Universidad de La Laguna

Grado en Ingeniería Informática
Administración y diseño de bases de datos

Modelo relacional. Vistas y disparadores

Objetivos

Los principales objetivos de esta práctica son los siguientes:

- Continuar desarrollando habilidades en las operaciones básicas con el SQL.
- Desarrollar actividades básicas con vistas y disparadores.

Descripción

Blockbuster LLC, conocida como Blockbuster Video, fue una franquicia estadounidense de videoclubes, especializada en alquiler de películas y videojuegos a través de tiendas físicas, servicios por correo y vídeo bajo demanda. Fue una de las precursoras de plataformas como la actual Netflix. Su modelo de negocios se basaba en el alquiler de DVD de juegos y películas.

Sobre el escenario descrito anteriormente, en esta práctica trabajaremos como una base de datos que típicamente se podría encontrar en un establecimiento en el que los clientes están registrados y pueden alquilar películas con un tiempo máximo de alquiler y su histórico de alquileres. Además, se incluyen datos sobre las películas disponibles, los empleados y las tiendas.

Partiendo de la base de datos alquilerdvd.tar que se encuentra disponible en GitHub. Realice las siguientes actividades con la ayuda de psql y el PL/pgSQL.

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.

   pg_restore -h ip -U postgres -d postgres -C AlquilerPractica.tar

2. Identifique las tablas, vistas y secuencias.

Tablas: \dt

```
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)
```

Vistas: \dv (Ahora mismo no hay ninguna creada)

```
alquilerdvd=# \dv
Did not find any relations.
```

Secuencias: \ds

```
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)
```

3. Identifique las tablas principales y sus principales elementos.

Tablas principales:

film (almacena la información de las películas, se relaciona con actores, idiomas y categorías y el inventario)

rental (almacena información sobre los alquileres, se relaciona con los trabajadores, el inventario, los clientes y los pagos)

payment (almacena información sobre los pagos, se relaciona con los trabajadores, los clientes y los alquileres)

customer (almacena información sobre los clientes, se relaciona con alquiler, pagos, direcciones y tiendas)

staff (almacena información sobre los trabajadores, se relaciona con las tiendas, los alquileres, los pagos y las direcciones)

store (almacena la información sobre las tiendas, se relaciona con los trabajadores, las direcciones, el inventario y los clientes)

4. Realice las siguientes consultas.
5.
    a. Obtenga las ventas totales por categoría de películas ordenadas descendentemente.

        select c.name as categoria, sum(p.amount) as venta_total from payment p inner join rental r on r.rental_id = p.rental_id inner join inventory i on i.inventory_id = r.inventory_id inner join film_category fc on fc.film_id = i.film_id inner join category c on c.category_id = fc.category_id group by categoria order by venta_total desc;

```
alquilerdvd=# select c.name as categoria, sum(p.amount) as venta_total
i on i.inventory_id = r.inventory_id inner join film_category fc on fc
categoria order by venta_total desc;
  categoria  | venta_total
-------------+-------------
 Sports      |     4892.19
 Sci-Fi      |     4336.01
 Animation   |     4245.31
 Drama       |     4118.46
 Comedy      |     4002.48
 New         |     3966.38
 Action      |     3951.84
 Foreign     |     3934.47
 Games       |     3922.18
 Family      |     3830.15
 Documentary |     3749.65
 Horror      |     3401.27
 Classics    |     3353.38
 Children    |     3309.39
 Travel      |     3227.36
 Music       |     3071.52
(16 rows)
```

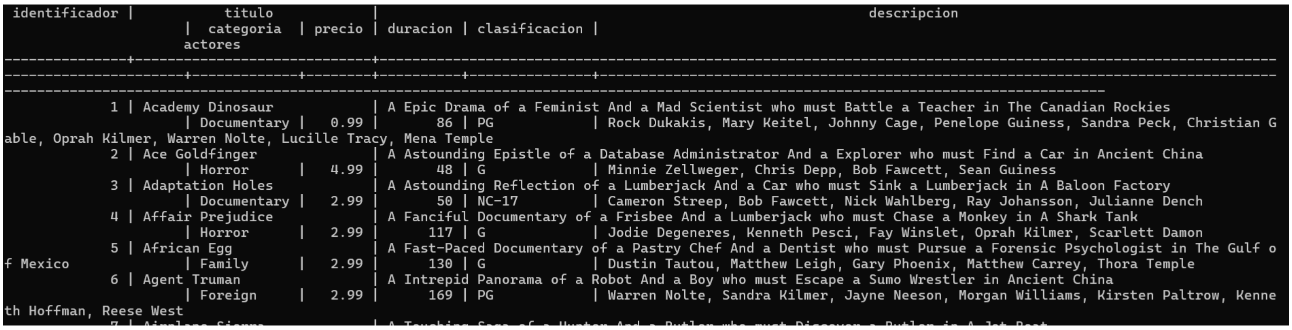
b. Obtenga las ventas totales por tienda, donde se refleje la ciudad, el país (concatenar la ciudad y el país empleando como separador la ","), y el encargado. Pudiera emplear `GROUP BY, ORDER BY`

select sum(p.amount) as ventas_por_tienda, concat(city, ',', country) as Ciudad_Pais, concat(st.first_name, ' ', st.last_name) as nombre_encargado from store s inner join address a on s.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 inner join staff st on s.manager_staff_id = st.staff_id inner join customer cu on s.store_id = cu.store_id inner join payment p on cu.customer_id = p.customer_id group by c.city_id, co.country_id, st.staff_id;

```
alquilerdvd=# select sum(p.amount) as ventas_por_tienda, concat(
gado from store s inner join address a on s.address_id = a.addre
country_id inner join staff st on s.manager_staff_id = st.staff_
 = p.customer_id group by c.city_id, co.country_id, st.staff_id;
 ventas_por_tienda |     ciudad_pais     | nombre_encargado
-------------------+---------------------+-----------------
          33621.42 | Lethbridge,Canada   | Mike Hillyer
          27690.62 | Woodridge,Australia | Jon Stephens
(2 rows)
```

c. Obtenga una lista de películas, donde se reflejen el identificador, el título, descripción, categoría, el precio, la duración de la película, clasificación, nombre y apellidos de los actores (puede realizar una concatenación de ambos). Pudiera emplear `GROUP BY`

select f.film_id as identificador, title as titulo, description as descripcion, c.name as categoria, rental_rate as precio, length as duracion, rating as clasificacion, string_agg(concat(first_name, ' ', last_name), ', ') as actores from film f inner join film_category fc on f.film_id = fc.film_id inner join category c on fc.category_id = c.category_id inner join film_actor fa on f.film_id = fa.film_id inner join actor a on a.actor_id = fa.actor_id group by f.film_id, c.name;



```
 identificador |          titulo             |                                        descripcion
                      |  categoria  | precio | duracion | clasificacion |
                      actores
---------------+-----------------------------+------------------------------------------------------------------------------
---------------------+------------+--------+----------+---------------+------------------------------------------------------
------------------------------------------------------------------------------------------------------------------
             1 | Academy Dinosaur            | A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies
                      | Documentary |   0.99 |       86 | PG            | Rock Dukakis, Mary Keitel, Johnny Cage, Penelope Guiness, Sandra Peck, Christian G
able, Oprah Kilmer, Warren Nolte, Lucille Tracy, 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, Chris Depp, Bob Fawcett, Sean 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         | Cameron Streep, Bob Fawcett, Nick Wahlberg, Ray Johansson, 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 Degeneres, Kenneth Pesci, Fay Winslet, Oprah Kilmer, Scarlett Damon
             5 | African Egg                 | A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf o
f Mexico              | Family      |   2.99 |      130 | G             | Dustin Tautou, Matthew Leigh, Gary Phoenix, Matthew Carrey, Thora Temple
             6 | Agent Truman                | A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China
                      | Foreign     |   2.99 |      169 | PG            | Warren Nolte, Sandra Kilmer, Jayne Neeson, Morgan Williams, Kirsten Paltrow, Kenne
th Hoffman, Reese West
```

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 a.first_name as nombre, a.last_name as appellido, concat(c.name, ':', f.title) as categoria_pelicula 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 fa.film_id = fc.film_id inner join category c on fc.category_id = c.category_id;

nombre	appellido	categoria_pelicula
Penelope	Guiness	Documentary:Academy Dinosaur
Penelope	Guiness	Animation:Anaconda Confessions
Penelope	Guiness	New:Angels Life
Penelope	Guiness	Games:Bulworth Commandments
Penelope	Guiness	Sci-Fi:Cheaper Clyde
Penelope	Guiness	Classics:Color Philadelphia
Penelope	Guiness	Horror:Elephant Trojan
Penelope	Guiness	Sports:Gleaming Jawbreaker
Penelope	Guiness	Games:Human Graffiti
Penelope	Guiness	Family:King Evolution
Penelope	Guiness	Horror:Lady Stage
Penelope	Guiness	Children:Language Cowboy
Penelope	Guiness	Foreign:Mulholland Beast
Penelope	Guiness	New:Oklahoma Jumanji
Penelope	Guiness	Horror:Rules Human
Penelope	Guiness	Family:Splash Gump
Penelope	Guiness	Comedy:Vertigo Northwest
Penelope	Guiness	Classics:Westward Seabiscuit
Penelope	Guiness	Music:Wizard Coldblooded
Nick	Wahlberg	Documentary:Adaptation Holes
Nick	Wahlberg	Family:Apache Divine
Nick	Wahlberg	Foreign:Baby Hall
Nick	Wahlberg	Action:Bull Shawshank

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

create view view_venta_por_categoria as select c.name as categoria, sum(p.amount) as venta_total from payment p inner join rental r on r.rental_id = p.rental_id inner join inventory i on i.inventory_id = r.inventory_id inner join film_category fc on fc.film_id = i.film_id inner join category c on c.category_id = fc.category_id group by categoria order by venta_total desc;

```
alquilerdvd=# create view view_venta_por_categoria as select c.name as categori
l_id = p.rental_id inner join inventory i on i.inventory_id = r.inventory_id in
c.category_id = fc.category_id group by categoria order by venta_total desc;
CREATE VIEW
```

create view view_venta_por_tienda as select sum(p.amount) as ventas_por_tienda, concat(city, ',', country) as Ciudad_Pais, concat(st.first_name, ' ', st.last_name) as nombre_encargado from store s inner join address a on s.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 inner join staff st on s.manager_staff_id = st.staff_id inner join customer cu on s.store_id = cu.store_id inner join payment p on cu.customer_id = p.customer_id group by c.city_id, co.country_id, st.staff_id;

```
alquilerdvd=# create view view_venta_por_tienda as select sum(p.amount) as ventas_por_tienda, concat(
e, ' ', st.last_name) as nombre_encargado from store s inner join address a on s.address_id = a.addre
join country co on c.country_id = co.country_id inner join staff st on s.manager_staff_id = st.staff_
nner join payment p on cu.customer_id = p.customer_id group by c.city_id, co.country_id, st.staff_id;
CREATE VIEW
```

create view view_actores_peliculas as select f.film_id as identificador, title as titulo, description as descripcion, c.name as categoria, rental_rate as precio, length as duracion, rating as clasificacion, string_agg(concat(first_name, ' ', last_name), ', ') as actores from film f inner join film_category fc on f.film_id = fc.film_id inner join category c on fc.category_id = c.category_id inner join film_actor fa on f.film_id = fa.film_id inner join actor a on a.actor_id = fa.actor_id group by f.film_id, c.name;

```
alquilerdvd=# create view view_actores_peliculas as select f.fil
tal_rate as precio, length as duracion, rating as clasificacion,
_category fc on f.film_id = fc.film_id inner join category c on
 actor a on a.actor_id = fa.actor_id group by f.film_id, c.name;
CREATE VIEW
```

create view view_informacion_actores as select a.first_name as nombre, a.last_name as appellido, concat(c.name, ':', f.title) as categoria_pelicula 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 fa.film_id = fc.film_id inner join category c on fc.category_id = c.category_id;

```
alquilerdvd=# create view view_informacion_actores as select a.first_nam
icula from actor a inner join film_actor fa on a.actor_id = fa.actor_id
d = fc.film_id inner join category c on fc.category_id = c.category_id;
CREATE VIEW
```

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

CHECKs para film:

check (rental_rate > 0) (Por si se intenta añadir un valor negativo)
check (length > 0) (Por si se intenta añadir un valor negativo)

```
alquilerdvd=# alter table film add constraint alquiler_positivo check (rental_rate > 0);
ALTER TABLE
alquilerdvd=# alter table film add constraint duracion_positiva check (length > 0);
ALTER TABLE
```

CHECKs para rental

check (return_date is null or return_date >= rental_date) (Para que se devuelvan después de ser prestadas)
check (rental_date <= current_timestamp) (No se puede alquilar una película en el futuro)

```
alquilerdvd=# alter table rental add constraint duracion_retorno check (return_date is null or return_date >= rental_date);
ALTER TABLE
alquilerdvd=# alter table rental add constraint alquiler_pasado check (rental_date <= current_timestamp);
ALTER TABLE
```

CHECK para payment

check (payment_date <= current_timestamp) (No se puede realizar un pago en el futuro)

```
alquilerdvd=# alter table payment add constraint pago_pasado check (payment_date <= current_timestamp);
ALTER TABLE
```

8. Explique la sentencia que aparece en la tabla `customer` (\d+ customer)

   Triggers:

   ```
   last_updated BEFORE UPDATE ON customer

   FOR EACH ROW EXECUTE PROCEDURE last_updated()
   ```

La sentencia hace que cada vez que se actualice un cliente se actualice también la fecha de última modificación (un campo en la tabla customer)

Identifique alguna tabla donde se utilice una solución similar.

Todas las tablas menos la tabla payment tienen la misma solución

9. 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 or replace function fecha_insercion()
returns trigger as $$
begin
   insert into inserciones(fecha_insercion) values (now());
   return new;
end;
$$ language plpgsql;
```

```
create trigger fecha_insercion
after insert on film
for each row
execute procedure fecha_insercion();
```

10. 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 or replace function fecha_eliminacion()
returns trigger as $$
```

```
begin
   insert into eliminaciones(fecha_eliminacion) values (now());
   return old;
end;
$$ language plpgsql;
```

```
create trigger fecha_eliminacion
after delete on film
for each row
execute procedure fecha_eliminacion();
```

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

Las secuencias son números que van incrementando según se van dando ciertos requisitos (la más común es la inserción en tablas). Son relevantes a la hora de crear identificadores para que se vayan creando de forma automática identificadores que no se repitan y que sean consecutivos

Entrega

Enlace a repositorio de GitHub donde se incluya al menos lo siguiente:

- Base de datos en un fichero comprimido en formato SQL.
- Breve informe en PDF donde incluya las interrogantes indicadas en la sección descripción. También puede escribir la información requerida usando el `Readme.md` del repositorio. Debe incluir las imágenes y fragmentos de código que sean necesarias para ilustrar claramente qué pasos ha realizado, los motivos de éstos y los resultados obtenidos.

Observaciones

- La práctica puede ser realizada por parejas. Si éste es su caso, incluya el nombre de los integrantes de la pareja en la entrega y realicen el envío ambos a través de la actividad habilitada en el campus virtual de la asignatura.
- Si el repositorio de GitHub es privado envite a cexposit@ull.edu.es para poder consultarlo.

Anexo. Esquema de la base de datos