Sistemas informaticos 1

Practica 3

Cambios realizados en el fichero actualiza.sql:

Hemos creado 3 tablas adicionales como se pide en el enunciado, con el fin de eliminar atributos multievaluados(Countries, Genres, Languages). Todas ellas se componen de los campos id y nombre. Siendo la clave primaria el id.

Una vez hechas estas 3 tablas cambiamos las tablas imdb\_moviecountries, imdb\_moviegenres, imdb\_movielanguages, en las que se ha cambiado el campo correspondiente por una referencia a las tablas anteriores. Ademas, imdb\_movielanguages tiene un campo adicional, extrainfo, a la que se le han modificado las restricciones permitiendo NULL. Este diseño podria mejorarse debido a que es preferible evitar los campos con NULL.

En lo relacionado a actormovies: hemos eliminado de la clave primaria numartici pation siendo la clave primaria resultante la combinacion de actorid y movieid. Estos campos han pasado a ser Foreign keys a la tablas imdb\_actors y imdb\_movies respectivamente. Por ultimo, se permiten NULL en la columna asCharacter.

EL único cambio en imdb\_actor hemos optado por reducir el campo gender a una única letra, M o F para ser consistentes con la tabla customers.

En la imdb\_directormovies se elimina de la primary key el campo numparticipations. De esta forma la clave primaria queda (movieid , directorid) que son referencias a la tablas imdb\_movies y imdb\_directors.

Los cambios realizados en imdb\_movies se reducen a eliminar la restriccionde NOT NULL movierelease.

En la tabla orderdetail faltaba la clave primaria. Consideramos que deberia ser (orderid, prod\_id) pero, si la utlizamos, tenemos claves primarias duplicadas. Para solucionarlo realizamos una query de las claves primarias repetidas agrupandolas por (orderid, prod\_id). De esta forma conseguimos actualizar la tabla en la que no tenemos duplicados manteniendo la informacion inicial. Se han añadido dos referencias a orders(orderid) y products(prod\_id).

En orders se ha añadido la clave extrangera customerid. Ademas la secuencia de orderid estaba en 1, y debido a la necesidad de insertar nuevos pedidos le hemos asignado el valor maximo de orderid.

En customers , se ha eliminado la restriccion de NOT NULL en todos los atributos no utilizados por nuestra aplicación. Se ha añadido NOT NULL en email e income. Ademas email a pasado a ser UNIQUE debido a que sera usado para identificar a los usuarios de nuestra tienda. El campo income ahora es de tipo NUMERIC ya que servira como saldo de los clientes de la aplicación.Ademas hemos cifrado el campo password en md5.

Al igual que en la tabla orders la secuencia del id de los usuarios estaba a 1 y la hemos puesto en su valor maximo.

Hemos creado 2 tablas adicones:

alertas, con campos prod\_id y mgs. Siendo la clave primaria la combinación de los mismos. Actualmente solo se considera el caso ‘producto agotado’ pero podrian existir distintos tipos de alerta para un mismo producto. Inicialmente se rellena con todos aquellos productos cuyo stock esta a 0.

topventas, esta tabla se crea con el fin de minimizar el tiempo de ejecución del procedimiento almacenado getTopVentas solicitado en el enunciado. Dicho procedimiento tardaba bastante tiempo en ser ejecutado (varios segundos), pero debido a que tendría escasa actualización hemos decidido añadirla a actualiza.sql.

SETPRICE.SQL:

Conseguimos la diferencia de años desde la venta de la pelicula hasta hoy. Tras esto dividimos el precio actual entre 1,02 elevado a la diferencia de años. Actualizamos el precio con el resultado obtenido por la cantidad de productos en cada venta.

SETORDERAMOUNT.SQL:

Conseguimos los pedidos con netamount or totalamount a NULL y los completamos con sus correspondientes valores.

En el caso de netamount tomamos la suma de los precios de los diferentes productos.

Mientras que totalamount actualizamos con netamount\*(taxes/100).

GETTOPVENTAS.SQL:

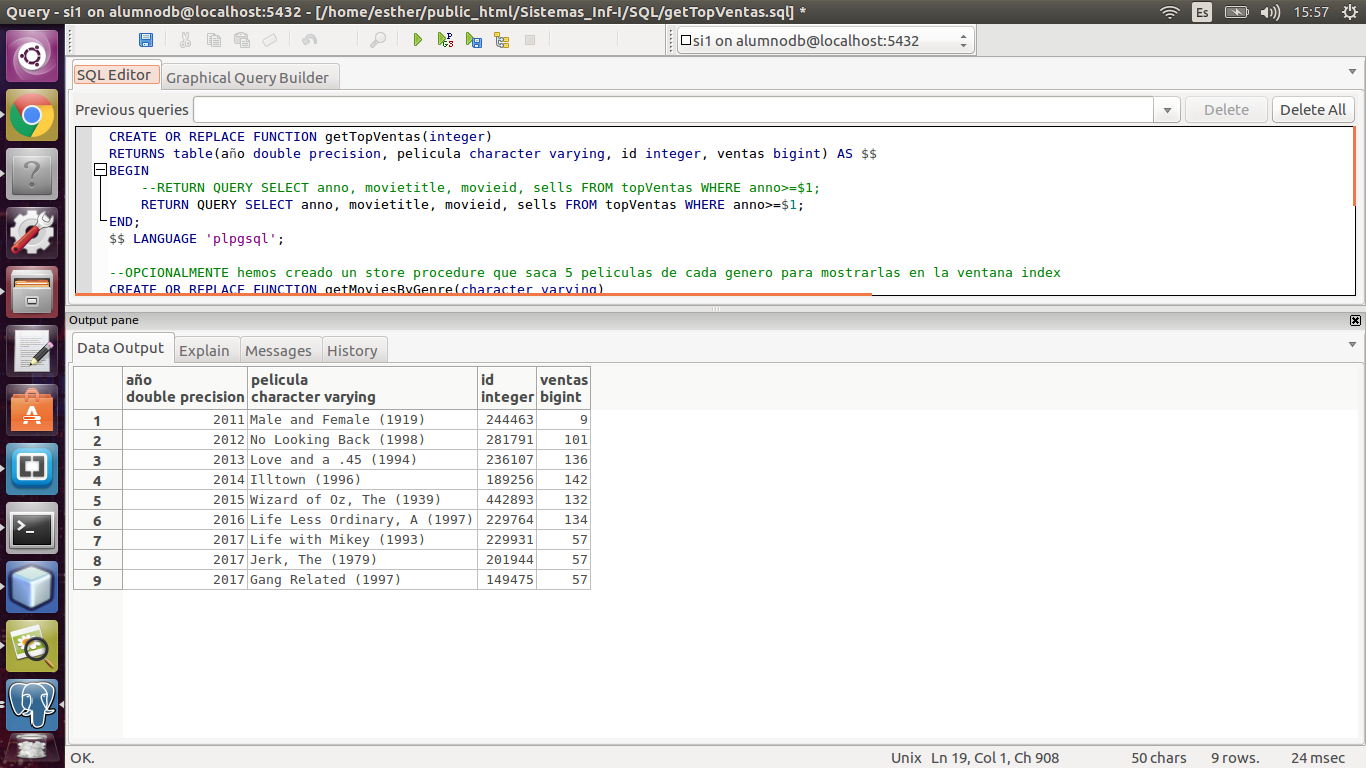
Debido a que hemos creado una tabla donde ya almacenamos los productos mas vendidos. El producto almacenado se reduce a la búsqueda de los registros en los que el año sea mayor o igual al pasado como argumento.

Ademas destacamos que en la tabla creada hemos añadido el id de las películas, para simplificar la programación de la pagina web.

Para la creación de la tabla agrupamos el JOIN de (orders, orderdetail, products) por fecha y película, sumando las cantidades de las ventas. Sobre este resultado seleccionamos el máximo.

Somos conscientes de que se realizan 2 consultas muy parecidas, pero no hemos sido capaces de obtener una solución mas optima. Una de las opciones que intentamos fue la creación de un procedimiento almacenado en PLPGSQL con instrucciones de control de flujo pero el resultado, a pesar de ser el mismo mostraba un peor rendimiento.

Nótese que en caso de tener para varias películas el mismo numero de ventas mostramos todas.



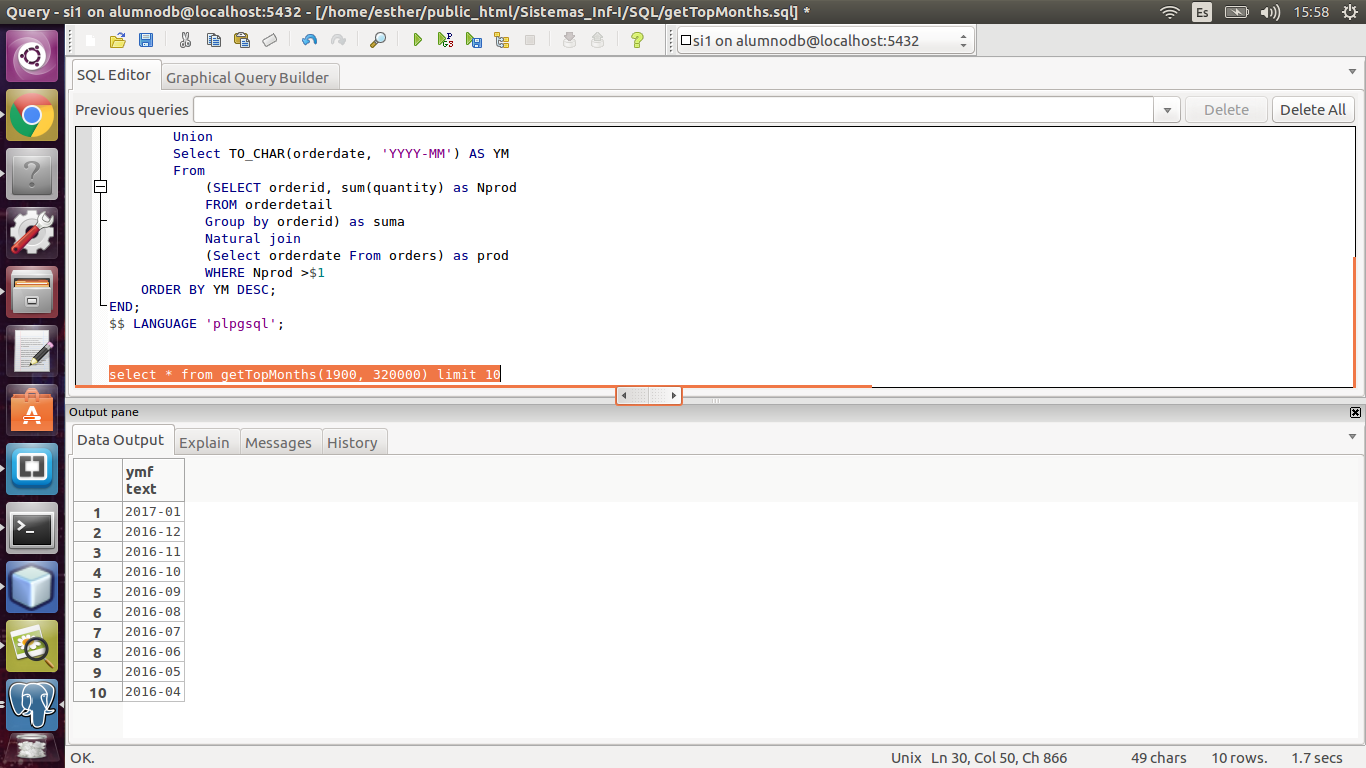
GETTOPMOUTHS.SQL:

Este procedimiento almacenado puede entenderse como la unión de dos consultas distintas.

La primera de ellas devuelve los meses-años en los que los ingresos obtenidos superan el argumento pasado a la función. Para ellos simplemente agrupamos por a mes-año y realizamos la operación de suma de totalamount.

La segunda devuelve los meses-años en los que el numero de productos vendidos supera el umbral pasado como parámetro. Para ellos simplemente sumamos el numero de ventas de ese producto a lo largo de ese mes.

Es importante notar que la unión de dos consultas debe tener las mismas columnas, condición que se cumple. Ademas notamos que la operación UNION opera como conjuntos, de manera que no aparecerán resultados repetidos.



UPDORDERS.SQL:

Este trigger realiza operaciones distintas en funcion de la operación realizada sobre orderdetail.

Caso INSERT:

Actualizamos el carrito con el nuevo objeto que se desea comprar, sumando su valor correspondiente.

Caso UPDATE:

Al añadir o quitar uno de los elementos del carrito se actualizan el precio del carro.

Caso DELETE:

Eliminamos el producto seleccionado del carro, restando el precio del carrito.

UPDINVENTORY.SQL: