Alquiler

Modelo Relacional. Vistas y disparadores

alu0101221953

Cánovas del Pino, Víctor

Identifique las tablas, vistas y secuencias.

- Con el comando **\dt** (display tables) podemos listar todas las tablas de la base de datos.

```
alquiler=# \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)
```

- Con el comando **\dv** (display views) podemos ver todas las vistas creadas.

```
alquiler=# \dv
No se encontró ninguna relación.
alquiler-#
```

En este caso no hay vistas creadas.

- Con el comando **\ds** (display sequences) podemos listar todas las secuencias de la base de datos.

```
alquiler=# \ds
                    Listado de relaciones
 Esquema |            Nombre             |   Tipo    |  Dueño
---------+-------------------------------+-----------+----------
 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)
```

Identifique las tablas principales y sus principales elementos.

actor

Contiene información sobre los actores de las películas.

- actor_id: Identificador único del actor (clave primaria).
- first_name, last_name: Nombre y apellido del actor.
- last_update: Fecha de la última actualización del registro.

address

Almacena las direcciones utilizadas para clientes, empleados y tiendas.

- address_id: Identificador único de la dirección (clave primaria).
- address, address2: Dirección principal y dirección secundaria.
- district: Distrito o región.
- city_id: Clave foránea que referencia la ciudad (city).
- postal_code, phone: Código postal y número de teléfono.
- last_update: Fecha de la última actualización del registro.

category

Define las categorías o géneros de las películas.

- category_id: Identificador único de la categoría (clave primaria).
- name: Nombre de la categoría (por ejemplo, Acción, Comedia).
- last_update: Fecha de la última actualización del registro.

customer

Contiene la información de los clientes que realizan alquileres.

- customer_id: Identificador único del cliente (clave primaria).
- first_name, last_name: Nombre y apellido del cliente.
- email: Correo electrónico del cliente.
- address_id: Clave foránea que referencia la dirección (address).
- activebool: Indica si el cliente está activo.
- create_date: Fecha en la que se registró el cliente.
- last_update: Fecha de la última actualización del registro.

film

Contiene la información de las películas disponibles en la base de datos.

- film_id: Identificador único de la película (clave primaria).
- title: Título de la película.
- description: Descripción breve de la película.
- release_year: Año de lanzamiento de la película.
- language_id: Clave foránea que referencia el idioma de la película (language).
- rental_duration: Duración del alquiler.
- rental_rate: Precio del alquiler.
- length: Duración de la película en minutos.
- rating: Clasificación de la película (por ejemplo, PG, R).
- last_update: Fecha de la última actualización del registro.

inventory

Representa el inventario de copias de películas disponibles en cada tienda.

- inventory_id: Identificador único del inventario (clave primaria).
- film_id: Clave foránea que referencia la película (film).
- store_id: Clave foránea que referencia la tienda (store).
- last_update: Fecha de la última actualización del registro.

payment

Registra los pagos realizados por los clientes.

- payment_id: Identificador único del pago (clave primaria).
- customer_id: Clave foránea que referencia al cliente (customer).
- staff_id: Clave foránea que referencia al empleado que procesó el pago (staff).
- rental_id: Clave foránea que referencia al alquiler (rental).
- amount: Monto del pago.
- payment_date: Fecha y hora en la que se realizó el pago.

rental

Registra la información de los alquileres de películas.

- rental_id: Identificador único del alquiler (clave primaria).
- rental_date: Fecha y hora en que se realizó el alquiler.
- inventory_id: Clave foránea que referencia el inventario (inventory).
- customer_id: Clave foránea que referencia al cliente (customer).
- return_date: Fecha y hora en que se devolvió la película.
- staff_id: Clave foránea que referencia al empleado que gestionó el alquiler (staff).
- last_update: Fecha de la última actualización del registro.

staff

Contiene información sobre el personal de la tienda.

- staff_id: Identificador único del empleado (clave primaria).
- first_name, last_name: Nombre y apellido del empleado.
- address_id: Clave foránea que referencia la dirección (address).
- store_id: Clave foránea que referencia la tienda (store).
- active: Indica si el empleado está activo.
- username: Nombre de usuario del empleado.
- last_update: Fecha de la última actualización del registro.

store

Define la información de cada tienda.

- store_id: Identificador único de la tienda (clave primaria).
- manager_staff_id: Clave foránea que referencia al gerente de la tienda (staff).
- address_id: Clave foránea que referencia la dirección (address).
- last_update: Fecha de la última actualización del registro.

Realice las siguientes consultas.

Ventas totales por categoría de películas ordenadas descendentemente.

```
SELECT
    c.name AS categoria,
    COUNT(*) AS ventas
FROM
    payment p
JOIN
    rental r ON p.rental_id = r.rental_id
JOIN
    inventory i ON r.inventory_id = i.inventory_id
JOIN
    film_category fc ON i.film_id = fc.film_id
JOIN
    category c ON fc.category_id = c.category_id
GROUP BY
    c.name
ORDER BY
    ventas DESC;
```

Ventas totales por tienda, donde se refleje la ciudad, el país (concatenar la ciudad y el país empleando como separador la “,”), y el encargado.

```
SELECT
   s.store_id AS tienda,
   CONCAT(ci.city, ', ', co.country) AS ubicacion,
   CONCAT(st.staff_id, ‘: ‘, st.first_name, ' ', st.last_name) AS encargado,
   COUNT(*) AS ventas
FROM
   payment p
JOIN
   rental r ON p.rental_id = r.rental_id
JOIN
   inventory i ON r.inventory_id = i.inventory_id
JOIN
   store s ON i.store_id = s.store_id
JOIN
   address a ON s.address_id = a.address_id
JOIN
   city ci ON a.city_id = ci.city_id
JOIN
   country co ON ci.country_id = co.country_id
JOIN
   staff st ON s.manager_staff_id = st.staff_id
GROUP BY
   s.store_id, ci.city, co.country, st.staff_id, st.first_name, st.last_name
ORDER BY
   ventas DESC;
```

Lista de películas, donde se reflejen el identificador, el título, descripción, categoría, el precio, la duración de la película, clasificación, nombre y apellidos de los actores (puede realizar una concatenación de ambos).

```
SELECT
   f.film_id AS pelicula_id,
   f.title AS titulo,
   f.description AS descripcion,
   c.name AS categoria,
   f.rental_rate AS precio,
   f.length AS duracion,
   f.rating AS clasificacion,
   STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ') AS actores
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;
```

Información de los actores, donde se incluya sus nombres y apellidos, las categorías y sus películas. Los actores deben de estar agrupados y, las categorías y las películas deben estar concatenados por ":"

```
SELECT
   a.first_name AS nombre,
   a.last_name AS apellidos,
   STRING_AGG(DISTINCT c.name || ': ' || f.title, ', ') AS categoria_pelicula
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.first_name, a.last_name;
```

Realice todas las vistas de las consultas anteriores. Colóqueles el prefijo **view_** a su denominación.

Ventas totales por categoría de películas ordenadas descendentemente.

```
CREATE VIEW view_ventas_categoria AS
SELECT
   c.name AS categoria,
   COUNT(*) AS ventas
FROM
   payment p
JOIN
   rental r ON p.rental_id = r.rental_id
JOIN
   inventory i ON r.inventory_id = i.inventory_id
JOIN
   film_category fc ON i.film_id = fc.film_id
JOIN
   category c ON fc.category_id = c.category_id
GROUP BY
   c.name
ORDER BY
   ventas DESC;
```

Ventas totales por tienda, donde se refleje la ciudad, el país (concatenar la ciudad y el país empleando como separador la “,”), y el encargado.

```
CREATE VIEW view_ventas_tienda AS
SELECT
   s.store_id AS tienda,
   CONCAT(ci.city, ', ', co.country) AS ubicacion,
   CONCAT(st.staff_id, ‘: ‘, st.first_name, ' ', st.last_name) AS encargado,
   COUNT(*) AS ventas
FROM
   payment p
JOIN
   rental r ON p.rental_id = r.rental_id
JOIN
   inventory i ON r.inventory_id = i.inventory_id
JOIN
   store s ON i.store_id = s.store_id
JOIN
   address a ON s.address_id = a.address_id
JOIN
   city ci ON a.city_id = ci.city_id
JOIN
   country co ON ci.country_id = co.country_id
JOIN
   staff st ON s.manager_staff_id = st.staff_id
GROUP BY
   s.store_id, ci.city, co.country, st.staff_id, st.first_name, st.last_name
ORDER BY
   ventas DESC;
```

Lista de películas, donde se reflejen el identificador, el título, descripción, categoría, el precio, la duración de la película, clasificación, nombre y apellidos de los actores (puede realizar una concatenación de ambos).

```
CREATE VIEW view_lista_peliculas AS
SELECT
   f.film_id AS pelicula_id,
   f.title AS titulo,
   f.description AS descripcion,
   c.name AS categoria,
   f.rental_rate AS precio,
   f.length AS duracion,
   f.rating AS clasificacion,
   STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ') AS actores
```

```
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;
```

Información de los actores, donde se incluya sus nombres y apellidos, las categorías y sus películas. Los actores deben de estar agrupados y, las categorías y las películas deben estar concatenados por ":"

```
CREATE VIEW view_informacion_actores AS
SELECT
   a.first_name AS nombre,
   a.last_name AS apellidos,
   STRING_AGG(DISTINCT c.name || ': ' || f.title, ', ') AS categoria_pelicula
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.first_name, a.last_name;
```

```
alquiler=# \dv
                 Listado de relaciones
 Esquema |           Nombre           | Tipo  |  Dueño
---------+----------------------------+-------+----------
 public  | view_informacion_actores   | vista | postgres
 public  | view_lista_peliculas       | vista | postgres
 public  | view_ventas_categoria      | vista | postgres
 public  | view_ventas_tienda         | vista | postgres
(4 filas)
```

Haga un análisis del modelo e incluya las restricciones CHECK que considere necesarias.

Análisis del Modelo

Tablas Principales

- **actor**: Contiene los datos de los actores, vinculada a film a través de la tabla intermedia **film_actor**.
- **address**: Almacena las direcciones, vinculada a city, que a su vez está relacionada con **country**.
- **category**: Define las categorías o géneros de las películas, relacionadas con film a través de la tabla **film_category**.
- **customer**: Contiene información sobre los clientes, con una relación de dirección y un vínculo a la tabla de **store**.
- **film**: Tabla principal de las películas, con información detallada como título, duración, clasificación y categoría.
- **inventory**: Registro de copias de cada película en cada tienda.
- **payment**: Registra los pagos realizados por los clientes en cada alquiler.
- **rental**: Contiene la información de los alquileres, vinculada a **customer**, **staff**, y **inventory**.
- **staff**: Contiene la información del personal, cada empleado está vinculado a una tienda.
- **store**: Define las tiendas, con su dirección y gerente.

Relaciones

- Se utilizan tablas intermedias (**film_actor** y **film_category**) para las relaciones de muchos a muchos entre **film** y **actor**, y **film** y **category**.

Restricciones **CHECK** Recomendadas

Tabla film

Duración de Alquiler (rental_duration): Asegura que la duración del alquiler sea positiva.
ALTER TABLE film ADD CONSTRAINT chk_rental_duration CHECK (rental_duration > 0);

Tarifa de Alquiler (rental_rate): Asegura que el precio sea positivo.
ALTER TABLE film ADD CONSTRAINT chk_rental_rate CHECK (rental_rate >= 0);

Costo de Reemplazo (replacement_cost): Asegura que el costo de reemplazo sea mayor que cero.
ALTER TABLE film ADD CONSTRAINT chk_replacement_cost CHECK (replacement_cost > 0);

Tabla payment

Monto (amount): Asegura que los montos de pago sean valores positivos.
ALTER TABLE payment ADD CONSTRAINT chk_amount CHECK (amount >= 0);

Tabla customer

Actividad (activebool): Asegura que el estado de actividad solo sea verdadero o falso (si usa un entero en lugar de booleano).
ALTER TABLE customer ADD CONSTRAINT chk_active_bool CHECK (activebool IN (true, false));

Tabla rental

Fecha de Devolución (return_date): Asegura que la fecha de devolución sea posterior a la fecha de alquiler.
ALTER TABLE rental ADD CONSTRAINT chk_return_date CHECK (return_date >= rental_date);

Tabla inventory

Filme Disponible (film_id): Asegura que cada inventario se vincule a una película existente.
ALTER TABLE inventory ADD CONSTRAINT chk_film_exists CHECK (film_id IS NOT NULL);

Explique la sentencia que aparece en la tabla customer

Triggers:

```
last_updated BEFORE UPDATE ON customer

FOR EACH ROW EXECUTE PROCEDURE last_updated()
```

El trigger **last_updated** en la tabla **customer** se activa antes de cada actualización (**UPDATE**) en cualquier registro de cliente. Su función es actualizar automáticamente el campo **last_update** con la fecha y hora actuales, permitiendo llevar un registro de la última modificación de cada cliente. Este trigger se ejecuta por cada fila modificada, asegurando que cada cambio en la información de un cliente quede registrado con su correspondiente fecha de actualización.

Una solución similar a la del trigger **last_updated** en la tabla **customer** se utiliza en otras tablas de esta base de datos, como **actor**, **address**, **category**, **film**, **inventory**, **rental**, y **staff**. En todas estas tablas, existe un campo llamado **last_update** que almacena la última fecha de modificación del registro, y un trigger **last_updated** se encarga de actualizar automáticamente este campo cada vez que se modifica el registro.

Construya un disparador que guarde en una nueva tabla creada por usted la fecha de cuando se insertó un nuevo registro en la tabla film.

Crear la Tabla **film_registers**

```
CREATE TABLE film_registers (
   reg_id SERIAL PRIMARY KEY,
   film_id INT NOT NULL,
   insert_date TIMESTAMP NOT NULL DEFAULT NOW()
);
```

Crear la Función del Disparador

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

Crear el Disparador en la Tabla **film**

```
CREATE TRIGGER after_film_insert
AFTER INSERT ON film
FOR EACH ROW
EXECUTE FUNCTION registers_film_insertion();
```

Construya un disparador que guarde en una nueva tabla creada por usted la fecha de cuando se eliminó un registro en la tabla **film** y el identificador del **film**.

Crear la Tabla **film_delete_reg**

```
CREATE TABLE film_delete_reg (
   reg_id SERIAL PRIMARY KEY,
   film_id INT NOT NULL,
   delete_date TIMESTAMP NOT NULL DEFAULT NOW()
);
```

Crear la Función del Disparador

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

Crear el Disparador en la Tabla **film**

```
CREATE TRIGGER after_film_delete
AFTER DELETE ON film
FOR EACH ROW
EXECUTE FUNCTION film_deletion();
```

Comente el significado y la relevancia de las secuencias.

Las secuencias en bases de datos son contadores automáticos que generan valores únicos y crecientes, utilizados comúnmente para asignar identificadores únicos a registros en tablas, como en claves primarias. Su relevancia radica en que aseguran valores únicos.