Universidad de La Laguna

AD BBDD.

Práctica 5 *INFORME*

Modelo Relacional. Vistas y disparadores

Integrantes:

VÍCTOR RODRÍGUEZ DORTA
alu0101540153

MARIO GUERRA PÉREZ
alu0101395036

4. Consultas.

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

```
select category.name as categoria,
sum(payment.amount) as ventas_totales
from payment
inner join rental on rental.rental_id = payment.rental_id
inner join inventory on inventory.inventory_id = rental.inventory_id
inner join film_category on film_category.film_id = inventory.film_id
inner join category on category.category_id = film_category.category_id
group by categoria
order by ventas_totales desc;
```

	A-Z categoria	123 ventas_totales
1	Sports	4.892,19
2	Sci-Fi	4.336,01
3	Animation	4.245,31
4	Drama	4.118,46
5	Comedy	4.002,48
6	New	3.966,38
7	Action	3.951,84
8	Foreign	3.934,47
9	Games	3.922,18
10	Family	3.830,15
11	Documentary	3.749,65
12	Horror	3.401,27
13	Classics	3.353,38
14	Children	3.309,39
15	Travel	3.227,36
16	Music	3.071,52

Y para crear la vista :

```
create view view_ventas_totales as
select category.name as categoria,
sum(payment.amount) as ventas_totales
from payment
inner join rental on rental.rental_id = payment.rental_id
inner join inventory on inventory.inventory_id = rental.inventory_id
inner join film_category on film_category.film_id = inventory.film_id
inner join category on category.category_id = film_category.category_id
group by categoria
order by ventas_totales 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

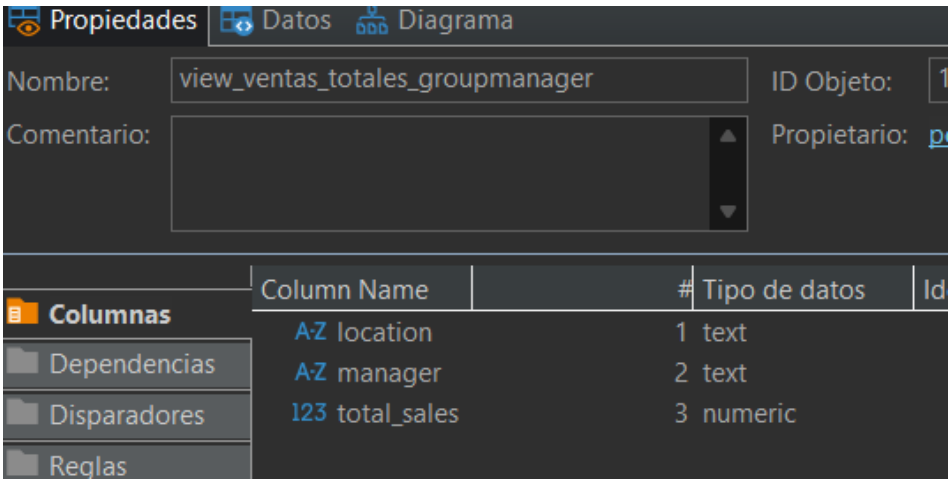
```
SELECT
 CONCAT(city.city, ', ', country.country) AS location,
 CONCAT(staff.first_name, ' ', staff.last_name) AS manager,
 SUM(payment.amount) AS total_sales
FROM payment
INNER JOIN rental ON rental.rental_id = payment.rental_id
INNER JOIN inventory ON inventory.inventory_id = rental.inventory_id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

	location	manager	total_sales
1	Woodridge, Australia	Jon Stephens	30.683,13
2	Lethbridge, Canada	Mike Hillyer	30.628,91

Y para crear la vista, se hace como antes :

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

Y vemos cómo se está creando la vista.



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

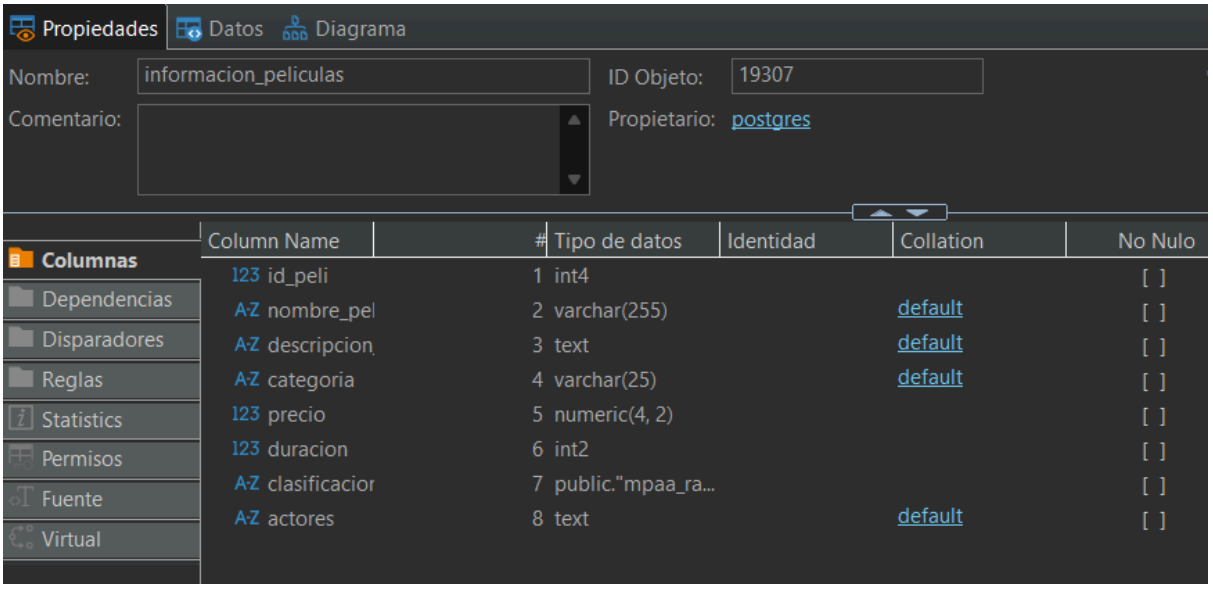
```
select f.film_id as id_peli,
        f.title as nombre_peli,
        f.description as descripcion_peli,
        c.name as categoria,
        f.rental_rate as precio,
        f.length as duracion,
        f.rating as clasificacion,
  STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ') AS actores
from film f
join film_category fc on f.film_id = fc.film_id
join category c on fc.category_id = c.category_id
join film_actor fa on f.film_id = fa.film_id
join actor a on fa.actor_id = a.actor_id
group by f.film_id , f.title , f.description , c.name, f.rental_rate, f.length , f.rating
```

	id_peli	nombre_peli	descripcion_peli	categoria	precio	duracion	clasificacion	actores
1	1	Academy Dinosaur	A Epic Drama of a Feminist And a Ma	Documentary	0,99	86	PG	Rock Dukakis, Mary Keitel, Johnny Ca
2	2	Ace Goldfinger	A Astounding Epistle of a Database A	Horror	4,99	48	G	Minnie Zellweger, Chris Depp, Bob Fa
3	3	Adaptation Holes	A Astounding Reflection of a Lumber	Documentary	2,99	50	NC-17	Cameron Streep, Bob Fawcett, Nick W
4	4	Affair Prejudice	A Fanciful Documentary of a Frisbee	Horror	2,99	117	G	Jodie Degeneres, Kenneth Pesci, Fay
5	5	African Egg	A Fast-Paced Documentary of a Pastr	Family	2,99	130	G	Dustin Tautou, Matthew Leigh, Gary P

Creamos la vista con :

```
create view view_Informacion_peliculas as
select f.film_id as id_peli,
        f.title as nombre_peli,
        f.description as descripcion_peli,
        c.name as categoria,
        f.rental_rate as precio,
        f.length as duracion,
        f.rating as clasificacion,
  STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ') AS actores
```

```
from film f
join film_category fc on f.film_id = fc.film_id
join category c on fc.category_id = c.category_id
join film_actor fa on f.film_id = fa.film_id
join actor a on fa.actor_id = a.actor_id
group by f.film_id , f.title , f.description , c.name, f.rental_rate, f.length , f.rating
```



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 ":"

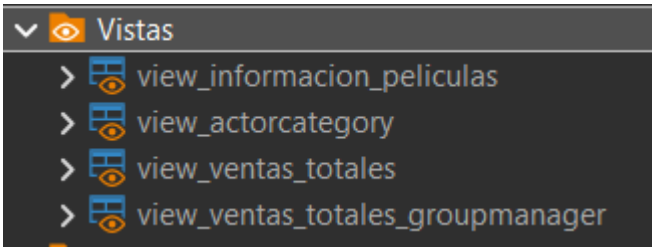
```
SELECT
 CONCAT(actor.first_name, ' ', actor.last_name) AS actor_name,
 STRING_AGG(category.name || ':' || film.title, ', ') AS categorias_peliculas
FROM actor
INNER JOIN film_actor ON film_actor.actor_id = actor.actor_id
INNER JOIN film ON film.film_id = film_actor.film_id
INNER JOIN film_category ON film_category.film_id = film.film_id
INNER JOIN category ON category.category_id = film_category.category_id
GROUP BY actor_name
ORDER BY actor_name;
```

	actor_name	categorias_peliculas
1	Adam Grant	Sci-Fi:Annie Identity, Foreign:Ballroom Mockingbird, Travel:Disciple Moth
2	Adam Hopper	Sci-Fi:Blindness Gun, Family:Blood Argonauts, Music:Chamber Italian, Doc
3	Al Garland	Documentary:Bill Others, New:Breakfast Goldfinger, Drama:Chitty Lock, D
4	Alan Dreyfuss	Sci-Fi:Badman Dawn, Sci-Fi:Barbarella Streetcar, Music:Birch Antitrust, Far
5	Albert Johansson	Music:Alaska Phantom, Foreign:Alley Evolution, Drama:Apollo Teen, Gam
6	Albert Nolte	Documentary:Bed Highball, Drama:Bright Encounters, Foreign:Brooklyn D

Crearemos también la vista:

```
create view view_ActorCategory as
SELECT
 CONCAT(actor.first_name, ' ', actor.last_name) AS actor_name,
 STRING_AGG(category.name || ':' || film.title, ', ') AS categorias_peliculas
FROM actor
INNER JOIN film_actor ON film_actor.actor_id = actor.actor_id
INNER JOIN film ON film.film_id = film_actor.film_id
INNER JOIN film_category ON film_category.film_id = film.film_id
INNER JOIN category ON category.category_id = film_category.category_id
GROUP BY actor_name
ORDER BY actor_name;
```

Después de haber realizado todas las vistas quedarían de esta forma :



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

- **Restricciones en la Entidad film**

Restricción CHECK en rental_duration:

Asegurar que la duración del alquiler sea estrictamente positiva (rental_duration > 0).

Evitar valores de cero o negativos. (la duración debe ser positiva):

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

Restricción CHECK en rental_rate:

Garantizar que la tarifa de alquiler sea mayor o igual a cero (rental_rate >= 0).

Permitir valores cero solo en casos especiales (por ejemplo, promociones).

```
ALTER TABLE film
ADD CONSTRAINT check_rental_rate_no_negativo
CHECK (rental_rate >= 0.00);
```

Restricción CHECK en replacement_cost:

Asegurar que el costo de reemplazo sea mayor o igual a cero (replacement_cost >= 0).

Impedir valores negativos.

```
ALTER TABLE film
ADD CONSTRAINT check_replacement_cost_positivo
CHECK (replacement_cost >= 0.00);
```

Restricción CHECK en rating:

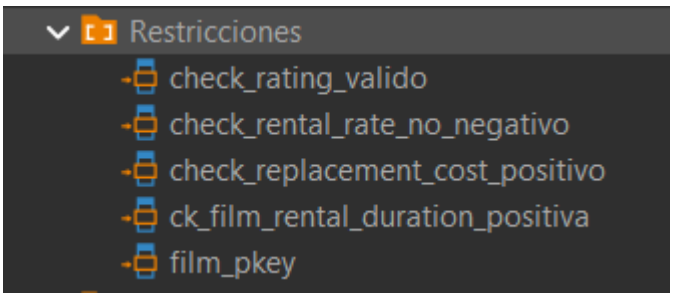
Limitar los valores del campo a las clasificaciones estándar:

'G', 'PG', 'PG-13', 'R', 'NC-17'.

Evitar la introducción de clasificaciones inválidas.

```
ALTER TABLE film
ADD CONSTRAINT check_rating_valido
```

```
CHECK (rating IN ('G', 'PG', 'PG-13', 'R', 'NC-17'));
```



- **Restricciones en las Entidades de Transacción**

Tabla rental

Restricción CHECK sobre las fechas:

Si return_date no es nula, debe ser mayor o igual que rental_date (return_date >= rental_date).

Evitar incoherencias temporales (no se puede devolver una película antes de alquilarla).

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

Restricción CHECK sobre rental_date:

Impedir que la fecha de alquiler sea posterior a la fecha actual (rental_date <= CURRENT_DATE).

Garantizar coherencia con el tiempo real.

```
ALTER TABLE rental
ADD CONSTRAINT check_fecha_alquiler_no_futura
CHECK (rental_date <= CURRENT_TIMESTAMP);
```

Restricciones en las Entidades de Gestión de Personal y Clientes

Tablas customer y staff

Restricción CHECK en active:

Limitar los valores posibles a 0 o 1 (active IN (0, 1)).

Garantizar que el estado del usuario o empleado esté claramente definido como activo (1) o inactivo (0).

```
ALTER TABLE customer
ADD CONSTRAINT check_customer_activo_valido
CHECK (active IN (0, 1));
```

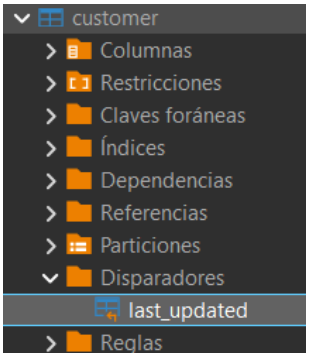
Porque el valor de esa tabla está declarado como numero

```
ALTER TABLE staff
ADD CONSTRAINT check_staff_activo_valido
CHECK (active IN (TRUE, FALSE));
```

Y en esta tabla está declarado como booleano.

7. Explicación de la sentencia de Customer.

En la tabla customer, podemos observar que posee un disparador:



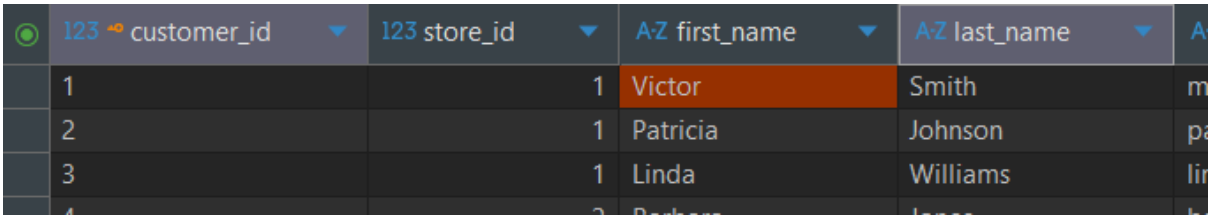

```
create trigger last_updated before
update
    on
    public.customer for each row execute function last_updated()
```

Este trigger tiene la función de detectar cuando se ha modificado un campo de la tabla customer, para así actualizar la fecha de modificación de dicha fila.
Para ello hace uso de la función last_updated():

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

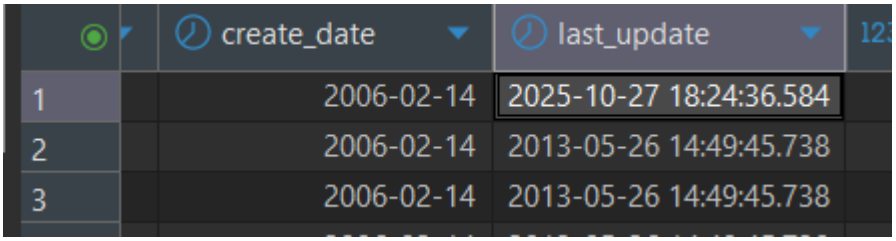
Lo que hace es modificar el valor de la columna con la fecha actual.

Por ejemplo si probamos a modificar una fila de la tabla customer:



customer_id	store_id	first_name	last_name
1	1	Victor	Smith
2	1	Patricia	Johnson
3	1	Linda	Williams

Por ejemplo, hacer que el id 1 sea de nombre Víctor, al guardarlo, veremos como la fecha de modificación cambia gracias a este trigger.



	create_date	last_update
1	2006-02-14	2025-10-27 18:24:36.584
2	2006-02-14	2013-05-26 14:49:45.738
3	2006-02-14	2013-05-26 14:49:45.738

Otra solución similar que podemos ver, es en la tabla Payment, donde, en vez de usar la función con un trigger, directamente cuando creas una fila de la tabla, activa el timestamp, para guardar ese valor.

```
CREATE TABLE public.payment (
	payment_id serial4 NOT NULL,
	customer_id int2 NOT NULL,
	staff_id int2 NOT NULL,
	rental_id int4 NOT NULL,
	amount numeric(5, 2) NOT NULL,
	payment_date timestamp NOT NULL,
	CONSTRAINT payment_pkey PRIMARY KEY (payment_id),
	CONSTRAINT payment_customer_id_fkey FOREIGN KEY (customer_id)
REFERENCES public.customer(customer_id) ON DELETE RESTRICT ON UPDATE
CASCADE,
	CONSTRAINT payment_rental_id_fkey FOREIGN KEY (rental_id) REFERENCES
public.rental(rental_id) ON DELETE SET NULL ON UPDATE CASCADE,
	CONSTRAINT payment_staff_id_fkey FOREIGN KEY (staff_id) REFERENCES
public.staff(staff_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
CREATE INDEX idx_fk_customer_id ON public.payment USING btree (customer_id);
CREATE INDEX idx_fk_rental_id ON public.payment USING btree (rental_id);
CREATE INDEX idx_fk_staff_id ON public.payment USING btree (staff_id);
```

Como podemos ver subrayado. El problema de hacerlo de esta forma es que no estaríamos registrando fechas ante posibles modificaciones.

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 if not exists film_insert_log (
    log_id serial primary key,
    film_id integer not null,
    insert_date timestamp not null default CURRENT_TIMESTAMP,
    -- no se si current_timestamp o NOW()
    constraint fk_film foreign key (film_id) references film(film_id) on delete cascade

);
```

En este apartado lo primero que se debe de hacer es crear la tabla donde se va a guardar la información cuando se active el trigger. Para ello, se crea la tabla con un atributo log_id, el cual se encarga de llevar las transacciones, film_id que guarda el id de la película con la que se va a hacer el registro, la fecha de inserción de la película con la variable por defecto CURRENT_TIMESTAMP, que coge la fecha del sistema por defecto; y por último, se añade una regla de integridad foránea para que si una peli se elimina, se elimina de esta tabla también.

```
create or replace function log_film_insert()
returns trigger as $$
-- solo las funciones que devuelven trigger se pueden
-- usar en triggers
begin
    insert into film_insert_log (film_id, insert_date)
    values (new.film_id, now());
    return new;
-- new es var auto creada por postgre que existe según la
-- operacion. New existe en update y insert
end;
$$ language plpgsql;
-- con $$ se delimita el cuerpo de la funcion
```

Lo siguiente es la función que se va a llamar en el trigger. Ésta lo que realiza es que se inserta en la tabla los valores de la nueva película con la hora del sistema (otra manera de llamar a la hora del sistema. Cabe destacar que new es una variable creada por Postgre que se crea cuando se inserta un nuevo valor en una tabla y permite operar con este en los trigger.

```
drop trigger if exists trigger_log_film_insert on film;
create trigger trigger_log_film_insert
    after insert on film
    for each row
    execute function log_film_insert();
```

Por último se crea el propio trigger el cual se adjudica a la tabla “film” y se activa una vez se inserte una nueva fila en dicha tabla, y posteriormente se llama a la función previamente declarada.

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 if not exists film_delete_log (
    log_id serial primary key,
    film_id integer not null,
    delete_date timestamp not null default current_timestamp
);

create or replace function log_film_delete()
returns trigger as $$
begin
    insert into film_delete_log (film_id, delete_date)
    values (old.film_id, now());
    return old;
end;
$$ language plpgsql;

drop trigger if exists trigger_log_film_delete on film;
create trigger trigger_log_film_delete
    before delete on film
    for each row
    execute function log_film_delete();
```

Para este trigger se realiza lo mismo que en el anterior. Las diferencias son que:

- No se hace regla de integridad ya que se va a borrar un dato directamente.
- Se usa la variable “Old” en lugar de “New”, ya que es un dato que ya existía.
- En el trigger se realiza la acción antes de que se realice el borrado (si no no se podrían guardar los datos.