Universidad de La Laguna

Modelo relacional. Vistas y disparadores

Realizado por: Marlon Eduardo Salazar Amador

Correo: alu0101433943@ull.edu.es

Introducción

En esta práctica de la asignatura Administración y diseño de bases de datos, trabajaremos con la base de datos *alquilerdvd*. El objetivo es aplicar los conceptos del modelo relacional, las vistas y los disparadores (triggers), así como analizar la estructura de la base de datos, identificar sus elementos principales y reforzar la integridad de los datos mediante restricciones y automatizaciones.

Durante el desarrollo se realiza la restauración de la base de datos, la exploración de sus tablas, vistas y secuencias, la creación de consultas avanzadas con joins y agrupamientos, y posteriormente se implementan vistas que facilitan su reutilización. Además, se añaden restricciones CHECK para validar la coherencia de los datos y se diseñan disparadores para registrar automáticamente las inserciones y eliminaciones de películas.

El informe recopila los pasos seguidos, las sentencias SQL empleadas y un análisis del modelo de datos.

Enunciados

1. **Realice la restauración de la base de datos alquilerdvd.tar. Observe que la base de datos no tiene un formato SQL como el empleado en actividades anteriores.**

Descargamos el archivo proporcionado. Lo trasladamos al directorio deseado y ejecutamos el siguiente comando:

```SQL
sudo -u postgres pg_restore -C -d postgres AlquilerPractica.tar
```

Este comando crea y restaura la base de datos *alquilerdvd*.

2. Identifique las tablas, vistas y secuencias.

```
alquilerdvd=# \d
                  List of relations
 Schema |            Name            |   Type   |  Owner
--------+----------------------------+----------+----------
 public | actor                      | table    | postgres
 public | actor_actor_id_seq         | sequence | postgres
 public | address                    | table    | postgres
 public | address_address_id_seq     | sequence | postgres
 public | category                   | table    | postgres
 public | category_category_id_seq   | sequence | postgres
 public | city                       | table    | postgres
 public | city_city_id_seq           | sequence | postgres
 public | country                    | table    | postgres
 public | country_country_id_seq     | sequence | postgres
 public | customer                   | table    | postgres
 public | customer_customer_id_seq   | sequence | postgres
 public | film                       | table    | postgres
 public | film_actor                 | table    | postgres
 public | film_category              | table    | postgres
 public | film_film_id_seq           | sequence | postgres
 public | inventory                  | table    | postgres
 public | inventory_inventory_id_seq | sequence | postgres
 public | language                   | table    | postgres
 public | language_language_id_seq   | sequence | postgres
 public | payment                    | table    | postgres
 public | payment_payment_id_seq     | sequence | postgres
 public | rental                     | table    | postgres
 public | rental_rental_id_seq       | sequence | postgres
 public | staff                      | table    | postgres
 public | staff_staff_id_seq         | sequence | postgres
 public | store                      | table    | postgres
 public | store_store_id_seq         | sequence | postgres
(28 rows)
```

El comando *\d* lista tablas, vistas y secuencias.

3. Identifique las tablas principales y sus principales elementos.

```
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)
```

El comando *\dt* lista todas las tablas de la base de datos actual.

```
                                    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 |
```

- Clave primaria: *film_id*
- Columnas importantes: *title*, *description*, *rental_rate*, *length*, *rating*, *language_id*

Accedemos a esta información sobre cada tabla mediante el comando *\d+ public.[tabla]*.

```
                                    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                     |           |          |
```

- Clave primaria: *customer_id*
- Columnas importantes: *email*, *address_id*, *store_id*

```
                                    Table "public.address"
   Column    |            Type             | Collation | Nullable |                  Default
-------------+-----------------------------+-----------+----------+-------------------------------------------
 address_id  | integer                     |           | not null | nextval('address_address_id_seq'::regclass)
 address     | character varying(50)       |           | not null |
 address2    | character varying(50)       |           |          |
 district    | character varying(20)       |           | not null |
 city_id     | smallint                    |           | not null |
 postal_code | character varying(10)       |           |          |
 phone       | character varying(20)       |           | not null |
 last_update | timestamp without time zone |           | not null | now()
```

- Clave primaria: *address_id*
- Columnas importantes: *city_id*, *phone*, *address*, *address2*

```
                                    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 |
```

- Clave primaria: *payment_id*
- Columnas importantes: *customer_id*, *staff_id*, *rental_id*, *amount*, *payment_date*

```
                                    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()
```

- Clave primaria: *rental_id*
- Columnas importantes: *rental_date, inventory_id, customer_id, return_id, staff_id*

```
                                        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                       |           |          |
```

- Clave primaria: *staff_id*
- Columnas importantes: *first_name, last_name, email, store_id, active, username, password*

```
                                      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()
```

- Clave primaria: *inventory_id*
- Columnas importantes: *film_id, store_id*

```
                                        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()
```

- Clave primaria: *store_id*
- Columnas importantes: *manager_staff_id, address_id*

4. Realice las siguientes consultas.

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

SQL

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

categoria	venta_total
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)

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`

SQL

```sql
SELECT s.store_id AS tienda, ci.city AS ciudad, co.country AS pais,
       st.first_name AS nombre_encagado, st.last_name AS apellido_encargado,
       SUM(p.amount) AS venta_total
  FROM payment p
       INNER JOIN rental r ON r.rental_id = p.rental_id
       INNER JOIN inventory i ON i.inventory_id = r.inventory_id
       INNER JOIN store s ON s.store_id = i.store_id
       INNER JOIN address a ON a.address_id = s.address_id
       INNER JOIN city ci ON ci.city_id = a.city_id
       INNER JOIN country co ON co.country_id = ci.country_id
       INNER JOIN staff st ON st.staff_id = s.manager_staff_id
         GROUP BY s.store_id, ci.city, co.country, st.first_name, st.last_name
         ORDER BY venta_total DESC;
```

tienda	ciudad	pais	nombre_encagado	apellido_encargado	venta_total
2	Woodridge	Australia	Jon	Stephens	30683.13
1	Lethbridge	Canada	Mike	Hillyer	30628.91

(2 rows)

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`

SQL

```sql
SELECT f.film_id AS id_pelicula, 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,
```

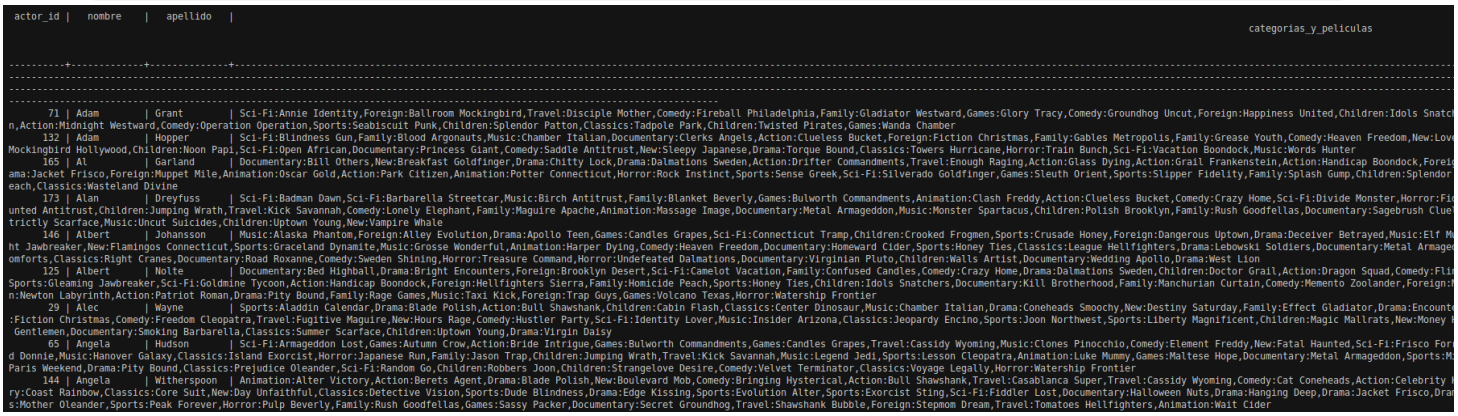
```
      string_agg(a.first_name || a.last_name, ',') AS actores
  FROM film f
      INNER JOIN film_category fc ON fc.film_id = f.film_id
      INNER JOIN category c ON c.category_id = fc.category_id
      INNER JOIN film_actor fa ON fa.film_id = f.film_id
      INNER JOIN actor a ON a.actor_id = fa.actor_id
        GROUP BY f.film_id, f.title, f.description, c.name, f.rental_rate,
f.length,
                 f.rating
        ORDER BY f.film_id;
```

id_pelicula	titulo	descripcion	categoria	precio	duracion	clasificacion	actores
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	RockDukakis,MaryKeitel,JohnnyCage,PenelopeGuiness,SandraPeck,ChristianGable,OprahKilmer,WarrenNolte,LucilleTracy,MenaTemple
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	MinnieZellweger,ChrisDepp,BobFawcett,SeanGuiness
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	CameronStreep,BobFawcett,NickWahlberg,RayJohansson,JulianneDench
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	JodieDegeneres,KennethPesci,FayWinslet,OprahKilmer,ScarlettDamon
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	DustinTautou,MatthewLeigh,CaryPhoenix,MatthewCarrey,ThoraTemple
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	WarrenNolte,SandraKilmer,JayneNeeson,MorganWilliams,KirstenPaltrow,KennethHoffman,ReeseWest
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	MenaHopper,JimMostel,MichaelBolger,OprahKilmer,RichardPenn
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	LucilleDee,SusanDavis,FayKilmer,GeneWillis
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	WilliamHackman,RipCrawford,RipWinslet,GretaKeitel,ChristianGable,MenaTemple,MerylAllen,WarrenNolte,ElvisMarx
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	GretaMalden,RockDukakis,RayJohansson,ReneeTracy,ValBolger,JudyDean,JadaRyder,AlecWayne
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	MichaelBening,JohnnyCage,ScarlettDamon,SeanGuiness
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	ValBolger,BurtPosey,GeneMckellen,JeffSilverstone,SylvesterDern,AlbertJohansson,SidneyCrowe
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	ChristopherBerry,KennethTorn,CaryMcconaughey,JonChase,MorganMcdormand
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	WoodyHoffman,RockDukakis,MinnieZellweger,MorganWilliams
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	SidneyCrowe,KennethPaltrow,HumphreyWillis,ReneeTracy,BurtDukakis,MenaHopper
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	GregoryGooding,JudeCruise,JohnSuvari,KarlBerry,AlbertJohansson
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	ReneeBall,EdChase,SpencerDepp,KarlBerry,LaurenceBullock,WoodyJolie,ChrisDepp,UmaWood
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	JadaRyder,AngelaWitherspoon,OprahKilmer,ReeseKilmer
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	PenelopeCronyn,KirkJovovich,JuliaMcqueen,JohnnyLollobrigida,ValBolger,JamesPitt
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	LauraBrody,CarmenHunt,EwanGooding,IanTandy,WalterTorn,EdMansfield

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 ":"

SQL

```
SELECT a.actor_id, a.first_name AS nombre, a.last_name AS apellido,
       string_agg(c.name || ':' || f.title, ',') AS categorias_y_peliculas
  FROM actor a
       INNER JOIN film_actor fa ON fa.actor_id = a.actor_id
       INNER JOIN film f ON f.film_id = fa.film_id
       INNER JOIN film_category fc ON fc.film_id = f.film_id
       INNER JOIN category c ON c.category_id = fc.category_id
         GROUP BY a.actor_id, a.first_name, a.last_name
         ORDER BY a.first_name, a.last_name;
```



actor_id	nombre	apellido	categorias_y_peliculas
71	Adam	Grant	Sci-Fi:Annie Identity,Foreign:Ballroom Mockingbird,Travel:Disciple Mother,Comedy:Fireball Philadelphia,Family:Gladiator Westward,Games:Glory Tracy,Comedy:Groundhog Uncut,Foreign:Happiness United,Children:Idols Snatch,Action:Midnight Westward,Comedy:Operation Operation,Sports:Seabiscuit Punk,Children:Splendor Patton,Classics:Tadpole Park,Children:Twisted Pirates,Games:Wanda Chamber
132	Adam	Hopper	Sci-Fi:Blindness Gun,Family:Blood Argonauts,Music:Chamber Italian,Documentary:Clerks Angels,Action:Clueless Bucket,Foreign:Fiction Christmas,Family:Gables Metropolis,Family:Grease Youth,Comedy:Heaven Freedom,New:Love Mockingbird Hollywood,Children:Noon Papi,Sci-Fi:Open African,Documentary:Princess Giant,Comedy:Saddle Antitrust,New:Sleepy Japanese,Drama:Torque Bound,Classics:Towers Hurricane,Horror:Train Bunch,Sci-Fi:Vacation Boondock,Music:Words Hunter
165	Al	Garland	Documentary:Bill Others,New:Breakfast Goldfinger,Drama:Chitty Lock,Drama:Dalmations Sweden,Action:Drifter Commandments,Travel:Enough Raging,Action:Glass Dying,Action:Grail Frankenstein,Action:Handicap Boondock,Foreigh:Jacket Frisco,Foreign:Muppet Mile,Animation:Oscar Gold,Action:Park Citizen,Animation:Potter Connecticut,Horror:Rock Instinct,Sports:Sense Greek,Sci-Fi:Silverado Goldfinger,Games:Sleuth Orient,Sports:Slipper Fidelity,Family:Splash Gump,Children:Splendor Beach,Classics:Wasteland Divine
173	Alan	Dreyfuss	Sci-Fi:Badman Dawn,Sci-Fi:Barbarella Streetcar,Music:Birch Antitrust,Family:Blanket Beverly,Games:Bulworth Commandments,Animation:Clash Freddy,Action:Clueless Bucket,Comedy:Crazy Home,Sci-Fi:Divide Monster,Horror:Flounted Antitrust,Children:Jumping Wrath,Travel:Kick Savannah,Comedy:Lonely Elephant,Family:Maguire Apache,Animation:Massage Image,Documentary:Metal Armageddon,Music:Monster Spartacus,Children:Polish Brooklyn,Family:Rush Goodfellas,Documentary:Sagebrush Cluetrictly Scarface,Music:Uncut Suicides,Children:Uptown Young,New:Vampire Whale
146	Albert	Johansson	Music:Alaska Phantom,Foreign:Alley Evolution,Drama:Apollo Teen,Games:Candles Grapes,Sci-Fi:Connecticut Tramp,Children:Crooked Frogmen,Sports:Crusade Honey,Foreign:Dangerous Uptown,Drama:Deceiver Betrayed,Music:Elf Mht Jawbreaker,New:Flamingos Connecticut,Sports:Graceland Dynamite,Music:Grosse Wonderful,Animation:Harper Dying,Comedy:Heaven Freedom,Documentary:Homeward Cider,Sports:Honey Ties,Classics:League Hellfighters,Drama:Lebowski Soldiers,Documentary:Metal Armageomforts,Classics:Right Cranes,Documentary:Road Roxanne,Comedy:Sweden Shining,Horror:Treasure Command,Horror:Undefeated Dalmations,Documentary:Virginian Pluto,Children:Walls Artist,Documentary:Wedding Apollo,Drama:West Lion
125	Albert	Nolte	Documentary:Bed Highball,Drama:Bright Encounters,Foreign:Brooklyn Desert,Sci-Fi:Camelot Vacation,Family:Confused Candles,Comedy:Crazy Home,Drama:Dalmations Sweden,Children:Doctor Grail,Action:Dragon Squad,Comedy:FlSports:Gleaming Jawbreaker,Sci-Fi:Goldmine Tycoon,Action:Handicap Boondock,Foreign:Hellfighters Sierra,Family:Homicide Peach,Sports:Honey Ties,Children:Idols Snatchers,Documentary:Kill Brotherhood,Family:Manchurian Curtain,Comedy:Memento Zoolander,Foreign:Nn:Newton Labyrinth,Action:Patriot Roman,Drama:Pity Bound,Family:Rage Games,Music:Taxi Kick,Foreign:Trap Guys,Games:Volcano Texas,Horror:Watership Frontier
29	Alec	Wayne	Sports:Aladdin Calendar,Drama:Blade Polish,Action:Bull Shawshank,Children:Cabin Flash,Classics:Center Dinosaur,Music:Chamber Italian,Drama:Coneheads Smoochy,New:Destiny Saturday,Family:Effect Gladiator,Drama:Encounte:Fiction Christmas,Comedy:Freedom Cleopatra,Travel:Fugitive Maguire,New:Hours Rage,Comedy:Hustler Party,Sci-Fi:Identity Lover,Music:Insider Arizona,Classics:Jeopardy Encino,Sports:Joon Northwest,Sports:Liberty Magnificent,Children:Magic Mallrats,New:Money HGentlemen,Documentary:Smoking Barbarella,Classics:Summer Scarface,Children:Uptown Young,Drama:Virgin Daisy
65	Angela	Hudson	Sci-Fi:Armageddon Lost,Games:Autumn Crow,Action:Bride Intrigue,Games:Bulworth Commandments,Games:Candles Grapes,Travel:Cassidy Wyoming,Music:Clones Pinocchio,Comedy:Element Freddy,New:Fatal Haunted,Sci-Fi:Frisco Fod Donnie,Music:Hanover Galaxy,Classics:Island Exorcist,Horror:Japanese Run,Family:Jason Trap,Children:Jumping Wrath,Travel:Kick Savannah,Music:Legend Jedi,Sports:Lesson Cleopatra,Animation:Luke Mummy,Games:Maltese Hope,Documentary:Metal Armageddon,Sports:MParis Weekend,Drama:Pity Bound,Classics:Prejudice Oleander,Sci-Fi:Random Go,Children:Robbers Joon,Children:Strangelove Desire,Comedy:Velvet Terminator,Classics:Voyage Legally,Horror:Watership Frontier
144	Angela	Witherspoon	Animation:Alter Victory,Action:Berets Agent,Drama:Blade Polish,New:Boulevard Mob,Comedy:Bringing Hysterical,Action:Bull Shawshank,Travel:Casablanca Super,Travel:Cassidy Wyoming,Comedy:Cat Coneheads,Action:Celebrity Ury:Coast Rainbow,Classics:Core Suit,New:Day Unfaithful,Classics:Detective Vision,Sports:Dude Blindness,Drama:Edge Kissing,Sports:Evolution Alter,Sports:Exorcist Sting,Sci-Fi:Fiddler Lost,Documentary:Halloween Nuts,Drama:Hanging Deep,Drama:Jacket Frisco,Drags:Mother Oleander,Sports:Peak Forever,Horror:Pulp Beverly,Family:Rush Goodfellas,Games:Sassy Packer,Documentary:Secret Groundhog,Travel:Shawshank Bubble,Foreign:Stepmom Dream,Travel:Tomatoes Hellfighters,Animation:Wait Cider

5. Realice todas las vistas de las consultas anteriores. Colóqueles el

prefijo `view_` a su denominación.

SQL

```sql
-- a. Obtenga las ventas totales por categoría de películas ordenadas
descendentemente.
CREATE VIEW view_ventas_por_categoria AS
            SELECT c.name AS categoria, SUM(p.amount) AS venta_total
              FROM payment p
                   INNER JOIN rental r ON r.rental_id = p.rental_id
                   INNER JOIN inventory i on i.inventory_id = r.inventory_id
                   INNER JOIN film_category fc ON fc.film_id = i.film_id
                   INNER JOIN category c ON c.category_id = fc.category_id
                     GROUP BY categoria
                     ORDER BY venta_total 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
CREATE VIEW view_ventas_por_tienda AS
            SELECT s.store_id AS tienda, ci.city AS ciudad, co.country AS pais,
                   st.first_name AS nombre_encagado,
                   st.last_name AS apellido_encargado, SUM(p.amount) AS
venta_total
              FROM payment p
                   INNER JOIN rental r ON r.rental_id = p.rental_id
                   INNER JOIN inventory i ON i.inventory_id = r.inventory_id
                   INNER JOIN store s ON s.store_id = i.store_id
                   INNER JOIN address a ON a.address_id = s.address_id
                   INNER JOIN city ci ON ci.city_id = a.city_id
                   INNER JOIN country co ON co.country_id = ci.country_id
                   INNER JOIN staff st ON st.staff_id = s.manager_staff_id
                     GROUP BY s.store_id, ci.city, co.country, st.first_name,
                              st.last_name
                     ORDER BY venta_total DESC;

-- c. 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
CREATE VIEW view_lista_peliculas AS
            SELECT f.film_id AS id_pelicula, 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(a.first_name || a.last_name, ',') AS actores
              FROM film f
                   INNER JOIN film_category fc ON fc.film_id = f.film_id
                   INNER JOIN category c ON c.category_id = fc.category_id
                   INNER JOIN film_actor fa ON fa.film_id = f.film_id
                   INNER JOIN actor a ON a.actor_id = fa.actor_id
                     GROUP BY f.film_id, f.title, f.description, c.name,
                              f.rental_rate, f.length, f.rating
                     ORDER BY f.film_id;
```

```sql
-- 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 “:”
CREATE VIEW view_info_actores AS
            SELECT a.actor_id, a.first_name AS nombre, a.last_name AS apellido,
                  string_agg(c.name || ':' || f.title, ',') AS
                  categorias_y_peliculas
              FROM actor a
                  INNER JOIN film_actor fa ON fa.actor_id = a.actor_id
                  INNER JOIN film f ON f.film_id = fa.film_id
                  INNER JOIN film_category fc ON fc.film_id = f.film_id
                  INNER JOIN category c ON c.category_id = fc.category_id
                    GROUP BY a.actor_id, a.first_name, a.last_name
                    ORDER BY a.first_name, a.last_name;
```

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

- Análisis:
  - *film*: los campos numéricos no deben ser negativos (*rental_rate* y *replacement_cost* ≥ 0, *rental_duration y length* > 0).
  - *customer*: *email* si existe debe tener un formato razonable, *create_date* no debe ser en el futuro.
  - *address*: *phone* debe tener formato razonable, *postal_code* longitud ≤ 10.
  - *payment*: *amount* >= 0, *payment_date* no en el futuro.
  - *rental*: *return_date* NULL o >= *rental_date*.
  - *staff*: *username* con tamaño mínimo, active boolean ya NOT NULL. *email* formato opcional.

- Creamos inicialmente las restricciones CHECK a añadir pero manteniéndolas *NOT VALID*:

```sql
SQL
-- film: tasas y costes no negativos, duracion y rental_duration positivas
ALTER TABLE film
ADD CONSTRAINT chk_film_rental_rate_nonneg
CHECK (rental_rate >= 0) NOT VALID;

ALTER TABLE film
ADD CONSTRAINT chk_film_replacement_cost_nonneg
CHECK (replacement_cost >= 0) NOT VALID;

-- film: rental_duration ya default 3 y NOT NULL pero aseguramos > 0
ALTER TABLE film
ADD CONSTRAINT chk_film_rental_duration_positive
CHECK (rental_duration > 0) NOT VALID;

-- film: length puede ser NULL en esquema. Si no es NULL, asegurar > 0
ALTER TABLE film
ADD CONSTRAINT chk_film_length_positive_or_null
CHECK (length IS NULL OR length > 0) NOT VALID;
```

```sql
-- customer: email (si no es NULL) debe tener formato clásico y create_date no debe
-- tener una fecha futura
ALTER TABLE customer
ADD CONSTRAINT chk_customer_email_format
CHECK (
     email IS NULL OR email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
) NOT VALID;

ALTER TABLE customer
ADD CONSTRAINT chk_customer_create_date_past
CHECK (create_date <= now()::date) NOT VALID;


-- address: telefono y postal_code longitud máxima razonable
ALTER TABLE address
ADD CONSTRAINT chk_address_phone_format
CHECK (phone IS NULL OR TRIM(phone) = '' OR phone ~ '^[0-9\+\-\(\) .]{6,20}$') NOT VALID;

ALTER TABLE address
ADD CONSTRAINT chk_address_postal_length
CHECK (postal_code IS NULL OR char_length(postal_code) <= 10) NOT VALID;


-- payment: importe positivo y payment_date no en el futuro
ALTER TABLE payment
ADD CONSTRAINT chk_payment_amount_positive
CHECK (amount >= 0) NOT VALID;

ALTER TABLE payment
ADD CONSTRAINT chk_payment_date_past
CHECK (payment_date <= now()) NOT VALID;


-- rental: return_date NULL o >= rental_date
ALTER TABLE rental
ADD CONSTRAINT chk_rental_return_after_rental
CHECK (return_date IS NULL OR return_date >= rental_date) NOT VALID;


-- staff: username con longitud mínima y email con formato clásico
ALTER TABLE staff
ADD CONSTRAINT chk_staff_username_len
CHECK (char_length(username) >= 3) NOT VALID;

ALTER TABLE staff
ADD CONSTRAINT chk_staff_email_format
CHECK (
     email IS NULL OR email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
) NOT VALID;
```

- Una vez comprobado que no habían filas que incumpliesen, valido las constraints.

```sql
ALTER TABLE film VALIDATE CONSTRAINT chk_film_rental_rate_nonneg;
ALTER TABLE film VALIDATE CONSTRAINT chk_film_replacement_cost_nonneg;
ALTER TABLE film VALIDATE CONSTRAINT chk_film_rental_duration_positive;
ALTER TABLE film VALIDATE CONSTRAINT chk_film_length_positive_or_null;

ALTER TABLE customer VALIDATE CONSTRAINT chk_customer_email_format;
ALTER TABLE customer VALIDATE CONSTRAINT chk_customer_create_date_past;

ALTER TABLE address VALIDATE CONSTRAINT chk_address_phone_format;
ALTER TABLE address VALIDATE CONSTRAINT chk_address_postal_length;

ALTER TABLE payment VALIDATE CONSTRAINT chk_payment_amount_positive;
ALTER TABLE payment VALIDATE CONSTRAINT chk_payment_date_past;

ALTER TABLE rental VALIDATE CONSTRAINT chk_rental_return_after_rental;

ALTER TABLE staff VALIDATE CONSTRAINT chk_staff_username_len;
ALTER TABLE staff VALIDATE CONSTRAINT chk_staff_email_format;
```

7. Explique la sentencia que aparece en la tabla `customer`

Triggers:

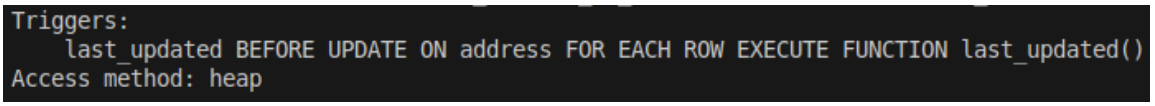
`last_updated BEFORE UPDATE ON customer`

`FOR EACH ROW EXECUTE PROCEDURE last_updated()`

Identifique alguna tabla donde se utilice una solución similar.

Esa sentencia indica que existe un trigger que se ejecuta antes de cada operación *UPDATE* sobre la tabla customer. *FOR EACH ROW* significa que se ejecuta una vez por cada fila modificada y *EXECUTE PROCEDURE last_updated()* llama a la función *last_updated()*. La función *last_updated()* actualiza el campo *last_update* de la fila a la fecha/hora actual.

Una tabla que utiliza una solución similar es la tabla *address*:



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` y el identificador del `film`.

```sql
-- Tabla para inserciones
CREATE TABLE film_insert_log (
    film_id      integer NOT NULL,
    last_date    timestamp,
    last_user    text
);
```

```sql
-- Función de trigger
CREATE FUNCTION film_insert_log_fn() RETURNS TRIGGER AS $film_insert_log_fn$
    BEGIN
        IF NEW.film_id IS NULL THEN
            RAISE EXCEPTION 'film_id cannot be null';
        END IF;

        INSERT INTO film_insert_log (film_id, last_date, last_user)
        VALUES (NEW.film_id, current_timestamp, current_user);

        RETURN NEW;
    END;
$film_insert_log_fn$ LANGUAGE plpgsql;

-- Trigger sobre film
CREATE TRIGGER trg_film_before_insert BEFORE INSERT ON film
FOR EACH ROW EXECUTE PROCEDURE film_insert_log_fn();
```

Comprobamos que funciona correctamente:

```sql
INSERT INTO film (title, language_id, rental_duration, rental_rate, replacement_cost,
fulltext)
VALUES ('test', 3, 3, 3, 3, to_tsvector('test'));
```

```
alquilerdvd=# SELECT *
alquilerdvd-# FROM film_insert_log;
 film_id |         last_date          | last_user
---------+----------------------------+-----------
    1001 | 2025-10-27 17:20:11.461305 | postgres
    1002 | 2025-10-27 17:26:12.938995 | postgres
(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`.**

```sql
-- Tabla para eliminaciones
CREATE TABLE film_delete_log (
  film_id      integer NOT NULL,
  last_date    timestamp,
  last_user    text
);
-- Función de trigger
CREATE FUNCTION film_delete_log_fn() RETURNS TRIGGER AS $film_delete_log_fn$
    BEGIN
      IF OLD.film_id IS NULL THEN
        RAISE EXCEPTION 'OLD.film_id cannot be null';
      END IF;
```

```
      INSERT INTO film_delete_log (film_id, last_date, last_user)
      VALUES (OLD.film_id, current_timestamp, current_user);

      RETURN OLD;
    END;
$film_delete_log_fn$ LANGUAGE plpgsql;

-- Trigger sobre film
CREATE TRIGGER trg_film_before_delete BEFORE DELETE ON film
FOR EACH ROW EXECUTE PROCEDURE film_delete_log_fn();
```

Comprobamos que funciona correctamente:

```
SQL
DELETE FROM film WHERE title = 'test';
```

```
alquilerdvd=# SELECT *
alquilerdvd-# FROM film_delete_log;
 film_id |         last_date          | last_user
---------+----------------------------+-----------
    1002 | 2025-10-27 17:40:38.539193 | postgres
(1 row)
```

10. Comente el significado y la relevancia de las secuencias.

Una secuencia es un objeto de la base de datos que entrega números enteros uno detrás de otro. Se usa habitualmente para asignar automáticamente identificadores (IDs) a las filas. Cuando una sesión pide el siguiente número con nextval() recibe un valor único y la secuencia avanza. Esto funciona bien aun cuando varias sesiones piden números a la vez porque PostgreSQL garantiza que no se repitan. Las secuencias se pueden configurar (valor inicial, incremento, caché) y, después de importar datos, es habitual sincronizarlas con setval(...) para que continúen desde el máximo ID existente. En resumen, las secuencias facilitan y hacen segura la generación automática de IDs en entornos con muchos usuarios simultáneos, aunque a costa de aceptar posibles huecos en la numeración.