

## Práctica 04. Disparadores y Vistas en SQL

Administracion y Diseño de Bases de Datos

Cheuk Kelly Ng Pante (alu0101364544@ull.edu.es)

12 de noviembre de 2023

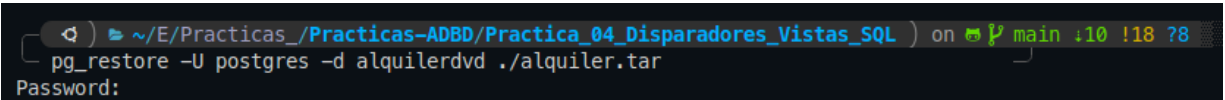
# Índice general

1. Realizar la restauración de la base de datos alquiler.tar	2
2. Identificación de las tablas, vistas y secuencias	2
3. Identifique las tablas principales y sus principales elementos	3
4. Realización de consultas	5
4.1. Ventas totales por categoria . . . . .	5
4.2. Ventas totales por tienda . . . . .	6
4.3. Lista de películas . . . . .	7
4.4. Información de los actores . . . . .	8
5. Vistas de las consultas realizadas	9
5.1. Ventas totales por categoria . . . . .	9
5.2. Ventas totales por tienda . . . . .	9
5.3. Lista de películas . . . . .	10
5.4. Información de los actores . . . . .	10
6. Análisis del modelo y restricciones <i>CHECK</i>	11
7. Funcionamiento del trigger	11
8. Construcción de disparadores	12
8.1. Disparador de inserción . . . . .	12
8.2. Disparador de eliminación . . . . .	13
9. Significado y relevancia de las <i>sequence</i>	14
10. Exportación de la base de datos en formato <i>sql</i>	14

## 1. Realizar la restauración de la base de datos alquiler.tar

Para realizar la restauración de la base de datos *alquiler.tar*, primero debemos crear la base de datos *ALQUILERDVD* y luego restaurar la base de datos con el comando *pg\_restore*, como se muestra a continuación:

```
usuario@ubuntu# pg_restore -U postgres -d alquilerdvd ./alquiler.tar
```



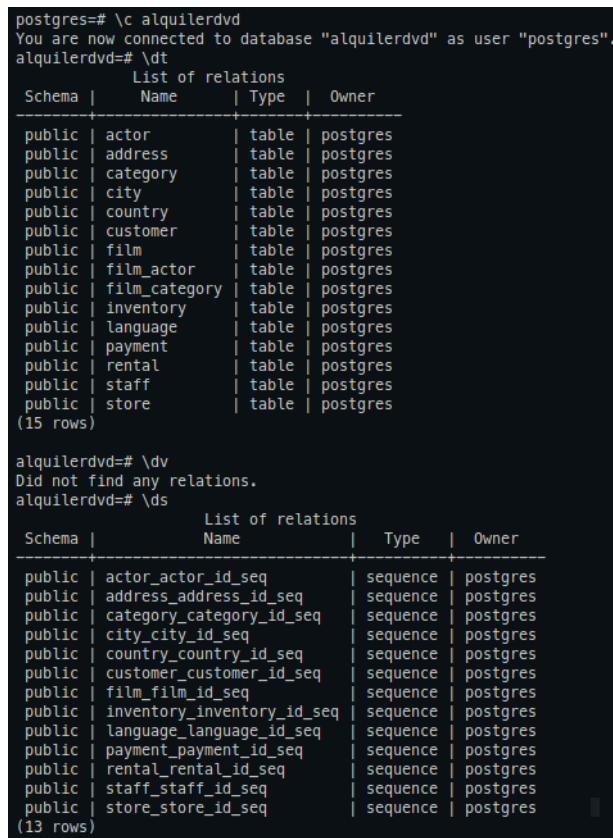
```
~ /E/Practicas_/Practicas-ADBD/Practica_04_Disparadores_Vistas_SQL on main .10 !18 ?8
pg_restore -U postgres -d alquilerdvd ./alquiler.tar
Password:
```

Figura 1.1: Restauración de la base de datos

## 2. Identificación de las tablas, vistas y secuencias

Para identificar las tablas, vistas y secuencias de la base de datos *ALQUILERDVD*, hay que usar la terminal interactiva de *PostgreSQL* y ejecutar los siguiente comandos:

```
usuario@ubuntu# sudo -u postgres psql
postgres=# \c alquilerdvd
alquilerdvd=# \dt
alquilerdvd=# \dv
alquilerdvd=# \ds
```



```
postgres=# \c alquilerdvd
You are now connected to database "alquilerdvd" as user "postgres".
alquilerdvd=# \dt
      List of relations
Schema | Name          | Type  | Owner
-----|-----|-----|-----
public | actor          | table | postgres
public | address         | table | postgres
public | category        | table | postgres
public | city            | table | postgres
public | country         | table | postgres
public | customer        | table | postgres
public | film            | table | postgres
public | film_actor      | table | postgres
public | film_category   | table | postgres
public | inventory       | table | postgres
public | language        | table | postgres
public | payment         | table | postgres
public | rental          | table | postgres
public | staff           | table | postgres
public | store           | table | postgres
(15 rows)

alquilerdvd=# \dv
Did not find any relations.
alquilerdvd=# \ds
      List of relations
Schema | Name                                | Type  | Owner
-----|-----|-----|-----
public | actor_actor_id_seq                 | sequence | postgres
public | address_address_id_seq             | sequence | postgres
public | category_category_id_seq           | sequence | postgres
public | city_city_id_seq                   | sequence | postgres
public | country_country_id_seq             | sequence | postgres
public | customer_customer_id_seq           | sequence | postgres
public | film_film_id_seq                   | sequence | postgres
public | inventory_inventory_id_seq          | sequence | postgres
public | language_language_id_seq           | sequence | postgres
public | payment_payment_id_seq             | sequence | postgres
public | rental_rental_id_seq               | sequence | postgres
public | staff_staff_id_seq                 | sequence | postgres
public | store_store_id_seq                 | sequence | postgres
(13 rows)
```

Figura 2.1: Identificación de las tablas, vistas y secuencias

### 3. Identifique las tablas principales y sus principales elementos

- **Tabla:** *actor*
  - **Descripción:** Contiene la información de los actores.
  - **Elementos:** *actor\_id, first\_name, last\_name, last\_update*
- **Tabla:** *address*
  - **Descripción:** Contiene la información de las direcciones.
  - **Elementos:** *address\_id, address, address2, district, city\_id, postal\_code, phone, last\_update*
- **Tabla:** *category*
  - **Descripción:** Contiene la información de las categorías.
  - **Elementos:** *category\_id, name, last\_update*
- **Tabla:** *city*
  - **Descripción:** Contiene la información de las ciudades.
  - **Elementos:** *city\_id, city, country\_id, last\_update*
- **Tabla:** *country*
  - **Descripción:** Contiene la información de los países.
  - **Elementos:** *country\_id, country, last\_update*
- **Tabla:** *customer*
  - **Descripción:** Contiene la información de los clientes.
  - **Elementos:** *customer\_id, store\_id, first\_name, last\_name, email, address\_id, activebool, create\_date, last\_update, active*
- **Tabla:** *film*
  - **Descripción:** Contiene la información de las películas.
  - **Elementos:** *film\_id, title, description, release\_year, language\_id, rental\_duration, rental\_rate, length, replacement\_cost, rating, last\_update, special\_features, fulltext*
- **Tabla:** *film\_actor*
  - **Descripción:** Contiene la información de los actores de las películas.
  - **Elementos:** *actor\_id, film\_id, last\_update*
- **Tabla:** *film\_category*
  - **Descripción:** Contiene la información de las categorías de las películas.
  - **Elementos:** *film\_id, category\_id, last\_update*

- **Tabla:** *inventory*
  - **Descripción:** Contiene la información de los inventarios.
  - **Elementos:** *inventory\_id, film\_id, store\_id, last\_update*
- **Tabla:** *language*
  - **Descripción:** Contiene la información de los lenguajes.
  - **Elementos:** *language\_id, name, last\_update*
- **Tabla:** *payment*
  - **Descripción:** Contiene la información de los pagos.
  - **Elementos:** *payment\_id, customer\_id, staff\_id, rental\_id, amount, payment\_date*
- **Tabla:** *rental*
  - **Descripción:** Contiene la información de los alquileres.
  - **Elementos:** *rental\_id, rental\_date, inventory\_id, customer\_id, return\_date, staff\_id, last\_update*
- **Tabla:** *staff*
  - **Descripción:** Contiene la información de los empleados.
  - **Elementos:** *staff\_id, first\_name, last\_name, address\_id, email, store\_id, active, username, password, last\_update, picture*
- **Tabla:** *store*
  - **Descripción:** Contiene la información de las tiendas.
  - **Elementos:** *store\_id, manager\_staff\_id, address\_id, last\_update*

## 4. Realizacion de consultas

### 4.1. Ventas totales por categoria

Para obtener las ventas totales por categoría de películas ordenadas descendientemente, se debe ejecutar la siguiente consulta:

```
alquilerdvd=# SELECT category.name, SUM(payment.amount) AS total_sales
FROM category
INNER JOIN film_category ON category.category_id = film_category.category_id
INNER JOIN film ON film_category.film_id = film.film_id
INNER JOIN inventory ON film.film_id = inventory.film_id
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
INNER JOIN payment ON rental.rental_id = payment.rental_id
GROUP BY category.name
ORDER BY total_sales DESC;
```

```
alquilerdvd=# SELECT category.name, SUM(payment.amount) AS total_sales
FROM category
INNER JOIN film_category ON category.category_id = film_category.category_id
INNER JOIN film ON film_category.film_id = film.film_id
INNER JOIN inventory ON film.film_id = inventory.film_id
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
INNER JOIN payment ON rental.rental_id = payment.rental_id
GROUP BY category.name
ORDER BY total_sales DESC;
```

name	total_sales
Sports	4892.19
Sci-Fi	4336.01
Animation	4245.31
Drama	4118.46
Comedy	4002.48
New	3966.38
Action	3951.84
Foreign	3934.47
Games	3922.18
Family	3830.15
Documentary	3749.65
Horror	3401.27
Classics	3353.38
Children	3309.39
Travel	3227.36
Music	3071.52

(16 rows)

Figura 4.1: Ventas totales por categoria

## 4.2. Ventas totales por tienda

Para obtener 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. Pusiera emplear GROUP BY, ORDER BY

```
alquilerdvd=# SELECT CONCAT(city.city, ', ', country.country) AS city_country,  
    CONCAT(staff.first_name, ' ', staff.last_name) AS manager,  
    SUM(payment.amount) AS total_sales  
FROM store  
INNER JOIN staff ON store.manager_staff_id = staff.staff_id  
INNER JOIN address ON store.address_id = address.address_id  
INNER JOIN city ON address.city_id = city.city_id  
INNER JOIN country ON city.country_id = country.country_id  
INNER JOIN inventory ON store.store_id = inventory.store_id  
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id  
INNER JOIN payment ON rental.rental_id = payment.rental_id  
GROUP BY city_country, manager  
ORDER BY total_sales DESC;
```

```
alquilerdvd=# SELECT CONCAT(city.city, ', ', country.country) AS city_country,  
    CONCAT(staff.first_name, ' ', staff.last_name) AS manager,  
    SUM(payment.amount) AS total_sales  
FROM store  
INNER JOIN staff ON store.manager_staff_id = staff.staff_id  
INNER JOIN address ON store.address_id = address.address_id  
INNER JOIN city ON address.city_id = city.city_id  
INNER JOIN country ON city.country_id = country.country_id  
INNER JOIN inventory ON store.store_id = inventory.store_id  
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id  
INNER JOIN payment ON rental.rental_id = payment.rental_id  
GROUP BY city_country, manager  
ORDER BY total_sales DESC;  
  
    city_country    |    manager    | total_sales  
-----+-----+-----  
Woodridge, Australia | Jon Stephens |    30683.13  
Lethbridge, Canada  | Mike Hillyer |    30628.91  
(2 rows)
```

Figura 4.2: Ventas totales por tienda

### 4.3. Lista de películas

Para obtener 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). Pusiera emplear GROUP BY

```
alquilerdvd=# SELECT film.film_id, title, description, category.name AS category_name,
rental_rate, length, rating,
actor.first_name || ' ' || actor.last_name AS actor_name
FROM film
INNER JOIN film_actor ON film.film_id = film_actor.film_id
INNER JOIN actor ON film_actor.actor_id = actor.actor_id
INNER JOIN film_category ON film.film_id = film_category.film_id
INNER JOIN category ON film_category.category_id = category.category_id
ORDER BY film_id;
```

film_id	title	description	category_name	rental_rate	length	rating	actor_name
1	Academy Dinosaur	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher In The Canadian Rockies	Documentary	0.99	86	PG	Rock Dukakis
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	Mary Keitel
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	Johnny Cage
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 Guinness
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	Sandra Peck
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	Christian Gable
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	Oprah Kliner
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	Warren Nolte
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	Lucille Tracy
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	Mena Temple
2	Ace Goldfinger	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China	Horror	4.99	48	G	Minnie Zellweger
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	Chris Depp
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	Bob Faucett
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	Sean Guinness
3	Adaptation Holes	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory	Documentary	2.99	50	NC-17	Cameron Streep
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	Bob Faucett
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
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	Ray Johansson
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	Julianne Dench
4	Affair Prejudice	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank	Horror	2.99	117	G	Jodie Degeneras
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	Kenneth Pesci
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	Fay Winslet
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	Oprah Kliner

Figura 4.3: Lista de películas



## 4.4. Información de los actores

Para obtener la información de los actores, donde se incluya sus nombres y apellidos, las categorías y sus películas. Los actores deben de estar agrupados y, las categorías y las películas deben estar concatenados por “.”

```
alquilerdvd=# SELECT actor.first_name || ' ' || actor.last_name AS actor_name,
string_agg(DISTINCT category.name, ':') AS categories,
string_agg(DISTINCT film.title, ':') AS films
FROM actor
INNER JOIN film_actor ON actor.actor_id = film_actor.actor_id
INNER JOIN film_category ON film_actor.film_id = film_category.film_id
INNER JOIN category ON film_category.category_id = category.category_id
INNER JOIN film ON film_category.film_id = film.film_id
GROUP BY actor_name;
```

actor_id	first_name	last_name	films_name
1	Penelope	Guinness	Academy Dinosaur : A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies ; Documentary
1	Penelope	Guinness	Anaconda Confessions : A Lacklustre Display of a Dentist And a Dentist who must Fight a Girl in Australia ; Animation
1	Penelope	Guinness	Angels Life : A Thoughtful Display of a Woman And a Astronaut who must Battle a Robot in Berlin ; New
1	Penelope	Guinness	Bulworth Commandments : A Amazing Display of a Mad Cow And a Pioneer who must Redeem a Suro Wrestler in The Outback ; Games
1	Penelope	Guinness	Cheaper Clyde : A Emotional Character Study of a Pioneer And a Girl who must Discover a Dog in Ancient Japan ; Sci-Fi
1	Penelope	Guinness	Color Philadelphia : A Thoughtful Panorama of a Car And a Crocodile who must Sink a Monkey in The Sahara Desert ; Classics
1	Penelope	Guinness	Elephant Trojan : A Beautiful Panorama of a Lumberjack And a Forensic Psychologist who must Overcome a Frisbee in A Baloon ; Horror
1	Penelope	Guinness	Gleaning Jawbreaker : A Amazing Display of a Composer And a Forensic Psychologist who must Discover a Car in The Canadian Rockies ; Sports
1	Penelope	Guinness	Human Graffiti : A Beautiful Reflection of a Womanizer And a Suro Wrestler who must Chase a Database Administrator in The Gulf of Mexico ; Games
1	Penelope	Guinness	King Evolution : A Action-Packed Tale of a Boy And a Lumberjack who must Chase a Madman in A Baloon ; Family
1	Penelope	Guinness	Lady Stage : A Beautiful Character Study of a Woman And a Man who must Pursue a Explorer in A U-Boat ; Horror
1	Penelope	Guinness	Language Cowboy : A Epic Yarn of a Cat And a Madman who must Vanquish a Dentist in An Abandoned Amusement Park ; Children
1	Penelope	Guinness	Mulholland Beast : A Awe-Inspiring Display of a Husband And a Squirrel who must Battle a Suro Wrestler in A Jet Boat ; Foreign
1	Penelope	Guinness	Oklahoma Juranji : A Thoughtful Drama of a Dentist And a Womanizer who must Meet a Husband in The Sahara Desert ; New
1	Penelope	Guinness	Rules Huran : A Beautiful Epistle of a Astronaut And a Student who must Confront a Monkey in An Abandoned Fun House ; Horror
1	Penelope	Guinness	Splash Gulp : A Taut Saga of a Crocodile And a Boat who must Conquer a Hunter in A Shark Tank ; Family
1	Penelope	Guinness	Vertigo Northwest : A Unbelievable Display of a Mad Scientist And a Mad Scientist who must Outgun a Mad Cow in Ancient Japan ; Comedy
1	Penelope	Guinness	Westward Seabiscuit : A Lacklustre Tale of a Butler And a Husband who must Face a Boy in Ancient China ; Classics
1	Penelope	Guinness	Wizard Coldblooded : A Lacklustre Display of a Robot And a Girl who must Defeat a Suro Wrestler in A MySQL Convention ; Music
2	Nick	Wahlberg	Adaptation Holes : A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory ; Documentary
2	Nick	Wahlberg	Apache Divine : A Awe-Inspiring Reflection of a Pastry Chef And a Teacher who must Overcome a Suro Wrestler in A U-Boat ; Family
2	Nick	Wahlberg	Baby Hall : A Boring Character Study of a A Shark And a Girl who must Outrace a Feminist in An Abandoned Mine Shaft ; Foreign
2	Nick	Wahlberg	Bull Shawshank : A Fanciful Drama of a Moose And a Squirrel who must Conquer a Pioneer in The Canadian Rockies ; Action
2	Nick	Wahlberg	Chainsaw Uptown : A Beautiful Documentary of a Boy And a Robot who must Discover a Squirrel in Australia ; Sci-Fi
2	Nick	Wahlberg	Chisum Behavior : A Epic Documentary of a Suro Wrestler And a Butler who must Kill a Car in Ancient India ; Family
2	Nick	Wahlberg	Destiny Saturday : A Touching Drama of a Crocodile And a Crocodile who must Conquer a Explorer in Soviet Georgia ; New
2	Nick	Wahlberg	Dracula Crystal : A Thrilling Reflection of a Feminist And a Cat who must Find a Frisbee in An Abandoned Fun House ; Classics

Figura 4.4: Información de los actores

## 5. Vistas de las consultas realizadas

Para crear las vistas de las consultas realizadas se usará el prefijo *VIEW* para identificarlas. A continuación se muestra el código de cada vista:

### 5.1. Ventas totales por categoría

Para crear la vista de las ventas totales por categoría, se debe ejecutar la siguiente consulta:

```
alquilerdvd=# CREATE VIEW total_rent_per_category AS
SELECT category.name, SUM(payment.amount) AS total_sales
FROM category
INNER JOIN film_category ON category.category_id = film_category.category_id
INNER JOIN film ON film_category.film_id = film.film_id
INNER JOIN inventory ON film.film_id = inventory.film_id
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
INNER JOIN payment ON rental.rental_id = payment.rental_id
GROUP BY category.name
ORDER BY total_sales DESC;
```

### 5.2. Ventas totales por tienda

Para crear la vista de las ventas totales por tienda, se debe ejecutar la siguiente consulta:

```
alquilerdvd=# CREATE VIEW total_rent_per_store AS
SELECT CONCAT(city.city, ', ', country.country) AS city_country,
CONCAT(staff.first_name, ' ', staff.last_name) AS manager,
SUM(payment.amount) AS total_sales
FROM store
INNER JOIN staff ON store.manager_staff_id = staff.staff_id
INNER JOIN address ON store.address_id = address.address_id
INNER JOIN city ON address.city_id = city.city_id
INNER JOIN country ON city.country_id = country.country_id
INNER JOIN inventory ON store.store_id = inventory.store_id
INNER JOIN rental ON inventory.inventory_id = rental.inventory_id
INNER JOIN payment ON rental.rental_id = payment.rental_id
GROUP BY city_country, manager
ORDER BY total_sales DESC;
```

### 5.3. Lista de películas

Para crear la vista de la lista de películas, se debe ejecutar la siguiente consulta:

```
alquilerdvd=# CREATE VIEW films_list AS
SELECT film.film_id, title, description, category.name AS category_name,
rental_rate, length, rating,
actor.first_name || ' ' || actor.last_name AS actor_name
FROM film
INNER JOIN film_actor ON film.film_id = film_actor.film_id
INNER JOIN actor ON film_actor.actor_id = actor.actor_id
INNER JOIN film_category ON film.film_id = film_category.film_id
INNER JOIN category ON film_category.category_id = category.category_id
ORDER BY film_id;
```

### 5.4. Información de los actores

Para crear la vista de la información de los actores, se debe ejecutar la siguiente consulta:

```
alquilerdvd=# CREATE VIEW actor_list AS
SELECT actor.first_name || ' ' || actor.last_name AS actor_name,
string_agg(DISTINCT category.name, ':') AS categories,
string_agg(DISTINCT film.title, ':') AS films
FROM actor
INNER JOIN film_actor ON actor.actor_id = film_actor.actor_id
INNER JOIN film_category ON film_actor.film_id = film_category.film_id
INNER JOIN category ON film_category.category_id = category.category_id
INNER JOIN film ON film_category.film_id = film.film_id
GROUP BY actor_name;
```

## 6. Análisis del modelo y restricciones *CHECK*

Dentro del modelo de la base de datos *ALQUILERDVD* se puede observar que existen tablas que emplean atributos que hacen uso de identificadores de tuplas que se encuentran en otras tablas, por lo que se puede decir que existen relaciones entre las tablas.

Se podrían implementar restricciones *CHECK* que se encarguen de comprobar que dichos números de identificación existan en las tablas relacionadas, de esta manera se asegura que no se inserten datos que no existan en la base de datos. Por ejemplo, en la tabla *film\_actor* se puede implementar una restricción *CHECK* que compruebe que el identificador de la película exista en la tabla *film* y que el identificador del actor exista en la tabla *actor*:

```
ALTER TABLE film_actor
ADD CONSTRAINT CHECK (film_id IN (SELECT film_id FROM film) AND
                      actor_id IN (SELECT actor_id FROM actor));
```

Otra restricción que se podría implementar es que el identificador de la película en la tabla *film\_category* exista en la tabla *film*:

```
ALTER TABLE film_category
ADD CONSTRAINT CHECK (film_id IN (SELECT film_id FROM film));
```

## 7. Funcionamiento del trigger

El siguiente trigger:

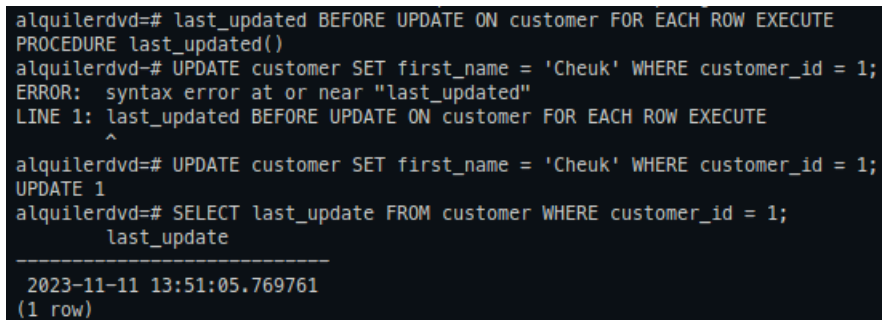
```
last_updated BEFORE UPDATE ON customer FOR EACH ROW EXECUTE
PROCEDURE last_updated()
```

Se encarga de actualizar el atributo *last\_update* de la tabla *customer* cada vez que se actualice una tupla de la tabla *customer*. Para probar el funcionamiento del trigger, se ejecutará la siguiente consulta:

```
alquilerdvd=# UPDATE customer SET first_name = 'Cheuk' WHERE customer_id = 1;
```

Para comprobar que se ha actualizado el atributo *last\_update* de la tupla con *customer\_id* igual a 1, se ejecutará la siguiente consulta:

```
alquilerdvd=# SELECT last_update FROM customer WHERE customer_id = 1;
```



```
alquilerdvd=# last_updated BEFORE UPDATE ON customer FOR EACH ROW EXECUTE
PROCEDURE last_updated()
alquilerdvd=# UPDATE customer SET first_name = 'Cheuk' WHERE customer_id = 1;
ERROR:  syntax error at or near "last_updated"
LINE 1: last_updated BEFORE UPDATE ON customer FOR EACH ROW EXECUTE
        ^
alquilerdvd=# UPDATE customer SET first_name = 'Cheuk' WHERE customer_id = 1;
UPDATE 1
alquilerdvd=# SELECT last_update FROM customer WHERE customer_id = 1;
 last_update
-----
2023-11-11 13:51:05.769761
(1 row)
```

Figura 7.1: Funcionamiento del trigger

## 8. Construcción de disparadores

### 8.1. Disparador de inserción

Para construir un disparador que se encargue de guardar en una nueva tabla creada la fecha de cuando se insertó un nuevo registro en la tabla *film*, se debe ejecutar la siguiente consulta:

```
CREATE TABLE updated_table_film (  
  id_updated_table_film SERIAL PRIMARY KEY,  
  last_update TIMESTAMP NOT NULL  
);
```

Una vez creada la tabla *updated\_table\_film*, se procederá a crear el disparador; primero se creará la función que se encargará de insertar la fecha en la tabla *updated\_table\_film*;

```
CREATE OR REPLACE FUNCTION insert_date_updated_table_film()  
RETURNS TRIGGER AS $$  
  BEGIN  
    INSERT INTO updated_table_film (last_update) VALUES (NOW());  
    RETURN NEW;  
  END;  
$$ LANGUAGE plpgsql;
```

y luego se creará el disparador que se encargará de ejecutar la función:

```
CREATE TRIGGER insert_date_updated_table_film  
AFTER INSERT ON film  
FOR EACH ROW  
EXECUTE PROCEDURE insert_date_updated_table_film();
```

Aquí se puede observar el funcionamiento del disparador:

```
alquilerdvd=# CREATE TABLE updated_table_film (  
  id_updated_table_film SERIAL PRIMARY KEY,  
  last_update TIMESTAMP NOT NULL  
);  
CREATE TABLE  
alquilerdvd=# CREATE OR REPLACE FUNCTION insert_date_updated_table_film()  
RETURNS TRIGGER AS $$  
  BEGIN  
    INSERT INTO updated_table_film (last_update) VALUES (NOW());  
    RETURN NEW;  
  END;  
$$ LANGUAGE plpgsql;  
CREATE FUNCTION  
alquilerdvd=# CREATE TRIGGER insert_date_updated_table_film  
AFTER INSERT ON film  
FOR EACH ROW  
EXECUTE PROCEDURE insert_date_updated_table_film();  
CREATE TRIGGER  
alquilerdvd=# INSERT INTO film (title, description, release_year, language_id, rental_duration,  
  rental_rate, length, replacement_cost, rating, special_features, fulltext)  
VALUES ('Independence Day',  
  'Epic film about an alien invasion of Earth',  
  1996, 1, 3, 4.99, 145, 19.99, 'PG-13',  
  '{Behind the Scenes, Deleted Scenes}',  
  'Aliens invade Earth and destroy everything');  
INSERT 0 1  
alquilerdvd=# ^[[200~SELECT * FROM updated_table_film;~  
ERROR:  syntax error at or near "  
LINE 1: SELECT * FROM updated_table_film;  
      ^  
alquilerdvd=# SELECT * FROM updated_table_film;  
ERROR:  syntax error at or near "~"  
LINE 1: ~  
      ^  
alquilerdvd=# SELECT * FROM updated_table_film;  
 id_updated_table_film | last_update  
-----  
1 | 2023-11-12 10:42:48.205047  
(1 row)
```

Figura 8.1: Funcionamiento del disparador de inserción

## 8.2. Disparador de eliminación

Para construir un disparador que se encargue de guardar en una nueva tabla creada la fecha de cuando se eliminó un nuevo registro en la tabla *film* y el identificador del film. Para esto, se debe ejecutar la siguiente consulta:

```
CREATE TABLE deleted_film_rows (  
  id_deleted_film_rows SERIAL PRIMARY KEY,  
  film_id INTEGER NOT NULL,  
  last_update TIMESTAMP NOT NULL  
);
```

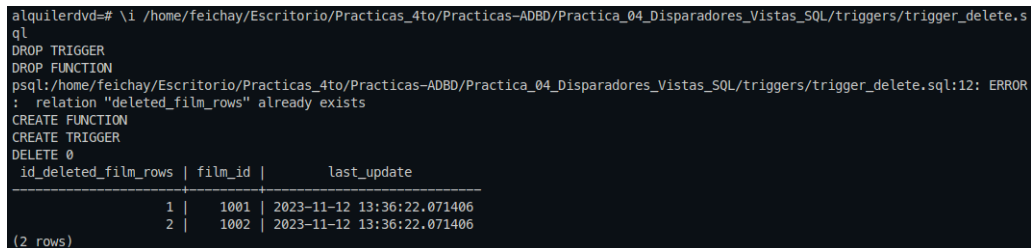
Una vez creada la tabla *deleted\_film\_rows*, se procederá a crear el disparador; primero se creará la función que se encargará de insertar el identificador del film y la fecha en la tabla *deleted\_film\_rows*:

```
CREATE OR REPLACE FUNCTION insert_deleted_film_rows()  
RETURNS TRIGGER AS $$  
  BEGIN  
    INSERT INTO deleted_film_rows (film_id, last_update) VALUES (OLD.film_id, NOW());  
    RETURN OLD;  
  END;  
$$ LANGUAGE plpgsql;
```

y luego se creará el disparador que se encargará de ejecutar la función:

```
CREATE TRIGGER insert_deleted_film_rows  
AFTER DELETE ON film  
FOR EACH ROW  
EXECUTE PROCEDURE insert_deleted_film_rows();
```

Aquí se puede observar el funcionamiento del disparador:



```
alquilerdvd=# \i /home/feichay/Escritorio/Practicas_4to/Practicas-ADBD/Practica_04_Disparadores_Vistas_SQL/triggers/trigger_delete.s  
ql  
DROP TRIGGER  
DROP FUNCTION  
psql:/home/feichay/Escritorio/Practicas_4to/Practicas-ADBD/Practica_04_Disparadores_Vistas_SQL/triggers/trigger_delete.sql:12: ERROR  
: relation "deleted_film_rows" already exists  
CREATE FUNCTION  
CREATE TRIGGER  
DELETE 0  
id_deleted_film_rows | film_id |      last_update  
-----  
1 | 1001 | 2023-11-12 13:36:22.071406  
2 | 1002 | 2023-11-12 13:36:22.071406  
(2 rows)
```

Figura 8.2: Funcionamiento del disparador de eliminacion

## 9. Significado y relevancia de las *sequence*

Las *sequence* son objetos que se encargan de generar números de identificación de tuplas de manera automática. Estos números de identificación son únicos y no se repiten. Las *sequence* son muy útiles cuando se quiere insertar una nueva tupla en una tabla y no se conoce el número de identificación que se le asignará a la tupla, ya que la *sequence* se encargará de generar un número de identificación único para la tupla.

Un ejemplo de creación de una *sequence* en PostgreSQL es el siguiente:

```
CREATE SEQUENCE sequence_name START 1 INCREMENT 1;
```

## 10. Exportación de la base de datos en formato *sql*

Para exportar la base de datos *ALQUILERDVD* en formato *sql*, se debe ejecutar el siguiente comando:

```
usuario@ubuntu# pg_dump -U postgres -d alquilerdvd > alquilerdvd.sql
```

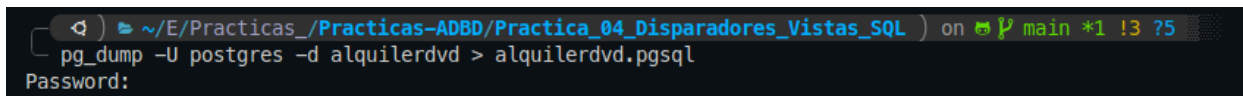


Figura 10.1: Exportación de la base de datos en formato *sql*