

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.

```
[I] ~ X pg_restore -h 10.6.130.74 -p 5432 -U postgres -d postgres -C ./Downloads/AlquilerPractica.tar
Password:
pg_restore: error: could not execute query: ERROR: role "cliente" does not exist
Command was: GRANT ALL ON DATABASE alquilerdvd TO cliente;

pg_restore: warning: errors ignored on restore: 1
```

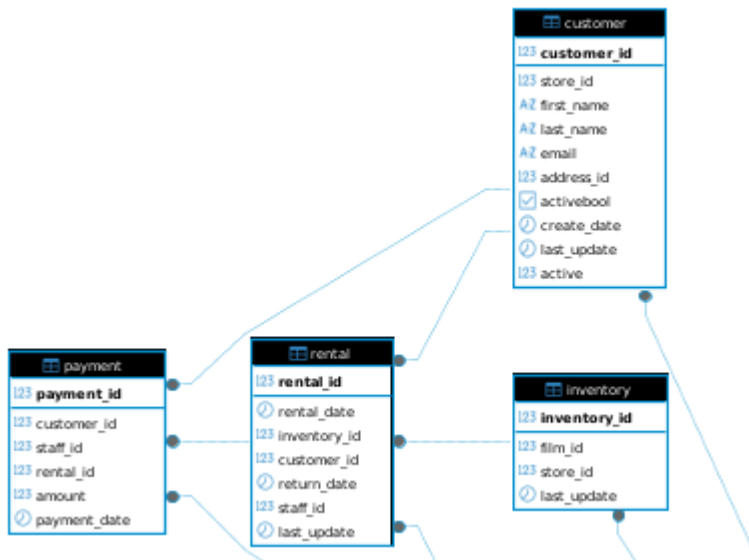
2. Identifique las tablas, vistas y secuencias.

```
postgres=# \d+
List of relations
Schema | Name | Type | Owner | Persistence | Access method | Size | Description
-----+-----+-----+-----+-----+-----+-----+-----
public | actor | table | postgres | permanent | heap | 40 kB | 
public | actor_actor_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | address | table | postgres | permanent | heap | 88 kB | 
public | address_address_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | category | table | postgres | permanent | heap | 8192 bytes | 
public | category_category_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | city | table | postgres | permanent | heap | 64 kB | 
public | city_city_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | country | table | postgres | permanent | heap | 8192 bytes | 
public | country_country_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | customer | table | postgres | permanent | heap | 96 kB | 
public | customer_customer_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | film | table | postgres | permanent | heap | 736 kB | 
public | film_actor | table | postgres | permanent | heap | 272 kB | 
public | film_category | table | postgres | permanent | heap | 72 kB | 
public | film_film_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | inventory | table | postgres | permanent | heap | 232 kB | 
public | inventory_inventory_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | language | table | postgres | permanent | heap | 8192 bytes | 
public | language_language_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | payment | table | postgres | permanent | heap | 896 kB | 
public | payment_payment_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | rental | table | postgres | permanent | heap | 1232 kB | 
public | rental_rental_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | staff | table | postgres | permanent | heap | 16 kB | 
public | staff_staff_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
public | store | table | postgres | permanent | heap | 8192 bytes | 
public | store_store_id_seq | sequence | postgres | permanent |  | 8192 bytes | 
(28 rows)
```

Cabe destacar que no hay vistas dado que no persisten en el tiempo y todavía no se han creado



3. Identifique las tablas principales y sus principales elementos.



Estos son los elementos que mas alto están en la jerarquía

4. Realice las siguientes consultas.

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

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



A-Z categoria ▼	123 ventas totales ▼
Sports	4,892.19
Sci-Fi	4,336.01
Animation	4,245.31
Drama	4,118.46
Comedy	4,002.48
New	3,966.38
Action	3,951.84
Foreign	3,934.47
Games	3,922.18
Family	3,830.15
Documentary	3,749.65
Horror	3,401.27
Classics	3,353.38

- b. 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

```
SELECT
    st.store_id AS numero_tienda,
    CONCAT(c.city, ', ', co.country) AS ciudad_pais,
    s.first_name || ' ' || s.last_name AS encargado,
    SUM(p.amount) AS ventas_totales
FROM payment p
INNER JOIN staff s ON p.staff_id = s.staff_id
INNER JOIN store st ON s.staff_id = st.manager_staff_id
INNER JOIN address a ON st.address_id = a.address_id
INNER JOIN city c ON a.city_id = c.city_id
INNER JOIN country co ON c.country_id = co.country_id
GROUP BY st.store_id, c.city, co.country, s.first_name, s.last_name
ORDER BY ventas_totales DESC;
```

123 numero tienda ▼	A-Z ciudad pais ▼	A-Z encargado ▼	123 ventas totales ▼
2	Woodridge, Australia	Jon Stephens	31,059.92
1	Lethbridge, Canada	Mike Hillyer	30,252.12



- c. 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 ":"

```
select
  a.first_name || ' ' || a.last_name as Nombre_y_Apellido,
  concat (f.title , ' : ', c."name")
from actor a
inner join film_actor fa on a.actor_id = fa.actor_id
inner join film f on fa.film_id = f.film_id
inner join film_category fc on f.film_id = fc.film_id
inner join category c on fc.category_id = c.category_id
group by a.first_name, a.last_name, f.title, c."name"
order by nombre_y_apellido desc;
```

Results 1 x

Enter a SQL expression to filter results (use Ctrl+Space)

AZ nombre y apellido	AZ concat
Zero Cage	Horn Working : Animation
Zero Cage	Story Side : Action
Zero Cage	Jersey Sassy : Children
Zero Cage	Dances None : Action
Zero Cage	Thin Sagebrush : Documentary
Zero Cage	Uptown Young : Children
Zero Cage	Ending Crowds : New
Zero Cage	Oleander Clue : Music
Zero Cage	Canyon Stock : Animation

refresh Save Cancel ... Export data

200 row(s) fetched - 0.1s, on 2025-10-23 at 10:38:40

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



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

stics 1 X

Value

Rows 0

time 0.0s

.me Thu Oct 23 10:20:19 WEST 2025

:ime Thu Oct 23 10:20:19 WEST 2025

create view view_ventas_por_cat as SELECT

c.name AS categoria,

SUM(p.amount) AS ventas_totales

FROM payment p

INNER JOIN rental r ON p.rental_id = r.rental_id



```
st.store_id AS número_tienda,  
CONCAT(c.city, ', ', co.country) AS ciudad_pais,  
s.first_name || ' ' || s.last_name AS encargado,  
SUM(p.amount) AS ventas_totales  
FROM payment p  
INNER JOIN staff s ON p.staff_id = s.staff_id  
INNER JOIN store st ON s.staff_id = st.manager_staff_id  
INNER JOIN address a ON st.address_id = a.address_id  
INNER JOIN city c ON a.city_id = c.city_id  
INNER JOIN country co ON c.country_id = co.country_id  
GROUP BY st.store_id, c.city, co.country, s.first_name, s.last_name  
ORDER BY ventas_totales DESC;
```

Statistics 1 X

Value

Rows 0

Time 0.0s

Time Thu Oct 23 10:20:19 WEST 2025

Time Thu Oct 23 10:20:19 WEST 2025

create view view_ventas_por_cat as SELECT

c.name AS categoria,

SUM(p.amount) AS ventas_totales

FROM payment p

INNER JOIN rental r ON p.rental_id = r.rental_id

INNER JOIN inventory i ON r.inventory_id = i.inventory_id



```
create view view_actor_film as select
    a.first_name || ' ' || a.last_name as Nombre_y_Apellido,
    concat (f.title , ' : ', c."name")
from actor a
inner join film_actor fa on a.actor_id = fa.actor_id
inner join film f on fa.film_id = f.film_id
inner join film_category fc on f.film_id = fc.film_id
inner join category c on fc.category_id = c.category_id
group by a.first_name, a.last_name , f.title, c."name"
order by nombre_y_apellido desc;
```

Statistics 1 X	Value
Rows	0
Elapsed time	0.0s
Start time	Thu Oct 23 10:40:14 WEST 2025
End time	Thu Oct 23 10:40:14 WEST 2025
SQL	<pre>create view view_actor_film as select a.first_name ' ' a.last_name as Nombre_y_Apellido, concat (f.title , ' : ', c."name") from actor a inner join film_actor fa on a.actor_id = fa.actor_id inner join film f on fa.film_id = f.film_id inner join film_category fc on f.film_id = fc.film_id</pre>

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

El modelo entidad-relación es extenso y cuenta con multitud de entidades, algunas con múltiples atributos que deben controlarse para que no haya funcionamientos inesperados, fuera de eso es una base de datos completa donde se han tenido en cuenta la mayoría, si no todos, los escenarios.

Las restricciones check añadidas son las siguientes:



▶

+

AI

....


⊖ ALTER TABLE film

ADD CONSTRAINT chk_publi_valida CHECK (release_year >= 1900);

Statistics 1 X

Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Thu Oct 23 11:17:03 WEST 2025
Finish time	Thu Oct 23 11:17:03 WEST 2025
Query	ALTER TABLE film ADD CONSTRAINT chk_publi_valida CHECK (release_year ≥ 1900)





```
ALTER TABLE film
ADD CONSTRAINT chk_duration_valida CHECK (rental_duration > 0);
```

Query (Ctrl+Enter)

Statistics 1 X

Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Thu Oct 23 11:18:11 WEST 2025
Finish time	Thu Oct 23 11:18:11 WEST 2025
Query	ALTER TABLE film ADD CONSTRAINT chk_duration_valida CHECK (rental_duration > 0)



<pre>ALTER TABLE payment ADD CONSTRAINT chk_amount_valida CHECK (amount >= 0);</pre>	
Statistics 1 X	
Name	Value
Updated Rows	0
Execute time	0.0s
Start time	Thu Oct 23 11:21:39 WEST 2025
Finish time	Thu Oct 23 11:21:39 WEST 2025
Query	ALTER TABLE payment ADD CONSTRAINT chk_amount_valida CHECK (amount ≥ 0)

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.

Este es un trigger que se ejecuta antes de actualizar una fila en la tabla `customer`, el cual, para cada fila ejecuta la funcion `last_updated()`.

Podemos encontrar un trigger similar en la tabla `actor`:



Properties			
Name:	last_updated	Column(s):	
Description:		Enabled:	0
Timing:	BEFORE	Object ID:	16956
Manipulation:	[UPDATE]	When Expression:	
Type:	ROW	Function:	public.last_updated

Source
<pre>create trigger last_updated before update on public.actor for each row execute function last_updated()</pre>

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`.

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



```
CREATE OR REPLACE FUNCTION registrar_insercion_film()  
RETURNS TRIGGER AS $film_trigger$  
BEGIN  
    INSERT INTO film_insert_log (film_id, fecha_insercion)  
    VALUES (NEW.film_id, NOW());  
    RETURN NEW; -- Permite que la inserción original en film continúe  
END;  
$film_trigger$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_registrar_insercion_film AFTER INSERT ON film  
FOR EACH row EXECUTE PROCEDURE registrar_insercion_film();
```

```
select * from film_insert_log;
```

insert_log 1 x		
from film_insert_log Enter a SQL expression to filter results (use Ctrl+Space)		
123 log id	123 film id	fecha insercion
1	1,004	2025-10-23 11:38:50.626



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`.

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

```
➤ create or replace function registrar_delete_film()  
returns trigger as $delete_trigger$  
begin  
    insert into film_delete_log(film_id, fecha_borrado)  
    values (new.film_id, now());  
    return new;  
end;  
$delete_trigger$ language plpgsql;
```

```
➤ CREATE TRIGGER trg_registrar_borrado_film  
AFTER delete ON film  
FOR EACH ROW  
EXECUTE PROCEDURE registrar_insercion_film();
```

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

En PostgreSQL, una secuencia es un objeto especial de la base de datos que genera valores enteros únicos y consecutivos de manera automática. Se usan principalmente para generar valores únicos para claves primarias (`PRIMARY KEY`) o cualquier columna que requiera un identificador incremental. Cada llamada a la secuencia devuelve el siguiente valor disponible. Estas permiten la generación de claves automáticas evitando posibles conflictos con las ya existentes.

