

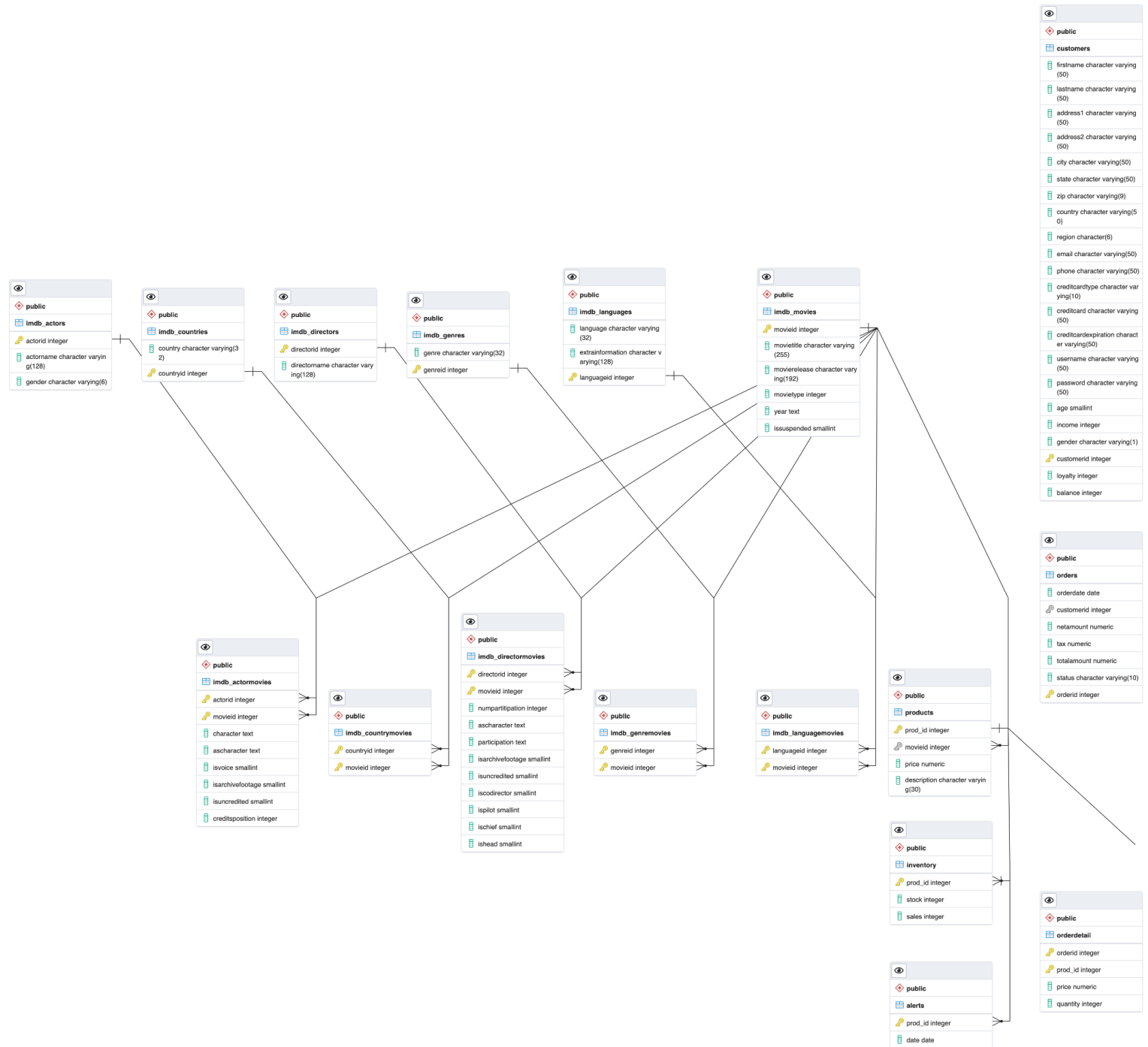
# **Memoria Práctica 2**

Autores:

César Ramírez Martínez

Pedro Urbina Rodríguez

# 1. Diagrama entidad-relación.



## 2. Cambios introducidos en la base de datos.

Con el fin de asegurar la integridad de los datos y de solventar algunos errores, hemos introducido una serie de cambios y modificaciones en la base de datos proporcionada.

Todos ellos pueden verse codificados en el archivo “actualiza.sql”.

Iremos comentando los cambios realizados por tablas:

### ❖ inventory

- Hacer “*prod\_id*” clave extranjera con el fin de identificar unívocamente cada uno de los inventarios.

### ❖ customers

- Añadir “*orderid*” como clave extranjera.
- Hacer la clave primaria “*customerid*” serial, de manera que al introducir una nueva fila en la tabla le sea asignada un identificador automáticamente.
- Añadir la columna “*balance*” con 0 como valor por defecto (más adelante daremos una explicación de cómo se rellenan los valores de esta columna).
- Añadir la columna “*loyalty*” con 0 como valor por defecto.

### ❖ orders

- Añadir “*customerid*” como clave extranjera.
- Hacer la clave primaria “*orderid*” serial, de manera que al introducir una nueva fila en la tabla le sea asignada un identificador automáticamente.

### ❖ orderdetail

- Añadir “*prod\_id*” como clave extranjera.
- Añadir “*orderid*” como clave extranjera.
- Hacer la dupla (“*prod\_id*”, “*orderid*”) clave primaria.

- Mediante una simple query (ver “actualiza.sql”) eliminamos los registros que quedan con la misma clave primaria tras el punto anterior.

#### ❖ imdb\_actormovies

- Añadir “*actorid*” como clave extranjera.
- Añadir “*movieid*” como clave extranjera.
- Hacer la dupla (“*actorid*”, “*movieid*”) clave primaria

#### ❖ imdb\_directormovies

- Eliminamos de la tupla que forma la clave primaria la columna “*numparticipation*”.

#### ❖ alerts

Tabla creada para almacenar las alertas generadas cuando el stock de un producto llega a 0.

- Añadir y hacer primaria la clave “*prod\_id*”. También la hemos establecido como serial, de manera que al introducir una nueva fila en la tabla le sea asignada un identificador automáticamente.
- Añadimos el campo “*date*” para almacenar el momento en el que el stock del producto llega a 0.

#### ❖ imdb\_countries

Tabla creada para almacenar los diferentes países de origen de las películas almacenadas.

- Mediante una simple query buscamos todos los posibles países de los que puede ser originaria una película y los añadimos todos a una nueva tabla.
- Además, creamos una clave primaria (serial) “*countryid*”.

#### ❖ imdb\_countrymovies

Tabla creada para crear la relación N-M entre países y películas.

- Añadimos la clave extranjera “*countryid*”.
- Añadimos la clave extranjera “*movieid*”.

- Hacemos clave primaria la dupla ("*countryid*", "*movieid*").
- Mediante una query rellenamos esta tabla para establecer las relaciones necesarias entre cada película y sus países de origen.
- Por último, eliminamos la tabla inicial imdb\_moviecountries, ya que su labor ha sido suplida por la de esta nueva tabla creada y, además, no nos parecía que su nombre fuese demasiado acertado

#### ❖ imdb\_genres

Tabla creada para almacenar los diferentes géneros que pueden tener las películas almacenadas.

- Mediante una simple query buscamos todos los posibles géneros que una película puede tener y los añadimos todos a una nueva tabla.
- Además, creamos una clave primaria (serial) "*genreid*".

#### ❖ imdb\_genremovies

Tabla creada para crear la relación N-M entre géneros y películas.

- Añadimos la clave extranjera "*genreid*".
- Añadimos la clave extranjera "*movieid*".
- Hacemos clave primaria la dupla ("*genreid*", "*movieid*").
- Mediante una query rellenamos esta tabla para establecer las relaciones necesarias entre cada película y sus géneros.
- Por último, eliminamos la tabla inicial imdb\_moviegenres, ya que su labor ha sido suplida por la de esta nueva tabla creada y, además, no nos parecía que su nombre fuese demasiado acertado.

#### ❖ imdb\_languages

Tabla creada para almacenar los diferentes lenguajes de las películas almacenadas.

- Mediante una simple query buscamos todos los posibles lenguajes de una película y los añadimos todos a una nueva tabla.
- Además, creamos una clave primaria (serial) "*languageid*".

❖ imdb\_languages\_movies

Tabla creada para crear la relación N-M entre géneros y películas.

- Añadimos la clave extranjera “*languageid*”.
- Añadimos la clave extranjera “*movieid*”.
- Hacemos clave primaria la dupla (“*languageid*”, “*movieid*”).
- Mediante una query rellenamos esta tabla para establecer las relaciones necesarias entre cada película y sus lenguajes.
- Por último, eliminamos la tabla inicial imdb\_movie\_languages, ya que su labor ha sido suplida por la de esta nueva tabla creada y, además, no nos parecía que su nombre fuese demasiado acertado.

## 2. Consultas, triggers y funciones.

❖ **setPrice.sql**

Archivo que almacena la query encargada de rellenar la columna “*price*” de la tabla orderdetail.

Esta query rellena esta columna con el precio del producto de cada orderdetail en el momento en el que se realizó la compra.

Se tiene en cuenta que el precio del producto ha ido aumentando un 2% cada año desde que se hizo la compra.

❖ **setOrderAmount.sql**

Archivo que contiene el procedimiento almacenado **setOrderAmount()**, encargado de rellenar las columnas “*netamount*” y “*totalamount*” de la tabla orders.

Este procedimiento rellena estas columnas teniendo en cuenta los orderdetails de cada uno de los orders.

El “*netamount*” de cada order simplemente se calcula sumando los precios totales (“*price*” \* “*quantity*”) de cada orderdetail que conforma un order en dado.

El “*totalamount*” de cada order se calcula obteniendo los impuestos sobre el “*netamount*” y sumándoselos al propio “*netamount*”.

#### ❖ **getTopSales.sql**

Archivo que almacena la función **getTopSales**(year1 INT, year2 INT, OUT Year INT, OUT Film varchar(255), OUT sales bigint).

Encargada de devolver una tabla con información sobre las películas que más se han vendido entre dos años pasados como parámetros de la función (“year1” y “year2”).

Para ver los detalles de su implementación ver el archivo **getTopSales.sql**.

#### ❖ **getTopActors.sql**

Archivo que almacena la función **getTopActors**(genre varchar(128), OUT Actor varchar(128), OUT Num bigint, OUT Debut text, OUT Film varchar(255), OUT Id integer, OUT Director varchar(128)).

Encargada de devolver una tabla con información sobre los actores que más veces han actuado en un género dado como parámetro (genre), con información sobre la película con la que debutaron dichos actores en ese género y con información sobre el director que dirigió dicha película. Siempre que hayan trabajado en más de 4 películas de dicho género.

Para ver los detalles de su implementación ver el archivo **getTopActors.sql**.

#### ❖ **updOrders.sql**

Archivo que almacena los triggers encargados de actualizar las columnas “*netamount*” y “*totalamount*” de la tabla orders, cuando la tabla orderdetail es modificada en algún aspecto.

En concreto este archivo almacena tres triggers; uno para cuando una nueva fila es introducida en la tabla orderdetail, otro para cuando una fila es eliminada y otro para cuando una fila es actualizada. Cada uno de ellos modifica de una forma distinta las columnas antes mencionadas en la tabla orders.

Véase el archivo **updOrders.sql** para mayor detalle.

#### ❖ **updInventoryAndCustomer.sql**

Archivo que almacena cinco triggers encargados de realizar diferentes labores sobre la base de datos.

El primero de ellos simplemente añade un nuevo order en tabla orders cuando detecta que un nuevo usuario ha sido dado de alta en la aplicación.

El segundo simplemente actualiza la fila que sea necesaria en la tabla inventory (reduciendo el stock disponible) cuando una compra se completa con éxito.

El tercero se encarga de crear una nueva alerta cuando el stock de un producto llega a cero.

El cuarto actualiza los puntos de fidelización del cliente que acaba de realizar una compra.

Y el último de ellos actualiza el balance del cliente que acaba de realizar una compra, restando el coste de esta.

Véase el archivo **updInventoryAndCustomer.sql** para mayor detalle.

### 3. Guía detallada de los ficheros.

Nuestra práctica consta de dos directorios principales, “app” y “sql”.

En “app” es donde almacenamos todos los ficheros relacionados con la funcionalidad, la visualización y los datos de la aplicación. Así como toda la información relacionada con los usuarios registrados.

En el directorio “sql” es donde hemos almacenado todos los ficheros que realizan la modificación de la base de datos, añadiendo funciones y triggers.

Dentro del directorio “app” observamos los siguientes archivos/directorios: “catalogue”, “routes.py”, “database.py”, “static” y “templates”.

#### -“static”:

En este directorio tenemos los subdirectorios: “css”, “images” y “scripts”.

##### -“css”:

Subdirectorio que simplemente contiene el archivo “estilo.css”. Archivo que contendrá toda la información en lo referido al estilo y aspecto de los documentos HTML de la aplicación.

##### -“images”:

Subdirectorio que almacena los posters de las películas disponibles en nuestro vídeo club y la imagen del logo de nuestra aplicación.

##### -“scripts”:

Subdirectorio que contiene todo el código de JavaScript, JQuery y AJAX del que hacemos uso en la aplicación.

#### -“templates”:

Directorio que almacena todos los documentos HTML de nuestra aplicación. Incluyendo “base.html”, la plantilla base del resto de archivos.

#### -“routes.py”:

Archivo escrito en Python que contiene las funciones que reaccionan a las interacciones del cliente con nuestra aplicación. Por ejemplo: en este archivo está definida la función que carga el detalle de la película que el cliente selecciona en la pantalla de inicio. Este archivo utiliza todas las funciones definidas en el archivo “database.py” para el acceso a la base de datos, con el objetivo de obtener los datos y mostrarlos al usuario.

#### -“database.py”:

Archivo escrito en Python que contiene las funciones que interactúan con la base de datos a través de sqlalchemy. Entre ellas se encuentran las necesarias para implementar el carrito accediendo a las bases de datos, las películas de las bases de datos a mostrar en la página de bienvenida, las necesarias para el manejo de



usuarios (login, register...), búsquedas y filtros. Para ello se ha utilizado la librería sqlalchemy de Python.

Dentro del directorio “sql” observamos los siguientes archivos/directorios: “makefile”, “actualiza.sql”, “getTopActors.sql”, “setOrderAmount.sql”, “setPrice.sql”, “updInventoryAndCustomer.sql”, “updOrders.sql”.

El primero es, como su propio nombre indica, un archivo makefile que programamos para no tener que repetir todos los comandos cada vez que realizamos una modificación en la base de datos. Para ello, ejecutando el comando “**make all**”, eliminamos, volvemos a crear y cargamos los datos en la base de datos. Después aplicamos nuestras modificaciones recogidas en el archivo actualiza.sql, creamos los triggers y probamos a ejecutar las funciones.

El resto de archivos implementan las consultas, funciones, procedimientos almacenados y triggers solicitados para completar los apartados a)-h).

## 4. Guía de instalación y ejecución.

Para ejecutar la aplicación deberemos extraer el zip adjunto en moodle.

Abriendo una terminal dentro del directorio “sql” empezamos por actualizar la base de datos.

Para ello podemos usar el makefile. Debemos incluir en la carpeta sql el dump “dump\_v1.4.sql.gz”. Con el comando “**make**” o “**make all**” la base de datos ya estará lista para su uso. La otra opción, si la base de datos está ya cargada y poblada es simplemente ejecutar el archivo “actualiza.sql” con el comando “**make update\_db**”.

Si simplemente se desea resetear la base de datos y repoblarla, se puede ejecutar el comando “**make reset\_db**” (siempre y cuando en la carpeta se encuentre el archivo dump).

Por defecto, los comandos “**make**” y “**make all**” ejecutarán las funciones getTopActors y getTopSales con los parámetros por defecto establecidos dentro de cada uno de los archivos donde están implementadas estas funciones.

Si tras resetear y poblar la base de datos se quiere ejecutar las funciones sin necesidad de resetear ni repoblar la base de datos de nuevo, se puede hacer uso de los comandos “**make getTopSales**” o “**make getTopActors**” para ejecutar cada una de las funciones de manera independiente.

(OJO. Si se ejecuta uno de estos dos comandos sin haber creado la base de datos o sin haberla poblado previamente, estos fallarán)

Una vez creada y modificada la base de datos simplemente deberemos ejecutar el comando “**python3 -m app**” con una terminal abierta al nivel de la carpeta “app”. Accediendo al enlace que nos indica la ejecución en la terminal podremos acceder a la página web.

### **¡IMPORTANTE!**

Para navegar por la web utilizar los botones introducidos e implementados por nosotros en los documentos HTML, y no los proporcionados por el navegador.

Por ejemplo, para volver a la página de inicio desde el detalle de una película, pulsar el botón “**Volver**”.

## 5. Resultados y evidencias.

Empezaremos describiendo los cambios realizados en la base de datos, probando procedimientos, funciones y triggers y acabaremos con una explicación de los cambios en la página web final.

Empezamos la práctica escribiendo en el archivo “actualiza.sql” los cambios que consideramos necesarios, explicados en la primera sección de esta misma memoria. La mayoría de cambios se ven reflejados en el diagrama E-R de la primera sección.

Para el apartado b) hemos realizado la consulta setPrice.sql. Antes de ejecutarla, la tabla orderdetail contiene orderdetails con el campo price a NULL, pero tras ejecutar la consulta obtenemos los resultados esperados, con el campo “price” con un valor coherente con lo que se nos pide en el enunciado:

1

select \* from orderdetail

Data Output

Explain

Messages

Notifications

	orderid integer	prod_id integer	price numeric	quantity integer
1	1043	4003	[null]	1
2	1050	466	[null]	1
3	1050	2480	[null]	1
4	1050	3140	[null]	1
5	1050	4571	[null]	1
6	1050	188	[null]	1
7	1050	6346	[null]	1
8	1050	3672	[null]	1
9	1050	3730	[null]	1

1

select \* from orderdetail

Data Output

Explain

Messages

Notifications

	orderid integer	prod_id integer	price numeric	quantity integer
1	1050	3672	18.06	1
2	1475	4850	17.31	1
3	1522	4161	12.26	1
4	1559	823	16.98	1
5	1619	4488	10.19	1
6	1922	1089	13.46	1
7	1946	6396	21.18	1
8	2026	3648	11.11	1
9	2080	1430	16.98	1
10	2242	3471	9.43	1

De la misma manera, para el apartado c) creamos el procedimiento almacenado setOrderAmount, que será llamado por actualiza.sql, para rellenar los campos netamount y totalamount de orders. Se muestran los resultados antes y después de ejecutar el procedimiento almacenado:

1 `select * from orders`

Data Output	Explain	Messages	Notifications				
	orderid [PK] integer	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying (10)
1	466	2019-03-31	26	[null]	15	[null]	Paid
2	1844	2017-08-29	128	[null]	15	[null]	Paid
3	2641	2016-04-28	194	[null]	15	[null]	Paid
4	4709	2019-07-25	339	[null]	15	[null]	Paid
5	7324	2019-06-29	534	[null]	15	[null]	Paid
6	7757	2016-08-02	562	[null]	15	[null]	Paid
7	9809	2020-03-16	719	[null]	15	[null]	Paid
8	10086	2019-04-03	741	[null]	15	[null]	Paid
9	10509	2019-05-04	778	[null]	15	[null]	Paid
10	10914	2017-02-23	806	[null]	15	[null]	Paid
11	12013	2020-12-05	896	[null]	18	[null]	Paid
12	14498	2019-05-27	1089	[null]	15	[null]	Paid
13	16013	2016-11-16	1209	[null]	15	[null]	Paid

1 `CALL setOrderAmount();`

2 `select * from orders`

Data Output	Explain	Messages	Notifications					
	orderid integer	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying (10)	orderid [PK] integer
1		2018-11-07	10232	40.00	15	46.00	Paid	550
2		2018-04-07	9670	100.00	15	115.00	Paid	125002
3		2018-12-31	10460	132.65	15	152.55	Paid	135006
4		2017-08-24	4022	9.62	15	11.06	Shipped	53347
5		2020-12-22	15	71.64	18	84.54	Shipped	1014
6		2018-10-11	12317	41.66	15	47.91	Shipped	158829
7		2019-09-09	11642	100.19	15	115.22	Shipped	150366
8		2020-03-22	6738	80.00	15	92.00	Paid	87858
9		2021-10-24	11460	67.16	21	81.26	[null]	181788
10		2019-02-15	13823	40.00	15	46.00	Paid	178159
11		2020-08-14	1978	160.00	15	184.00	Paid	26508
12		2018-05-18	11425	80.00	15	92.00	Paid	147416
13		2017-06-02	13150	40.00	15	46.00	Paid	169557

Para el apartado d) creamos la función getTopSales que dado dos años devuelve las películas que más se han vendido entre esos dos años, una por año, ordenadas

de mayor a menor por número de ventas. Se muestran los resultados al llamar a esta función con los años 2017 a 2020:

```

pedro@pedro-GF63-Thin-10SC:~/Videos/sil/sil/P2/sql$ psql -U alumnodb sil < getTopSales.sql
CREATE FUNCTION
year |          film          | sales
-----+-----+-----
2017 | Roman Holiday (1953)   |   137
2019 | Shopping (1994)        |   136
2018 | They Made Me a Criminal (1939) |   135
2020 | Band of the Hand (1986) |   133
(4 rows)


```

Para el apartado e) realizamos la función getTopActors que dada un parámetro “género” devuelva los actores o actrices que más veces han actuado en dicho género, ordenados de más actuaciones a menos, siempre que hayan trabajado en más de 4 películas de ese género. Se adjunta un resultado de la ejecución de esta función tanto en la terminal como en su integración ya en la página web, ambas con el género “Fantasy”, y vemos que los resultados son iguales (salvo que en la página están limitados a 10 por defecto):

```

pedro@pedro-GF63-Thin-10SC:~/Videos/sil/sil/P2/sql$ psql -U alumnodb sil < getTopActors.sql
CREATE FUNCTION
actor | num | debut |          film          | id | director
-----+-----+-----+-----+-----+-----
Welker, Frank | 20 | 1984 | Gremlins (1984) | 162128 | Dante, Joe (I)
Cummings, Jim (I) | 10 | 1988 | Who Framed Roger Rabbit (1988) | 439315 | Zemeckis, Robert
Williams, Robin (I) | 9 | 1991 | Fisher King, The (1991) | 138165 | Gilliam, Terry
Williams, Robin (I) | 9 | 1991 | Hook (1991) | 180529 | Spielberg, Steven (I)
Purvis, Jack | 8 | 1977 | Star Wars (1977) | 378000 | Lucas, George
Malet, Arthur | 8 | 1964 | Mary Poppins (1964) | 249956 | Stevenson, Robert (I)
Foray, June | 8 | 1950 | Cinderella (1950) | 76909 | Jackson, Wilfred
Foray, June | 8 | 1950 | Cinderella (1950) | 76909 | Luske, Hamilton
Englund, Robert | 8 | 1984 | Nightmare On Elm Street, A (1984) | 280322 | Craven, Wes
Foray, June | 8 | 1950 | Cinderella (1950) | 76909 | Geronimi, Clyde
Polito, Jon | 7 | 1986 | Highlander (1986) | 175309 | Mulcahy, Russell
Baker, Kenny (I) | 7 | 1977 | Star Wars (1977) | 378000 | Lucas, George

```



## Video Club César y Pedro

MENÚ

Inicio

Registrar

Iniciar Sesión

Cesta

Usuarios en línea: 216

Not logged in

BEST ACTORS

Actor	Numero de películas	Año debut	Película debut	Director película debut
Welker, Frank	20	1984	<a href="#">Gremlins (1984)</a>	Dante, Joe (I)
Cummings, Jim (I)	10	1988	<a href="#">Who Framed Roger Rabbit (1988)</a>	Zemeckis, Robert
Williams, Robin (I)	9	1991	<a href="#">Fisher King, The (1991)</a>	Gilliam, Terry
Williams, Robin (I)	9	1991	<a href="#">Hook (1991)</a>	Spielberg, Steven (I)
Purvis, Jack	8	1977	<a href="#">Star Wars (1977)</a>	Lucas, George
Malet, Arthur	8	1964	<a href="#">Mary Poppins (1964)</a>	Stevenson, Robert (I)
Foray, June	8	1950	<a href="#">Cinderella (1950)</a>	Jackson, Wilfred
Foray, June	8	1950	<a href="#">Cinderella (1950)</a>	Luske, Hamilton
Englund, Robert	8	1984	<a href="#">Nightmare On Elm Street, A (1984)</a>	Craven, Wes
Foray, June	8	1950	<a href="#">Cinderella (1950)</a>	Geronimi, Clyde

Para el apartado f) hemos creado las tablas solicitadas, que podemos ver tras ejecutar el archivo “actualiza.sql”. Estas tablas actúan como relación m-n por lo que solo contienen pares de claves extranjeras de las tablas a las que hacen referencia. Se adjuntan imágenes de algunas de las filas de las tablas creadas:

1	select * from imdb_countries			
Data Output	Explain	Messages	Notifications	
country character varying (32)	countryid [PK] integer			
1	Algeria		1	
2	Argentina		2	
3	Aruba		3	
4	Australia		4	
5	Austria		5	
6	Belgium		6	
7	Bhutan		7	
8	Botswana		8	

1	select * from imdb_genremovies			
Data Output	Explain	Messages	Notifications	
genreid [PK] integer	movieid [PK] integer			
1	9	103		
2	14	103		
3	26	103		
4	1	147		
5	3	147		
6	6	147		

1	select * from imdb_languagemovies			
Data Output	Explain	Messages	Notifications	
languageid [PK] integer	movieid [PK] integer			
1	21	103		
2	20	103		
3	19	103		
4	18	103		
5	17	103		
6	21	147		

Para el apartado g) y h) creamos el trigger y las funciones solicitadas. Para ello implementamos los archivos “updOrders.sql” y “updInventoryAndCustomer”. Para comprobar su funcionamiento usaremos directamente la página web, usando el carrito, que tan solo modifica la tabla orderdetail, y veremos como se actualiza automáticamente la tabla orders. Aprovechando el proceso probaremos también el login y register del apartado j) y del carrito, el apartado l) de integración en el portal:

Tras completar un registro y un login exitoso en la página web como en la práctica anterior, por la terminal del servidor obtenemos el customerid del nuevo usuario que usamos para comprobar que la tabla customers se ha actualizado correctamente, como se muestra a continuación:

1	select * from customers where customerid = 14094			
Data Output	Explain	Messages	Notifications	
customerid [PK] integer	customerid [PK] integer			
1	14094			

La tabla orders crea un nuevo carrito automáticamente para este nuevo usuario gracias al trigger “tr\_updInventoryAndCustomerNewCustomer”, como podemos ver a continuación. Vemos además que es el carrito activo porque el estado es NULL:

1

select \* from orders where customerid = 14094

Data Output

Explain

Messages

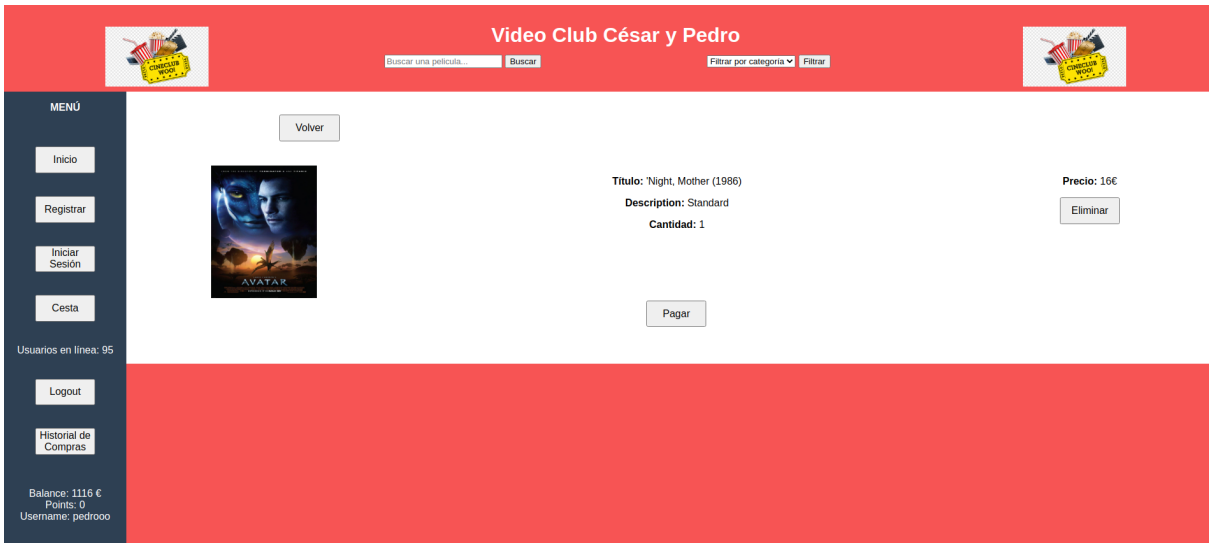
Notifications

	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying (10)	orderid [PK] integer
1	2021-11-19	14094	0	0	0	[null]	181791

Podemos ver que todavía no hay ningún orderdetail asociado a este carrito:

1	select * from orderdetail where orderid = 181791			
Data Output	Explain	Messages	Notifications	
orderid [PK] integer	prod_id [PK] integer	price numeric	quantity integer	

Tras añadir un producto a la cesta desde la página web, queda reflejado en el carrito como en la práctica anterior:



También queda reflejado en la base de datos:

```
1 select * from orderdetail where orderid = 181791
```

	orderid [PK] integer	prod_id [PK] integer	price numeric	quantity integer	
1	181791	5	16	1	

Además la tabla orders se ha actualizado automáticamente gracias al trigger como esperábamos, con unos nuevos campos netamount, totalamount y tax (0 por defecto):

```
1 select * from orders where orderid = 181791
```

	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying (10)	orderid [PK] integer
1	2021-11-19	14094	16	0	16.00	[null]	181791

Lo mismo ocurre cuando añadimos más artículos del mismo producto o los eliminamos, tanto la tabla orderdetail como la tabla orders se actualizan de la manera esperada, por ejemplo, tras añadir otra película a la cesta la fila de orders actualiza correctamente los campos del precio:

```
1 select * from orders where orderid = 181791
```

Data Output		Explain	Messages	Notifications			
	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying (10)	orderid [PK] integer
1	2021-11-19	14094	29	0	29.00	[null]	181791

Además, al pagar este carrito el saldo del usuario es actualizado correctamente, como se puede ver en la web y en la base de datos ( $1116 - 29 = 1087$ ), y los puntos de fidelidad se han añadido correctamente:



# Video Club César y Pedro



MENÚ

Inicio

Registrar

Iniciar Sesión

Cesta

Usuarios en línea: 132

Logout

Historial de Compras

Username: pedriooo

Balance: 1087 €

Points: 145



Título: 'Night, Mother' (1986)

Description: Standard

Cantidad: 1

Precio: 16€



Título: 'Breaker' Morant (1980)

Description: Standard

Cantidad: 1

Precio: 13€

Continuar Comprando


Felicidades! Ha realizado su compra de 29.00€ con éxito.

```
1 select customerid, balance, loyalty from customers where customerid = 14094
```

Data Output		Explain	Messages	Notifications
	customerid [PK] integer	balance integer	loyalty integer	
1	14094	1087	145	

Comprobamos por último que cuando se compra un producto que tenía solo un stock, se crea una alerta en la tabla alerts. Elegimos un producto con stock = 1 en el inventario:

```
1 select * from inventory where prod_id = 220
```

Data Output		Explain	Messages	Notifications
	prod_id [PK] integer	 stock integer	 sales integer	
1		220	1	161

Averiguando a qué película corresponde este producto, compramos desde la página web la última unidad de este producto:



De manera que ahora el inventario queda:

Query Editor Query History

```
1 select * from inventory where prod_id = 220
```

Data Output Explain Messages Notifications

	prod_id [PK] integer	stock integer	sales integer
1	220	0	161

Y, en efecto, en la tabla alerts hay una nueva alerta de este producto:

```
1 select * from alerts where prod_id = 220
```

Data Output Explain Messages Notifications

	prod_id [PK] integer	date date
1	220	2021-11-19

Por último, el funcionamiento de la página web es muy similar al de la práctica anterior, la única diferencia es que ahora casi toda la información se almacena en la base de datos. Este comportamiento era el deseado, ya que uno de nuestros objetivos era cambiar toda la estructura subyacente de la página web sin que el usuario note las diferencias. El cambio más notable en el “funcionamiento aparente” (es decir desde el punto de vista del usuario que accede a la página web) es el ya mostrado de la página principal, en el que ahora se muestra la lista generada por el procedimiento getTopActors, pudiendo el usuario elegir el género y el número de actores que aparecerán por pantalla. El resto de funcionalidad se mantiene igual que en la práctica anterior, excepto que ahora toda la información se almacena en la base de datos.