Práctica 3 Sistemas Informáticos I

Juan Riera Gomez, Luis Cárabe Fernández-Pedraza

November 28, 2018

1 Introducción

En esta práctica hemos continuado con nuestra aplicación de venta online de películas *Peordede*, a la cual hemos cambiado el funcionamiento interno, añadiendo el acceso a una base de datos PostgreSQL, sustituyendo la funcionalidad anterior que obtenía los datos de ficheros json y distribuciones de directorios.

2 Descripción de los archivos de entrega

Se entregan varias carpetas y archivos.

- Static: contiene todo aquello necesario para construir los html's, a parte de los templates html en sí mismos. Esto incluye los archivos css, las imágenes y los archivos javascript, todo ello organizado en subcarpetas.
- Templates: contiene las directrices que utilizará flask para construir el html de respuesta a las peticiones, así como las bases del html en sí mismo.
- Actualiza.sql: fichero con las intrucciones SQL para actualizar la base de datos que nos daban en moodle al modelo ajustado a nuestras necesidades.
- ConfirmaCompra.sql: fichero que contiene la funcion auxiliar que utilizamos para confirmar las compras de los usuarios.
- CreateCarrito.sql: fichero en sql que contiene una funcion auxiliar utilizada para crear el carrito dado un usuario.
- Diagrama.png: diagrama entidad-relacion de nuestra basde de datos tras las ediciones pertinentes
- getTopMonths.sql: fichero sql que contiene el código de la función getTopMonths() descrita en el enunciado.
- getTopVentas.sql: fichero en sql del código de la ufncion descrita en el enunciado getTopVentas().
- init.sh: script en bash que se puede ejecutar en los ordenadores docentes, y que debería funcionar bien en cualquier ordenador linux con los paquetes pertinentes instalados y con un usuario creado llamado alumnodb con contraseña alumnodb, que prepara la base de datos

para el funcionamiento de la aplicación y que borra la base de datos anterior, si existía, llamada *si1*. Tarda algo más de un minuto en ejecutarse. NOTA IMPORTANTE: este script supone la existencia, en la misma carpeta de ejecución, del fichero llamado *dump_v1.2.sql.gz* que nos daban en el enunciado, este fichero no lo hemos podido incluir en la entrega porque era demasiado pesado para subirlo a moodle. Las instrucciones de ejecución se explican en más detalle más adelante.

- Peordede.py: es nuestra aplicación en flask, este es el archivo a ejecutar para iniciar nuestra aplicación.
- Peordede5.wsgi: es nuestra aplicación wsgi, que debe ser la ejecutada en caso de estar funcionando dentro de un servidor Apache 2.
- setOrderAmount.sql: fichero que contiene la función setOrderAmount() descrita en el enunciado.
- setPrize.sql: fichero que contiene la función setPrize() descrita en el enunciado.
- updInventory.sql: fichero en sql con el código de la función updInventory descrita en el enunciado.
- updOrders.sql: fichero que contiene el código sql de la función updOrders descrita en el enunciado.

3 Instrucciones de arranque del servidor en condiciones normales

Ya vaya a funcionar con apache o en modo *standalone*, para que la aplicación funcione sin errores, se debe de ejecutar antes el script *init.sh*, que supone que está todo en la posición en la que se ha realizado la entrega, es decir, que dentro de la carpeta de la entrega no hemos movido nada de directorio. Además el script supone que la arquitectura de usuarios se corresponde con la de los laboratorios docentes, o que al menos tiene un usuario llamado *alumnodb* que tiene como contraseña *alumnodb*. Por otro lado, el script supone que postgreSQL está correctamente instalado en el ordenador. Si alguno de estos requisitos no se satisface, o si el script da algún error, mirar la siguiente sección en la que se detalla métodos de arranque alternativos.

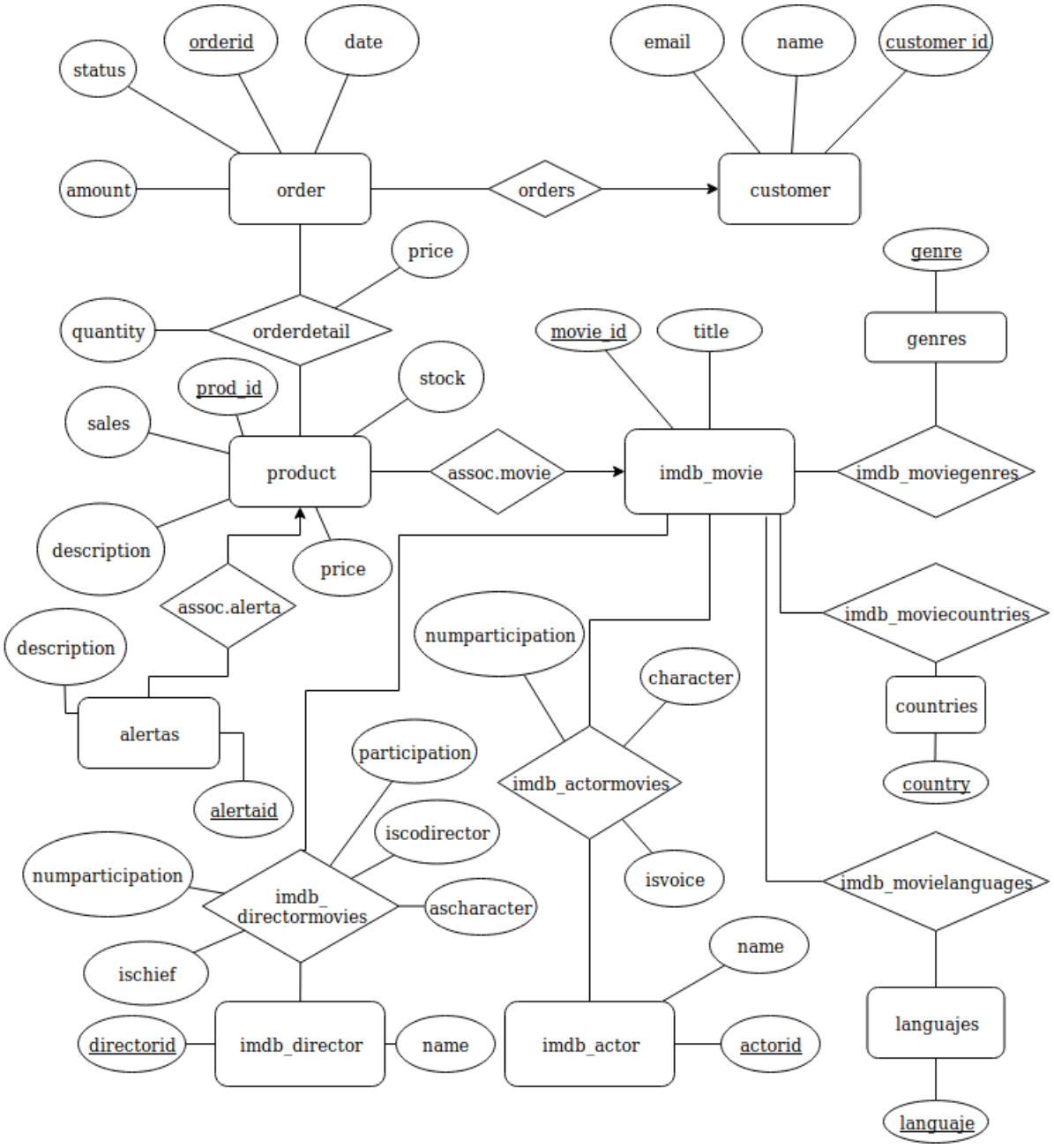
- Descargarse el archivo *dump_v1.2.sql.gz*, que puede descargarse de moodle
- Meter el archivo mencionado anteriormente en la carpeta raíz de la entrega, es decir, en el mismo directorio en el que se encuentre init.sh
- Ejecutar el comando *./init.sh* y esperar hasta que termine la ejecución.
- Cuando termine la ejecución del script se habrá eliminado la base de datos anterior, si existía, y se habrá sustituido por la nueva base de datos *si1* lista para funcionar.
- Para arrancar el servidor en local, ejecutar simplemente *python peordede.py*

4 Instrucciones alternativas de arranque del servidor

Si alguno de los requisitos no se satisface o si el script da algún error comprobar lo siguiente:

- Si el problema es que el archivo *dump_v1.2.sql.gz* ya se ha ejecutado y no se quiere que el script lo ejecute de nuevo, simplemente se puede comentar o borrar esa línea del script, es decir, añadir un símbolo # al inicio de la línea que ahora comienza con *gunzip...*
- Si el error viene porque no existía una base de datos llamada *si1*, no hay ningún problema, el script seguirá adelante.
- Si el error viene porque falta algún paquete, cosa que no debería ocurrir en los laboratorios docentes, instalarlo y volver a ejecutar el script.
- Si el error viene porque se está utilizando otra arquitectura sql, otro usuario distinto, o algo por el estilo se debe descomprimir dump_v1.2.sql y ejecutarlo, así como todos los demás scripts sql continuando con actualiza.sql.

5 Diagrama entidad-relación



Como se aprecia en el diagrama, hemos editado bastantes cosas sobre el diseño inicial que nos daban en moodle. Detallamos aquí los cambios principales que se han realizado sobre la base de datos que nos daban:

- En la tabla *customers* hemos añadido las restrucciones de que el email no pueda ser NULL ya que no nos interesa tener usuarios sin email, La columna income, que hemos utilizado como el saldo de cada usuario, sea de tipo numeric en vez de integer por si los usuarios añaden o gastan números no enteros de saldo. Hemos añadido algunos valores por defecto en algunas columnas que en nuestro formulario de inicio de sesión son campos obligatrios, pero que al poblar la base de datos no se rellenan. Por otro lado hemos eliminado las columnas address2, state, region, phone, age y gender porque no las usamos y no las consideramos importantes. Concretamente state y region no las hemos utilizado porque nuestra página es para España.

- Hemos fusionado la tabla inventory y products en una sola, ya que por nuestra implementación

estaban representando la misma entidad.

- Hemos creado una secuencias para que las claves primarias de products se generen automáticamente.
- En la tabla orderdetail, hemos añadido las restricción de que prod_id y orderid sean claves extranjeras ya que son claves primarias de sus respectivas tablas. También hemos añadido valores por al precio y a la cantidad cuando estas sean NULL y hemos añadido la restriccion de que no puedan serlo más, ya que un detalle de una order no tiene sentido que no tenga cantidad o precio.
- En la tabla imdb_actormovies que relaciona los actores con las películas en las que aparecen, hemos añadido la restriccion deque movieid y actorid sean claves extranjeras ya que son claves primarias de sus respectivas tablas.
- Hemos añadido la columna 'poster' a la tabla de películas, ya que deberían de tener una imagen.
- Hemos poblado las tablas languages y countries que contienen los idiomas y los países de las películas respectivamente.
- A la tabla genres le hemos añadido la restricción de que genrid sea clave primaria, y la hemos puesto en imdb_moviegenres como clave extranjera.
- Hemos creado la tabla alertas, que contiene un identificador como clave primaria que va aumentando con una secuencia, un identificador de producto como clave extranjera y una descripción.
- Por ultimo, reajustamos las secuencias, ya que al crearlas más tarde que las claves que se vaían de ellas en unos casos, y al poblar la tabla indicando los valores de estas claves en otros, quedaban desajustadas.

6 Análisis y/o resultados de las consultas pedidas en el enunciado

6.1 actualiza.sql

En este fichero se llevan a cabo los cambios enumerados en la sección anterior.

6.2 setPrice.sql

Este script se encarga de actualizar el precio de las filas de orderdetail con el correspondiente a su año, teniendo en cuenta que ha ido aumentando un 2% anual. Esto quiere decir que por cada año se iban multiplicando por 1.02 los precios.

De esta manera, el procedimiento mira el precio del producto, y lo divide por 1.02 tantas veces como años hayan pasado desde la venta. Este es el precio que guarda en orderdetail.

6.3 setOrderAmount.sql

Esta función actualiza la tabla orders para que el campo totalamount sea correcto, es decir, sea igual a netamount sumándole los impuestos, que es el resultado de dividir el campo tax entre 100 y multiplicarlo por netamount.

Para facilitar todo esto nos hemos valido de una view, que hemos llamado ordersWithPrices, que es una tabla que contiene todos los orderid con la suma de los precios de sus orderdetail's asociados.

6.4 getTopVentas.sql

Esta funcion recibe un año y devuelve para ese año y los siguientes la película que más se ha vendido.

Para ello se vale de una view llamada dateMovieQuantity que contiene tres columnas: un año, el nombre de una película para ese año y el número de copias de esa película que se vendieron.

Cuandos e ejecuta la función, simplemente selecciona de dateMovieQuantity los años superiores al de entrada, mira el máximo de ventas de cada año y el resultado lo cruza mediante producto cartesiano nuevamente con dateMovieQuantity para obtener los nombres de las peliculas con máximo de ventas de cada año.

He aquí una captura del resultado:

```
si1=# select getTopMonths(19000, 320000);
                    gettopmonths
-------------------------------------------------
 (2014,"October  ",19086,261821.39673307575004)
 (2015,"May      ",19001,266805.37651431195736)
 (2014,September,19133,261099.41153263602322)
 (2016,"July     ",19280,276240.96501345634130)
(4 rows)
```

6.5 getTopMonths.sql

Script que contiene el código de la funcion getTopMonths que recibe como argumentos dos números enteros y devuelve, junto con su importe y productos, los meses y años en los que se ha superado uno de los dos umbrales, económicamente (primer argumento) o en número de productos (segundo argumento)

Para su implementación, ejecutamos una consulta que pida el año y el mes con las sumas de las cantidades de articulos vendidos y de los precios, y esta consulta la anidamos en otra que compruebe que solo nos quedamos con los precios o cantidades que superene los umbrales.

He aquí una captura del resultado:

```
si1=# select getTopVentas(2000);
                gettopventas
-----------------------------------------
 (2012,"Male and Female (1919)",9)
 (2013,"No Looking Back (1998)",101)
 (2014,"Love and a .45 (1994)",136)
 (2015,"Illtown (1996)",142)
 (2016,"Wizard of Oz, The (1939)",134)
 (2017,"Life Less Ordinary, A (1997)",134)
 (2018,"Gang Related (1997)",57)
 (2018,"Jerk, The (1979)",57)
 (2018,"Life with Mikey (1993)",57)
(9 rows)
```

6.6 updOrders.sql

Este script sql crea una triggers asociados a la tabla orderdetail que mantiene actualizado el carrito, presente en la tabla orders.

En cuanto a la implementación, realmente de orderdetail solo se añaden o eliminan filas cuando se añaden o eliminan películas del carrito. Por tanto, cada vez que se edita en una fila de orderdetail para aumentar la cantidad se ejecuta una función updAddOrders que resta el precio viejo a netamount y suma el nuevo. Cada vez que se añade una fila nueva a orderdetail se ejecuta una función AddOrders, que añade el nuevo precio a la fila de orders correspondiente. Análogamente ocurre con la eliminación de filas.

6.7 updInventory.sql

La misión de este Trigger es realmente sencilla, simplemente actualiza el producto una vez se confirma una compra, es decir cada vez que una fila de orders actualiza su atributo status a 'Paid', resta el 'stock' de los productos asociados a esa compra y suma la misma cantidad en 'sales', como es lógico.

A parte, cuando un producto tiene stock 0, crea una alerta para ese producto avisando de que no hay stock.

7 Otras mejoras y decisiones de implementación

En nuestra página principal se muestran las películas mas vendidas. Dado que esta consulta es muy pesada hemos creado un thread, que almacena el resultado de la consulta en una variable global, de donde tomamos los datos para construir la página principal. Este mismo thread se encarga de refrescar la variable una vez cada minuto.

8 Conclusiones

En esta práctica hemos aprendido las bases de sql y sqlalchemy y nos hemos enfrentado a problemas reales de eficiencia e integridad de bases de datos. Hemos aprendido sobre el uso de triggers, funciones y queries sql en un entorno de PostgreSQL.