Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Modelo relacional. Vistas y disparadores

Néstor Rubén Delgado Feliciano

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

T: 900 43 25 26

ull.es

Índice

Introducción

Este informe presenta las actividades realizadas en la práctica de SQL y PL/pgSQL, centradas en la simulación de una base de datos para una videoteca, inspirada en la franquicia Blockbuster LLC. A través de esta práctica, se busca afianzar habilidades en operaciones básicas de SQL, el manejo de vistas y disparadores. Se trabajará con una base de datos restaurada de ***alquilerdvd.tar***, realizando consultas sobre ventas, películas y actores, así como la implementación de triggers y el análisis del modelo de datos.

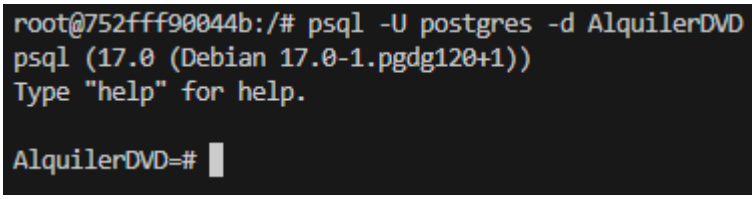
Realice la restauración de la base de datos alquilerdvd.tar.

Para realizar la restauración de la base de datos y después conectarnos a la misma ejecutamos los siguientes comandos:

```
Unset
pg_restore -d AlquilerDVD -U postgres -h localhost -p 5432 ./docker-entrypoint-initdb.d/AlquilerPractica.tar

psql -U postgres -d AlquilerDVD
```

En la siguiente imagen se puede comprobar que funciona correctamente.



Identifique las tablas, vistas y secuencias.

Ejecutar el comando **\dt** para mostrar las tablas:

```
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)
```

Ejecutar los comandos **\dv** y **\ds** para mostrar las vistas y secuencias respectivamente:

```
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)
```

Identifique las tablas principales y sus principales elementos.

Para ver la información específica de cada tabla use el comando **\d nombre_tabla.**

Tabla “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 |
Indexes:
    "film_pkey" PRIMARY KEY, btree (film_id)
    "film_fulltext_idx" gist (fulltext)
    "idx_fk_language_id" btree (language_id)
    "idx_title" btree (title)
Foreign-key constraints:
    "film_language_id_fkey" FOREIGN KEY (language_id) REFERENCES language(language_id) ON UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "film_actor" CONSTRAINT "film_actor_film_id_fkey" FOREIGN KEY (film_id) REFERENCES film(film_id) ON UPDATE CASCADE ON DELETE RESTRICT
    TABLE "film_category" CONSTRAINT "film_category_film_id_fkey" FOREIGN KEY (film_id) REFERENCES film(film_id) ON UPDATE CASCADE ON DELETE RESTRICT
    TABLE "inventory" CONSTRAINT "inventory_film_id_fkey" FOREIGN KEY (film_id) REFERENCES film(film_id) ON UPDATE CASCADE ON DELETE RESTRICT
Triggers:
    film_fulltext_trigger BEFORE INSERT OR UPDATE ON film FOR EACH ROW EXECUTE FUNCTION tsvector_update_trigger('fulltext', 'pg_catalog.english', 'title', 'description')
    last_updated BEFORE UPDATE ON film FOR EACH ROW EXECUTE FUNCTION last_updated()
```

Tabla “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()
Indexes:
    "rental_pkey" PRIMARY KEY, btree (rental_id)
    "idx_fk_inventory_id" btree (inventory_id)
    "idx_unq_rental_rental_date_inventory_id_customer_id" UNIQUE, btree (rental_date, inventory_id, customer_id)
Foreign-key constraints:
    "rental_customer_id_fkey" FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON UPDATE CASCADE ON DELETE RESTRICT
    "rental_inventory_id_fkey" FOREIGN KEY (inventory_id) REFERENCES inventory(inventory_id) ON UPDATE CASCADE ON DELETE RESTRICT
    "rental_staff_id_key" FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
Referenced by:
    TABLE "payment" CONSTRAINT "payment_rental_id_fkey" FOREIGN KEY (rental_id) REFERENCES rental(rental_id) ON UPDATE CASCADE ON DELETE SET NULL
Triggers:
    last_updated BEFORE UPDATE ON rental FOR EACH ROW EXECUTE FUNCTION last_updated()
```

Tabla “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 |
Indexes:
    "payment_pkey" PRIMARY KEY, btree (payment_id)
    "idx_fk_customer_id" btree (customer_id)
    "idx_fk_rental_id" btree (rental_id)
    "idx_fk_staff_id" btree (staff_id)
Foreign-key constraints:
    "payment_customer_id_fkey" FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON UPDATE CASCADE ON DELETE RESTRICT
    "payment_rental_id_fkey" FOREIGN KEY (rental_id) REFERENCES rental(rental_id) ON UPDATE CASCADE ON DELETE SET NULL
    "payment_staff_id_fkey" FOREIGN KEY (staff_id) REFERENCES staff(staff_id) ON UPDATE CASCADE ON DELETE RESTRICT
```

Tabla “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                     |           |          |
Indexes:
    "customer_pkey" PRIMARY KEY, btree (customer_id)
    "idx_fk_address_id" btree (address_id)
    "idx_fk_store_id" btree (store_id)
    "idx_last_name" btree (last_name)
Foreign-key constraints:
    "customer_address_id_fkey" FOREIGN KEY (address_id) REFERENCES address(address_id) ON UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "payment" CONSTRAINT "payment_customer_id_fkey" FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON UPDATE CASCADE ON DELETE RESTRICT
    TABLE "rental" CONSTRAINT "rental_customer_id_fkey" FOREIGN KEY (customer_id) REFERENCES customer(customer_id) ON UPDATE CASCADE ON DELETE RESTRICT
Triggers:
    last_updated BEFORE UPDATE ON customer FOR EACH ROW EXECUTE FUNCTION last_updated()
```

Tabla “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                       |           |          |
Indexes:
    "staff_pkey" PRIMARY KEY, btree (staff_id)
Foreign-key constraints:
    "staff_address_id_fkey" FOREIGN KEY (address_id) REFERENCES address(address_id) ON UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "payment" CONSTRAINT "payment_staff_id_fkey" FOREIGN KEY (staff_id) REFERENCES staff(staff_id) ON UPDATE CASCADE ON DELETE RESTRICT
    TABLE "rental" CONSTRAINT "rental_staff_id_key" FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
    TABLE "store" CONSTRAINT "store_manager_staff_id_fkey" FOREIGN KEY (manager_staff_id) REFERENCES staff(staff_id) ON UPDATE CASCADE ON DELETE RESTRICT
Triggers:
    last_updated BEFORE UPDATE ON staff FOR EACH ROW EXECUTE FUNCTION last_updated()
```

Tabla “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()
Indexes:
    "store_pkey" PRIMARY KEY, btree (store_id)
    "idx_unq_manager_staff_id" UNIQUE, btree (manager_staff_id)
Foreign-key constraints:
    "store_address_id_fkey" FOREIGN KEY (address_id) REFERENCES address(address_id) ON UPDATE CASCADE ON DELETE RESTRICT
    "store_manager_staff_id_fkey" FOREIGN KEY (manager_staff_id) REFERENCES staff(staff_id) ON UPDATE CASCADE ON DELETE RESTRICT
Triggers:
    last_updated BEFORE UPDATE ON store FOR EACH ROW EXECUTE FUNCTION last_updated()
```

Tabla “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()
Indexes:
    "inventory_pkey" PRIMARY KEY, btree (inventory_id)
    "idx_store_id_film_id" btree (store_id, film_id)
Foreign-key constraints:
    "inventory_film_id_fkey" FOREIGN KEY (film_id) REFERENCES film(film_id) ON UPDATE CASCADE ON DELETE RESTRICT
Referenced by:
    TABLE "rental" CONSTRAINT "rental_inventory_id_fkey" FOREIGN KEY (inventory_id) REFERENCES inventory(inventory_id) ON UPDATE CASCADE ON DELETE RESTRICT
Triggers:
    last_updated BEFORE UPDATE ON inventory FOR EACH ROW EXECUTE FUNCTION last_updated()
```

Estas tablas son fundamentales en una base de datos para un videoclub porque cubren las entidades clave y los procesos operativos: ofrecer películas, gestionar alquileres, recibir pagos, y organizar el inventario y personal en las distintas sucursales.

1. **film**: Esta tabla almacena información sobre cada película disponible, incluyendo su título, descripción, año de lanzamiento, el coste de reemplazarla, su puntuación de críticas o su duración. Es crucial para definir el inventario de películas que el videoclub ofrece a sus clientes.

2. **rental**: Registra cada transacción de alquiler, incluyendo fechas de inicio y retorno. Es fundamental para rastrear el historial de alquileres y gestionar la disponibilidad de películas.

3. **payment**: Almacena los pagos realizados por los clientes por cada alquiler así como su fecha. Es indispensable para el control financiero y el registro de ingresos derivados de las transacciones de alquiler. Se guardan datos importantes como la cantidad del pago y su fecha.

4. **customer**: Contiene información sobre cada cliente registrado, como su nombre, email de contacto o si está activo o no, lo cual es esencial para gestionar quiénes pueden alquilar y llevar un control de sus transacciones y pagos.

5. **staff**: La tabla del personal almacena los datos de los empleados de cada tienda como su nombre, email de contacto o si está activo o no, permitiendo gestionar la atención y soporte a los clientes, así como la asignación de responsabilidades dentro del sistema.

6. **store**: Esta tabla representa cada sucursal o tienda física de la franquicia. Distinguir entre tiendas es clave para el sistema, ya que cada una tiene su propio inventario, empleados y base de clientes locales. Guarda información como la dirección o el manager a cargo.

7. **inventory**: Permite saber la disponibilidad de cada película en cada tienda, ya que una película puede estar en varias tiendas a la vez. La gestión de inventario es vital para responder a la demanda y saber cuántas copias de cada película están disponibles para alquiler en cada ubicación.

Realice las siguientes consultas.

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

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

```
    name     | total_sells
-------------+-------------
 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.

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

```
     city_country     | manager_staff_id | manager_name  | total_sells
----------------------+------------------+---------------+-------------
 Woodridge, Australia |                2 | Jon, Stephens |    30683.13
 Lethbridge, Canada   |                1 | 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).**

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

film_id	title	description	category	rental_rate	length	rating	actor
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	Penelope Guiness
23	Anaconda Confessions	A Lacklusture Display of a Dentist And a Dentist who must Fight a Girl in Australia	Animation	0.99	92	R	Penelope Guiness
25	Angels Life	A Thoughtful Display of a Woman And a Astronaut who must Battle a Robot in Berlin	New	2.99	74	G	Penelope Guiness
106	Bulworth Commandments	A Amazing Display of a Mad Cow And a Pioneer who must Redeem a Sumo Wrestler in The Outback	Games	2.99	61	G	Penelope Guiness
140	Cheaper Clyde	A Emotional Character Study of a Pioneer And a Girl who must Discover a Dog in Ancient Japan	Sci-Fi	0.99	87	G	Penelope Guiness
166	Color Philadelphia	A Thoughtful Panorama of a Car And a Crocodile who must Sink a Monkey in The Sahara Desert	Classics	2.99	149	G	Penelope Guiness
277	Elephant Trojan	A Beautiful Panorama of a Lumberjack And a Forensic Psychologist who must Overcome a Frisbee in A Baloon	Horror	4.99	126	PG-13	Penelope Guiness
361	Gleaming Jawbreaker	A Amazing Display of a Composer And a Forensic Psychologist who must Discover a Car in The Canadian Rockies	Sports	2.99	89	NC-17	Penelope Guiness
438	Human Graffiti	A Beautiful Reflection of a Womanizer And a Sumo Wrestler who must Chase a Database Administrator in The Gulf of Mexico	Games	2.99	68	NC-17	Penelope Guiness
499	King Evolution	A Action-Packed Tale of a Boy And a Lumberjack who must Chase a Madman in A Baloon	Family	4.99	184	NC-17	Penelope Guiness
506	Lady Stage	A Beautiful Character Study of a Woman And a Man who must Pursue a Explorer in A U-Boat	Horror	4.99	67	PG	Penelope Guiness
509	Language Cowboy	A Epic Yarn of a Cat And a Madman who must Vanquish a Dentist in An Abandoned Amusement Park	Children	0.99	78	NC-17	Penelope Guiness
605	Mulholland Beast	A Awe-Inspiring Display of a Husband And a Squirrel who must Battle a Sumo Wrestler in A Jet Boat	Foreign	2.99	157	PG	Penelope Guiness
635	Oklahoma Jumanji	A Thoughtful Drama of a Dentist And a Womanizer who must Meet a Husband in The Sahara Desert	New	0.99	58	PG	Penelope Guiness
749	Rules Human	A Beautiful Epistle of a Astronaut And a Student who must Confront a Monkey in An Abandoned Fun House	Horror	4.99	153	R	Penelope Guiness
832	Splash Gump	A Taut Saga of a Crocodile And a Boat who must Conquer a Hunter in A Shark Tank	Family	0.99	175	PG	Penelope Guiness
939	Vertigo Northwest	A Unbelieveable Display of a Mad Scientist And a Mad Scientist who must Outgun a Mad Cow in Ancient Japan	Comedy	2.99	90	R	Penelope Guiness
970	Westward Seabiscuit	A Lacklusture Tale of a Butler And a Husband who must Face a Boy in Ancient China	Classics	0.99	52	NC-17	Penelope Guiness
980	Wizard Coldblooded	A Lacklusture Display of a Robot And a Girl who must Defeat a Sumo Wrestler in A MySQL Convention	Music	4.99	75	PG	Penelope 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	Nick Wahlberg
31	Apache Divine	A Awe-Inspiring Reflection of a Pastry Chef And a Teacher who must Overcome a Sumo Wrestler in A U-Boat	Family	4.99	92	NC-17	Nick Wahlberg
47	Baby Hall	A Boring Character Study of a A Shark And a Girl who must Outrace a Feminist in An Abandoned Mine Shaft	Foreign	4.99	153	NC-17	Nick Wahlberg
105	Bull Shawshank	A Fanciful Drama of a Moose And a Squirrel who must Conquer a Pioneer in The Canadian Rockies	Action	0.99	125	NC-17	Nick Wahlberg
132	Chainsaw Uptown	A Beautiful Documentary of a Boy And a Robot who must Discover a Squirrel in Australia	Sci-Fi	0.99	114	PG	Nick Wahlberg
145	Chisum Behavior	A Epic Documentary of a Sumo Wrestler And a Butler who must Kill a Car in Ancient India	Family	4.99	124	G	Nick Wahlberg

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

```
-[ RECORD 1 ]------------------------------------------------------------------
actor      | Penelope Guiness
categories | Animation:Children:Classics:Comedy:Documentary:Family:Foreign:Games:Horror:Music:New:Sci-Fi:Sports
films      | Academy Dinosaur:Anaconda Confessions:Angels Life:Bulworth Commandments:Cheaper Clyde:Color Philadelphia:Elephant Trojan:Gleaming Jawbreaker:Human Graffiti:King Evolution:Lady Stage:Language Cowboy:Mulholland Beast:Oklahoma Jumanji:Rules Human:Splas
h Gump:Vertigo Northwest:Westward Seabiscuit:Wizard Coldblooded
-[ RECORD 2 ]------------------------------------------------------------------
actor      | Nick Wahlberg
categories | Action:Animation:Children:Classics:Comedy:Documentary:Drama:Family:Foreign:Games:Music:New:Sci-Fi:Travel
films      | Adaptation Holes:Apache Divine:Baby Hall:Bull Shawshank:Chainsaw Uptown:Chisum Behavior:Destiny Saturday:Dracula Crystal:Fight Jawbreaker:Flash Wars:Gilbert Pelican:Goodfellas Salute:Happiness United:Indian Love:Jekyll Frogmen:Jersey Sassy:Liaisons
Sweet:Lucky Flying:Maguire Apache:Mallrats United:Mask Peach:Roof Champion:Rushmore Mermaid:Smile Earring:Wardrobe Phantom
-[ RECORD 3 ]------------------------------------------------------------------
actor      | Ed Chase
categories | Action:Classics:Documentary:Drama:Foreign:Music:New:Sci-Fi:Sports:Travel
films      | Alone Trip:Army Flintstones:Artist Coldblooded:Boondock Ballroom:Caddyshack Jedi:Cowboy Doom:Eve Resurrection:Forrest Sons:French Holiday:Frost Head:Halloween Nuts:Hunter Alter:Image Princess:Jeepers Wedding:Luck Opus:Necklace Outbreak:Platoon Insti
nct:Spice Sorority:Wedding Apollo:Weekend Personal:Whale Bikini:Young Language
-[ RECORD 4 ]------------------------------------------------------------------
actor      | Jennifer Davis
categories | Action:Animation:Comedy:Documentary:Drama:Family:Horror:Music:New:Sci-Fi:Sports:Travel
films      | Anaconda Confessions:Angels Life:Barefoot Manchurian:Bed Highball:Blade Polish:Boondock Ballroom:Ghostbusters Elf:Greedy Roots:Hanover Galaxy:Instinct Airport:Jumanji Blade:National Story:Oklahoma Jumanji:Poseidon Forever:Raiders Antitrust:Random Go
:Reds Pocus:Silverado Goldfinger:Splash Gump:Submarine Bed:Treasure Command:Unforgiven Zoolander
```

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

Añadí **CREATE VIEW nombre AS** a cada una de las consultas anteriores.

```sql
-- Vista para 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_sells
FROM film_category fc
JOIN category c ON fc.category_id = c.category_id
JOIN film f ON fc.film_id = f.film_id
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
JOIN payment p ON r.rental_id = p.rental_id
GROUP BY c.name
ORDER BY total_sells DESC;

-- Vista para las ventas totales por tienda, incluyendo ciudad y país concatenados, y el e
CREATE VIEW view_total_sales_by_store AS
SELECT CONCAT(ci.city, ', ', co.country) AS city_country,
    s.manager_staff_id,
    CONCAT(st.first_name, ', ', st.last_name) AS manager_name,
    SUM(p.amount) AS total_sells
FROM store s
JOIN address a ON s.address_id = a.address_id
JOIN city ci ON a.city_id = ci.city_id
JOIN country co ON ci.country_id = co.country_id
JOIN inventory i ON s.store_id = i.store_id
JOIN rental r ON i.inventory_id = r.inventory_id
JOIN payment p ON r.rental_id = p.rental_id
JOIN staff st ON st.staff_id = s.manager_staff_id
GROUP BY s.store_id, ci.city, co.country, s.manager_staff_id, st.first_name, st.last_name
ORDER BY total_sells DESC;
```

```sql
-- Vista para la lista de películas con detalles específicos
CREATE VIEW view_film_list_with_details AS
SELECT f.film_id,
       f.title,
       f.description,
       c.name AS category,
       f.rental_rate AS price,
       f.length AS duration,
       f.rating AS classification,
       CONCAT(a.first_name, ' ', a.last_name) AS actor
FROM film f
JOIN film_actor fa ON f.film_id = fa.film_id
JOIN actor a ON fa.actor_id = a.actor_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id;

-- Vista para la información de actores, incluyendo categorías y películas concatenadas
CREATE VIEW view_actor_info_with_categories_and_films AS
SELECT CONCAT(a.first_name, ' ', a.last_name) AS actor,
       STRING_AGG(DISTINCT c.name, ':') AS categories,
       STRING_AGG(DISTINCT f.title, ':') AS films
FROM actor a
JOIN film_actor fa ON a.actor_id = fa.actor_id
JOIN film f ON fa.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category c ON fc.category_id = c.category_id
GROUP BY a.actor_id;
```

Imagen que muestra la correcta creación de las vistas:

```
CREATE VIEW
CREATE VIEW
CREATE VIEW
CREATE VIEW
AlquilerDVD=# \dv
                        List of relations
 Schema |                    Name                     | Type |  Owner
--------+---------------------------------------------+------+----------
 public | view_actor_info_with_categories_and_films | view | postgres
 public | view_film_list_with_details               | view | postgres
 public | view_total_sales_by_category              | view | postgres
 public | view_total_sales_by_store                 | view | postgres
(4 rows)
```

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

Revisé cada tabla, analizando cada atributo para identificar posibles restricciones que pudieran aplicarse. Cuando consideré que una restricción podía aportar valor, la añadí. En algunos casos, como en las expresiones regulares para campos como *distrito* o *dirección*, las restricciones no eran estrictamente necesarias. Sin embargo, bajo mi criterio, considero que agregar restricciones adicionales es beneficioso para asegurar la integridad y consistencia de los datos.

```
ALTER TABLE category
ADD CONSTRAINT chk_name CHECK (LENGTH(TRIM(name)) > 0 AND name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$');

ALTER TABLE language
ADD CONSTRAINT chk_name CHECK (LENGTH(TRIM(name)) > 0 AND name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$');

ALTER TABLE actor
ADD CONSTRAINT chk_first_name CHECK (LENGTH(TRIM(first_name)) > 0 AND first_name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$'),
ADD CONSTRAINT chk_last_name CHECK (LENGTH(TRIM(last_name)) > 0 AND last_name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$');

ALTER TABLE film
ADD CONSTRAINT chk_length CHECK (length > 0),
ADD CONSTRAINT chk_rating CHECK (rating IN ('G', 'PG', 'PG-13', 'R', 'NC-17')),
ADD CONSTRAINT chk_release_year CHECK (release_year BETWEEN 1900 AND EXTRACT(YEAR FROM CURRENT_DATE)),
ADD CONSTRAINT chk_rental_duration CHECK (rental_duration > 0),
ADD CONSTRAINT chk_rental_rate CHECK (rental_rate >= 0),
ADD CONSTRAINT chk_replacement_cost CHECK (replacement_cost >= 0),
ADD CONSTRAINT chk_title CHECK (LENGTH(TRIM(title)) > 0 AND title ~* '^[A-Za-z0-9À-ÖØ-öø-ÿ\s!?:&\-]+$'); -- \'

ALTER TABLE rental
ADD CONSTRAINT chk_rental_date CHECK (rental_date <= CURRENT_TIMESTAMP),
ADD CONSTRAINT chk_return_date CHECK (return_date > rental_date);


ALTER TABLE payment
ADD CONSTRAINT chk_payment_date CHECK (payment_date <= CURRENT_TIMESTAMP);

ALTER TABLE customer
ADD CONSTRAINT chk_create_date CHECK (create_date <= CURRENT_TIMESTAMP),
ADD CONSTRAINT chk_email CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}$'),
ADD CONSTRAINT chk_first_name CHECK (LENGTH(TRIM(first_name)) > 0 AND first_name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$'),
ADD CONSTRAINT chk_last_name CHECK (LENGTH(TRIM(last_name)) > 0 AND last_name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$');

ALTER TABLE staff
ADD CONSTRAINT chk_email CHECK (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}$'),
ADD CONSTRAINT chk_first_name CHECK (LENGTH(TRIM(first_name)) > 0 AND first_name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$'),
ADD CONSTRAINT chk_last_name CHECK (LENGTH(TRIM(last_name)) > 0 AND last_name ~* '^[A-Za-zÀ-ÖØ-öø-ÿ\s\-]+$'),
ADD CONSTRAINT chk_username_length CHECK (LENGTH(username) >= 3),
ADD CONSTRAINT chk_username_format CHECK (username ~* '^[A-Za-z0-9._-]+$'),
ADD CONSTRAINT chk_password_length CHECK (LENGTH(password) >= 6),
ADD CONSTRAINT chk_password_format CHECK (password ~* '^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d!@#$%^&*()_+=-]+$');
```

```
ALTER TABLE address
ADD CONSTRAINT chk_address CHECK (address ~ '^[0-9A-Za-zÀ-ÖØ-öø-ÿ\s\-()\.\,\/&]*$'),
ADD CONSTRAINT chk_address2 CHECK (address2 ~ '^[0-9A-Za-zÀ-ÖØ-öø-ÿ\s\-()\.\,\/\&]*$'),
ADD CONSTRAINT chk_district CHECK (district ~ '^[0-9A-Za-zÀ-ÖØ-öø-ÿ\s\-()\.\,\/&]*$'),
ADD CONSTRAINT chk_phone_format CHECK (phone ~ '^[0-9]*$'),
ADD CONSTRAINT chk_postal_code_length CHECK (LENGTH(postal_code) <= 5),
ADD CONSTRAINT chk_postal_code_format CHECK (postal_code ~ '^[0-9]*$');

ALTER TABLE city
ADD CONSTRAINT chk_city CHECK (city ~ '^[0-9A-Za-zÀ-ÖØ-öø-ÿ\s\-()\.\,\/&]+$');

ALTER TABLE country
ADD CONSTRAINT chk_country CHECK (country ~ '^[0-9A-Za-zÀ-ÖØ-öø-ÿ\s\-()\.\,\/&]+$');
```

Imagen que muestra cómo se han añadido las restricciones correctamente:

```
ALTER TABLE film
ADD CONSTRAINT chk_length CHECK (length > 0),
ADD CONSTRAINT chk_rating CHECK (rating IN ('G', 'PG', 'PG-13', 'R', 'NC-17')),
ADD CONSTRAINT chk_release_year CHECK (release_year BETWEEN 1900 AND EXTRACT(YEAR FROM CURRENT_DATE)),
ADD CONSTRAINT chk_rental_duration CHECK (rental_duration > 0),
ADD CONSTRAINT chk_rental_rate CHECK (rental_rate >= 0),
ADD CONSTRAINT chk_replacement_cost CHECK (replacement_cost >= 0),
ADD CONSTRAINT chk_title CHECK (LENGTH(TRIM(title)) > 0 AND title ~* '^[A-Za-z0-9À-ÖØ-öø-ÿ\s!?:&\-]+$'); -- \'

ALTER TABLE rental
ADD CONSTRAINT chk_rental_date CHECK (rental_date <= CURRENT_TIMESTAMP),
ADD CONSTRAINT chk_return_date CHECK (return_date > rental_date);


ALTER TABLE payment
ADD CONSTRAINT chk_payment_date CHECK (payment_date <= CURRENT_TIMESTAMP);

ALTER TABLE customer
ADD CONSTRAINT chk_create_date CHECK (create_date <= CURRENT_TIMESTAMP),
ADD CONSTRAINT chk_country CHECK (country ~ '^[0-9A-Za-zÀ-ÖØ-öø-ÿ\s\-()\.\,\/&]+$');),,!@#$%^&*()_+=-]+$');),'),
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

Resumen de las restricciones:

category y **language**: Ambos requieren que el campo name no esté vacío y que contenga solo letras, espacios y guiones.

actor: Los campos first_name y last_name deben tener contenido y solo permitir letras, espacios y guiones.

film:

- length debe ser mayor a 0.
- rating solo puede contener valores específicos (G, PG, PG-13, R, NC-17).
- release_year debe estar entre 1900 y el año actual.
- rental_duration debe ser mayor a 0.
- rental_rate y replacement_cost deben ser no negativos.
- title no puede estar vacío y debe permitir solo caracteres alfanuméricos, signos de puntuación limitados y espacios.

rental:

- rental_date debe ser anterior o igual a la fecha y hora actual.
- return_date debe ser posterior a rental_date.

payment: La fecha de payment_date no puede estar en el futuro.

customer:

- create_date no puede ser en el futuro.
- email debe estar en un formato válido de correo electrónico.
- first_name y last_name deben estar llenos y solo permitir letras, espacios y guiones.

staff:

- email debe tener formato de correo válido.
- first_name y last_name deben estar llenos y permitir letras, espacios y guiones.
- username debe tener al menos 3 caracteres, permitiendo letras, números, puntos, guiones y guiones bajos.
- password debe tener al menos 6 caracteres y contener al menos una letra y un número.

address:

- address, address2 y district solo permiten letras, números y algunos caracteres especiales.
- phone solo permite números.
- postal_code debe tener un máximo de 5 dígitos y solo contener números.

city y **country**: Permiten letras, números y ciertos caracteres especiales en sus nombres.

Explique la sentencia que aparece en la tabla customer “last_updated BEFORE UPDATE ON customer FOR EACH ROW EXECUTE PROCEDURE last_updated()”. Identifique alguna tabla donde se utilice una solución similar.

La sentencia del trigger en la tabla customer se utiliza para actualizar automáticamente el campo last_update cada vez que se modifica un registro.

```
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                     |           |          |
```

- **Nombre del trigger**: last_updated.
- **Evento**: Se ejecuta antes de cada actualización (BEFORE UPDATE) en la tabla.
- **Tipo**: FOR EACH ROW, se activa para cada fila modificada.
- **Función**: Ejecuta la función last_updated(), que actualiza el campo last_update con la fecha y hora actuales.

La función last_update() es la siguiente (usé el comando **\sf nombre_función**):

```
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 $function$
```

La utilidad del trigger last_updated junto con la función last_updated() es mantener un registro automático de la última modificación de cada fila en la base de datos. Esto es especialmente útil para:

1. **Auditoría y Seguimiento**: Facilita el rastreo de cambios, permitiendo ver cuándo se actualizó un registro por última vez. Esto es valioso en contextos de auditoría o en sistemas donde es importante conocer el historial de modificaciones de datos.
2. **Consistencia y Automatización**: Evita que la aplicación o los usuarios tengan que actualizar manualmente el campo last_update en cada modificación. El trigger asegura que esta actualización sea automática y consistente.
3. **Optimización de Consultas de Actualización**: Facilita la recuperación de registros modificados recientemente, lo que es útil en sistemas que requieren identificar cambios recientes o sincronizar datos.

Tablas como ***film***, ***staff***, ***rental***, ***store***, ***inventory***, ***language***, ***actor***, ***film_actor***, ***category***, ***film_category***, ***address*** y ***city*** también tienen un trigger last_updated para mantener actualizada la marca de tiempo de la última modificación.

```
Triggers:
    film_fulltext_trigger BEFORE INSERT OR UPDATE ON film FOR EACH ROW EXECUTE FUNCTION
    last_updated BEFORE UPDATE ON film FOR EACH ROW EXECUTE FUNCTION last_updated()
```

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

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

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

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

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.

```
CREATE TABLE film_insert_log (
    log_id SERIAL PRIMARY KEY,
    film_id INTEGER NOT NULL,
    insertion_date TIMESTAMP NOT NULL
);

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

CREATE TRIGGER film_insert_log
AFTER INSERT ON film
FOR EACH ROW
EXECUTE FUNCTION log_film_insertion();
```

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

```
CREATE TABLE film_delete_log (
    log_id SERIAL PRIMARY KEY,
    film_id INTEGER NOT NULL,
    deletion_date TIMESTAMP NOT NULL
);

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

CREATE TRIGGER film_delete_log_trigger
BEFORE DELETE ON film
FOR EACH ROW
EXECUTE FUNCTION log_film_deletion();
```

Comente el significado y la relevancia de las secuencias.

Las **secuencias** son objetos en bases de datos relacionales, como PostgreSQL, que generan una serie de números enteros de forma secuencial. Se utilizan principalmente para generar identificadores únicos, como las claves primarias de las tablas.

Significado de las Secuencias

1. **Generación Automática de Números**: Las secuencias permiten la generación automática de números únicos, evitando la necesidad de que los desarrolladores gestionen manualmente los identificadores de cada fila.
2. **Independencia de Tablas**: Las secuencias son independientes de las tablas, lo que significa que pueden ser utilizadas por múltiples tablas o registros sin interferir entre sí.
3. **Configurabilidad**: Las secuencias son altamente configurables, permitiendo a los usuarios definir el valor inicial, el incremento (cuánto se suma en cada llamada a la secuencia), los valores máximos y mínimos, y si deben ser cíclicas o no.

Relevancia de las Secuencias

1. **Integridad de los Datos**: Al proporcionar identificadores únicos, las secuencias ayudan a mantener la integridad de los datos. Esto es especialmente importante en sistemas donde múltiples usuarios o procesos pueden intentar insertar registros simultáneamente.
2. **Optimización de Rendimiento**: Usar secuencias para generar claves primarias puede ser más eficiente que otras técnicas, como utilizar un campo con un valor fijo o calcular identificadores manualmente. Las secuencias permiten a la base de datos manejar la generación de identificadores de manera rápida y eficiente.
3. **Facilitación de Migraciones**: Cuando se requiere migrar datos entre sistemas o bases de datos, las secuencias permiten asegurar que los nuevos registros tengan identificadores que no colisionen con los existentes.
4. **Soporte para Funcionalidades Avanzadas**: Las secuencias pueden ser utilizadas en una variedad de contextos, como en la generación de claves foráneas y en la implementación de funcionalidades avanzadas, como auditorías y trazabilidad de registros.

Usando comando **\ds** para ver las diferentes secuencias:

```
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)
```

Usando el comando **\d nombre_secuencia** para ver una secuencia específica:

```
AlquilerDVD=# \d actor_actor_id_seq
                      Sequence "public.actor_actor_id_seq"
  Type  | Start | Minimum |       Maximum       | Increment | Cycles? | Cache
--------+-------+---------+---------------------+-----------+---------+-------
 bigint |     1 |       1 | 9223372036854775807 |         1 | no      |     1

AlquilerDVD=# \d staff_staff_id_seq
                      Sequence "public.staff_staff_id_seq"
  Type  | Start | Minimum |       Maximum       | Increment | Cycles? | Cache
--------+-------+---------+---------------------+-----------+---------+-------
 bigint |     1 |       1 | 9223372036854775807 |         1 | no      |     1

AlquilerDVD=# \d film_film_id_seq
                       Sequence "public.film_film_id_seq"
  Type  | Start | Minimum |       Maximum       | Increment | Cycles? | Cache
--------+-------+---------+---------------------+-----------+---------+-------
 bigint |     1 |       1 | 9223372036854775807 |         1 | no      |     1

AlquilerDVD=# \d payment_payment_id_seq
                     Sequence "public.payment_payment_id_seq"
  Type  | Start | Minimum |       Maximum       | Increment | Cycles? | Cache
--------+-------+---------+---------------------+-----------+---------+-------
 bigint |     1 |       1 | 9223372036854775807 |         1 | no      |     1
```

Conclusiones

A lo largo de esta práctica, se logró un entendimiento más profundo de las operaciones fundamentales de SQL y PL/pgSQL, especialmente en la gestión de bases de datos relacionales. La creación de vistas y disparadores demostró su utilidad para optimizar consultas y mantener la integridad de los datos. Además, se adquirieron habilidades prácticas en la restauración y manipulación de bases de datos, lo que facilita el análisis y la gestión de información en entornos reales. Esta experiencia fortalece las competencias necesarias para trabajar en proyectos que involucren bases de datos complejas y mejorará la capacidad para desarrollar soluciones eficientes en el futuro.

Referencias y Recursos Utilizados

1. Repositorio de práctica de Alquiler en GitHub - Universidad de La Laguna: https://github.com/ull-cs/adbd/blob/main/relational-models/AlquilerPractica.tar
2. Documentación oficial de PostgreSQL - Guía completa sobre el uso de psql: https://www.postgresql.org/docs/current/app-psql.html
3. Docker - Herramienta de virtualización y contenedores: https://www.docker.com/
4. ChatGPT - Asistente de IA para generación de contenidos: https://chatgpt.com/