

# PRÁCTICA 3

## PROGRAMACIÓN WEB Y BASES DE DATOS



### AUTORES:

Alberto Estepa Fernández  
Sergio Teso Lorenzo

## Descripción Esquema Entidad Relación y Base de datos

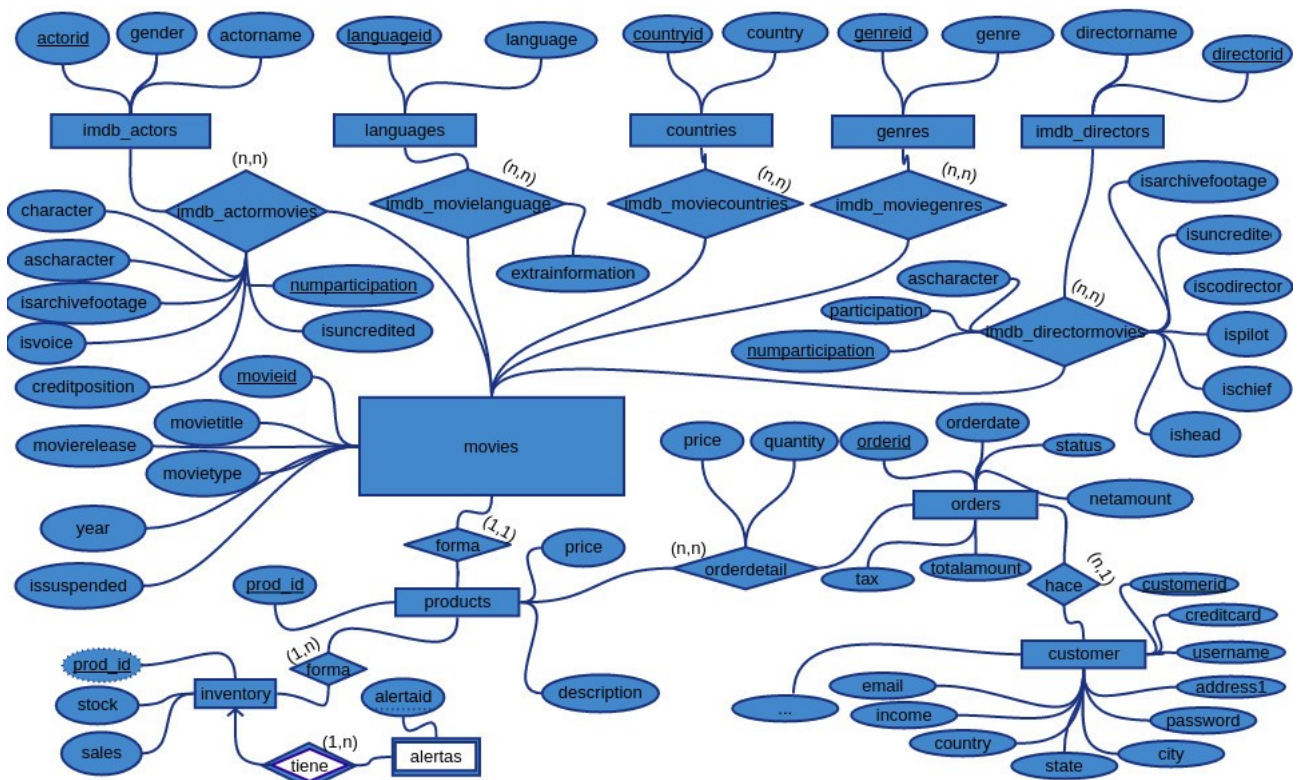
El esquema final de la base de datos una vez realizado el actualiza.sql (apartado siguiente) se presenta de la siguiente forma:

Como centro del esquema tenemos las películas relacionadas con los productos que formarán nuestro inventario y los pedidos de la página web.

Las películas por una parte están representadas por actores (relación n-n, ya que una película está representada por varios actores y un actor actúa en varias películas), están dirigidas por directores (relación n-n por el mismo motivo que con actores, donde podemos ver que una película esta dirigida por varios directores) y están en varios idiomas, se encuentran en varios países y tienen varios géneros temáticos.

Todos los productos forman parte de un inventario. Cuando se acaba un determinado producto en el inventario, salta una alarma en el inventario, por ésto la relación esta íntegramente relacionada donde el identificador de producto es clave en la entidad alerta.

Por otro lado, los productos forman parte de varios pedidos y un pedido esta formado por determinados productos diferentes o iguales. Estos pedidos son ejercidos por clientes en relación (n-1, ya que un cliente efectúa varios pedidos pero un pedido es ejercido por un solo cliente).



Aunque se visualiza dicho diagrama arriba, para una mejor apreciación de los detalles de ésta, se adjunta en la entrega. Además incluimos el diagrama generado por SchemaSpy.

## Descripción actualiza.sql

Hemos visto que las tablas no estaban relacionadas entre sí. Lo primero que hemos modificado son añadir Foreign Keys a aquellas tablas que era necesario y relacionarlas unas con otras permitiendo los cambios en cascada.

Por otra parte, como indica el enunciado, hemos convertido las tablas 'moviecountries', 'moviegenres' y 'movielanguages' donde cada película le correspondía un país, genero e idioma respectivamente (todos atributos cuyo tipo eran varchar) creando las tablas countries, genres y languages que almacenan todos los países, géneros e idiomas con un índice para cada uno, modificando así las tablas originales donde en vez de almacenar varchar, almacenamos índices que se corresponden con la información anterior.

Por otro lado, cuando registramos el usuario, le pedimos ciertos requisitos a rellenar en el formulario. Otros datos que podemos guardar en la base de datos no son necesarios rellenarlos por el usuario y pueden estar a null. Así eliminamos la condición de NOT NULL de aquellos atributos que no entren dentro del formulario pedido.

Además, hemos asignado un check  $> 0$  al income (saldo) de manera que no pueda haber un saldo negativo.

Para la inserción de datos si no se proporciona el id en el insert into de varias tablas no se podría insertar al ser este único. Para facilitar este trabajo hemos actualizado los ids de alertas, countries, customers, genres, languages, orders, products, imdb\_actors, imdb\_directors, imdb\_movies con un *SELECT setval( 'secuencia correspondiente', SELECT max(id) FROM tabla).*

Por último, como última función de actualiza.sql y como nos pide el enunciado al hacer el trigger updInventory, hemos añadido una tabla alertas para almacenar las alertas que nos indica si la cantidad en stock llega a cero.

## Descripción setPrize.sql

La base de datos proporcionada presenta el problema que el precio de venta de los pedidos no están relleno. Para calcularlos, sabemos que cada año que pasa desde el año del pedido sube un 2%, teniendo el precio actual en la tabla 'products' y el año de venta en la tabla 'orderdetail'. La fórmula que aplicaremos será multiplicar el precio actual de la película por 0.98 (se corresponde al incremento 2 por ciento por año) tantas veces como años entre el actual y el de venta hayan transcurrido.

Output pane

Data Output Explain Messages History

Query returned successfully: 1000112 rows affected, 12241 ms execution time.

	orderid integer	prod_id integer	price numeric	quantity integer		orderid integer	prod_id integer	price numeric	quantity integer
1	1043	4003		1	1	1043	4003	12.6548911	1
2	1050	466		1	2	1050	466	13.8355224	1
3	1050	2480		1	3	1050	2480	22.1368358	1
4	1050	3140		1	4	1050	3140	22.1368358	1
5	1050	4571		1	5	1050	4571	15.6802587	1
6	1050	188		1	6	1050	188	13.8355224	1
7	1050	6346		1	7	1050	6346	14.3889432	1
8	1050	3672		1	8	1050	3672	17.9861791	1
9	1050	3730		1	9	1050	3730	13.2821015	1
10	1050	1009		1	10	1050	1009	10.1460497	1
11	1050	899		1	11	1050	899	14.7578905	1
12	1057	5916		1	12	1057	5916	9.94312876	1
13	1057	5312		1	13	1057	5312	20.3382179	1
14	1057	4381		1	14	1057	4381	16.2705743	1
15	1057	4519		1	15	1057	4519	16.2705743	1
16	1057	5530		1	16	1057	5530	9.94312876	1
17	1057	187		1	17	1057	187	10.8470495	1
18	1057	4692		1	18	1057	4692	17.1744951	1
					19	1057	449	12.6548911	1
					20	1065	4890	22.344	1
					21	1065	5538	11.76	1
					22	1065	1200	18.62	1

## Descripción setOrderAmount.sql

Otro problema de la base de datos proporcionada es que en la tabla orders(pedidos) las columnas que indican el netamount(sumas del precio de las películas del pedido) y totalamount(igual que el netamount pero añadiéndole los impuestos) están vacías.

Para solucionar dicho problema, hemos creado un procedimiento almacenado que se encarga de esto. Se divide en dos partes fundamentales, una para cada columna pedida.

Así básicamente para actualizar la columna netamount, seleccionamos las películas que corresponden al pedido y según la cantidad que correspondan a cada película sumamos el precio de dichas películas.

Por otro lado, para actualizar la columna totalamount, seleccionamos las películas que corresponden al pedido y como tenemos calculado ya el netamount, multiplicamos dicho valor por el porcentaje de las tasas (tomándolo en cuenta como un porcentaje) y se lo añadimos al valor sin aplicar las tasas.

id	netamount numeric	tax numeric	totalamount numeric
		15	
		15	
		18	
		15	
		15	
		15	
		18	
		15	
		15	
		15	
		15	
		15	
		15	
		18	

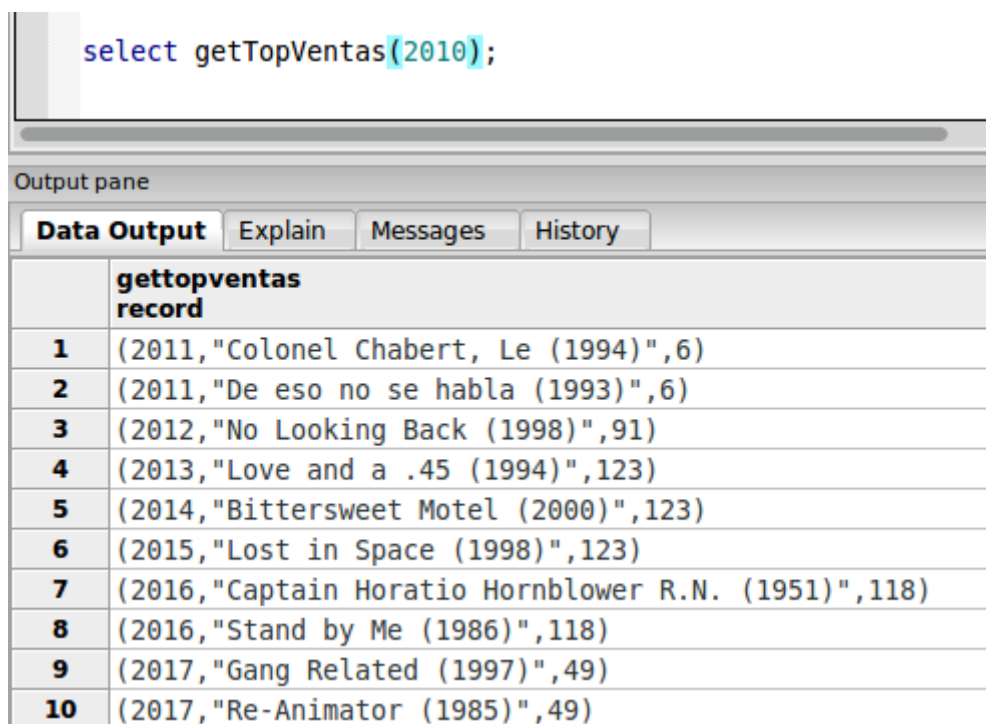
id	netamount numeric	tax numeric	totalamount numeric
	31.6932	15	36.44718000
	29.40	15	33.81000000
	158.368	18	186.87424000
	132.43916	15	152.30503400
	86.589664	15	99.57811360
	61.2715992	15	70.46233904
	12	18	14.16000000
	136.37680	15	156.83332000
	497.248675	15	571.83597625
	51.3427012	15	59.04410640
	112.266962	15	129.10700730
	65.9862181	15	75.88415082
	139.955256	15	160.94853740
	209.72	15	241.17800000
	11	18	12.98000000

## Descripción getTopVentas.sql

Queremos hallar las películas más vendidas a partir de un cierto año. Para resolverlo creamos una función a la que se le pasa el parámetro del año y nos devuelva una tabla con la solución a partir de ese año.

La implementación de dicha función básicamente se divide en dos partes: por una parte calculamos el máximo de las películas vendidas cada año ordenadas por año y por otro damos la información de todas las películas vendidas en esos años agrupadas por la cantidad de ventas en ese año. Así juntamos dicha información y solo mostramos las películas que se corresponden con dicho máximo de ventas por año.

Vemos que sucede que hay años en que hay más de una película, eso es porque comparten el liderato como películas más vendidas. Hemos considerado correcto permitir dicho comportamiento.



The screenshot shows a SQL query editor with the query `select getTopVentas(2010);` entered. Below the editor is an "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is active, displaying the results of the query in a table format. The table has two columns: "gettopventas" and "record". The results are as follows:

	gettopventas	record
1	(2011,"Colonel Chabert, Le (1994)",6)	
2	(2011,"De eso no se habla (1993)",6)	
3	(2012,"No Looking Back (1998)",91)	
4	(2013,"Love and a .45 (1994)",123)	
5	(2014,"Bittersweet Motel (2000)",123)	
6	(2015,"Lost in Space (1998)",123)	
7	(2016,"Captain Horatio Hornblower R.N. (1951)",118)	
8	(2016,"Stand by Me (1986)",118)	
9	(2017,"Gang Related (1997)",49)	
10	(2017,"Re-Animator (1985)",49)	

## Descripción getTopMonths.sql

Queremos hallar los meses según las ventas de películas en dichos meses cumplan ciertos umbrales estadísticos, como son el número de películas vendidas y el dinero que se ha obtenido de las ventas dicho mes.

Para ello creamos una función en la que devolvemos una tabla con los resultados. En dicha función tenemos una consulta con la que separamos los pedidos por meses y en una subconsulta obtenemos el total de películas vendidas y dinero ganado por ventas por cada mes. Filtramos por los datos pasados por parámetro y obtenemos la solución.

Adjuntamos varias ejecuciones de la función, para demostrar su funcionamiento.

```
select * from getTopMonths(20000,3000000);
```

Output pane

	<b>mes</b> double precision	<b>anyo</b> double precision	<b>prod_totales</b> numeric	<b>precio_total</b> numeric
1	9	2013	19133	3073179.763
2	5	2015	18350	3026621.098
3	7	2015	19379	3153091.487
4	12	2015	18414	3060438.448
5	1	2016	18626	3058010.703
6	8	2016	18522	3024235.414
7	9	2016	18387	3078367.721
8	10	2016	18576	3159435.016

```
select * from getTopMonths(19000,320000);
```

Output pane

	<b>mes</b> double precision	<b>anyo</b> double precision	<b>prod_totales</b> numeric	<b>precio_total</b> numeric
1	12	2011	2502	370433.1291
2	1	2012	3887	604555.3086
3	2	2012	4761	717624.4606
4	3	2012	7262	1083194.278
5	4	2012	7897	1156070.402
6	5	2012	10146	1559783.752
7	6	2012	11078	1685378.467
8	7	2012	12547	1908740.976
9	8	2012	14916	2294920.196
10	9	2012	15639	2379792.045
11	10	2012	18178	2740420.765
12	11	2012	17917	2745273.608
13	12	2012	17671	2684272.418
14	1	2013	18867	2977201.693
15	2	2013	16012	2617070.866



## Descripción updOrders.sql

Nos ponemos en la situación de actualizar el carrito de la compra. Así al añadir un producto (insert) cambiar un producto (update) o eliminar un producto (delete), el carrito de la compra debe actualizarse automáticamente. Así necesitamos un trigger que realice dicha tarea.

Distinguimos cada caso:

- Añadimos un producto (insert):  
Al añadir un producto, lo que debemos modificar del pedido es netamount y totalamount, añadiendo el nuevo precio de la/las película/s añadidas.
- Eliminamos un producto (delete):  
Al eliminar un producto, lo que debemos modificar del pedido es netamount y totalamount, eliminando el precio de la/las película/s eliminadas.
- Cambiamos un producto (update):  
Al modificar un producto, lo que debemos cambiar del pedido es netamount y totalamount, eliminando el precio de la/las película/s eliminadas y añadiendo el precio de la/las película/s añadidas.

Como detalle de implementación, para distinguir cada caso nos servimos de la variable TG\_OP que se genera automáticamente al llamar al trigger.

Adjuntamos un ejemplo de insercción (análogo con eliminación y modificación):

```
select * from orders where orderid = 181790
```

Output pane							
Data Output Explain Messages History							
	orderid integer	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying(10)
1	181790	2015-03-20	14093	70.7328	15	81.3427200000000000	Shipped

Después insertamos tres productos con prod\_id '4003'.

```
INSERT INTO orderdetail(orderid, prod_id, price, quantity)
Values(181790, '4003', 20, 3);
```

Así, obtenemos dichos resultados en la tabla orders, referido a ese pedido (observamos que se modifica automáticamente lo que queríamos):

```
select * from orders where orderid = 181790
```

Output pane							
Data Output Explain Messages History							
	orderid integer	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying(10)
1	181790	2015-03-2	14093	130.7328	15	150.3427200000000000	Shipped



## Descripción updInventory.sql

Estamos en la situación que se termina de realizar una compra. Tendremos que actualizar nuestro inventario quitando de stock la cantidad de productos comprados y añadiendo a ventas esa misma cantidad. También hay que tener en cuenta que el stock de un producto llegue a 0 y se active alguna especie de alerta en el sistema.

Para realizar esta funcionalidad se han realizado los siguientes procedimientos:

- Crear una tabla alertas, la cual cada una tendrá su propio ID y el id del producto del cual se nos ha acabado el stock.
- Crear un trigger que cuando termine una venta actualizara las cantidades de stock y de vendidos de un producto.

El trigger es disparado cuando en un pedido su status se convierte en '*Paid*'. Entonces se considera que se ha comprado el producto y se actualiza la tabla de productos, restando al stock y añadiendo a las ventas la cantidad comprada de cada producto del pedido.

Se adjunta las pruebas realizadas del correcto funcionamiento del trigger:

### Caso 1:

Data Output						
	prod_id integer	orderid integer	price numeric	quantity integer	stock integer	sales integer
1	1938	1	10.5644	1	962	155
2	1014	1	10.5644	1	201	186
3	1288	1	10.5644	1	160	173

Podemos ver los tres productos de la orden 1 con sus cantidades, stock y ventas. Realizamos el disparador del trigger y obtenemos:

Data Output						
	prod_id integer	orderid integer	price numeric	quantity integer	stock integer	sales integer
1	1938	1	10.5644	1	961	156
2	1014	1	10.5644	1	200	187
3	1288	1	10.5644	1	159	174

Podemos ver que se han restado y sumado correctamente las cantidades necesarias.

## Caso 2:

	Data Output	Explain	Messages	History		
	prod_id integer	orderid integer	price numeric	quantity integer	stock integer	sales integer
1	1288	1	10.5644	159	159	174
2	1938	1	10.5644	1	961	156
3	1014	1	10.5644	1	200	187

Aumentamos la cantidad del porducto 1288 de manera que el stock se quede a 0.

	Data Output	Explain	Messages	History		
	prod_id integer	orderid integer	price numeric	quantity integer	stock integer	sales integer
1	1288	1	10.5644	159	0	333
2	1938	1	10.5644	1	960	157
3	1014	1	10.5644	1	199	188

Disparamos el trigger y vemos como se efectua correctamente. Ahora miramos la tabla de alertas:

	Data Output	Explain
	alertaid integer	prod_id integer
1	1	1288

Podemos ver como la alerta del producto 1288 se ha registrado con éxito y el trigger ha cumplido satisfactoriamente su funcionalidad.

## Login y Register con Database

Una vez desarrollada nuestra database toca implementarla con nuestra página web desarrollada anteriormente. Nuestro objetivo poder realizar un login y register de los customers de nuestra web.

Lo primero que tenemos que hacer es conectarnos a la database mediante PHP. Para ello usaremos un objeto PDO por la flexibilidad que este nos ofrece, ya que si migráramos nuestra database de pgsq a mysql, sqlite... etc solo tendríamos que cambiar la url que le pasamos como parámetro al constructor del objeto PDO. Una vez asegurados que la conexión ha sido exitosa podemos ejecutar nuestras sentencias SQL en el servidor. Usaremos PDO → prepare para introducir nuestras secuencias sql ya que nos permite pasarle parámetros posteriormente. Usando bindParam añadimos los parametros deseados y con → execute ejecutamos la sentencia deseada.

Para el login haremos un select de la password de la tabla customers donde el username es igual al username que recibimos por el formulario. Si la query devuelve 0 columnas el usuario no existe mientras que si nos devuelve la password comprobamos que la introducida por el formulario y la recibida por base de datos es la misma. **Nota:** Tanto si el usuario no existe o la password introducida es incorrecta el mensaje de error dado es password incorrecta para evitar dar información comprometedor de la base de datos.

Para el register una vez los datos han sido validados por el formulario y enviados al servidor se comprueba que el usuario no existe ya, sino se muestra un mensaje de error. Si el usuario no existe se crea la sentencia SQL correspondiente a la inserción de información en la base de datos y se ejecuta la sentencia. Una vez todo ha resultado correctamente se activa la variable login de sesión y se redirige al usuario al index donde todo aparece como lo vería un usuario loggeado.