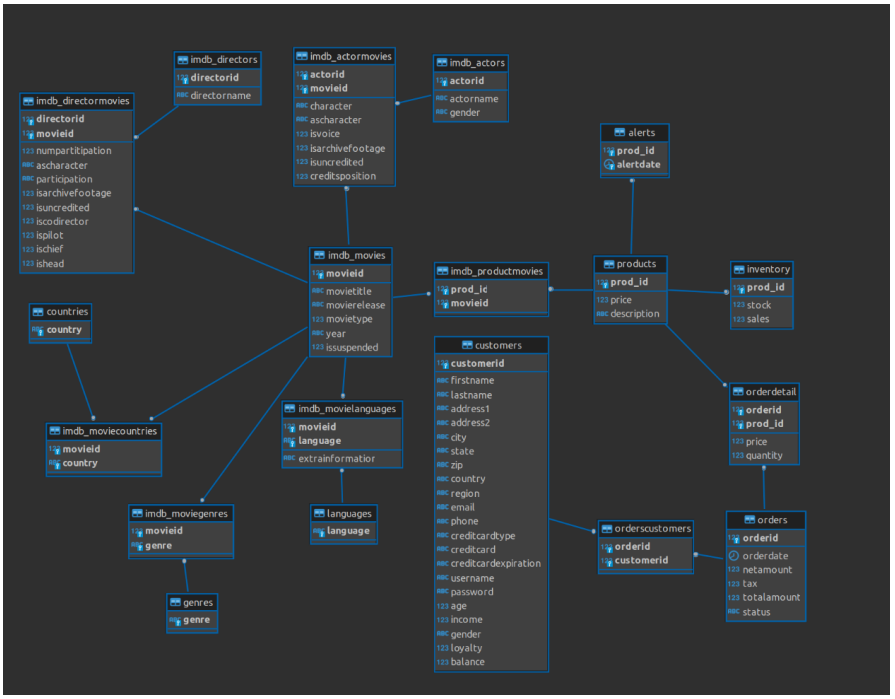
MEMORIA ENTREGA PRÁCTICA 2 SI:

Programación Web y Bases de Datos

Guillermo Martín-Coello y Daniel Varela
Grupo 1391

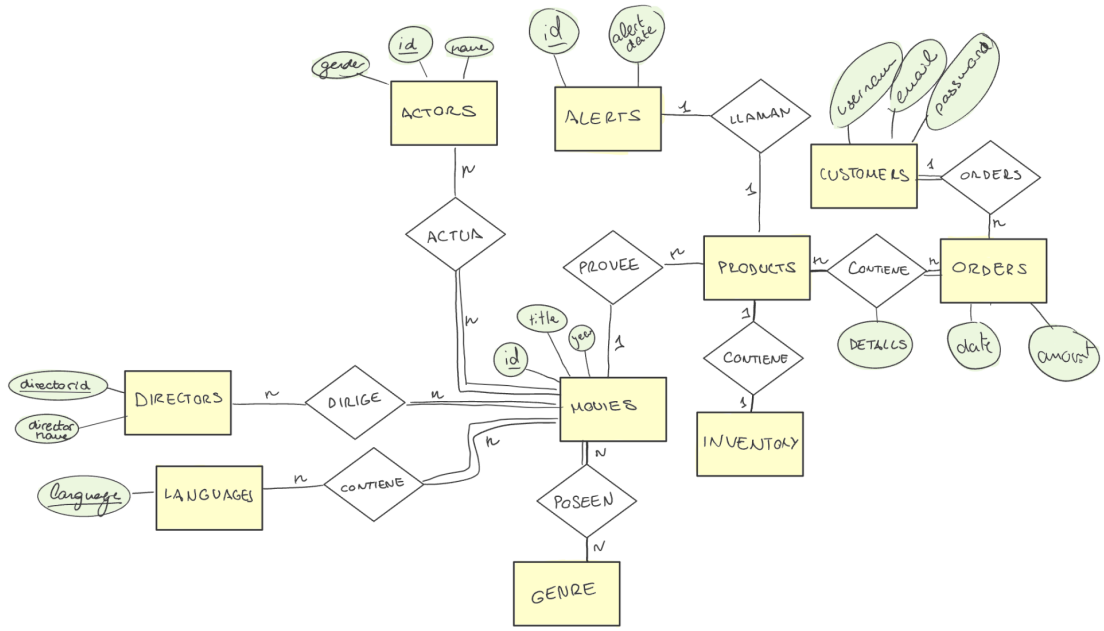
DIAGRAMA E-R

Tras realizar los cambios en acualiza.sql obtenemos las siguientes tablas y relaciones:



Lo cual traducido a un diagrama E-R se mostraría como la siguiente imagen:

DIAGRAMA E-R



Hemos obviado los atributos que no eran claves primarias y foráneas para obtener un diagrama más claro

Cambios en actualiza.sql

Los cambios que hemos realizado en la base de datos original, a parte de los especificados en el enunciado son los siguientes:

- El campo **“numpartitipation”** en **“imdb_directormovies”** pasa a ser una clave **no primaria** ya que no es necesario para relacionar las películas con su/s directores
- El campo **“extrainformation”** en **“imdb_movielanguages”** pasa a ser una clave **no primaria** ya que no es necesario para relacionar las películas con sus idiomas
- Añadimos una **tabla “imdb_productmovies”** la cual **relaciona** la tabla “**products”** con la tabla **“imdb_movies”** mediante sus primary keys.
- Las claves **“actorid”** y **“movieid”** en” **imdb_actormovies”** pasan a ser **primary keys** ya que son necesarias para relacionar cada película con sus actores. Pasan a ser también **claves foráneas** de sus respectivas claves primarias (“actorid” en “imdb_actors” y “movieid” en “imdb_movies” respectivamente)
- Creamos una **clave foránea** que relaciona el atributo **“productid”** en **“inventory”** con la clave primaria **“productid”** en **“products”**.
- Hacemos que los atributos **“orderid”** y **“prod_id”** en “**orderdetail”** sean **claves foráneas** de sus respectivas claves primarias (orderid en orders y prod_id en products respectivamente) y se conviertan también en **claves primarias** de **“orderdetail”**
- Creamos una tabla **“customerorders”** la cual relaciona **“customers”** y **“orders”** con sus primary keys

setPrice.sql

```sql
UPDATE orderdetail
    SET price = q.price
    FROM (SELECT products.price, orderdetail.orderid, orderdetail.prod_id
            FROM products, orderdetail
             WHERE products.prod_id = orderdetail.prod_id) AS q
    WHERE orderdetail.orderid = q.orderid AND orderdetail.prod_id = q.prod_id
```

En esta función hacemos un update a la tabla orderdetail el cual actualiza el valor price a el valor que se le corresponde a cada producto. Lo logramos igualando los campos prod_id de ambas tablas (products y orderdetail)

setOrderAmount.sql

```sql
drop function if exists setOrderAmount() cascade;
CREATE OR REPLACE FUNCTION setOrderAmount() returns void AS $$
BEGIN
 UPDATE orders
    SET netamount = q.netamount,
        totalamount = q.totalamount
    FROM    (SELECT orders.orderid, SUM(products.price * orderdetail.quantity) as netamount, SUM(products.price * orderdetail.quantity) + orders.tax as totalamount
                FROM orderdetail, orders, products
                WHERE orders.orderid = orderdetail.orderid and products.prod_id = orderdetail.prod_id
                GROUP BY orders.orderid) AS q
    WHERE orders.orderid = q.orderid;
end; $$
LANGUAGE 'plpgsql';

select setOrderAmount();
```

En esta función, actualizamos los campos netamount y totalamount de orders según los campos de orderdetail. Hacemos esto obteniendo todas las filas de orderdetails, agrupandolas por el orderid y sumando la multiplicación de cantidad del producto por el precio del mismo, y para el totalprice simplemente le sumamos la cantidad del campo tax. Así con cada order hasta rellenar los campos totalamount y netamount de la tabla orders

getTopActors.sql

```sql
drop function if exists getTopActors() cascade;
CREATE OR REPLACE FUNCTION getTopActors(genre CHAR) RETURNS TABLE (Actor VARCHAR, Num bigint, Debut TEXT,  Film VARCHAR, Director VARCHAR) AS $$
BEGIN
    RETURN QUERY
    select firstquery.actorname as Actor, firstquery.cont as Num, secondquery.debut as Debut, secondquery.film as Film, secondquery.director as Director from
    (
    -- actor genero y pelis por genero si <4 se elimina del resultado
    select q.Actor, q.cont, q.actorname from
        (select imdb_actors.actorid as Actor, count(*) as cont, imdb_actors.actorname as actorname
            from imdb_actors, imdb_actormovies, imdb_movies, imdb_moviegenres
            where imdb_moviegenres.movieid = imdb_movies.movieid
            and imdb_movies.movieid = imdb_actormovies.movieid
            and imdb_actors.actorid = imdb_actormovies.actorid
            and imdb_moviegenres.genre = $1
            group by imdb_actors.actorid
        ) as q
        where q.cont > 4
    ) as firstquery,
    (
    -- primera pelicula de cada actor y genero de la pelicula titulo a
    select distinct on (imdb_actormovies.actorid) imdb_actormovies.actorid as Actor, imdb_movies.movietitle as film, imdb_movies.year as debut, imdb_directors.directorname as director
        from imdb_movies, imdb_actormovies, imdb_moviegenres, imdb_directors, imdb_directormovies
        where imdb_movies.movieid=imdb_actormovies.movieid
        and imdb_moviegenres.movieid=imdb_movies.movieid
        and imdb_movies.movieid=imdb_directormovies.movieid
        and imdb_directormovies.directorid=imdb_directors.directorid
        and imdb_moviegenres.genre = $1
        group by imdb_actormovies.actorid, imdb_movies.movietitle, imdb_movies.year, imdb_movies.movieid, imdb_directors.directorname
        order by imdb_actormovies.actorid, imdb_directors.directorname, imdb_movies.year
    ) as secondquery
    where firstquery.Actor = secondquery.Actor
    order by num desc;
END; $$
LANGUAGE 'plpgsql';
```

Esta función la dividimos en dos partes, en la primera, recopilamos los nombres de los actores que han participado en alguna película del género especificado junto con el número de veces que han participado en películas de dicho género y el id del actor. En el segundo obtenemos el resto de campos necesarios para la función los cuales son el año de debut en el género especificado, la película con la que debutó el actor y el director/directores de dicha película. Finalmente juntamos ambas queries por el id del actor, ordenamos por numero de apariciones y obtenemos un resultado tal que así para el genero de acción:

actor	num	debut	film	director
Rosales Jr., Thomas	19	1998	U.S. Marshals (1998)	Baird, Stuart
Stallone, Sylvester	18	1976	Rocky (1976)	Avildsen, John G.
Welker, Frank	18	2000	Road to El Dorado, The (2000)	Bergeron, Bibo
Thorsen, Sven-Ole	16	1999	13th Warrior, The (1999)	Crichton, Michael
Connery, Sean	15	1962	Longest Day, The (1962)	Annakin, Ken
Ford, Harrison (I)	14	1979	Apocalypse Now (1979)	Coppola, Francis Ford
Llewelyn, Desmond	14	1999	World Is Not Enough, The (1999)	Apted, Michael
Trejo, Danny	13	2000	Reindeer Games (2000)	Frankenheimer, John
Schwarzenegger, Arnold	13	1984	Terminator, The (1984)	Cameron, James (I)
Gibson, Mel (I)	13	1990	Bird on a Wire (1990)	Badham, John
Van Damme, Jean-Claude	12	1988	Bloodsport (1988)	Arnold, Newt
Jackson, Samuel L.	12	1998	Negotiator, The (1998)	Gray, F. Gary

getTopSales.sql

```sql
drop function if exists getTopSales() cascade;
CREATE OR REPLACE FUNCTION getTopSales(IN year1 integer, IN year2 integer) returns table("year" text, movie varchar, sales bigint, id integer) AS $$
BEGIN
    RETURN QUERY
    SELECT distinct
        ON (imdb_movies.year) imdb_movies.year as "year",  imdb_movies.movietitle as movie, sum(inventory.sales) as sales, imdb_movies.movieid as id
        from inventory, imdb_productmovies, imdb_movies
        where inventory.prod_id = imdb_productmovies.prod_id
            and imdb_movies.movieid = imdb_productmovies.movieid
            and imdb_movies.year >= cast(year1 as text) and imdb_movies.year <= cast(year2 as text)
        group by imdb_movies.year, imdb_movies.movietitle, imdb_movies.movieid
        order by imdb_movies.year, sales desc;
END; $$
LANGUAGE 'plpgsql';
```

En esta función, simplemente seleccionamos aquellas películas con mayor número de ventas por cada año en el intervalo especificado. Para hacerlo, relacionamos las películas con el número de sales, las agrupamos por año y simplemente cogemos la primera en número de sales para cada año.
El resultado que obtenemos queda tal que así para los años 1990-1998:

year	movie	sales	id
1990	Hunt for Red October, The (1990)	514	184891
1991	Fisher King, The (1991)	537	138165
1992	My Cousin Vinny (1992)	533	270059
1993	Johnny cien pesos (1993)	522	204107
1994	Only You (1994)	577	291835
1995	Heavy (1995)	543	171259
1996	Glimmer Man, The (1996)	568	156403
1997	Life Less Ordinary, A (1997)	600	229764
1998	Living Out Loud (1998)	542	233000

updOrders.sql

```sql
drop function if exists updateOrder() cascade;
-- Funcion updateOrder()
create function updateOrder() returns trigger as $$
begin
    -- Actualizar la información de la tabla 'orders'
    -- Si se elimina un artículo del carrito, actualizar la información de la tabla 'orders'

    if (tg_op = 'DELETE') then
        update orders
        set totalamount =  tax + (cost-(old.quantity*old.price)), netamount = (cost-(old.quantity*old.price))
        from (
            select orderid, sum(price*quantity) as cost
            from orderdetail
            where orderid = old.orderid
            group by orderid
        ) as q
        where orders.orderid = old.orderid;
        return old;
    elsif (tg_op = 'INSERT') then
        update orders
        set totalamount = tax + (cost+(new.quantity*new.price)), netamount = (cost+(new.quantity*new.price))
        from (
            select orderid, sum(price*quantity) as cost
            from orderdetail
            where orderid = new.orderid
            group by orderid
        ) as q,
        where orders.orderid = new.orderid;
        return new;


    elsif (tg_op = 'UPDATE') then
        update orders
        set totalamount = tax + (cost+((new.quantity-old.quantity)*new.price)), netamount = (cost+((new.quantity-old.quantity)*new.price))
        from (
            select orderid, sum(price*quantity) as cost
            from orderdetail
            where orderid = new.orderid
            group by orderid
        ) as q
        where orders.orderid = new.orderid;
        return new;


    end if;
    -- Si

    return new;
end;
$$ language plpgsql;

-- Realizar un trigger, updOrders, que actualice la información de la tabla 'orders' cuando se
-- añada, actualice o elimine un artículo del carrito.

drop trigger updOrders on orderdetail;
create trigger updOrders before insert or update or delete on orderdetail for each row execute procedure updateOrder();
```

Este trigger se activa siempre que se añada, elimine o se actualice un campo de orderdetail. lo que hace es actualizar el totalamount y el netamount de la tabla orders para que se adecue a la nueva cantidad o precio de un campo de orderdetails, para hacer esto, usamos diferentes metodologías para cada tipo de cambio en la tabla orderdetail ya que por ejemplo para un cambio de tipo delete, no tendríamos campo new y para un cambio de tipo insert , no tenemos campo old. Pero a grandes rasgos todos hacen lo mismo que es mirar la tabla actualizada y cambiar los valores de cantidad o precio de acuerdo con la actualización.

updateInventoryAndCustomer.sql

```
drop function if exists updInventoryAndCustomer() cascade;

create function updInventoryAndCustomer() returns trigger as $$
begin
    if old.status is null and new.status is not null then
    -- actualice las tablas ‘orders’ e 'inventory'
        update orders set orderdate = now() where orderid = new.orderid;
        update inventory set stock = stock - q.quantity
        from( select quantity, prod_id from orderdetail where orderdetail.orderid=new.orderid) as q
        where q.prod_id = inventory.prod_id;
    -- cree una alerta en la tabla 'alertas' para aquellos productos comprados cuya cantidad en
    -- stock en el inventario llega a cero
        insert into alerts(prod_id)
        select q.prod_id
        from (
            select inventory.prod_id, inventory.stock
            from orderdetail natural join products natural join inventory
            where orderdetail.orderid = new.orderid
        ) as q
        where q.stock <= 0;
        -- sume los puntos de fidelización conseguidos por la compra en la tabla ‘customers’
        -- descuente en la tabla ‘customers’ el precio total de la compra

        update customers set loyalty = loyalty + (q.price*(5/100)), balance=balance-q.price
        from (
            select totalamount as price, customerid
            from orders natural join orderscustomers
            where orders.orderid=new.orderid
        ) as q
        where q.customerid = customers.customerid;
    end if;
    return new;

end;
$$
language plpgsql;



drop trigger updInventoryAndCustomer on orders;
create trigger updInventoryAndCustomer after update on orders for each row execute procedure updInventoryAndCustomer();
```

Para este trigger vamos a dividirlo en 3 pasos, antes que nada decir que este se activa cuando hay un cambio en la tabla orders, pero solo realizara acciones cuando dicho cambio sea pasar de un estado null a un estado diferente (es decir, cuando se confirma una compra). La primera fase actualiza el campo orderdate en orders para que contenga la fecha en que se ha confirmado el pago, posteriormente actualizaremos el campo stock en la tabla inventory para que substraiga la cantidad de cada producto que se haya comprado del stock de dicho producto. En la fase 2, comprobamos si el stock se ha agotado, y en dicho caso añadimos un registro a la tabla alerts con el id del producto. Por último actualizamos la tabla customers, sustrayendo al usuario comprador (el que posee la order), el precio de la order del campo balance y añadiendo los puntos de recompensa al campo loyalty

(Para probar ambos triggers se puede hacer en la página)

Contenido Adicional:

En el apartado de implementación a la página, hemos añadido también la metodología de búsqueda con la base de datos, la metodología de registro en la base de datos, la metodología de pago con la base de datos (aunque falta implementar el pago con puntos), la metodología del historial con la base de datos y por último la metodología de añadir saldo con la base de datos.