INGENIERÍA INFORMÁTICA
Escuela Politécnica Superior
Universidad Autónoma De Madrid

PRÁCTICA 2

Práctica 2 Uso de Bases de Datos Flask

David Teófilo Garitagoitia Romero
Daniel Cerrato Sánchez

11/21/2021

Índice de Contenidos

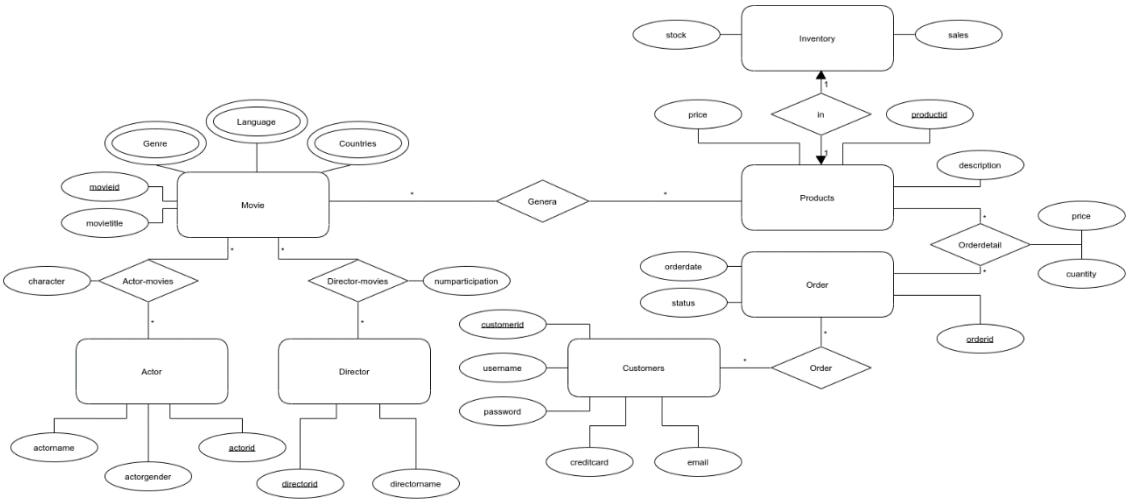
1. Introducción

En esta práctica el objetivo es expandir los conocimientos sobre bases de datos empleando PostgreSQL para dejar de emplear ficheros JSON como bases de datos y en su lugar emplear bases de datos relacionales.

2. Ejercicio 0

Lo primero que se nos pide es realizar un trabajo de ingeniería inversa sobre la base de datos para obtener el diagrama entidad relación.

Tras examinar con la terminal de postgre la base de datos, llegamos al siguiente diagrama resultado del proceso de ingeniería inversa:



2.1. Analizar la Base de Datos

Tras examinar la base de Datos y obtener su diagrama entidad- relación, procedemos a evaluar y analizar la base de datos. Entre otras cosas, se observa que la entidad customers no guarda ningún registro sobre lo puntos (“loyalty”), que se obtienen como recompensa al comprar productos, por lo que será necesario agregar dicho cambio. Este campo, empezará inicializándose a 0, a su vez, tampoco se guarda registro del dinero de los usuarios por lo que crearemos el campo “balance” para guardar dicha información, este se inicializará con un valor aleatorio. Además, por cómo está expresado, pueden existir múltiples customers con mismo username y password por lo que haremos únicos la unión de ambos campos, de forma que puedan existir varios usuarios con mismo username o usuarios con misma password pero nunca usuarios con misma password y username.

Los campos multi evaluados también los separaremos para crear entidades propias.

Este cambio generará las entidades genre, language y countries.

Dentro de Orders, se crea customerid clave foránea para customers.

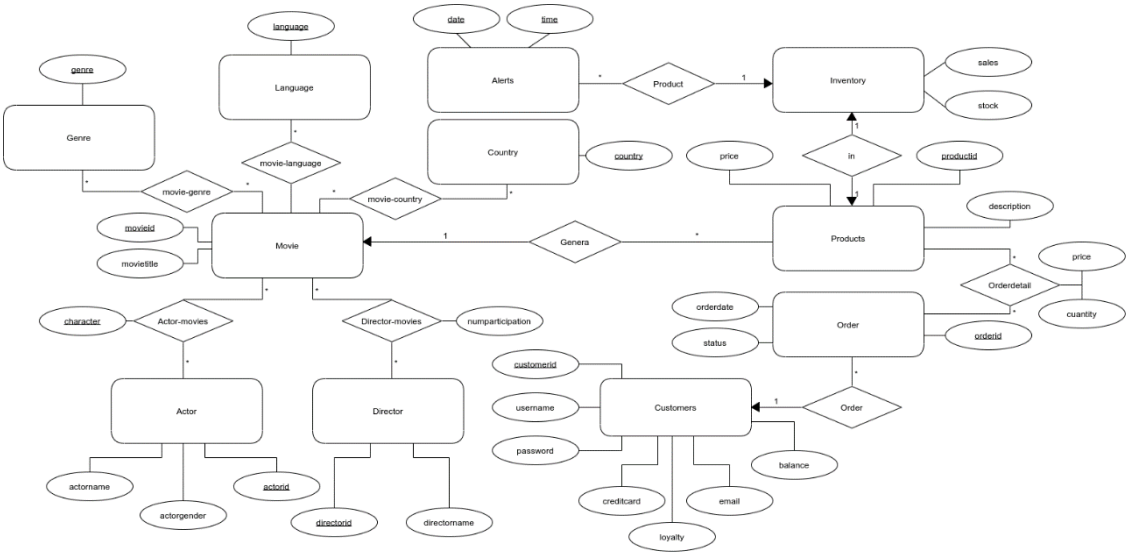
En actor-movies será necesario añadir clave primaria compuesta (actorid, movieid, character)

Por su parte para director-movies cambiaremos a clave primaria compuesta (directorid, movieid).

En orderdetail, orderid y prod_id los emplearemos como claves foráneas y orderid, prod_id como clave primaria.

Por último, crearemos la tabla alerts con campos producto, fecha y hora y un trigger de cuando stock llega a 0

Tras estos cambios llegamos al siguiente diagrama:



3. Procedimientos y consultas SQL

Pasamos al siguiente apartado de la práctica donde se nos pide elaborar una serie de consultas y procedimientos SQL

3.1. actualiza.sql

En este archivo, generamos los cambios presentados en el apartado anterior sobre la base de datos.

3.2. setPrice.sql

En este otro archivo, actualizamos el precio de todos lo productos de orderdetail en base al año de compra con relacion al primer año que se crearon, aumentando en un 2% el precio cada año de diferencia.

3.3. setOrderAmount.sql

Aquí creamos una función que actualice los campos de netamount y totalamount de la tabla orders, calculando la suma total de los precios obtenidos de cada producto multiplicado por su cantidad en cada pedido.

3.4. getTopSales.sql y getTopActors.sql

getTopSales.sql es un archivo que contiene una función con la que obtendremos las películas con mayores ventas de productos por cada año entre dos años pasados por argumento.

Esto se ha realizado en tres partes:

1. Crear la tabla salesPerYear: donde conseguimos calcular el numero de ventas por película y año. Es necesario crear una tabla temporal puesto que vamos a utilizar esta misma query dos veces. De esta forma, ganamos en rendimiento.

2. Crear la tabla topSales: donde conseguimos el propósito de esta función. Generamos una tabla para que el bucle de a continuación no quede muy engorroso.

3. Usar un bucle: para sacar por argumento todas las tuplas obtenidas tras la query.

getTopActors.sql es un archivo que contiene una función con la que obtendremos información sobre los actores que han participado en más películas de un género que se pase por argumento, la/s primera/s película/s que hicieron para ese género y el/los director/es que la/s dirigieron.

Este proceso se ha realizado en 4 partes:

1. Crear la tabla genreMovies: donde conseguimos todas las películas del género y su año de estreno. Es necesario crear una tabla temporal puesto que usaremos esta query en varias ocasiones posteriormente, ganando así algo de rendimiento.

2. Crear la tabla actorsGenreDebut: donde conseguimos los actores, el número de películas en las que ha participado del género y el año de debut en este. Esta vez, hemos creado la tabla para una mejor visualización de la query total.

3. Crear la tabla topActors: donde cumplimos finalmente el requisito de esta función. De nuevo se ha generado una tabla temporal auxiliar para que el bucle que viene a continuación no quede demasiado cargado de código.

4. Usar un bucle: para sacar por argumento todas las tuplas obtenidas tras la query

4. Triggers

- **updOrders:** Este trigger actualiza los campos netamount y totalamount de la tabla orders cuando se inserta, se elimina o se actualiza algo en el carrito de la compra, véase, en orderdetail.

  Gracias al id del pedido modificado podemos actualizar en la tabla orders, los datos correspondientes, estos son netamount (con la suma de los precios de los productos del pedido multiplicados por su cantidad) y totalamount (aplicando un impuesto marcado en el pedido)

- **updInventoryAndCustomer:** Este otro trigger es la unión de varias sub-funciones que se llevan a cabo cuando se actualiza el status de un pedido a 'Paid' (pagado).

  Primero actualizamos la fecha del pedido a la fecha en la que se realiza el pago de la compra.
  Después, creamos una tabla temporal auxiliar con los ids de los productos comprados y sus cantidades. Esta tabla la hemos generado porque vamos a usar esta query en dos ocasiones posteriormente, mejorando algo el rendimiento del trigger.
  Luego, actualizamos el inventario retirando del stock la cantidad de productos comprados y añadiéndolos al campo sales.
  Seguidamente, comprobamos si alguno de estos productos se ha quedado con el stock vacío, en cuyo caso añadiremos una alerta a la tabla de alertas creada en actualiza.sql con el id del producto agotado y la fecha y hora del momento de agotamiento.
  Finalmente, actualizamos los puntos de fidelización y el saldo de dinero del cliente que ha realizado el pedido.

5. Integración en el portal

Para integrarlo en el portal lo primero será eliminar todo lo que teníamos empleando los ficheros JSON.

Como ahora el contenido es demasiado grande y por tanto la carga pesada y poco user-friendly emplearemos paginación, simplemente mediante un offset y un limit (preestablecido) iremos recortando los resultados para que se muestren en pequeñas porciones, al pulsar el botón de next, el offset se incrementará y así es como implementamos una paginación simple, si el offset es superior a 0, quiere decir que existe una página previa por lo que podemos mostrar el botón de prev, por otro lado si la suma del offset y el límite es inferior a la suma de resultados posibles quiere decir que aún existen páginas superiores por lo que podemos mostrar el botón de next.

Con respecto a los filtros simplemente cuando pinchemos en uno de los filtros la paginación se reinicia, el número de resultados es recalculado para el género de película seleccionado y las películas mostradas son aquellas que cumplan las condiciones de búsqueda (género o en su caso también título), el título a buscar se guarda en una variable global la cual únicamente será modificada cuando se envía una nueva string en el formulario para preguntar por un título, de esta forma al buscar por un título y mirar en una categoría, los resultados mostrados son los que coinciden con ese título y categoría, en ese mismo formulario, creamos el campo para preguntar por el número de actores de getTopActors que se quiera mostrar, de esta forma, para obtener por ej. Los Y primeros resultados de topActors para el género X, bastará con filtrar por género seleccionando dicho género en el menú lateral y posteriormente introducir en el formulario el número Y en el campo de número de actores.

De esta forma todos los datos mostrados son relativos a la categoría seleccionada en el filtro.

Para dicho menú lateral, al existir muchos géneros y poder agregarse más se utiliza una consulta básica para obtener los distintos géneros, estos son pasados a jinja y finalmente se crea el menú lateral con todos los géneros distintos.

Al pulsar en cualquier película, se redirige a la página de la película, en la cual se muestra esta a detalle, para esta página requerimos consular por los productos de la película para poder seleccionar y comprar uno, además consultaremos sus actores , todos estos resultados se obtienen de consultas de database.py en los que empleando el id de la película (que se manda como argumento en el url_for) obtenemos los datos de dicha película.

Al decidir comprar una película, está se añade a la sesión de compra en caso de estar logeado, en caso de no estarlo, se emplea el carrito de la base de datos.

Para saber si estas o no logeado existe un valor en sesión que actúa como diccionario donde se guardan, su id (el cual emplearemos para consultas en la base de datos como puede ser obtener su carrito), su dinero y su username (que se emplea para que se pueda mostrar en todo momento en la página web)

Si en sesión no se encuentra ningún campo para usuario, quiere decir que aún no se ha logeado y por tanto se empleará la sesión para guardar la información del carrito.

En la sesión guardaremos el nº de veces que se añade cada producto junto con su id de producto para que se pueda calcular de esta forma el precio total en el carrito y se puedan añadir o quitar películas.

Para añadir o quitar simplemente si estamos logeados, buscaremos si tenemos un orderdetail para el carrito para el producto que queremos quitar o añadir, si es el caso, añadiremos o restaremos la quantity, si esta llega a cero simplemente deletearemos dicho orderdetail.

Para un usuario que no está logeado simplemente cambiaremos los campos de la sesión como hacíamos en iteraciones anteriores.

Para la página de regitro, se sigue una dinámica similar a la empleada con el menú lateral, solicitamos los campos obligatorios (aquellos que no pueden ser NULL) y que no tengamos explícitos con sus comprobantes como pueden ser password o email.

Con esos datos comprobaremos que no exista ningún usuario con mismos credenciales (como ya se expuso anteriormente, el campo de password y customername son únicos), en caso de ser así mostraremos un mensaje para indicar que el usuario ya existe, si todo es correcto simplemente redirigiremos a login para que pueda iniciar sesión con el usuario recién creado.

La página de usuario se ha mantenido muy similar a la práctica 2, con un formulario simple que permita al usuario ingresar dinero, y tablas por cada uno de los productos adquiridos donde se muestra la imagen (con un link a la página de la película), el título de la película, precio y descripción del producto adquirido (“Estándar”, “Gold”…)

Con estos cambios, conseguimos integrar la nueva base de datos en el portal de la iteración anterior.

[FINAL DE DOCUMENTO]