ADMINISTRACIÓN Y DISEÑO DE BASE DE DATOS

Modelo relacional. Vistas y disparadores

Enrique Hernández Cabrera.
Airam Herrera Plasencia.

Universidad de La Laguna

ÍNDICE

1) RESTAURACIÓN DE LA BASE DE DATOS.

Preparamos la base de datos.

Antes de realizar la restauración, creamos una nueva base de datos donde se cargará el contenido del archivo.

```
C:\Users\Hyssen>createdb -U postgres AlquilerDVD
Password:
```

Ejecutamos **pg_restore** para restaurar el fichero .tar a la base de datos.

```
pg_restore -d AlquilerDVD -U postgres -h localhost -p 5432
C:/Users/Hyssen/Downloads/AlquilerPractica.tar
```

```
C:\Users\Hyssen>pg_restore -d AlquilerDVD -U postgres -h localhost -p 5432 C:/Users/Hyssen/Downloads/AlquilerPractica.tar
Password:
```

```
C:\Users\Hyssen>psql -h localhost -U postgres -d AlquilerDVD
Password for user postgres:
psql (17.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.
```

2) IDENTIFICAR TABLAS, VISTAS Y SECUENCIAS.

Mostramos las tablas con el comando **\dt** y las vistas y secuencias con **\dv** y **\ds** subsecuentemente.

```
AlquilerDVD-# \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 | 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 | rental        | table | postgres
 public | staff         | table | postgres
 public | store         | table | postgres
(15 rows)
```

```
AlquilerDVD-# \dv
Did not find any relations.
AlquilerDVD-# \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 | 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 | rental_rental_id_seq       | sequence | postgres
 public | staff_staff_id_seq         | sequence | postgres
 public | store_store_id_seq         | sequence | postgres
(13 rows)
```

3) IDENTIFICAR LAS TABLAS PRINCIPALES Y SUS PRINCIPALES ELEMENTOS.

Podemos ver la información específica de cada tabla usando el comando **\d nombre_tabla**.

FILM

```
AlquilerDVD-# \d film
                                        Table "public.film"
      Column      |            Type             | Collation | Nullable |                Default
------------------+-----------------------------+-----------+----------+---------------------------------------
 film_id          | integer                     |           | not null | nextval('film_film_id_seq'::regclass)
 title            | character varying(255)      |           | not null |
 description      | text                        |           |          |
 release_year     | year                        |           |          |
 language_id      | smallint                    |           | not null |
 rental_duration  | smallint                    |           | not null | 3
 rental_rate      | numeric(4,2)                |           | not null | 4.99
 length           | smallint                    |           |          |
 replacement_cost | numeric(5,2)                |           | not null | 19.99
 rating           | mpaa_rating                 |           |          | 'G'::mpaa_rating
 last_update      | timestamp without time zone |           | not null | now()
 special_features | text[]                      |           |          |
 fulltext         | tsvector                    |           | not null |
```

RENTAL

```
AlquilerDVD-# \d rental
                                      Table "public.rental"
    Column    |            Type             | Collation | Nullable |                 Default
--------------+-----------------------------+-----------+----------+-----------------------------------------
 rental_id    | integer                     |           | not null | nextval('rental_rental_id_seq'::regclass)
 rental_date  | timestamp without time zone |           | not null |
 inventory_id | integer                     |           | not null |
 customer_id  | smallint                    |           | not null |
 return_date  | timestamp without time zone |           |          |
 staff_id     | smallint                    |           | not null |
 last_update  | timestamp without time zone |           | not null | now()
```

PAYMENT

```
AlquilerDVD-# \d payment
                                      Table "public.payment"
    Column    |            Type             | Collation | Nullable |                  Default
--------------+-----------------------------+-----------+----------+-------------------------------------------
 payment_id   | integer                     |           | not null | nextval('payment_payment_id_seq'::regclass)
 customer_id  | smallint                    |           | not null |
 staff_id     | smallint                    |           | not null |
 rental_id    | integer                     |           | not null |
 amount       | numeric(5,2)                |           | not null |
 payment_date | timestamp without time zone |           | not null |
```

CUSTOMER

```
AlquilerDVD-# \d customer
                                      Table "public.customer"
   Column    |            Type             | Collation | Nullable |                  Default
-------------+-----------------------------+-----------+----------+-------------------------------------------
 customer_id | integer                     |           | not null | nextval('customer_customer_id_seq'::regclass)
 store_id    | smallint                    |           | not null |
 first_name  | character varying(45)       |           | not null |
 last_name   | character varying(45)       |           | not null |
 email       | character varying(50)       |           |          |
 address_id  | smallint                    |           | not null |
 activebool  | boolean                     |           | not null | true
 create_date | date                        |           | not null | 'now'::text::date
 last_update | timestamp without time zone |           |          | now()
 active      | integer                     |           |          |
```

STAFF

```
AlquilerDVD-# \d staff
                                    Table "public.staff"
   Column    |            Type             | Collation | Nullable |                 Default
-------------+-----------------------------+-----------+----------+----------------------------------------
 staff_id    | integer                     |           | not null | nextval('staff_staff_id_seq'::regclass)
 first_name  | character varying(45)       |           | not null |
 last_name   | character varying(45)       |           | not null |
 address_id  | smallint                    |           | not null |
 email       | character varying(50)       |           |          |
 store_id    | smallint                    |           | not null |
 active      | boolean                     |           | not null | true
 username    | character varying(16)       |           | not null |
 password    | character varying(40)       |           |          |
 last_update | timestamp without time zone |           | not null | now()
 picture     | bytea                       |           |          |
```

STORE

```
AlquilerDVD-# \d store
                                    Table "public.store"
     Column      |            Type             | Collation | Nullable |                Default
-----------------+-----------------------------+-----------+----------+---------------------------------------
 store_id        | integer                     |           | not null | nextval('store_store_id_seq'::regclass)
 manager_staff_id | smallint                    |           | not null |
 address_id      | smallint                    |           | not null |
 last_update     | timestamp without time zone |           | not null | now()
```

INVENTORY

```
AlquilerDVD-# \d inventory
                                   Table "public.inventory"
   Column    |            Type             | Collation | Nullable |                   Default
-------------+-----------------------------+-----------+----------+-----------------------------------------------
 inventory_id | integer                     |           | not null | nextval('inventory_inventory_id_seq'::regclass)
 film_id     | smallint                    |           | not null |
 store_id    | smallint                    |           | not null |
 last_update | timestamp without time zone |           | not null | now()
```

Para el escenario de una video tienda donde los clientes pueden alquilar películas y/o videojuegos hemos identificado las siguientes como las tablas principales:

1. **FILM:** Esta es la tabla central, ya que contiene todos los datos necesarios sobre las películas en alquiler, incluye:
    - film_id: para enlazar esta tabla con otras.
    - title para búsquedas y visualización del inventario.
    - rental_duration y rental_rate Duración máxima del alquiler y tarifa del alquiler.
    - replacement_cost: costo de reemplazo.
2. **RENTAL:** Esta tabla es esencial para la operación de la videotienda, ya que permite registrar y rastrear cada transacción de alquiler, controlando qué película ha alquilado cada cliente y su fecha de devolución. Sin este registro, no sería posible administrar los alquileres ni el histórico de transacciones de cada cliente.
    - tiene diferentes ids únicos que permiten referenciar este registro con otras tablas como rental_id, inventory_id, customer_id, staff_id.
    - rental_date: importante para calcular el tiempo de alquiler y posibles cargos por demora.

3. **PAYMENT:** Esta tabla es crucial para registrar los pagos asociados a los alquileres. Permite mantener el control financiero, asegurando que cada alquiler tenga un pago asociado y que los ingresos estén debidamente registrados.
    - amount: Monto pagado, que permite calcular los ingresos generados y detectar posibles discrepancias.
    - distintos ids como, payment_id, customer_id, staff_id, rental_id.
4. **CUSTOMER:** es fundamental en la gestión de relaciones con los clientes, ya que almacena toda la información necesaria para identificar a los clientes y realizar un seguimiento de su actividad en la videotienda.
    - customer_id: necesario para enlazar esta tabla con las tablas de alquiler y pago.
    - email: útil para notificaciones o recordatorios de devolución.
    - create_date: Fecha de creación del registro del cliente, importante para el historial de membresía.
    - active y activebool: Indica si el cliente está activo, útil para administrar a clientes con membresías activas.
5. **STAFF:** Esta tabla es fundamental para el control y la gestión del personal, permitiendo que cada transacción de alquiler o pago esté asociada a un empleado. Además, las credenciales permiten una gestión segura de acceso al sistema.
    - staff_id: Identificador único para cada empleado, que se utiliza como clave primaria y para referencias en otras tablas.
    - active: Indicador booleano que señala si el empleado está activo o no, lo cual ayuda en la gestión del personal.
    - username y password: Credenciales del empleado, necesarias para el acceso al sistema.
    - store_id: Identifica la tienda en la que trabaja el empleado, permitiendo asociar al personal con diferentes sucursales.
6. **STORE:** La tabla store permite gestionar múltiples sucursales dentro del sistema, proporcionando flexibilidad para que una videotienda crezca en diferentes ubicaciones. La asociación de un gerente (staff) a cada tienda permite una estructura organizacional clara y facilita el análisis del rendimiento por tienda.
    - store_id: Identificador único de la tienda, que actúa como clave primaria y permite referenciar esta sucursal en la tabla inventory.
    - manager_staff_id: Identificador del empleado que es el gerente de la tienda, enlazado con staff_id en la tabla staff.
    - address_id: Identificador de la dirección de la tienda, enlazado con la tabla address para obtener la ubicación de la sucursal.

7. **INVENTORY:** Esta tabla es crucial para gestionar qué películas están disponibles en cada sucursal, optimizando el control del inventario y asegurando que cada sucursal tenga un registro claro de las películas que posee.
    - inventory_id: Identificador único de cada registro de inventario, utilizado como clave primaria.
    - film_id: Enlaza con la tabla film para identificar la película específica en el inventario.
    - store_id: Enlaza con la tabla store para identificar en qué tienda está disponible la copia de la película.
    - last_update: Fecha de la última actualización del registro, importante para mantener actualizado el estado del inventario.

4) REALICE LAS SIGUIENTES CONSULTAS

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

```
AlquilerDVD=# SELECT C.NAME AS CATEGORY, SUM(P.AMOUNT) AS SALES
AlquilerDVD-# FROM CATEGORY C
AlquilerDVD-# JOIN FILM_CATEGORY FC ON C.CATEGORY_ID = FC.CATEGORY_ID
AlquilerDVD-# JOIN FILM F ON FC.FILM_ID = F.FILM_ID
AlquilerDVD-# JOIN INVENTORY I ON F.FILM_ID = I.FILM_ID
AlquilerDVD-# JOIN RENTAL R ON I.INVENTORY_ID = R.INVENTORY_ID
AlquilerDVD-# JOIN PAYMENT P ON R.RENTAL_ID = P.RENTAL_ID
AlquilerDVD-# GROUP BY C.NAME
AlquilerDVD-# ORDER BY SALES DESC;
```

```
  category   |  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 rows)
```

- 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.

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

```
       location        | manager_name | total_sales
-----------------------+--------------+-------------
 Woodridge, Australia  | Jon Stephens |    30683.13
 Lethbridge, Canada    | Mike Hillyer |    30628.91
(2 rows)
```

- 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)

```
AlquilerDVD=# SELECT
AlquilerDVD-#     f.film_id,
AlquilerDVD-#     f.title,
AlquilerDVD-#     f.description,
AlquilerDVD-#     c.name AS category,
AlquilerDVD-#     f.rental_rate AS price,
AlquilerDVD-#     f.length AS duration,
AlquilerDVD-#     f.rating,
AlquilerDVD-#     STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ') AS actors
AlquilerDVD-# FROM
AlquilerDVD-#     film f
AlquilerDVD-# JOIN
AlquilerDVD-#     film_category fc ON f.film_id = fc.film_id
AlquilerDVD-# JOIN
AlquilerDVD-#     category c ON fc.category_id = c.category_id
AlquilerDVD-# JOIN
AlquilerDVD-#     film_actor fa ON f.film_id = fa.film_id
AlquilerDVD-# JOIN
AlquilerDVD-#     actor a ON fa.actor_id = a.actor_id
AlquilerDVD-# GROUP BY
AlquilerDVD-#     f.film_id, f.title, f.description, c.name, f.rental_rate, f.length, f.rating
AlquilerDVD-# ORDER BY
AlquilerDVD-#     f.title;
```

```
 film_id |          title          |                                                                                                description                                                                                                |  category   | price | duration | rating |                                                                                                actors
---------+-------------------------+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+-------------+-------+----------+--------+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
       1 | Academy Dinosaur        | A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies                                                                                                      | Documentary |  0.99 |       86 | PG     | Rock Dukakis, Mary Ke
itel, Johnny Cage, Penelope Guiness, Sandra Peck, Christian Gable, Oprah Kilmer, Warren Nolte, Lucille Tracy, Mena Temple
       2 | Ace Goldfinger          | A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China                                                                                                 | Horror      |  4.99 |       48 | G      | Minnie Zellweger, Chr
is Depp, Bob Fawcett, Sean Guiness
       3 | Adaptation Holes        | A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory                                                                                                      | Documentary |  2.99 |       50 | NC-17  | Cameron Streep, Bob F
awcett, Nick Wahlberg, Ray Johansson, Julianne Dench
       4 | Affair Prejudice        | A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank                                                                                                          | Horror      |  2.99 |      117 | G      | Jodie Degeneres, Kenn
eth Pesci, Fay Winslet, Oprah Kilmer, Scarlett Damon
       5 | African Egg             | A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico                                                                                 | Family      |  2.99 |      130 | G      | Dustin Tautou, Matthe
w Leigh, Gary Phoenix, Matthew Carrey, Thora Temple
       6 | Agent Truman            | A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China                                                                                                             | Foreign     |  2.99 |      169 | PG     | Warren Nolte, Sandra
Kilmer, Jayne Neeson, Morgan Williams, Kirsten Paltrow, Kenneth Hoffman, Reese West
       7 | Airplane Sierra         | A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat                                                                                                                     | Comedy      |  4.99 |       62 | PG-13  | Mena Hopper, Jim Most
el, Michael Bolger, Oprah Kilmer, Richard Penn
       8 | Airport Pollock         | A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India                                                                                                                         | Horror      |  4.99 |       54 | R      | Lucille Dee, Susan Da
vis, Fay Kilmer, Gene Willis
       9 | Alabama Devil           | A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat                                                                                   | Horror      |  2.99 |      114 | PG-13  | William Hackman, Rip
Crawford, Rip Winslet, Greta Keitel, Christian Gable, Mena Temple, Meryl Allen, Warren Nolte, Elvis Marx
      10 | Aladdin Calendar        | A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist in Ancient China                                                                                                             | Sports      |  4.99 |       63 | NC-17  | Greta Malden, Rock Du
kakis, Ray Johansson, Renee Tracy, Val Bolger, Judy Dean, Jada Ryder, Alec Wayne
      11 | Alamo Videotape         | A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention                                                                                                             | Foreign     |  0.99 |      126 | G      | Michael Bening, Johnn
y Cage, Scarlett Damon, Sean Guiness
      12 | Alaska Phantom          | A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Australia                                                                                                                    | Music       |  0.99 |      136 | PG     | Val Bolger, Burt Pose
y, Gene Mckellen, Jeff Silverstone, Sylvester Dern, Albert Johansson, Sidney Crowe
      13 | Ali Forever             | A Action-Packed Drama of a Dentist And a Crocodile who must Battle a Feminist in The Canadian Rockies                                                                                                 | Horror      |  4.99 |      150 | PG     | Christopher Berry, Ke
nneth Torn, Cary Mcconaughey, Jon Chase, Morgan Mcdormand
      14 | Alice Fantasia          | A Emotional Drama of a A Shark And a Database Administrator who must Vanquish a Pioneer in Soviet Georgia                                                                                            | Classics    |  0.99 |       94 | NC-17  | Woody Hoffman, Rock D
ukakis, Minnie Zellweger, Morgan Williams
      15 | Alien Center            | A Brilliant Drama of a Cat And a Mad Scientist who must Battle a Feminist in A MySQL Convention                                                                                                       | Foreign     |  2.99 |       46 | NC-17  | Sidney Crowe, Kenneth
 Paltrow, Humphrey Willis, Renee Tracy, Burt Dukakis, Mena Hopper
      16 | Alley Evolution         | A Fast-Paced Drama of a Robot And a Composer who must Battle a Astronaut in New Orleans                                                                                                               | Foreign     |  2.99 |      180 | NC-17  | Gregory Gooding, Jude
 Cruise, John Suvari, Karl Berry, Albert Johansson
      17 | Alone Trip              | A Fast-Paced Character Study of a Composer And a Dog who must Outgun a Boat in An Abandoned Fun House                                                                                                 | Music       |  0.99 |       82 | R      | Renee Ball, Ed Chase,
 Spencer Depp, Karl Berry, Laurence Bullock, Woody Jolie, Chris Depp, Uma Wood
      18 | Alter Victory           | A Thoughtful Drama of a Composer And a Feminist who must Meet a Secret Agent in The Canadian Rockies                                                                                                  | Animation   |  0.99 |       57 | PG-13  | Jada Ryder, Angela Wi
therspoon, Oprah Kilmer, Reese Kilmer
      19 | Amadeus Holy            | A Emotional Display of a Pioneer And a Technical Writer who must Battle a Man in A Baloon                                                                                                             | Action      |  0.99 |      113 | PG     | Penelope Cronyn, Kirk
 Jovovich, Julia Mcqueen, Johnny Lollobrigida, Val Bolger, James Pitt
      20 | Amelie Hellfighters     | A Boring Drama of a Woman And a Squirrel who must Conquer a Student in A Baloon                                                                                                                       | Music       |  4.99 |       79 | R      | Laura Brody, Carmen H
unt, Ewan Gooding, Ian Tandy, Walter Torn, Ed Mansfield
```

La lista es demasiado grande y hemos puesto una parte en la imagen.

- 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 ":"

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

5) REALICE TODAS LAS VISTAS DE LAS CONSULTAS ANTERIORES. COLÓQUELES EL PREFIJO VIEW_ A SU DENOMINACIÓN.

- Vista sobre las ventas totales por categoría de películas ordenadas descendentemente.

```
CREATE VIEW view_total_sales_by_category AS
SELECT
c.name AS category_name,
    SUM(p.amount) AS total_sales
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 f ON i.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
    c.name
ORDER BY
    total_sales DESC;
```

- Vista sobre las ventas totales por tienda, donde se refleja la ciudad, el país, y el encargado.

```
CREATE VIEW view_total_sales_by_store AS
SELECT
    CONCAT(ci.city, ', ', co.country) AS location,
    CONCAT(s.first_name, ' ', s.last_name) AS manager_name,
    SUM(p.amount) AS total_sales
FROM
    store st
JOIN
    staff s ON st.manager_staff_id = s.staff_id
JOIN
    address a ON st.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
    inventory i ON st.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
    location, manager_name
ORDER BY
    total_sales DESC;
```

- Vista sobre la lista de películas, donde se refleja el identificador, el título, descripción, categoría, el precio, la duración de la película, clasificación, nombre y apellidos de los actores.

```
CREATE VIEW view_movie_list 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 sobre la información de los actores, donde se incluyen sus nombres y apellidos, las categorías y sus películas.

```
CREATE VIEW view_actor_info AS
SELECT
    CONCAT(a.first_name, ' ', a.last_name) AS actor_name,
    STRING_AGG(DISTINCT c.name || ':' || f.title, ', ') AS
categories_and_movies
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
    actor_name
ORDER BY
    actor_name;
```

6) HAGA UN ANÁLISIS DEL MODELO E INCLUYA LAS RESTRICCIONES CHECK QUE CONSIDERE NECESARIAS

Vamos a realizar varios checks para las siguientes tablas:

Customer

```
ALTER TABLE customer
   ADD CONSTRAINT customer_first_name_chk CHECK (first_name ~* '^[A-Za-zÀ-ÿ ]+$'
AND first_name <> ''),
   ADD CONSTRAINT customer_last_name_chk CHECK (last_name ~* '^[A-Za-zÀ-ÿ ]+$'
AND last_name <> ''),
   ADD CONSTRAINT customer_email_chk CHECK (email ~*
'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' OR email IS NULL)
ALTER COLUMN active TYPE boolean USING active::boolean;
```

Restricciones sugeridas:

- **first_name y last_name**: deben contener solo caracteres alfabéticos y no pueden estar vacíos.
- **email**: debe tener un formato válido de correo electrónico si es proporcionado.
- **active**: restringido a valores TRUE o FALSE.

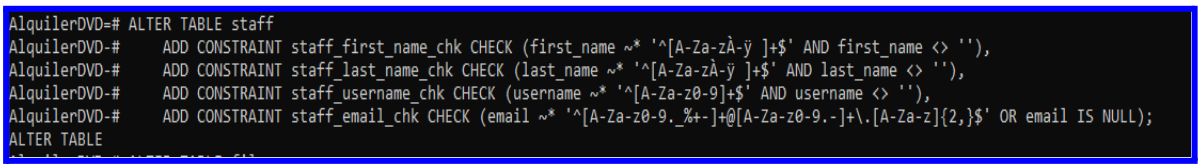
```
AlquilerDVD=# ALTER TABLE customer
AlquilerDVD-#     ADD CONSTRAINT customer_first_name_chk CHECK (first_name ~* '^[A-Za-zÀ-ÿ ]+$' AND first_name <> ''),
AlquilerDVD-#     ADD CONSTRAINT customer_last_name_chk CHECK (last_name ~* '^[A-Za-zÀ-ÿ ]+$' AND last_name <> ''),
AlquilerDVD-#     ADD CONSTRAINT customer_email_chk CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' OR email IS NULL);
ALTER TABLE
```

staff

```
ALTER TABLE staff
   ADD CONSTRAINT staff_first_name_chk CHECK (first_name ~* '^[A-Za-zÀ-ÿ ]+$' AND first_name <> ''),
   ADD CONSTRAINT staff_last_name_chk CHECK (last_name ~* '^[A-Za-zÀ-ÿ ]+$' AND last_name <> ''),
   ADD CONSTRAINT staff_username_chk CHECK (username ~* '^[A-Za-z0-9]+$' AND username <> ''),
   ADD CONSTRAINT staff_email_chk CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' OR email IS NULL);
```

Restricciones sugeridas:

- **first_name, last_name y username**: deben contener caracteres alfanuméricos y no pueden estar vacíos.
- **email**: debe tener un formato válido de correo electrónico



film

```
ALTER TABLE film
   ADD CONSTRAINT film_title_chk CHECK (title <> ''),
   ADD CONSTRAINT film_release_year_chk CHECK (release_year BETWEEN 1900 AND EXTRACT(YEAR FROM CURRENT_DATE)),
   ADD CONSTRAINT film_rental_duration_chk CHECK (rental_duration > 0),
   ADD CONSTRAINT film_rental_rate_chk CHECK (rental_rate > 0),
   ADD CONSTRAINT film_replacement_cost_chk CHECK (replacement_cost > 0),
   ADD CONSTRAINT film_rating_chk CHECK (rating IN ('G', 'PG', 'PG-13', 'R', 'NC-17'));
```

Restricciones sugeridas:

- **title**: no debe estar vacío.
- **release_year**: debe ser un año válido (por ejemplo, entre 1900 y el año actual).
- **rental_duration**: valor positivo.
- **rental_rate y replacement_cost**: valores positivos.
- **rating**: limitar a valores válidos de clasificación.

```
AlquilerDVD=# ALTER TABLE film
AlquilerDVD-#     ADD CONSTRAINT film_title_chk CHECK (title <> ''),
AlquilerDVD-#     ADD CONSTRAINT film_release_year_chk CHECK (release_year BETWEEN 1900 AND EXTRACT(YEAR FROM CURRENT_DATE)),
AlquilerDVD-#     ADD CONSTRAINT film_rental_duration_chk CHECK (rental_duration > 0),
AlquilerDVD-#     ADD CONSTRAINT film_rental_rate_chk CHECK (rental_rate > 0),
AlquilerDVD-#     ADD CONSTRAINT film_replacement_cost_chk CHECK (replacement_cost > 0),
AlquilerDVD-#     ADD CONSTRAINT film_rating_chk CHECK (rating IN ('G', 'PG', 'PG-13', 'R', 'NC-17'));
ALTER TABLE
```

rental

```
ALTER TABLE rental
   ADD CONSTRAINT rental_return_date_chk CHECK (return_date IS NULL OR return_date
> rental_date);
```

Restricciones sugeridas:

- **return_date**: debe ser posterior a rental_date si existe.

```
AlquilerDVD=# ALTER TABLE rental
AlquilerDVD-#     ADD CONSTRAINT rental_return_date_chk CHECK (return_date IS NULL OR return_date > rental_date);
ALTER TABLE
```

payment

```
ALTER TABLE payment
   ADD CONSTRAINT payment_amount_chk CHECK (amount > 0);
```

Restricciones sugeridas:

- **amount**: valor positivo.

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

7) EXPLIQUE LA SENTENCIA QUE APARECE EN LA TABLA CUSTOMER.

```
AlquilerDVD=# \d customer
                                        Table "public.customer"
   Column    |            Type             | Collation | Nullable |                  Default
-------------+-----------------------------+-----------+----------+-------------------------------------------
 customer_id | integer                     |           | not null | nextval('customer_customer_id_seq'::regclass)
 store_id    | smallint                    |           | not null |
 first_name  | character varying(45)       |           | not null |
 last_name   | character varying(45)       |           | not null |
 email       | character varying(50)       |           |          |
 address_id  | smallint                    |           | not null |
 activebool  | boolean                     |           | not null | true
 create_date | date                        |           | not null | 'now'::text::date
 last_update | timestamp without time zone |           |          | now()
 active      | integer                     |           |          |
```

Triggers:

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

```
Triggers:
    last_updated BEFORE UPDATE ON address FOR EACH ROW EXECUTE FUNCTION last_updated()
```

```
AlquilerDVD=# \sf last_updated
CREATE OR REPLACE FUNCTION public.last_updated()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
BEGIN
    NEW.last_update = CURRENT_TIMESTAMP;
    RETURN NEW;
END $function$
```

Es un disparador (trigger) en SQL que está configurado para ejecutarse cada vez que se realiza una actualización (UPDATE) en la tabla customer. Este disparador utiliza una función llamada last_updated() para actualizar la columna last_updated en la tabla customer, manteniendo automáticamente un registro de la última fecha y hora en que cada fila fue modificada.

Vamos a desglosar cada parte de esta sentencia para entender mejor su funcionamiento:

1. last_updated

Este es el nombre del disparador. Se utiliza para identificar y referirse al disparador en la base de datos. Aquí, el nombre last_updated indica su propósito: mantener actualizada la columna last_updated en la tabla customer.

2. BEFORE UPDATE

Este es el momento en que se activa el disparador. BEFORE significa que el disparador se ejecutará **antes** de que ocurra la operación UPDATE en la fila. Esto permite modificar la fila antes de que se guarden los cambios en la base de datos.

3. ON customer

Este es el nombre de la tabla a la cual se aplica el disparador. En este caso, el disparador se activará cada vez que se intente realizar una actualización en la tabla customer.

4. FOR EACH ROW

Indica que el disparador se ejecutará **para cada fila** afectada por la operación UPDATE. Si la sentencia UPDATE modifica varias filas al mismo tiempo, el disparador se activará una vez por cada fila.

5. EXECUTE PROCEDURE last_updated()

Esta parte define la acción que realizará el disparador. EXECUTE PROCEDURE llama a la función last_updated(), que contiene el código para actualizar la columna last_updated en la tabla customer con la fecha y hora actual. La función last_updated() normalmente se define en PL/pgSQL y se utiliza para modificar la fila antes de que se complete la operación de actualización.

La tabla actor, address, category, city, country, film_actor, film_category, inventory, language, rental, staff y store también poseen el trigger last_updated.

8) 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

1. Creamos la tabla de auditoría **FILM_AUDIT**.

Aquí:

- film_id almacenará el identificador único del registro en la tabla film.
- inserted_at almacenará la fecha y hora exacta en que se insertó el registro en la tabla film.

```
AlquilerDVD=# CREATE TABLE film_audit (
AlquilerDVD(#     film_id INT PRIMARY KEY,
AlquilerDVD(#     inserted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
AlquilerDVD(# );
CREATE TABLE
```

2. Crear la función para el disparador que inserte un registro en film_audit cada vez que se inserte un registro en film.

```
AlquilerDVD=# CREATE OR REPLACE FUNCTION log_film_insertion()
AlquilerDVD-# RETURNS TRIGGER AS $$
AlquilerDVD$# BEGIN
AlquilerDVD$#     INSERT INTO film_audit (film_id, inserted_at)
AlquilerDVD$#     VALUES (NEW.film_id, now());
AlquilerDVD$#     RETURN NEW;
AlquilerDVD$# END;
AlquilerDVD$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

3. Crear el disparador en la tabla film.

Ahora, crearemos el disparador en la tabla film para que se active tras cada INSERT y ejecute la función creada anteriormente.

```
AlquilerDVD=# CREATE TRIGGER after_film_insert
AlquilerDVD-# AFTER INSERT ON film
AlquilerDVD-# FOR EACH ROW
AlquilerDVD-# EXECUTE FUNCTION log_film_insertion();
CREATE TRIGGER
```

4. Verificar el funcionamiento.

Primero, insertamos un registro en la tabla film.

```
INSERT INTO film ( film_id, language_id,  title, description,
release_year, rental_duration, rental_rate, length,
replacement_cost, rating, special_features
) VALUES (DEFAULT,  1,  'The Matrix', 1999,5,  4.99, 136, 19.99,
'R', ARRAY['Commentaries', 'Deleted Scenes', 'Behind the
Scenes']);
```

Luego, podemos consultar la tabla film_audit para ver el registro.

```
INSERT INTO film (
film_id, language_id, title, description, release_year,
rental_duration, rental_rate, length, replacement_cost, rating,
special_features
) VALUES (DEFAULT, 1, 'Star Wars', 'The epic tale of a galaxy far,
far away, where a young farm boy becomes a hero and fights the
forces of darkness.', 1977, 7, 3.99, 121, 17.99, 'PG',
ARRAY['Deleted Scenes', 'Behind the Scenes', 'Interviews']
);
```

```
AlquilerDVD=# SELECT * FROM film_audit;
 film_id |        inserted_at
---------+----------------------------
    1001 | 2024-11-12 17:28:14.309281
    1002 | 2024-11-12 17:28:53.546265
(2 rows)
```

9) 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

1. Creamos la tabla de auditoría **FILM_DELETION_AUDIT**.

```
AlquilerDVD=# CREATE TABLE film_deletion_audit (
AlquilerDVD(#     film_id INT,                          deleted_at TIMESTAMP
AlquilerDVD(# );
CREATE TABLE
```

2. Creamos la función para el disparador.

```
AlquilerDVD=# CREATE OR REPLACE FUNCTION record_film_deletion()
AlquilerDVD-# RETURNS TRIGGER AS $$
AlquilerDVD$# BEGIN
AlquilerDVD$#     INSERT INTO film_deletion_audit (film_id, deleted_at)
AlquilerDVD$#     VALUES (OLD.film_id, CURRENT_TIMESTAMP);
AlquilerDVD$#     RETURN OLD;
AlquilerDVD$# END;
AlquilerDVD$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

3. Creamos el disparador en la tabla **film**.

Ahora creamos el disparador para que se active cuando se elimine un registro en la tabla **film**.

```
AlquilerDVD=# CREATE TRIGGER log_film_deletion
AlquilerDVD-# BEFORE DELETE ON film
AlquilerDVD-# FOR EACH ROW
AlquilerDVD-# EXECUTE FUNCTION record_film_deletion();
CREATE TRIGGER
```

4. Probamos el funcionamiento.

Para ello, eliminamos una entrada de la tabla **film**.

```
AlquilerDVD=# DELETE FROM FILM
AlquilerDVD-# WHERE film_id = 1001;
DELETE 1
```

Consultamos la tabla de auditoría para verificar el funcionamiento.

```
AlquilerDVD=# SELECT * FROM film_deletion_audit;
 film_id |         deleted_at
---------+----------------------------
    1001 | 2024-11-12 17:37:14.376805
(1 row)
```

10) COMENTE EL SIGNIFICADO Y RELEVANCIA DE LAS SECUENCIAS

Las **secuencias** son objetos de base de datos utilizados para generar números únicos y secuenciales. Son comúnmente usadas para generar **valores de claves primarias** o cualquier otro tipo de identificador único en tablas.

Relevancia de las Secuencias:

1. **Generación de claves primarias**: Son frecuentemente usadas con el tipo de dato `SERIAL` para asegurar que cada nuevo registro en una tabla tenga un valor único y secuencial, evitando colisiones.
2. **Flexibilidad y control**: Las secuencias permiten configurar el valor inicial, el incremento, y otros parámetros como el límite mínimo y máximo de los valores generados. También se puede **reiniciar** o **modificar** la secuencia según sea necesario.
3. **Rendimiento**: PostgreSQL maneja las secuencias de manera eficiente, incluso en entornos de alta concurrencia, garantizando que los valores generados sean consistentes y sin conflictos.

   Ejemplo de creación de una secuencia:

```
CREATE SEQUENCE film_id_seq START 1 INCREMENT 1;
```

Esto crea una secuencia que comienza en 1 y aumenta en 1 con cada solicitud.

En resumen, las secuencias son fundamentales para la **gestión automática de identificadores únicos**, garantizando la integridad de los datos y mejorando el rendimiento en bases de datos con altas tasas de inserción.
Fuente: *https://www.postgresql.org/docs/current/sql-createsequence.html*