Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Modelo relacional: Vistas y disparadores

Grupo 3
Laura Álvarez Zamora - alu0101349824
Emilio González Díaz - alu0101316734
Pablo Fernández Herrera - alu0100817515

C/ Padre Herrera s/n
38207 The Lagoon
Santa Cruz of Tenerife. Spain

T: 900 43 25 26

ull.es

Universidad
de La Laguna

Índice

1. Identifique las tablas, vistas y secuencias

Tablas: Hay un total de 15 tablas en la base de datos inicial, 17 si contamos las dos tablas adicionales que se crean para los disparadores de los puntos 7 y 8, el total de tablas se puede obtener mediante el comando /dt:

- **Tablas iniciales**
  - Actor (actor_id, first_name, last_name, last_update)
  - Address (address_id, address, address2, district, city_id, postal_code, phone, last_update)
  - Category (category_id, name, last_update)
  - City (city_id, city, country_id, last_update)
  - Country (country_id, country, last_update)
  - Customer (customer_id integer, nextval, store_id, first_name, last_name, email, address_id, activebool, create_date, last_update, active integer)
  - film (film_id, title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost, rating, last_update, special_features, fulltext)
  - film_actor (actor_id, film_id, last_update)
  - film_category (film_id, category_id, last_update)
  - inventory (inventory_id, film_id, store_id, last_update)
  - language (language_id, name, last_update)
  - payment (payment_id, customer_id, staff_id, rental_id, amount, payment_date)
  - rental (rental_id, rental_date, inventory_id, customer_id, return_date, staff_id, last_update)
  - staff (staff_id, first_name, last_name, address_id, email, store_id, active, username, password, last_update, picture)
  - store (store_id, manager_staff_id, address_id, last_update)

- **Tablas añadidas por nosotros**
  - eliminar_registro_pelicula(log_id, film_id, delete_date)
  - registro_insersiones_pelicula(log_id, film_id, insert_date)

```
Alquiler=# \dt
                    List of relations
 Schema |              Name               | Type  |  Owner
--------+---------------------------------+-------+----------
 public | actor                           | table | postgres
 public | address                         | table | postgres
 public | category                        | table | postgres
 public | city                            | table | postgres
 public | country                         | table | postgres
 public | customer                        | table | postgres
 public | eliminar_registro_pelicula      | table | LauraAZ
 public | film                            | table | postgres
 public | film_actor                      | table | postgres
 public | film_category                   | table | postgres
 public | inventory                       | table | postgres
 public | language                        | table | postgres
 public | payment                         | table | postgres
 public | registro_inserciones_pelicula   | table | LauraAZ
 public | rental                          | table | postgres
 public | staff                           | table | postgres
 public | store                           | table | postgres
(17 rows)
```

Imagen 1: Tablas

Vistas: Las vistas se pueden mostrar mediante el comando /dv, aunque inicialmente la base de datos por defecto no disponía de vistas tras el desarrollo de la práctica se implementaran 4 vistas nuevas:

- **Vistas añadidas por nosotros**
  - view_informacion_actores
  - view_lista _peliculas
  - view_ventas_totales_por_categoria
  - view_ventas_totales_por_tienda

```
Alquiler=# \dv
                        List of relations
 Schema |                  Name                  | Type |  Owner
--------+----------------------------------------+------+---------
 public | view_informacion_actores               | view | LauraAZ
 public | view_lista_peliculas                   | view | LauraAZ
 public | view_ventas_totales_por_categoria      | view | LauraAZ
 public | view_ventas_totales_por_tienda         | view | LauraAZ
(4 rows)
```

Imagen 2: Vistas

Secuencias: Inicialmente había en la base de datos un total de 13 secuencias, pero durante la práctica se añadieron dos más, hasta un total de 15, las secuencias se pueden mostrar mediante el comando /ds:

- **Secuencias iniciales**
  - actor_actor_id_seq
  - address_address_id_seq
  - category_category_id_seq
  - city_city_id_seq
  - country_country_id_seq
  - customer_customer_id_seq
  - film_film_id_seq
  - inventory_inventory_id_seq
  - language_language_id_seq
  - payment_payment_id_seq
  - Rental_rental_id_seq
  - staff_staff_id_seq
  - store_store_id_seq

- **Secuencias añadidas por nosotros**
  - registro_inserciones_pelicula_log_id_seq
  - eliminar_registro_pelicula_og_id_seq

```
Alquiler=# \ds
                        List of relations
 Schema |                    Name                     |   Type   |  Owner
--------+---------------------------------------------+----------+----------
 public | actor_actor_id_seq                          | sequence | postgres
 public | address_address_id_seq                      | sequence | postgres
 public | category_category_id_seq                    | sequence | postgres
 public | city_city_id_seq                            | sequence | postgres
 public | country_country_id_seq                      | sequence | postgres
 public | customer_customer_id_seq                    | sequence | postgres
 public | eliminar_registro_pelicula_log_id_seq       | sequence | LauraAZ
 public | film_film_id_seq                            | sequence | postgres
 public | inventory_inventory_id_seq                  | sequence | postgres
 public | language_language_id_seq                    | sequence | postgres
 public | payment_payment_id_seq                      | sequence | postgres
 public | registro_inserciones_pelicula_log_id_seq    | sequence | LauraAZ
 public | rental_rental_id_seq                        | sequence | postgres
 public | staff_staff_id_seq                          | sequence | postgres
 public | store_store_id_seq                          | sequence | postgres
(15 rows)
```

Imagen 3: Secuencias

2. Identifique las tablas principales y sus principales elementos

Para considerar una tabla como principal creemos que los criterios más importantes son el número de relaciones con otras tablas y la relevancia de la información contenida en la tabla. Por ello consideramos que las siguientes tablas son las más importantes:

2.1. Tabla customer

Esta tabla contiene toda la información relacionada con los clientes de la plataforma de renta de videos, nombre, dirección, estado, dirección completa, tienda asociada, etc.

2.2. Tabla film

Esta tabla contiene toda la información relacionada con las películas sobre las que se está trabajando en la plataforma de renta de videos, tales como su categoría, actores que aparecen, precio de renta, precio de remplazo, descripción completa, duración, etc.

2.3. Tabla staff

Esta tabla contiene toda la información relacionada con los empleados de la plataforma de renta de videos, tales como nombre completo, estado de actividad, email. Dirección completa, foto, etc.

2.4. Tabla store

Esta tabla contiene información relacionada con las tiendas que administra la plataforma de renta de videos, dirección completa, identidad del administrador, etc.

2.5. Tabla payment

Esta tabla contiene información sobre los pagos de los alquileres de películas de los clientes. Contiene campos para identificar al cliente, su pago, lo que ha alquilado, la fecha del pago, etc.

2.6. Tabla inventory

Esta tabla contiene un inventario completo de todas las películas que tiene la plataforma de renta de películas y su correspondiente tienda asociada.

```
Alquiler=# SELECT
    tc.constraint_name,
    tc.table_name,
    kcu.column_name,
    ccu.table_name AS foreign_table_name,
    ccu.column_name AS foreign_column_name
FROM
    information_schema.table_constraints AS tc
JOIN
    information_schema.key_column_usage AS kcu
    ON tc.constraint_name = kcu.constraint_name
JOIN
    information_schema.constraint_column_usage AS ccu
    ON ccu.constraint_name = tc.constraint_name
WHERE
    tc.constraint_type = 'FOREIGN KEY';
        constraint_name         |  table_name   |   column_name    | foreign_table_name | foreign_column_name
--------------------------------+---------------+------------------+--------------------+---------------------
 film_actor_actor_id_fkey       | film_actor    | actor_id         | actor              | actor_id
 film_actor_film_id_fkey        | film_actor    | film_id          | film               | film_id
 film_category_category_id_fkey | film_category | category_id      | category           | category_id
 film_category_film_id_fkey     | film_category | film_id          | film               | film_id
 film_language_id_fkey          | film          | language_id      | language           | language_id
 fk_address_city                | address       | city_id          | city               | city_id
 fk_city                        | city          | country_id       | country            | country_id
 inventory_film_id_fkey         | inventory     | film_id          | film               | film_id
 payment_customer_id_fkey       | payment       | customer_id      | customer           | customer_id
 payment_rental_id_fkey         | payment       | rental_id        | rental             | rental_id
 payment_staff_id_fkey          | payment       | staff_id         | staff              | staff_id
 rental_customer_id_fkey        | rental        | customer_id      | customer           | customer_id
 rental_inventory_id_fkey       | rental        | inventory_id     | inventory          | inventory_id
 rental_staff_id_key            | rental        | staff_id         | staff              | staff_id
 staff_address_id_fkey          | staff         | address_id       | address            | address_id
 store_address_id_fkey          | store         | address_id       | address            | address_id
 store_manager_staff_id_fkey    | store         | manager_staff_id | staff              | staff_id
 customer_address_id_fkey       | customer      | address_id       | address            | address_id
(18 rows)
```

Imagen 4: FK de las tablas

3. Realice las siguientes consultas

A. Obtenga las ventas totales por categoría de películas ordenadas descendentemente.

```
-- 4a
SELECT CATEGORY.name AS CATEGORIA, COUNT(*) AS TOTAL_VENTAS
FROM PAYMENT
INNER JOIN RENTAL ON PAYMENT.rental_id = RENTAL.rental_id
INNER JOIN INVENTORY ON INVENTORY.inventory_id = RENTAL.inventory_id
INNER JOIN FILM ON FILM.film_id = inventory.film_id
INNER JOIN FILM_CATEGORY ON FILM_CATEGORY.film_id = film.film_id
INNER JOIN CATEGORY ON CATEGORY.category_id = film_category.category_id
GROUP BY CATEGORY.category_id
ORDER BY TOTAL_VENTAS DESC;
```

B. Obtenga las ventas totales por tienda, donde se refleje la ciudad, el país (concatenar la ciudad y el país empleando como separador la ","), y el encargado. Pudiera emplear GROUP BY, ORDER BY

```
-- 4b
SELECT STORE.store_id, CITY.city || ', ' || COUNTRY.country AS LOCALIZACION,
STAFF.first_name AS ENCARGADO ,COUNT(*) AS VENTAS_TOTALES
FROM STORE
INNER JOIN INVENTORY ON STORE.store_id = INVENTORY.store_id
INNER JOIN RENTAL ON RENTAL.inventory_id = INVENTORY.inventory_id
INNER JOIN PAYMENT ON PAYMENT.rental_id = RENTAL.rental_id
INNER JOIN ADDRESS ON ADDRESS.address_id = STORE.address_id
INNER JOIN CITY ON CITY.city_id = ADDRESS.city_id
INNER JOIN COUNTRY ON COUNTRY.country_id = CITY.country_id
INNER JOIN STAFF ON STAFF.staff_id = STORE.manager_staff_id
GROUP BY STORE.store_id, CITY.city_id, COUNTRY.country_id, STAFF.staff_id;
```

C. Obtenga una 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). Pudiera emplear GROUP BY

```
-- 4c
SELECT FILM.film_id, title, description, CATEGORY.name AS CATEGORY,
replacement_cost AS COST, rating, ACTOR.first_name || ' ' || ACTOR.last_name AS
ACTOR_NAME
FROM FILM
INNER JOIN FILM_CATEGORY ON FILM.film_id = FILM_CATEGORY.film_id
INNER JOIN CATEGORY ON CATEGORY.category_id = FILM_CATEGORY.category_id
INNER JOIN FILM_ACTOR ON FILM_ACTOR.film_id = FILM.film_id
INNER JOIN ACTOR ON ACTOR.actor_id = FILM_ACTOR.actor_id;
```

D. Obtenga la 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 ":"

```
-- 4d
SELECT first_name, last_name, STRING_AGG(FILM.title || ', ' || CATEGORY.NAME,' : ')
FROM FILM
INNER JOIN FILM_CATEGORY ON FILM.film_id = FILM_CATEGORY.film_id
INNER JOIN CATEGORY ON CATEGORY.category_id = FILM_CATEGORY.category_id
INNER JOIN FILM_ACTOR ON FILM_ACTOR.film_id = FILM.film_id
INNER JOIN ACTOR ON ACTOR.actor_id = FILM_ACTOR.actor_id
GROUP BY ACTOR.actor_id;
```

4. Realice todas las vistas de las consultas anteriores

A continuación desarrollamos el código que hemos utilizado para desarrollar las vistas de las consultas del apartado anterior:

```
-- 5a
CREATE VIEW view_ventas_totales_por_categoria AS
SELECT CATEGORY.name AS CATEGORIA, COUNT(*) AS TOTAL_VENTAS
FROM PAYMENT
INNER JOIN RENTAL ON PAYMENT.rental_id = RENTAL.rental_id
INNER JOIN INVENTORY ON INVENTORY.inventory_id = RENTAL.inventory_id
INNER JOIN FILM ON FILM.film_id = INVENTORY.film_id
INNER JOIN FILM_CATEGORY ON FILM_CATEGORY.film_id = FILM.film_id
INNER JOIN CATEGORY ON CATEGORY.category_id = FILM_CATEGORY.category_id
GROUP BY CATEGORY.category_id
ORDER BY TOTAL_VENTAS DESC;

-- 5b
CREATE VIEW view_ventas_totales_por_tienda AS
SELECT STORE.store_id, CITY.city || ', ' || COUNTRY.country AS LOCALIZACION,
    STAFF.first_name AS ENCARGADO, COUNT(*) AS VENTAS_TOTALES
FROM STORE
INNER JOIN INVENTORY ON STORE.store_id = INVENTORY.store_id
INNER JOIN RENTAL ON RENTAL.inventory_id = INVENTORY.inventory_id
INNER JOIN PAYMENT ON PAYMENT.rental_id = RENTAL.rental_id
INNER JOIN ADDRESS ON ADDRESS.address_id = STORE.address_id
INNER JOIN CITY ON CITY.city_id = ADDRESS.city_id
INNER JOIN COUNTRY ON COUNTRY.country_id = CITY.country_id
INNER JOIN STAFF ON STAFF.staff_id = STORE.manager_staff_id
GROUP BY STORE.store_id, CITY.city_id, COUNTRY.country_id, STAFF.staff_id;

-- 5c
CREATE VIEW view_informacion_actores AS
SELECT FILM.film_id, title, description, CATEGORY.name AS CATEGORY,
    replacement_cost AS COST, rating,
    ACTOR.first_name || ' ' || ACTOR.last_name AS ACTOR_NAME
```

```
FROM FILM
INNER JOIN FILM_CATEGORY ON FILM.film_id = FILM_CATEGORY.film_id
INNER JOIN CATEGORY ON CATEGORY.category_id = FILM_CATEGORY.category_id
INNER JOIN FILM_ACTOR ON FILM_ACTOR.film_id = FILM.film_id
INNER JOIN ACTOR ON ACTOR.actor_id = FILM_ACTOR.actor_id;

-- 5d
CREATE VIEW view_lista_peliculas AS
SELECT first_name, last_name,
    STRING_AGG(FILM.title || ', ' || CATEGORY.NAME, ' : ') AS FILM_LIST
FROM FILM
INNER JOIN FILM_CATEGORY ON FILM.film_id = FILM_CATEGORY.film_id
INNER JOIN CATEGORY ON CATEGORY.category_id = FILM_CATEGORY.category_id
INNER JOIN FILM_ACTOR ON FILM_ACTOR.film_id = FILM.film_id
INNER JOIN ACTOR ON ACTOR.actor_id = FILM_ACTOR.actor_id
GROUP BY ACTOR.actor_id;
```

5. Incluir restricciones Check

Las restricciones que hemos decidido añadir a la base de datos, tras analizarla han sido:

- Una restricción para asegurar que el coste de reemplazo de una película debía ser siempre mayor a 0.

```
ALTER TABLE FILM
ADD CONSTRAINT coste_pelicula_mayor_cero CHECK (replacement_cost > 0);
```

- Una restricción para asegurar que el pago de una película alquilada debía ser siempre mayor a 0.

```
ALTER TABLE PAYMENT
ADD CONSTRAINT coste_pagos_mayor_igual_cero CHECK (amount >= 0);
```

- Una restricción para asegurar que la devolución de una película siempre ocurre después de haber rentado la película.

```
ALTER TABLE RENTAL
ADD CONSTRAINT duracion_menor_fecha_devolucion CHECK (rental_date <
return_date);
```

- Una restricción para comprobar el formato del correo de un cliente.

```
ALTER TABLE CUSTOMER
ADD CONSTRAINT email_valido_cliente CHECK (email ~*
'^[A-Za-z0-9._-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');
```

- Una restricción para comprobar el formato del correo de un empleado.

```
ALTER TABLE STAFF
ADD CONSTRAINT email_valido_empleado CHECK (email ~*
'^[A-Za-z0-9._-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');
```

```
Alquiler=# INSERT INTO CUSTOMER (store_id, first_name, last_name, email, address_id, activebool, create_date, last_update, active)
VALUES (1, 'John', 'Doe', 'john.doe@example.com', 530, true, '2024-11-08', '2024-11-08 10:00:00', 1);
INSERT 0 1
Alquiler=# SELECT * FROM CUSTOMER WHERE email = 'john.doe@example.com';
 customer_id | store_id | first_name | last_name |        email         | address_id | activebool | create_date |     last_update     | active
-------------+----------+------------+-----------+----------------------+------------+------------+-------------+---------------------+--------
         601 |        1 | John       | Doe       | john.doe@example.com |        530 | t          | 2024-11-08  | 2024-11-08 10:00:00 |      1
(1 row)

Alquiler=# INSERT INTO CUSTOMER (store_id, first_name, last_name, email, address_id, activebool, create_date, last_update, active)
VALUES (1, 'Jane', 'Doe', 'jane.doe@invalid', 530, true, '2024-11-08', '2024-11-08 10:00:00', 1);
ERROR:  new row for relation "customer" violates check constraint "email_valido_cliente"
DETAIL:  Failing row contains (602, 1, Jane, Doe, jane.doe@invalid, 530, t, 2024-11-08, 2024-11-08 10:00:00, 1).
Alquiler=#
```

Imagen 5: Ejemplo de restricción Check sobre los emails

6. Explique el siguiente trigger

```
last_updated BEFORE UPDATE ON customer
FOR EACH ROW EXECUTE PROCEDURE last_updated()
```

Esto es un trigger que se activa automáticamente antes de cada actualización de cualquiera de los campos de la tabla customer, y llama a la función last_updated().

Esta función sobrescribe el campo last_update de la fila, por la fecha actual del sistema en el momento en el que se activa el trigger y lo guarda en la nueva entrada que se va a crear en la tabla.

Otras tablas donde se ejecuta un trigger similar son:

- Actor
- Address
- Category
- City
- Country
- Film
- Film_actor
- Film_category
- Inventory
- Rental
- Staff
- Store

7. Disparador para guardar en una nueva tabla la fecha de inserción de un nuevo registro en la tabla film

A continuación representa la creación del disparador para guardar, en una nueva tabla registro_insersiones_pelicula, compuesta por los campos, log_id (clave primaria), film_id (id de la película insertada) e insert_date (fecha de inserción del nuevo registro en la tabla film).

Se creará la función funcion_inserciones_pelicula() donde se hará la inserción de los correspondientes campos en la tabla creada anteriormente con la información necesaria.

Y por último se crea el trigger disparador_inserciones_pelicula, que utilizará la función anterior para cada inserción en la tabla Film.

```
-- 7
CREATE TABLE registro_inserciones_pelicula (
  log_id SERIAL PRIMARY KEY,
  film_id INT,
  insert_date TIMESTAMP
);

CREATE OR REPLACE FUNCTION funcion_inserciones_pelicula()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO registro_inserciones_pelicula(film_id, insert_date)
  VALUES (NEW.film_id, CURRENT_TIMESTAMP);
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER disparador_inserciones_pelicula
AFTER INSERT ON FILM
FOR EACH ROW EXECUTE FUNCTION funcion_inserciones_pelicula();
```

8. Disparador para guardar en una nueva tabla la fecha de eliminación de un registro en la tabla film y el identificador del film

A continuación representa la creación del disparador para guardar, en una nueva tabla eliminar_registro_pelicula , compuesta por los campos, log_id (clave primaria), film_id (id de la película eliminada) e delete_date (fecha de eliminacion del nuevo registro en la tabla film).

Se creará la función funcion_eliminar_pelicula() donde se hará la inserción de los correspondientes campos en la tabla creada anteriormente con la información necesaria.

Y por último se crea el trigger disparador_eliminar_pelicula, que utilizará la función anterior para cada inserción en la tabla Film.

```
-- 8
CREATE TABLE eliminar_registro_pelicula (
  log_id SERIAL PRIMARY KEY,
  film_id INT,
  delete_date TIMESTAMP
);

CREATE OR REPLACE FUNCTION funcion_eliminar_pelicula()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO eliminar_registro_pelicula(film_id, delete_date)
  VALUES (OLD.film_id, CURRENT_TIMESTAMP);
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER disparador_eliminar_pelicula
AFTER DELETE ON FILM
FOR EACH ROW EXECUTE FUNCTION funcion_eliminar_pelicula();
```

9. Significado y la relevancia de las sequence

Las secuencias se utilizan en esta base de datos para generar los valores únicos, para las claves primarias de las tablas, y de esta forma asegurar que cada fila en una tabla tenga un identificador único.

También hemos implementado en esta práctica dos nuevas secuencias, que permiten llevar un historial de las inserciones y eliminaciones realizadas en la tabla film.