Francisco Lerma Martínez y Jesús Morato Martín Grupo 1401 SI

MEMORIA SISTEMAS INFORMÁTICOS: PRÁCTICA 2

Guía detallada de ficheros

Dentro del directorio principal nos encontramos los siguientes directorios y archivos:
-start.wsgi: que nos permite ejecutar con Apache la aplicación web.
-Memoria-P2: este fichero.
-/app: directorio con los datos y programas de la aplicación.
-thesessions
-/SQL: directorio con los scripts.sql y las imágenes en .png de los modelos ER.

Dentro del repositorio /app tenemos lo siguiente:
-catalogue/catalogue.json: json con el catálogo de películas de la aplicación.
-static/css: contiene el fichero .css común para todos los templates html.
-static/js: que a su vez contiene:
    -js/ajax.js: código ajax que nos permite comunicar asíncronamente con el servidor el número de clientes online.
    -js/historial-toggle.js: JQuery que nos permite desplegar pedidos en el historial.
    -js/password-strength.js: JQuery que calcula la fuerza de la contraseña y la muestra en la barra de progreso dinámicamente.
    -js/register.js: JavaScript que valida los campos de registro del usuario.
-static/img: imágenes que cargar.
-templates/: directorio que contiene todos los templates que se muestran como capturas más adelante, junto con su estilo.
-__init__.py: inicializa Flask y Flask Sessions.
-__main__py: ejecuta la app.
-routes.py: fichero python con las funciones views que se enlazan a las urls, y funciones auxiliares que se explican y documentan en el propio código.
-database.py: fichero python que utiliza SQLAlchemy para realizar queries en la base de datos especificada como engine. Es utilizada para el login, register, top ventas y el carrito.

Dentro del repositorio /SQL tenemos lo siguiente:
-actualiza.sql: script sql que actualiza la base de datos, la mejora y optimiza. Se explica mas adelante.
-setPrice.sql: script sql que rellena el precio de los orderdetail.
-setOrderAmounts.sql: procedimiento almacenado sql que completa las orders una vez se han calculado los precios de las orderdetail.
-getTopVentas: función SQL que devuelve el top ventas de cada año ordenado de mayor a menor. Recibe los dos años que marcan el intervalo como argumentos.
-getTopMonths: función SQL que muestra que meses se ha vendido más de X productos o se ha superado Y ingresos.
-updOrders: trigger que actualiza orders cuando se añaden objetos al carrito.

-updInventory: trigger que actualiza el inventario al comprar un carrito y crea alertas si se queda sin stock.
-reset.sh: script bash que realiza todos los anteriores scripts en orden adeás de crear la base de datos de 0.
-ModeloERBaseDatosBasica.png: es la imagen que contiene el Modelo ER de la base de datos que nos proporcionan originalmente, sin los cambios que le realizamos.
-ModeloERNuestraBD.png: contiene el diagrama ER de la base de datos tras el actualiza.sql.

Instrucciones de uso e Implementación:

Para poder ejecutar la plataforma necesitaremos crear un entorno virtual de python3 en el directorio principal, que tiene que ser public_html, llamado si1pyenv en el que tengamos python 3.6, flask, postgresql y SQLAlchemy instalados para poder ejecutar la aplicación.

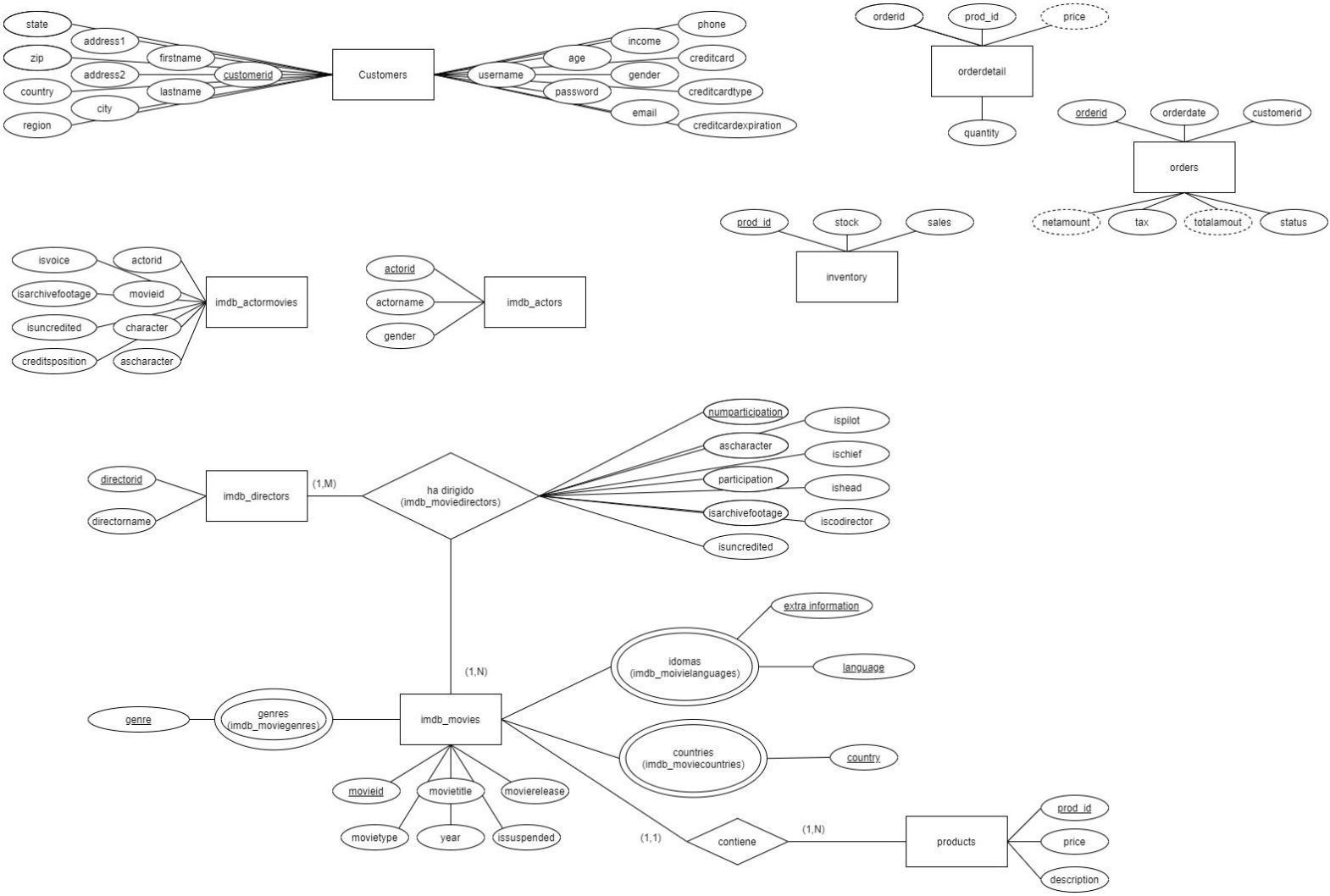
La aplicación se puede ejecutar desde la terminal en public_html con el siguiente comando:
$python3 -m app
Y accediendo a la url http://0.0.0.0:5001/index podemos ir a la página principal de la aplicación y hacer lo que deseemos.

Otra opción es mediante el servidor Apache, antes de ejecutarlo necesitamos darle permisos a los directorios de nuestra aplicación para que no de error al crear los directorios de los usuarios. Desde la terminal en public_html podemos ejecutar
$chmod -R 777 ./
comando con el damos los permisos necesarios. Una vez hecho si accedemos la url http://localhost/~eps/start.wsgi/ accederemos mediante Apache a la aplicación.

Para ejecutar la base de datos y actualizarla es necesario crear una llamada si1 con el dump que se proporciona en el campus, y ejecutar todos los scripts en el orden que aparece arriba o bien ejecutar el script bash reset.sh.

Apartado A (actualiza.sql):

Una vez cargada la base de datos básica, vimos las tablas creadas a partir del pgAdmin y conseguimos ver las relaciones y atributos que tenían las entidades. La imagen que se encuentra en el directorio SQL/ModeloERBaseDatosBasica.png muestra el diagrama ER de esta base de datos sin modificar. También se añade a continuación, pero se ve con mejor calidad en la imagen original.Se puede observar que faltan muchas relaciones entre las entidades, algunos tipos de datos no son los más correctos, faltan claves primarias, las secuencias no están bien inicializadas, etc.



Se puede observar que faltan muchas relaciones entre las entidades, algunos tipos de datos no son los más correctos, faltan claves primarias, las secuencias no están bien inicializadas, etc.

A continuación enumeramos los cambios que realizamos en cada tabla en actualiza.sql:

customers:

Lo primero que nos dimos cuenta es que hay nombres de usuario repetidos que son de diferentes clientes. Como eliminar el segundo customer no es eficaz, lo que hacemos es concatenar al username el id del customer para hacer que sea único. Y así una vez ponemos la restricción de que sea único el username no da error de duplicados. Por último, hacemos que algunos atributos puedan ser null, ya que los únicos obligatorios son username, password, email y la tarjeta de crédito.

imdb_actormovies:
Esta tabla no tiene PK por lo que la generamos a partir de sus dos foreign key que son movieid y actorid. Estas dos FK tampoco están definidas, así que hay que crearlas también. Se puede ver que esta tabla será una relación entre los actores que actúan en las pelis. También se cambian los smallint por boolean ya que nos parece más clarificador.

idmdb_directormovies:
En esta tabla sí que existen las FK a movieid y directorid, pero no están bien creadas ya que les falta el correcto borrado y actualización con CASCADE, el cual se añade en actualiza. También se añade la PK con movieid y directorid, de manera que también será una relación entre los directores y las pelis que han dirigido. Por último se cambian a boolean los smallint.

Apartado f) Atributos multivaluados
Para el apartado f hemos creado 3 tablas que se llaman countries, genres y languages que relacionan de manera única un país, género o idioma con su id. De manera que ahora en las tablas imdb_moviecountries, imdb_moviegenres, imdb_movielanguages se relaciona el id de cada pelicula con el id de cada idioma, género o lenguaje de manera que no hay strings repetidas y se aumenta la consistencia de la base de datos, se elima la columna con el nombre para sustituirla con el de su id. Por lo tanto estas tablas serán relaciones (tendrán como PK la FK de movieid y la FK de genresid, languageid o countryid) y relacionarán las películas con sus idiomas, géneros y países.

Para crear las tablas simplemente generamos id autoincrementales de manera que solo insertamos los diferentes idiomas, países y géneros.

imdb_movies:
Quitamos parámetros obligatorios y cambiamos los smallint por boolean.

orderdetail:
Nos hemos dado cuenta de que no tiene PK la tabla por lo que la tenemos que crear. La PK más lógica es la combinada por orderid y prodid, ambas serán FK de la tabla final. Pero da error ya que existen claves duplicadas, lo cual nos indica que hay valores incorrectos en la tabla ya que no pueden haber dos orderdetail que repitan esos campos, ya que en un mismo order no puedes tener dos orderdetail del mismo producto. De manera que antes de generar la PK, creamos una nueva tabla en la que meteremos los prodid y orderid sin repetición, la única diferencia con la tabla anterior es que ahora en la cantidad sumamos las cantidades de todos los duplicados. Es decir, que si tenemos dos duplicados con cantidades 2 y 4 por ejemplo, en la nueva tabla tendremos el orderid y prod id único con una cantidad de 6. De esta manera agrupamos todas las repeticiones y sumamos las cantidades de los productos. Por último borramos la primera tabla y renombramos esta última con orderdetail. También comprobamos que la cantidad siempre sea positiva con un check.

orders:
Hacemos una nuev FK de manera que se actualiza y borra en CASCADE, también metemos valores por defecto y cambiamos la obligatoriedad de algunos.

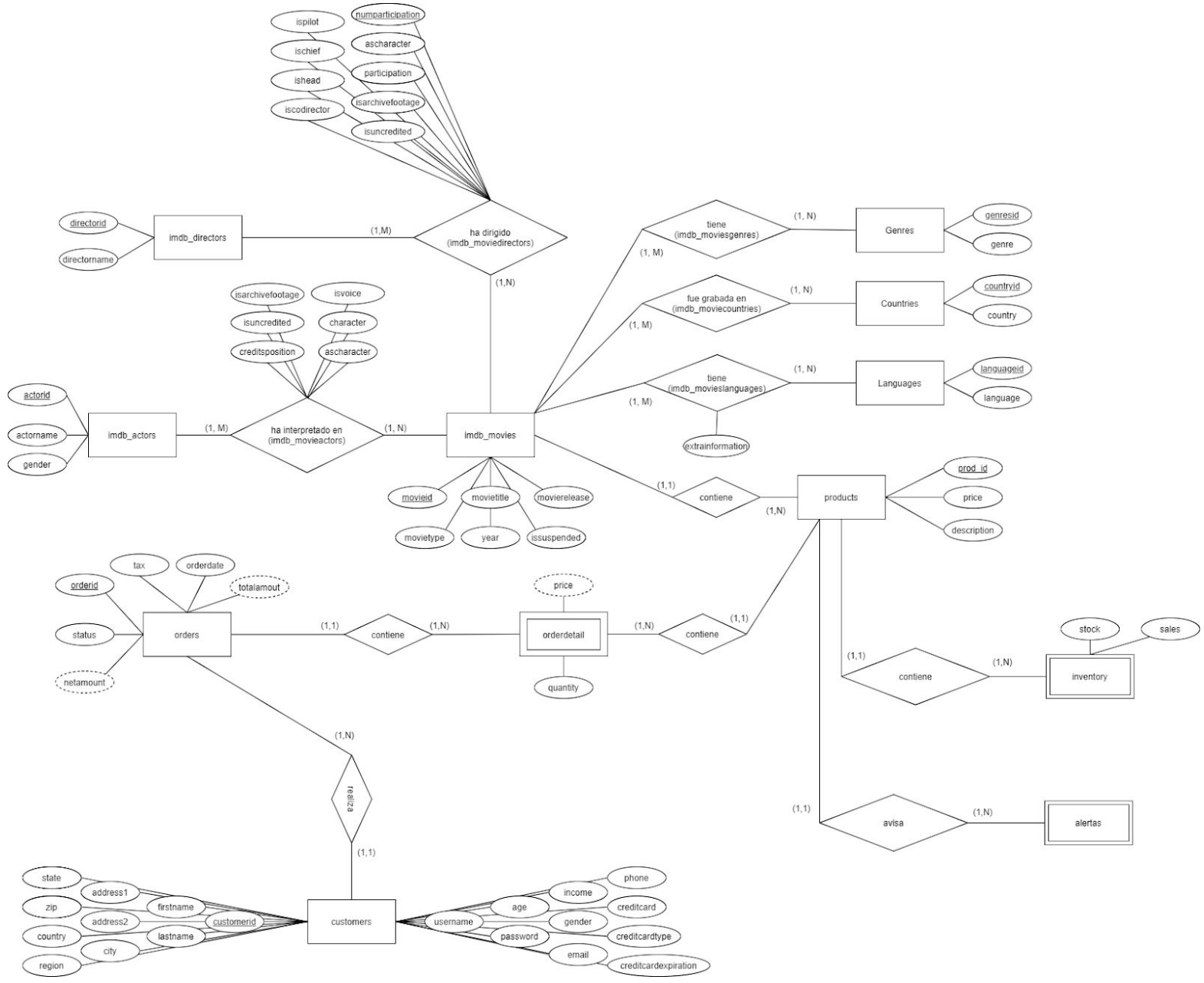
alertas:

Nos creamos la tabla de aletas para el apartado h, que relaciona una alerta con un producto, la cual nos avisa que su stock está a 0.

Secuencias:

Hacemos que el valor de las secuencias de los id de las tablas anteriores estén inicializados en el valor más 1 del último id de cada tabla, de manera que al crear nuevas filas los id se creen correctamente y no haya repeticiones.

Una vez introducido todos estos cambios el diagrama ER quedaría como sigue (también se encuentra en SQL/ModeloERNuestraBD.png para una mejor calidad y claridad):



Apartado B (setPrice.sql):

Para calcular el precio de cada producto simplemente extraemos el año de la fecha actual y le restamos el año de fecha del pedido del producto. Eso nos dice la cantidad de años que ha pasado. Como cada año que pasa se incrementa el precio un 2%, si dividimos el precio actual por 1.02 elevado a la diferencia de años nos dará el precio en el momento que se compró. En orderdetail guardamos el precio INDIVIDUAL de cada producto (no guardamos el precio individual * cantidad).

Apartado C (setOrderAmount.sql):

Primero hacemos una query que agrupa las orderdetails por el id del pedido, y sumamos los precio*cantidad de cada orderdetail. De manera que cada una de estas sumas será el precio total sin impuestos del pedido (netamount). Una vez esta calculado, si multiplicamos por (1+tax/100) nos da el precio con los impuestos que se guardará en totalamount.

Apartado D (getTopVentas.sql):

Hacemos un join de orders, orderdetail y los productos para sacar las columnas necesarias de año, título y ventas. La función recibe dos parámetros, que será el intervalo de los años que queremos. Si hacemos SELECT * FROM getTopVentas(2017, 2020) nos dará las películas más vendidas en 2017, 2018, 2019 y 2020 ordenadas de mayor a menor por número de ventas.

Apartado E (getTopMonths.sql):

Primero agrupamos los orderdetail por oderid y sumamos las cantidades para así obtener el número de productos vendidos por order. Por último agrupamos las orders por año y mes sumamos todas las cantidades de las orders que se han vendido ese mes asi como del precio de todas esas orders, de manera que obtenemos los ingresos de ese mes y el número de productos vendidos. Si alguno de los dos son superiores a los argumentos que le pasamos entonces mantenemos la fila. Para llamar a la función usamos SELECT * FROM getTopMonths(320000, 19000), siendo el primer argumento el mínimo número de ingresos o importe y el segundo argumento el mínimo número de productos vendidos.

Apartado G (updOrders.sql):

El trigger se ejecuta cada vez que se actualiza el orderdetail, se crea o se borra, de manera que se actualiza el precio de orders. Si el orders se queda sin orderdetails, entonces borramos el order.

Apartado H (updInventory.sql):

El trigger se ejecuta cuando el carrito actual que es el está a null en status en orders, pasa a estar pagado ‘Paid’ . Entonces actualizamos el stock y las sales del inventory. Si el stock llega a 0 entonces creamos una alerta.

SQLAlchemy:

El código se encuentra en database.py comentado. Para el login y register ya no creamos las carpetas de los usuarios ya que guardamos y consultamos la información de la base de datos. Hemos hecho el json manualmente del catálogo ya que creíamos que había que coger un subconjunto de pelis de la BD. Mostramos en el catálogo las 6 películas más vendidas de los últimos 6 años que se muestra en la tabla. Ya que lo hemos implementado así solo podemos meter al carrito estas películas. Para el carrito hemos hecho que un usuario registrado si tenía un pedido en la base de datos sin pagar sea el carrito activo de ese momento, sino, se crea un nuevo pedido sin orderdetail. Por otro lado, cada vez que añadimos o eliminamos una película del carrito se modifica su correspondiente orderdetail. Cuando le damos a comprar, primero comprobamos que haya suficiente stock de todos los productos, además de todas las comprobaciones anteriores que hacíamos. Si se cumplen, cambiamos el estado de ese order a “Paid” y en el historial mostraremos todos los pedidos del usuario que tiene ya pagados.