de fechas, lo que permite desglosar los alquileres por año, mes y día. La tabla es utilizada para realizar análisis temporales de las transacciones de alquileres.

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

6.3.5 Tabla de Dimensiones: dim_store

La tabla `dim_store` almacena información sobre las tiendas donde se realizaron los alquileres, lo que permite analizar los datos a nivel de sucursal.

```
CREATE TABLE dim_store (  
    store_id INT PRIMARY KEY,  
    store_name VARCHAR(100)  
);
```

6.4 Procedimientos Almacenados para la Carga de Datos

Para alimentar las tablas del modelo multidimensional, se han desarrollado los siguientes procedimientos almacenados. Estos procedimientos se encargan de extraer los datos desde la base de datos transaccional y cargarlos en las tablas del esquema en estrella.

6.4.1 Procedimiento: load_fact_rentals

Este procedimiento inserta los datos agregados sobre los alquileres en la tabla `fact_rentals`, calculando el número de alquileres y el total de ingresos por transacción.

```
CREATE OR REPLACE FUNCTION load_fact_rentals()
RETURNS VOID AS $$
BEGIN
    -- Eliminar registros existentes para evitar duplicaciones
    DELETE FROM fact_rentals;

    -- Insertar nuevos datos de alquiler agregados
    INSERT INTO fact_rentals (film_id, address_id, store_id, rental_date,
        rental_count, total_amount)
    SELECT
        i.film_id,                -- ID de la pelicula
        a.address_id,            -- ID de la direccion
        s.store_id,              -- ID de la tienda
        r.rental_date,           -- Fecha de alquiler
        COUNT(r.rental_id) AS rental_count, -- Numero de alquileres
        SUM(p.amount) AS total_amount      -- Suma de ingresos
    FROM rental r
    JOIN payment p ON r.rental_id = p.rental_id
    JOIN customer c ON r.customer_id = c.customer_id
    JOIN address a ON c.address_id = a.address_id
    JOIN inventory i ON r.inventory_id = i.inventory_id
    JOIN store s ON i.store_id = s.store_id
    GROUP BY i.film_id, a.address_id, s.store_id, r.rental_date;
END;
$$ LANGUAGE plpgsql;
```


6.4.2 Procedimiento: load_dim_film

Este procedimiento carga la tabla de dimensión de películas con datos provenientes de las tablas de películas, categorías y actores.

```
CREATE OR REPLACE FUNCTION load_dim_film()
RETURNS VOID AS $$
BEGIN
    INSERT INTO dim_film (film_id, title, category, actor_name)
    SELECT
        f.film_id,
        f.title,
        COALESCE(c.name, 'Unknown') AS category,
        COALESCE(STRING_AGG(a.first_name || ' ' || a.last_name, ', '), 'No actors
        listed') AS actor_name
    FROM film f
    LEFT JOIN film_category fc ON f.film_id = fc.film_id
    LEFT JOIN category c ON fc.category_id = c.category_id
    LEFT JOIN film_actor fa ON f.film_id = fa.film_id
    LEFT JOIN actor a ON fa.actor_id = a.actor_id
    GROUP BY f.film_id, f.title, c.name
    ON CONFLICT (film_id) DO NOTHING;
END;
$$ LANGUAGE plpgsql;
```

6.4.3 Procedimiento: load_dim_address

Este procedimiento carga la tabla dim_address con información jerárquica de ubicaciones.

```
CREATE OR REPLACE FUNCTION load_dim_address()
RETURNS VOID AS $$
BEGIN
    INSERT INTO dim_address (address_id, city, country)
    SELECT
        a.address_id,
        ci.city,
        co.country
    FROM address a
    JOIN city ci ON a.city_id = ci.city_id
    JOIN country co ON ci.country_id = co.country_id
    ON CONFLICT (address_id) DO NOTHING;
END;
$$ LANGUAGE plpgsql;
```

6.4.4 Procedimiento: load_dim_rental_date

Este procedimiento carga las fechas en la tabla de dimensión `dim_rental_date`.

```
CREATE OR REPLACE FUNCTION load_dim_rental_date()
RETURNS VOID AS $$
BEGIN
    INSERT INTO dim_rental_date (rental_date, year, month, day)
    SELECT
        DISTINCT r.rental_date,
        EXTRACT(YEAR FROM r.rental_date) AS year,
        EXTRACT(MONTH FROM r.rental_date) AS month,
        EXTRACT(DAY FROM r.rental_date) AS day
    FROM rental r
    ON CONFLICT (rental_date) DO NOTHING;
END;
$$ LANGUAGE plpgsql;
```

6.4.5 Procedimiento: load_dim_store

Este procedimiento carga los datos de las tiendas en la tabla de dimensión `dim_store`.

```
CREATE OR REPLACE FUNCTION load_dim_store()
RETURNS VOID AS $$
BEGIN
    INSERT INTO dim_store (store_id, store_name)
    SELECT
        s.store_id,
        'Store ' || s.store_id AS store_name
    FROM store s
    ON CONFLICT (store_id) DO NOTHING;
END;
$$ LANGUAGE plpgsql;
```

6.4.6 Procedimiento: load_star_schema

Este procedimiento ejecuta todos los procedimientos de carga individuales para poblar el esquema en estrella. Primero se cargan las tablas de dimensiones y luego la tabla de hechos.

```
CREATE OR REPLACE FUNCTION load_star_schema()
RETURNS VOID AS $$
BEGIN
    PERFORM load_dim_film();
    PERFORM load_dim_address();
    PERFORM load_dim_rental_date();
    PERFORM load_dim_store();
    PERFORM load_fact_rentals();
END;
$$ LANGUAGE plpgsql;
```

6.5 Pruebas y Validación

Las siguientes consultas SQL se pueden utilizar para verificar que las tablas del modelo multidimensional han sido correctamente pobladas:

```
-- Verificar los datos en dim_film
SELECT * FROM dim_film LIMIT 10;

-- Verificar los datos en dim_address
SELECT * FROM dim_address LIMIT 10;

-- Verificar los datos en dim_rental_date
SELECT * FROM dim_rental_date LIMIT 10;

-- Verificar los datos en dim_store
SELECT * FROM dim_store LIMIT 10;

-- Verificar los datos en fact_rentals
SELECT * FROM fact_rentals LIMIT 10;
```

Además, se pueden realizar consultas agregadas para verificar los datos de alquileres por película, tienda o por fecha:

```
-- Datos agregados por película
SELECT film_id, SUM(rental_count) AS total_rentals, SUM(total_amount) AS
       total_revenue
FROM fact_rentals
GROUP BY film_id
LIMIT 10;

-- Datos agregados por tienda
SELECT store_id, SUM(rental_count) AS total_rentals, SUM(total_amount) AS
       total_revenue
FROM fact_rentals
GROUP BY store_id
LIMIT 10;

-- Datos agregados por año y mes
SELECT EXTRACT(YEAR FROM rental_date) AS year, EXTRACT(MONTH FROM rental_date) AS
       month,
       SUM(rental_count) AS total_rentals, SUM(total_amount) AS total_revenue
FROM fact_rentals
GROUP BY year, month
LIMIT 10;
```

7 Visualización y acceso por medio de interfaz gráfica al modelo estrella