Memoria Práctica 3:

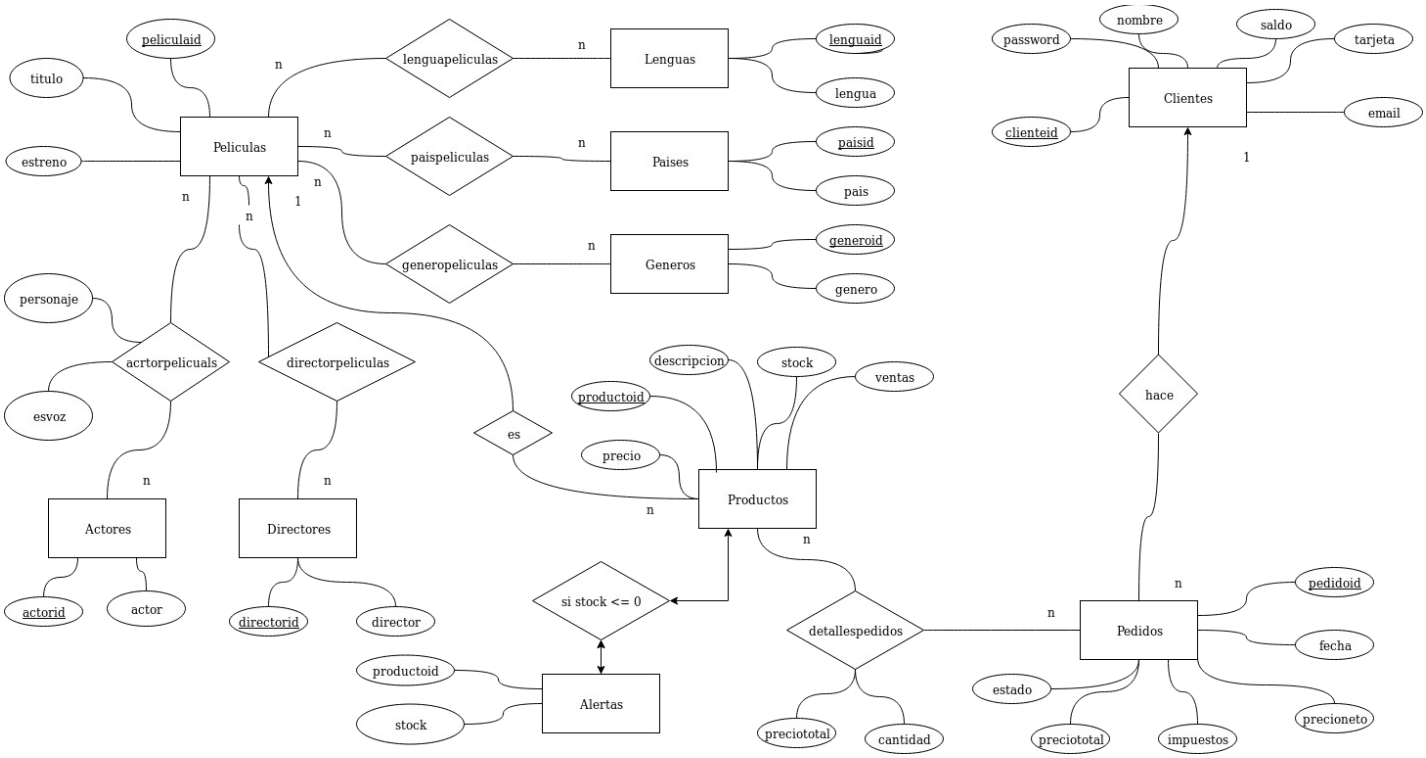
Uso de una base de datos relacional para nuestra aplicación web.

Óscar Gómez Borzdynski y Lucía Colmenarejo Pérez

Modificación de la base de datos aportada:

En primer lugar, hemos creado el diagrama E-R de la base de datos proporcionada. Vemos que no tiene claves extranjeras ni claves primarias de las entidades.

Pensamos que todo lo que termina en id puede ser considerado una clave, ya sea extranjera o primaria. Por ello procedemos a crear el E-R como pensamos que debería ser:



Por ello vamos a realizar los siguientes cambios:

- Especificar las claves primarias en las entidades: Customer, Orders, Products, Imdb_Movies, Imdb_Actors e Imdb_Directors.
- Especificar claves “foreign” en las entidades: OrderDetail, Inventory, Imdb_ActorMovies, Imdb_DirectorMovies, Imdb_Movielanguages, Imdb_MovieCountries, Imdb_MovieGenres.
- Eliminar atributos que consideramos innecesarios como por ejemplo el género de un cliente.
- Añadimos la clase de alertas para el caso en el que se quiera realizar una compra de un producto del cual no queda ninguna existencia en el stock.

Decidimos realizar estas acciones para poder definir las tablas que son entidades y las que son relaciones. Además cribamos datos no necesarios para aligerar el peso de la base de datos y acelerar las consultas.

Descripción del actualiza.sql:

En cuanto a este archivo hemos tomado la decisión de prescindir de ciertos campos de ciertas clases, ya que en el planteamineto inicial de nuestra página no eran necesarios.

Como hemos explicado anterorimente en el diagrama ER hemos creado una tabla llamada *alertas*. Dicha tabla se irá creando a medida que se llame al trigger implementado updInventory. Este trigger creará una nueva alerta cada vez que se desee comprar un producto cuyo stock sea menor o igual que 0.

Por otro lado, hemos creado las tablas *lenguapeliculas*, *paispeliculas* y *generopeliculas*, tal y como se nos pedía en el enunciado.

Por último, hemos actualizado la forma de declarar el id de un cliente para que cuando se cree una nueva instancia de tipo cliente esta se cree con el id siguiente respecto al último ya registrado.

Análisis de las consultas reaizadas:

setPrice.sql:

Esta consulta se encarga de asignar un precio en la clase de detallespedidos. Hay que tener en cuenta que dicho precio se incrementa un 2% por cada uno de los años desde su venta, por lo que habrá que multiplicar por 1,02 el precio inicial tantas veces como años hay entre el año de compra y el actual.

Data Output | Explain | Messages | History

	pedidoid integer	productoid integer	preciototal numeric	cantidad integer
1	179118	1	12.2501903491116	1
2	105560	1	13	1
3	107690	1	11.7745005277889	1
4	119075	1	11.7745005277889	1
5	40492	1	11.7745005277889	1
6	92546	1	12.7450980392157	1
7	172251	1	12.4951941560938	1
8	33920	1	11.7745005277889	1
9	69967	1	12.7450980392157	1
10	45228	1	12.4951941560938	1
11	29740	1	24.5003806982232	2
12	111509	1	24.0199810766894	2
13	137253	1	36.7505710473347	3
14	41204	1	12.2501903491116	1
15	20099	1	12.2501903491116	1
16	80338	1	12.2501903491116	1
17	54522	1	11.7745005277889	1
18	175516	1	12.4951941560938	1
19	53528	1	12.4951941560938	1
20	124977	1	12.7450980392157	1
21	135115	1	12.2501903491116	1
22	67815	1	12.2501903491116	1
23	14397	1	12.7450980392157	1
24	178455	1	11.7745005277889	1
25	137178	1	12.0099905383447	1
26	166633	1	12.2501903491116	1
27	125166	1	12.0099905383447	1
28	138725	1	12.7450980392157	1

Salida (parte) de la consulta para aquellos pedidos con un productoid = 1.

setOrderAmount.sql:

Esta consulta se trata de un procedimiento almacenado que se encarga de rellenar los campos de *precioneto* y *preciototal* de la tabla de *pedidos*.

El *precioneto* contiene la suma de los precios de las películas del pedido; mientras que el *preciototal* contiene dicha suma junto con los impuestos.

Data Output | Explain | Messages | History

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	1	2016-07-21	693	31.7185697808534	15	46.7185697808534	Shipped

Salida de la consulta para el pedido con id=1

getTopVentas.sql:

Esta consulta recibe como argumento un año y devuelva para ese año y los siguientes la película que más se ha vendido.

En esta consulta lo que hemos hecho ha sido realizar la suma de todas las películas (para cada película distinta) cada año. Posteriormente, nos quedamos con la película con mayor ventas de cada año. Para finalizar nos quedamos con los años que son siguientes e igual al que nos han pasado como argumento en la consulta.

Data Output | Explain | Messages | History

	gettopventas record
1	(2012.9."Male and Female (1919)")
2	(2013.101."No Looking Back (1998)")
3	(2014.136."Love and a .45 (1994)")
4	(2015.142."Illtown (1996)")
5	(2016.134."Wizard of Oz, The (1939)")
6	(2017.134."Life Less Ordinary, A (1997)")
7	(2018.57."Jerk, The (1979)")

Salida de la consulta cuando se le pasa como argumento el año 2012.

getTopMonths.sql:

Esta consulta recibe unos umbrales de número de productos y de importe (*preciototal*) acumulados, y devuelva los meses (en realidad la pareja año-mes) en los que se ha superado alguno de los dos umbrales, junto con su importe y productos.

	gettopmonths record
1	(2015.5.291060.19001)
2	(2014.10.286378.19086)
3	(2016.7.300902.19280)
4	(2014.9.287139.19133)

Salida de la consulta cuando le pasamos como umbrales: 19000 productos y 320000 euros.

UpdOrders.sql:

Se trata de un trigger que actualiza la tabla de *detallespedidos* cuando se añade o elimina un pedido de nuestro carrito.

Nuestro trigger se llama *updOrders* y llama la función *updOrders_function* la cual realizará una acción u otra dependiendo de si se hace un INSERT, UPDATE o DELETE sobre nuestra tabla *detllespedidos.*

Los campos que van a ser modificados independientemente de la acción realizada son precioneto y preciototal. Como se puede observar la manera de calcular estos dos campos es similar. Lo único que cambia es la variable *result.* Este campo si estamos en DELETE contendrá el preciototal del pedido a eliminar , OLD.preciototal (en negativo, para que a la hora de sumarlo se reste); si es el caso de INSERT contendrá el preciototal del nuevo elemento a insertar, NEW.preciototal; y si es el caso de UPDATE será el resultado de NEW.preciototal- OLD.preciototal.

CASO INSERT:

Data Output | Explain | Messages | History

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	50	2015-02-03	5605	13.5694416174774	15	28.5694416174774	Shipped

Insertamos:
INSERT INTO detallespedidos(pedidoid, productoid, preciototal, cantidad) VALUES (50, 2660,10, 1);

Data Output | Explain | Messages | History

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	50	2015-02-03	5605	23.5694416174774	15	38.5694416174774	Shipped

Se ve como el precio ha aumentado en 10.

CASO UPDATE:

Data Output | Explain | Messages | History

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	161010	2018-10-25	12499	91.3	21	112.3	

Vemos que al hacer la consulta:

```
UPDATE detallespedidos SET cantidad=2, preciototal=24 WHERE pedidoid=161010 AND productoid= 2453;
```

EL resultado obtenido es el siguiente, donde vemos que se modifica tanto el preciototal como el precioneto con una cantidad de 24.

Data Output | Explain | Messages | History

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	161010	2018-10-25	12499	115.3	21	136.3	

CASO DELETE:

Vamos a eliminar del pedido 161010 el producto con id 2453. Para ello vemos cúal es el precioneto y el preciototal inicial de nuestro pedido:

Data Output | Explain | Messages | History

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	161010	2018-10-25	12499	194.6	21	215.6	Paid

Posteriormente vemos los productos que tiene:

	pedidoid integer	productoid integer	preciototal numeric	cantidad integer
1	161010	3599	21.6	1
2	161010	4746	22.5	1
3	161010	1388	13.2	1
4	161010	2639	18	1
5	161010	4048	16	1
6	161010	2453	12	1
7	161010	3599	21.6	1
8	161010	4746	22.5	1
9	161010	1388	13.2	1
10	161010	2639	18	1
11	161010	4048	16	1

De estos productos seleccionamos el 2453 y lo eliminamos de la tabla:

```
DELETE FROM detallespedidos WHERE pedidoid=161010 AND productoid=2453;
```

A continuación podemos ver que en los productos del pedido con id 161010 ya no se encuentra dicho producto:

	pedidoid integer	productoid integer	preciototal numeric	cantidad integer
1	161010	3599	21.6	1
2	161010	3599	21.6	1
3	161010	4746	22.5	1
4	161010	1388	13.2	1
5	161010	2639	18	1
6	161010	4048	16	1
7	161010	4746	22.5	1
8	161010	1388	13.2	1
9	161010	2639	18	1
10	161010	4048	16	1

Al igual que también se puede comprobar que el preciototal y el precioneto del pedido han disminuido justo el precio del producto eliminado:

Data Output | Explain | Messages | History

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	161010	2018-10-25	12499	182.6	21	203.6	Paid

updInvetory.sql:

En este caso tenemos que actualizar la tablas *productos* y *pedidos* cuando se finalice la compra. El trigger también deberá crear una alerta en una nueva tabla llamada *alertas* si la cantidad en stock llega a cero.

Cogemos un pedido en el que el estado esté a NULL:

	pedidoid integer	fecha date	clienteid integer	precioneto numeric	impuestos numeric	preciototal numeric	estado character varying(10)
1	147769	2018-10-24	11460	92.7	21	113.7	

Posteriormente, escogemos uno de los productos de ese pedido, eligiendo uno entre los siguientes:

	pedidoid integer	productoid integer	preciototal numeric	cantidad integer
1	147769	3541	19.2	1
2	147769	578	18	1
3	147769	4524	19	1
4	147769	414	14	1
5	147769	6413	22.5	1

Nos quedamos con el producto con id=578:

	productoid integer	peliculaid integer	precio numeric	descripcion character varying(30)	stock integer	ventas integer
1	578	39414	18	Ultra	443	176

En este resultado vemos que el stock de este producto es de 443. Realizando una consulta de tipo UPDATE sobre el pedido con esa id, cambiando el estado de NULL a ‘Paid’, vemos que el stock disminuye en uno:

	productoid integer	peliculaid integer	precio numeric	descripcion character varying(30)	stock integer	ventas integer
1	578	39414	18	Ultra	442	177

La serir de consultas realizadas para obtener cada uno de estos resultados son las siguientes:

```
SELECT * FROM pedidos where pedidoid = 147769;

SELECT * FROM detallespedidos where pedidoid = 147769;

SELECT * FROM productos where productoid = 578; -> stock = 443

UPDATE pedidos set estado = 'Paid' WHERE pedidoid= 147769;

SELECT * FROM productos where productoid = 578; -> stock = 442
```