Universidad de La Laguna

Modelo Relacional. Vistas y Disparadores.

Mikel Mugica e Ibai Heras

C/ Padre Herrera s/n
38207 La Laguna
Santa Cruz de Tenerife. España

T: 900 43 25 26

ull.es

1.Restauración de la bases de datos

El comando que hemos ejecutado para hacer la restauración de la base de datos ha sido la siguiente:

& "C:\Program Files\PostgreSQL\17\bin\pg_restore.exe" -U postgres -d alquilerdvd --clean --verbose "C:\Users\ibaih\Desktop\AlquilerPractica.tar"

2. Identifique las Tablas, Vistas y Secuencias

Tablas(\dt)

- **actor**: Contiene información de los actores.
- **address**: Información de direcciones.
- **category**: Las categorías de películas.
- **city**: Datos sobre las ciudades.
- **country**: Información sobre países.
- **customer**: Información sobre los clientes.
- **film**: Contiene los detalles de las películas.
- **film_actor**: Relación entre películas y actores.
- **film_category**: Relación entre películas y categorías.
- **inventory**: Información de inventario de películas.
- **language**: Información sobre los idiomas de las películas.
- **payment**: Detalles sobre los pagos de los alquileres.
- **rental**: Información de los alquileres realizados.
- **staff:** Información sobre los empleados o encargados.
- **store**: Información sobre las tiendas de alquiler de películas.

```
alquilerdvd=# \dt
           Listado de relaciones
 Esquema |    Nombre     | Tipo  |  Dueño
---------+---------------+-------+----------
 public  | actor         | tabla | postgres
 public  | address       | tabla | postgres
 public  | category      | tabla | postgres
 public  | city          | tabla | postgres
 public  | country       | tabla | postgres
 public  | customer      | tabla | postgres
 public  | film          | tabla | postgres
 public  | film_actor    | tabla | postgres
 public  | film_category | tabla | postgres
 public  | inventory     | tabla | postgres
 public  | language      | tabla | postgres
 public  | payment       | tabla | postgres
 public  | rental        | tabla | postgres
 public  | staff         | tabla | postgres
 public  | store         | tabla | postgres
(15 filas)
```

Vistas(\dv)

Las vistas no están especificadas inicialmente, pero pueden crearse a partir de consultas complejas. Las vistas de las consultas que se mencionan a continuación podrían crearse con el prefijo view_ en sus nombres.

En esta base de datos no se han encontrado vistas definidas:

```
alquilerdvd=# \dv
No se encontró ninguna relación.
```

Secuencias(\ds)

Las secuencias generalmente se usan para generar valores únicos, típicamente para las claves primarias. Por ejemplo:

- film_id en la tabla film
- actor_id en la tabla actor
- customer_id en la tabla customer

Las secuencias se crean de forma automática cuando se utiliza un tipo de columna SERIAL o BIGSERIAL, o se pueden crear explícitamente mediante un comando CREATE SEQUENCE.

En esta baso de datos, estas son las secuencias definidas:

```
alquilerdvd=# \ds
                  Listado de relaciones
 Esquema |           Nombre           |   Tipo    |  Due±o
---------+----------------------------+-----------+----------
 public  | actor_actor_id_seq         | secuencia | postgres
 public  | address_address_id_seq     | secuencia | postgres
 public  | category_category_id_seq   | secuencia | postgres
 public  | city_city_id_seq           | secuencia | postgres
 public  | country_country_id_seq     | secuencia | postgres
 public  | customer_customer_id_seq   | secuencia | postgres
 public  | film_film_id_seq           | secuencia | postgres
 public  | inventory_inventory_id_seq | secuencia | postgres
 public  | language_language_id_seq   | secuencia | postgres
 public  | payment_payment_id_seq     | secuencia | postgres
 public  | rental_rental_id_seq       | secuencia | postgres
 public  | staff_staff_id_seq         | secuencia | postgres
 public  | store_store_id_seq         | secuencia | postgres
(13 filas)
```

3. Identifique las Tablas Principales y sus Principales Elementos

Las tablas principales de este modelo de base de datos son las siguientes:

film

- film_id: Identificador único de la película.
- title: Título de la película.
- description: Descripción de la película.
- release_year: Año de estreno.
- rental_rate: Precio de alquiler.
- length: Duración de la película.
- rating: Clasificación de la película.

category

- category_id: Identificador único de la categoría.
- name: Nombre de la categoría.

customer

- customer_id: Identificador único del cliente.
- first_name: Primer nombre del cliente.
- last_name: Apellido del cliente.
- email: Correo electrónico del cliente.
- address_id: Identificador de la dirección del cliente.

rental

- rental_id: Identificador único del alquiler.
- rental_date: Fecha de alquiler.
- inventory_id: Identificador del inventario.
- customer_id: Identificador del cliente que alquila.
- return_date: Fecha de devolución.

payment

- payment_id: Identificador único del pago.
- amount: Monto del pago.
- payment_date: Fecha del pago.

store

- store_id: Identificador de la tienda.
- manager_staff_id: Identificador del empleado encargado.
- address_id: Identificador de la dirección de la tienda.

4. Realice las siguientes Consultas

4.1. Ventas totales por categoría de películas ordenadas descendentemente:

```
SELECT c.name AS category, SUM(p.amount) AS total_sales
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
JOIN film f ON fc.film_id = f.film_id
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
JOIN payment p ON r.rental_id = p.rental_id
GROUP BY c.name
ORDER BY total_sales DESC;
```

```
  category   | total_sales
-------------+-------------
 Sports      |     4892.19
 Sci-Fi      |     4336.01
 Animation   |     4245.31
 Drama       |     4118.46
 Comedy      |     4002.48
 New         |     3966.38
 Action      |     3951.84
 Foreign     |     3934.47
 Games       |     3922.18
 Family      |     3830.15
 Documentary |     3749.65
 Horror      |     3401.27
 Classics    |     3353.38
 Children    |     3309.39
 Travel      |     3227.36
 Music       |     3071.52
(16 filas)
```

4.2. Ventas totales por tienda, con ciudad, país y encargado:

```
SELECT s.store_id,
    CONCAT(ct.city, ', ', co.country) AS location,
    CONCAT(st.first_name, ' ', st.last_name) AS manager,
    SUM(p.amount) AS total_sales
FROM store s
JOIN address a ON s.address_id = a.address_id
JOIN city ct ON a.city_id = ct.city_id
JOIN country co ON ct.country_id = co.country_id
JOIN staff st ON s.manager_staff_id = st.staff_id
JOIN inventory i ON s.store_id = i.store_id
JOIN rental r ON i.inventory_id = r.inventory_id
JOIN payment p ON r.rental_id = p.rental_id
GROUP BY s.store_id, ct.city, co.country, st.first_name, st.last_name

ORDER BY total_sales DESC;
```

```
 store_id |       location       |   manager    | total_sales
----------+----------------------+--------------+-------------
        2 | Woodridge, Australia | Jon Stephens |    30683.13
        1 | Lethbridge, Canada   | Mike Hillyer |    30628.91
(2 filas)
```

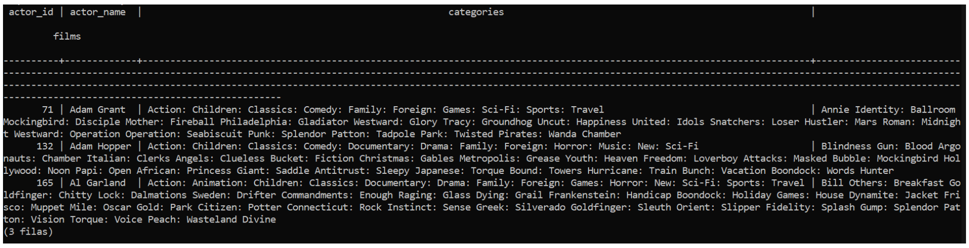
4.3. Lista de películas con detalles, actores y categoría:

```
SELECT f.film_id,
f.title,
    f.description,
    c.name AS category,
    f.rental_rate AS price,
    f.length AS duration,
    f.rating,
    STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ') AS actors
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, f.description, c.name, f.rental_rate, f.length, f.rating
ORDER BY f.title
LIMIT 3;
```

```
 actor_id | actor_name  |                                                categories                                                |
        films
----------+-------------+----------------------------------------------------------------------------------------------------------+-----------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------
       71 | Adam Grant  | Action: Children: Classics: Comedy: Family: Foreign: Games: Sci-Fi: Sports: Travel                       | Annie Identity: Ballroom
Mockingbird: Disciple Mother: Fireball Philadelphia: Gladiator Westward: Glory Tracy: Groundhog Uncut: Happiness United: Idols Snatchers: Loser Hustler: Mars Roman: Midnigh
t Westward: Operation Operation: Seabiscuit Punk: Splendor Patton: Tadpole Park: Twisted Pirates: Wanda Chamber
      132 | Adam Hopper | Action: Children: Classics: Comedy: Documentary: Drama: Family: Foreign: Horror: Music: New: Sci-Fi     | Blindness Gun: Blood Argo
nauts: Chamber Italian: Clerks Angels: Clueless Bucket: Fiction Christmas: Gables Metropolis: Grease Youth: Heaven Freedom: Loverboy Attacks: Masked Bubble: Mockingbird Hol
lywood: Noon Papi: Open African: Princess Giant: Saddle Antitrust: Sleepy Japanese: Torque Bound: Towers Hurricane: Train Bunch: Vacation Boondock: Words Hunter
      165 | Al Garland  | Action: Animation: Children: Classics: Documentary: Drama: Family: Foreign: Games: Horror: New: Sci-Fi: Sports: Travel | Bill Others: Breakfast Go
ldfinger: Chitty Lock: Dalmations Sweden: Drifter Commandments: Enough Raging: Glass Dying: Grail Frankenstein: Handicap Boondock: Holiday Games: House Dynamite: Jacket Fri
sco: Muppet Mile: Oscar Gold: Park Citizen: Potter Connecticut: Rock Instinct: Sense Greek: Silverado Goldfinger: Sleuth Orient: Slipper Fidelity: Splash Gump: Splendor Pat
ton: Vision Torque: Voice Peach: Wasteland Divine
(3 filas)
```

4.4. Información de los actores con categorías y películas:

```
SELECT a.actor_id,
    CONCAT(a.first_name, ' ', a.last_name) AS actor_name,
    STRING_AGG(DISTINCT c.name, ': ') AS categories,
    STRING_AGG(DISTINCT f.title, ': ') AS films
FROM actor a
JOIN film_actor fa ON a.actor_id = fa.actor_id
JOIN film f ON fa.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
GROUP BY a.actor_id, a.first_name, a.last_name
ORDER BY actor_name
LIMIT 3;
```



```
 actor_id | actor_name  |                                   categories                                   |
        films
----------+-------------+--------------------------------------------------------------------------------+------------------------
       71 | Adam Grant  | Action: Children: Classics: Comedy: Family: Foreign: Games: Sci-Fi: Sports: Travel | Annie Identity: Ballroom
Mockingbird: Disciple Mother: Fireball Philadelphia: Gladiator Westward: Glory Tracy: Groundhog Uncut: Happiness United: Idols Snatchers: Loser Hustler: Mars Roman: Midnigh
t Westward: Operation Operation: Seabiscuit Punk: Splendor Patton: Tadpole Park: Twisted Pirates: Wanda Chamber
      132 | Adam Hopper | Action: Children: Classics: Comedy: Documentary: Drama: Family: Foreign: Horror: Music: New: Sci-Fi | Blindness Gun: Blood Argo
nauts: Chamber Italian: Clerks Angels: Clueless Bucket: Fiction Christmas: Gables Metropolis: Grease Youth: Heaven Freedom: Loverboy Attacks: Masked Bubble: Mockingbird Hol
lywood: Noon Papi: Open African: Princess Giant: Saddle Antitrust: Sleepy Japanese: Torque Bound: Towers Hurricane: Train Bunch: Vacation Boondock: Words Hunter
      165 | Al Garland  | Action: Animation: Children: Classics: Documentary: Drama: Family: Foreign: Games: Horror: New: Sci-Fi: Sports: Travel | Bill Others: Breakfast Go
ldfinger: Chitty Lock: Dalmations Sweden: Drifter Commandments: Enough Raging: Glass Dying: Grail Frankenstein: Handicap Boondock: Holiday Games: House Dynamite: Jacket Fri
sco: Muppet Mile: Oscar Gold: Park Citizen: Potter Connecticut: Rock Instinct: Sense Greek: Silverado Goldfinger: Sleuth Orient: Slipper Fidelity: Splash Gump: Splendor Pat
ton: Vision Torque: Voice Peach: Wasteland Divine
(3 filas)
```

5. Realice Todas las Vistas de las Consultas Anteriores

-- Vista para las ventas totales por categoría de películas

```
CREATE VIEW view_sales_by_category AS
SELECT c.name AS category, SUM(p.amount) AS total_sales
FROM category c
JOIN film_category fc ON c.category_id = fc.category_id
JOIN film f ON fc.film_id = f.film_id
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
JOIN payment p ON r.rental_id = p.rental_id
GROUP BY c.name
ORDER BY total_sales DESC;
```

-- Vista para las ventas totales por tienda, ciudad, país y encargado

```
CREATE VIEW view_sales_by_store AS
SELECT s.store_id,
    CONCAT(ct.city, ', ', co.country) AS location,
    CONCAT(st.first_name, ' ', st.last_name) AS manager,
    SUM(p.amount) AS total_sales
FROM store s
JOIN address a ON s.address_id = a.address_id
JOIN city ct ON a.city_id = ct.city_id
JOIN country co ON ct.country_id = co.country_id
JOIN staff st ON s.manager_staff_id = st.staff_id
JOIN inventory i ON s.store_id = i.store_id
JOIN rental r ON i.inventory_id = r.inventory_id
```

```
JOIN payment p ON r.rental_id = p.rental_id
GROUP BY s.store_id, ct.city, co.country, st.first_name, st.last_name
ORDER BY total_sales DESC;

-- Vista para la lista de películas con detalles, actores y categoría
CREATE VIEW view_film_details AS
SELECT f.film_id,
    f.title,
    f.description,
    c.name AS category,
    f.rental_rate AS price,
    f.length AS duration,
    f.rating,
    STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ') AS actors
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, f.description, c.name, f.rental_rate, f.length, f.rating
ORDER BY f.title;
```

-- Vista para la información de los actores, categorías y películas

```
CREATE VIEW view_actors_categories_films AS
SELECT a.actor_id,
    CONCAT(a.first_name, ' ', a.last_name) AS actor_name,
    STRING_AGG(DISTINCT c.name, ': ') AS categories,
    STRING_AGG(DISTINCT f.title, ': ') AS films
FROM actor a
JOIN film_actor fa ON a.actor_id = fa.actor_id
JOIN film f ON fa.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
GROUP BY a.actor_id, a.first_name, a.last_name
ORDER BY actor_name;
```

Aquí están todas las vistas definidas anteriormente:

```
alquilerdvd=# \dv
                  Listado de relaciones
 Esquema |            Nombre             | Tipo  | Dueto
---------+-------------------------------+-------+-------
 public  | view_actors_categories_films  | vista | Mikel
 public  | view_film_details             | vista | Mikel
 public  | view_sales_by_category        | vista | Mikel
 public  | view_sales_by_store           | vista | Mikel
(4 filas)
```

6. Análisis del Modelo e Inclusión de Restricciones CHECK

Restricciones CHECK: Pueden agregarse restricciones para asegurar la integridad de los datos. Ejemplo:

En la tabla payment, se podría agregar una restricción CHECK para asegurarse de que el amount sea siempre mayor o igual a 0:

ALTER TABLE payment

ADD CONSTRAINT check_amount_positive CHECK (amount >= 0);

```
alquilerdvd=# ALTER TABLE payment
alquilerdvd-# ADD CONSTRAINT check_amount_positive CHECK (amount >= 0);
ALTER TABLE
```

En la tabla film, se podría agregar una restricción para verificar que la duración de la película no sea negativa:

ALTER TABLE film

ADD CONSTRAINT check_duration_positive CHECK (length > 0);

```
alquilerdvd=# ALTER TABLE film
alquilerdvd-# ADD CONSTRAINT check_duration_positive CHECK (length > 0);
ALTER TABLE
```

En la tabla rental, se podría agregar una restricción para verificar que la fecha de devolución es mayor que la fecha de alquiler, esta no se puede devolver antes de alquilarla:

ALTER TABLE rental

ADD CONSTRAINT check_rental_before_devolution CHECK (rental_date < return_date)

```
alquilerdvd=# ALTER TABLE rental
alquilerdvd-# ADD CONSTRAINT check_rental_before_devolution CHECK (rental_date < return_date)
alquilerdvd-# ;
ALTER TABLE
```

En la tabla film, se podría agregar una restricción CHECK para asegurarse de que el rental_rate sea siempre mayor a 0:

ALTER TABLE film

ADD CONSTRAINT check_rental_rate_positive CHECK (rental_rate > 0);

```
alquilerdvd=# ALTER TABLE film
alquilerdvd-# ADD CONSTRAINT check_rental_rate_positive CHECK (rental_rate > 0);
ALTER TABLE
```

7. Explicación de la Sentencia en la Tabla customer (Trigger)

La sentencia last_updated BEFORE UPDATE ON customer FOR EACH ROW EXECUTE PROCEDURE last_updated() se usa para crear un trigger que se ejecuta antes de actualizar cualquier registro en la tabla customer. Esta acción permite modificar automáticamente los datos de un registro antes de ser actualizado, como agregar la fecha de última modificación.

Ejemplo de uso:

CREATE TRIGGER last_updated

BEFORE UPDATE ON customer

FOR EACH ROW

EXECUTE FUNCTION last_updated();

Tabla Similar:

Una solución similar se utiliza en la tabla rental, donde se pueden actualizar automáticamente los campos como la fecha de devolución cuando se actualiza un alquiler.

```
 trigger_name | table_name | trigger_events | trigger_timing |        trigger_function
--------------+------------+----------------+----------------+--------------------------------
 last_updated | rental     | UPDATE         | BEFORE         | EXECUTE FUNCTION last_updated()
(1 fila)
```

Para visualizar los triggers de una tabla hemos usado un comando sql que encontramos en internet.

8. Crear Trigger para Insertar en Nueva Tabla

Hemos creado esta nueva tabla para almacenar los registros de las nuevas inserciones de los datos en la tabla film. En esta tabla se registra la fecha en la que se insertó dicho dato.

```
CREATE TABLE film_insert_log (
    film_id INT,
    insert_date TIMESTAMP
);
```

En esta parte definimos el funcionamiento (la función) del disparador.

```
CREATE OR REPLACE FUNCTION log_film_insert() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO film_insert_log (film_id, insert_date)
    VALUES (NEW.film_id, NOW());
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

En la última parte se crea el disparador que llama a la función después de una inserción en la tabla film

```
CREATE TRIGGER trigger_film_insert
AFTER INSERT ON film
FOR EACH ROW EXECUTE FUNCTION log_film_insert();
```

```
alquilerdvd=# INSERT INTO public.film (title, description, release_year, language_id, rental_duration, rental_rate, leng
th, replacement_cost, rating, special_features, fulltext)
alquilerdvd-# VALUES ('New Film', 'Description of the new film', 2023, 1, 3, 4.99, 120, 19.99, 'PG', '{}', to_tsvector('
New Film Description'));
INSERT 0 1
alquilerdvd=# SELECT * FROM public.film_insert_log;
 film_id |        insert_date
---------+----------------------------
    1001 | 2024-11-12 16:25:02.589178
```

9. Crear Trigger para Eliminar en Nueva Tabla

Hemos creado esta nueva tabla para almacenar los registros de los datos eliminados de la tabla film. En esta tabla se registra la fecha en la que se eliminó dicho dato.

```
CREATE TABLE film_delete_log (
    film_id INT,
    delete_date TIMESTAMP
);
```

En esta parte definimos el funcionamiento (la función) del disparador.

```
CREATE OR REPLACE FUNCTION log_film_delete() RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO film_delete_log (film_id, delete_date)
    VALUES (OLD.film_id, NOW());
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

En la última parte se crea el disparador que llama a la función después de eliminar una fila en la tabla film

```
CREATE TRIGGER trigger_film_delete
AFTER DELETE ON film
FOR EACH ROW EXECUTE FUNCTION log_film_delete();
```

```
alquilerdvd=# DELETE FROM public.film WHERE title = 'New Film';
DELETE 1
alquilerdvd=# SELECT * FROM public.film_delete_log;
 film_id |        delete_date
---------+----------------------------
    1001 | 2024-11-12 16:42:43.771204
(1 fila)
```

10. Significado y Relevancia de las Secuencias

Las secuencias son objetos en bases de datos que permiten generar valores únicos, normalmente usados para claves primarias. Son importantes para asegurar la unicidad de los identificadores en una tabla y evitar colisiones. Se utilizan para asignar valores automáticos y consecutivos sin la necesidad de intervención manual, lo que mejora la eficiencia y la consistencia de los datos.