Eva Lacaba
Rodrigo Lardiés

PRACTICA 2: Programación Web y Bases de Datos

Explicación de actualiza.sql:

A continuación enumeramos los distintos cambios que hemos realizado en la base de datos, y los explicamos:

- **Actualizamos los seriales de customers, products y orders**

Los id's "customerid", "orderid" y "prod_id" son secuenciales en nuestra base de datos, y queremos que cuando se añada alguno de ellos a la base de datos, estos se incrementen en uno respecto del último id presente en la base. Es por ello que realizamos una actualización de estos por una posible pérdida de secuencia para asegurarnos que el siguiente id añadido sea correcto. Para ello usamos además "lock table" para que mientras estemos actualizando estos id's no se produzcan otras transacciones.

- **Cambios en las tablas**

Decidimos borrar algunas de las columnas de la tabla *customers* que no nos eran útiles para el desarrollo de nuestra aplicación, como *address1, address2, city, state, zip, country, region, phone, creditcardtype, creditcardexpiration, age* e *income.* Todos estos campos serían necesarios si hubiésemos implementado una pasarela de pago, pero no es el caso. El campo *income* nos es completamente indiferente en cualquier caso.

Además en esta tabla hemos añadido el campo de saldo, que necesitábamos para nuestra aplicación y que hemos inicializado a 0; y también hemos modificado el campo de email para que este no pueda ser null (aunque en nuestra aplicación ya obligamos al usuario a poner un email válido).

También hemos modificado la tabla *imdb_directormovies* donde hemos eliminado las columnas *ascharacter* y *participation, la* tabla *imdb_movielanguages* de donde hemos eliminado la columna *extrainformation,* y la tabla *imdb_actormovies,* de la cual hemos eliminado las columnas de *ascharacter, isvoice, isarchivefootage, isuncredited* y *creditsposition.* Todos ellos son irrelevantes para la implementación de nuestra página pues estos datos no son útiles para el usuario que compra la película, por lo que no los mostraremos.

- **Creación de nuevas tablas**

Creamos tres tablas nuevas: *countries, genres* y *languages.*
En ellas guardaremos los distintos países, géneros e idiomas de las películas con los id's asignados a cada uno. De esta manera evitamos que estén repitiéndose constantemente en las tablas los países, los géneros y los idiomas.

Para lograrlo creamos la tabla y después utilizamos "insert" y "select distinct" para insertar en nuestra nueva tabla los países, géneros e idiomas que sean distintos de su respectiva tabla antigua.

Por último para enlazar las tablas nuevas con las antiguas tendremos que actualizar las tablas antiguas, cambiando ahora los países, géneros e idiomas por sus id's correspon-

dientes, así como cambiar el tipo de esta columna nueva que ahora será un entero, y añadir las FK correspondientes (que lo haremos un poco más adelante).

- **Establecer primary keys**

Establecemos las primary key de aquellas tablas que no tengan, como por ejemplo *imdb_actormovies* cuya primary key estará compuesta de *actorid,movieid* y la tabla *imdb_movielanguages* que tendrá como primary key *movieid, language.*

- **Cambios en la tabla orderdetail**

Observamos que al intentar crear la PK de esta tabla con la tupla (orderid, prod_id) nos daba error por duplicado. Esto se debía a que había dos entradas en la tabla con mismo *orderid* y *prod_id* con cantidades distintas. Lo que debería ocurrir es que las cantidades se sumasen, por lo que creamos una nueva tabla que hiciese esto.
De esta forma en nuestra nueva tabla, agrupábamos por la tupla mencionada anteriormente y sumábamos las cantidades de todas las filas que encontrase con la misma tupla.
Renombramos ahora esta nueva tabla como la nueva *orderdetail* y añadimos ahora la primary key.

- **Añadimos las FK necesarias**

Observamos que en la base de datos faltaban ciertas relaciones entre tablas, luego también añadimos las foreign keys que faltaban para relacionar nuestras tablas en la base de datos.
Por ejemplo, *imdb_actormovies* está relacionada con *imdb_actors e imdb_movies.*

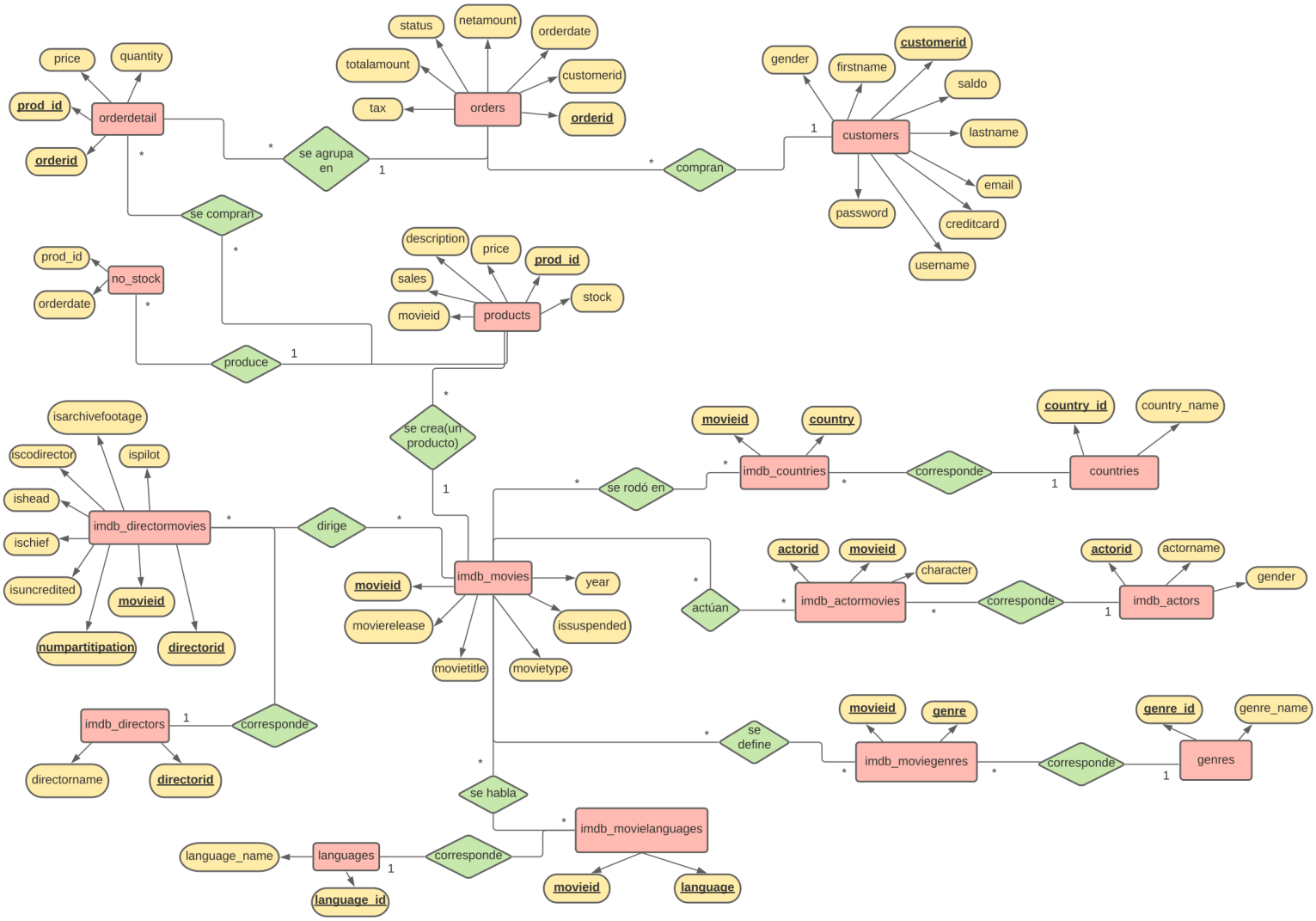
- **Fusión de las tablas inventory y products**

Decidimos fusionar estas dos tablas porque no vimos ninguna ventaja en tenerlas por separado, y nos parecía visualmente más fácil tener toda la información de un producto en una misma tabla, ya que añadir dos columnas más (stock y sales) no hacía que la tabla fuese demasiado extensa. Dado que había productos que podían no estar en inventory decidimos hacer un left join para asegurarnos de no perder nada de información.
Una vez fusionadas las dos en una nueva tabla, simplemente añadimos las PK y FK correspondientes, así como actualizamos los campos a not null o a cero según como estaba en nuestras tablas antiguas.

- **Creación de alert**

Creamos una tabla llamada *no_stock* para alertar de la falta de stock de una película con los campos prod_id y orderdate. Decidimos no ponerle una primary key por si se daba la situación de que un mismo producto se agotase en la misma fecha. La fecha la añadimos con el fin de saber qué películas son las más populares, que podría usarse (como propuesta) para colocar las películas más populares del momento al inicio de nuestra página.

DIAGRAMA E-R


price
quantity
prod_id
orderdetail
orderid
se agrupa en
status
netamount
orderdate
totalamount
customerid
tax
orders
orderid
compran
gender
firstname
customerid
saldo
customers
lastname
email
password
creditcard
username
se compran
prod_id
no_stock
orderdate
description
price
prod_id
sales
stock
movieid
products
produce
se crea(un producto)
isarchivefootage
iscodirector
ispilot
ishead
imdb_directormovies
ischief
isuncredited
movieid
numpartitipation
directorid
dirige
imdb_directors
corresponde
directorname
directorid
movieid
imdb_countries
country
se rodó en
corresponde
country_id
country_name
countries
movieid
imdb_movies
year
movierelease
issuspended
movietitle
movietype
actúan
actorid
movieid
imdb_actormovies
character
corresponde
actorid
actorname
gender
imdb_actors
se define
movieid
genre
imdb_moviegenres
corresponde
genre_id
genre_name
genres
se habla
imdb_movielanguages
movieid
language
corresponde
language_name
languages
language_id

Explicación de las consultas:

setPrice.sql:

Para esta consulta, realizamos un "*update*" de la tabla *orderdetail* y actualizamos el precio de todas las películas. Para ello obtenemos la diferencia entre el año actual y el año del pedido, ya que se ha incrementado un 2% anualmente. A continuación solo tenemos que realizar la siguiente operación sobre el precio actual del producto:

setOrderAmount.sql:

Para esta consulta, creamos la función setOrderAmount(). Ésta se encargará de actualizar la tabla *orders*, obteniendo el *netAmount* y el *totalAmount* de cada pedido. Para ello obtenemos todos las filas de la tabla *orderdetail* que cuyo *orderid* coincida con el de la tabla *orders*. Una vez tenemos estas filas, basta con sumar el producto de (*price* * *quantity*) para todas las filas y obtenemos el *netAmount*. Para el *totalAmount* basta con añadir el porcentaje de impuestos (*tax*) correspondiente.

getTopVentas.sql:

Para esta consulta creamos la funcion getTopVentas(). En ella, seleccionamos el titulo, el id, el año en el que se realizó el pedido de dicha película (anoVenta), y la cantidad de unidades de dicha película vendidas en ese año (totalVentas). Filtramos para obtener unicamente los años que nos interesan, siendo estos mayores que el año inicial, y menores que el final. Agrupamos por id y titulo, y las ordenamos por año y cantidad de ventas.

Finalmente, sobre esta tabla auxiliar realizamos un "*select distinct"* para quedarnos únicamente con la película más vendida de cada año, y devolvemos el resultado.

getTopMonths.sql:

Para esta consulta creamos la función *getTopMonths()*. Queremos obtener los meses en los que se supera alguno de los parámetros dados. Para ello obtenemos la tabla con el año, el mes, el número de ventas de ese mes (*monthVenta*) y la ganancia obtenida ese mes (*totalGanancia*). Por último filtramos de esa tabla únicamente las filas que cumplan las condiciones de número de ventas o ganancia seleccionadas.

UpdOrders.sql:

Para esta consulta, creamos un *trigger* que se active y ejecute la función *update_orders()* cuando haya una operación de *delete*, *update* o *insert* en la tabla *orderdetail*.

Si la operación es *delete* (*TG_OP = 'delete'*) actualizamos el *netAmount* y el *totalAmount* de las filas afectadas por la operación (para ello usamos la funcion **old** de donde obtendremos las filas afectadas). La sentencia *sql* es similar a la de *setOrderAmount()*. Además si el *netAmount* es 0, significa que ese pedido no tiene artículos, por lo que borramos el pedido de la tabla *orders*.

Si la operación es *insert* o *update*, realizamos la actualización de las tablas de la misma manera que para la operación *delete*, pero para obtener las filas afectadas usaremos la función **new**.

UpdInventory.sql:

Para esta consulta, creamos un *trigger* que se active y ejecute la función *update_inventory()* cuando se realice una operación de *update* sobre la tabla *orders*. Esta función actualizará la tabla *products*, que tiene la información de la tabla *inventory* original.

Comprobamos que en la fila modificada en la tabla se ha puesto el *status* de ese order a '*Paid*'. Si es así, es que el pedido ha sido pagado, por lo que tenemos que actualizar el *stock* de ese pedido. Para ello, realizamos un *update* de la tabla *products* sumando al *stock* la cantidad comprada en el pedido para cada producto y sumando la misma cantidad a la columna *sales.*

Por último, si el stock de un producto llega a 0, lo añadiremos a la tabla de 'no_stock' (esta será la tabla 'alertas' del enunciado), incluyendo su *prod_id* y la fecha del pedido en el que se agotó.

Evidencias de los resultados de las consultas:

Las consultas setPrice.sql, setOrderAmount, updOrders, updInventory realizan operaciones de delete y update pricipalmente, por lo que no devuelven una lista de elementos que se pueda mostrar. A continuación mostramos los resultados obtenidos para las consulas getTopVentas y getTopMonths usando los umbrales propuestos

getTopVentas.sql: (entre 2016 y 2020)

```
rodrigo@rodrigo-SATELLITE-PRO-A50-D:~/Escritorio/PRACTICA2_SI/SI1/sql$ psql si1 alumnodb < getTopVentas.sql
NOTICE:  function gettopventas(pg_catalog.int4,pg_catalog.int4) does not exist, skipping
DROP FUNCTION
CREATE FUNCTION
 anoventa |            titulo            |   id   | totalventas
----------+------------------------------+--------+-------------
     2016 | Love and a .45 (1994)        | 236107 |         136
     2017 | Illtown (1996)               | 189256 |         142
     2018 | Wizard of Oz, The (1939)     | 442893 |         134
     2019 | Life Less Ordinary, A (1997) | 229764 |         134
     2020 | Life with Mikey (1993)       | 229931 |          57
(5 rows)
```

getTopMonths.sql: (para mas de 19000 articulos o 320000€)

```
rodrigo@rodrigo-SATELLITE-PRO-A50-D:~/Escritorio/PRACTICA2_SI/SI1/sql$ psql si1 alumnodb < getTopMonths.sql
CREATE FUNCTION
 anoventa | monthventa | totalventas | totalganancia
----------+------------+-------------+---------------
     2015 |          1 |        3881 |     462390.34
     2015 |          2 |        4629 |     549535.26
     2015 |          3 |        7262 |     864386.39
     2015 |          4 |        7897 |     934748.93
     2015 |          5 |       10146 |    1188991.68
     2015 |          6 |       11078 |    1312751.65
     2015 |          7 |       12547 |    1478059.09
     2015 |          8 |       14916 |    1757810.33
     2015 |          9 |       15639 |    1852971.01
     2015 |         10 |       18178 |    2130898.98
     2015 |         11 |       17917 |    2139981.49
     2015 |         12 |       17671 |    2070692.58
     2016 |          1 |       18867 |    2273328.22
     2016 |          2 |       16942 |    2041425.84
     2016 |          3 |       18302 |    2238843.78
     2016 |          4 |       17642 |    2127115.21
     2016 |          5 |       18194 |    2184253.31
     2016 |          6 |       17366 |    2097522.59
     2016 |          7 |       18760 |    2298658.60
     2016 |          8 |       18662 |    2262339.21
     2016 |          9 |       19133 |    2323816.30
     2016 |         10 |       19086 |    2291806.42
     2016 |         11 |       18329 |    2245016.17
     2016 |         12 |       18789 |    2299496.66
     2017 |          1 |       18252 |    2242992.07
     2017 |          2 |       17437 |    2164075.03
     2017 |          3 |       18713 |    2326681.13
     2017 |          4 |       17409 |    2119636.91
     2017 |          5 |       18915 |    2311182.94
     2017 |          6 |       18495 |    2290714.37
     2017 |          7 |       18214 |    2243267.17
     2017 |          8 |       18503 |    2274653.43
     2017 |          9 |       18148 |    2242292.38
     2017 |         10 |       18745 |    2299842.86
     2017 |         11 |       18273 |    2257425.38
     2017 |         12 |       18684 |    2330543.05
     2018 |          1 |       18394 |    2298270.11
     2018 |          2 |       17173 |    2177088.32
     2018 |          3 |       18683 |    2332224.09
     2018 |          4 |       18541 |    2307779.90
     2018 |          5 |       18322 |    2321485.78
     2018 |          6 |       18136 |    2267338.34
     2018 |          7 |       19280 |    2435806.95
     2018 |          8 |       18058 |    2288865.62
     2018 |          9 |       17971 |    2258278.00
     2018 |         10 |       18256 |    2319004.12
     2018 |         11 |       17756 |    2228710.03
     2018 |         12 |       18275 |    2304968.91
     2019 |          1 |       18598 |    2373107.79
     2019 |          2 |       16472 |    2117781.95
     2019 |          3 |       17807 |    2263288.03
     2019 |          4 |       17641 |    2263104.39
     2019 |          5 |       18068 |    2310089.23
     2019 |          6 |       17708 |    2266704.54
     2019 |          7 |       17755 |    2284796.99
     2019 |          8 |       18522 |    2341851.62
     2019 |          9 |       18387 |    2401916.51
     2019 |         10 |       18576 |    2439395.99
     2019 |         11 |       16948 |    2214074.42
     2019 |         12 |       15634 |    2055349.76
     2020 |          1 |       14671 |    1975304.11
     2020 |          2 |       12133 |    1622829.72
     2020 |          3 |       11089 |    1504071.90
     2020 |          4 |        9396 |    1262077.12
     2020 |          5 |        8762 |    1177652.73
     2020 |          6 |        6240 |     827251.83
     2020 |          7 |        5649 |     769087.39
     2020 |          8 |        3906 |     531642.23
(68 rows)
```

Mejoras realizadas:

Implementamos también el historial y la opción de búsqueda de películas por nombre pues nuestra aplicación daba esa opción en la práctica anterior y creímos conveniente integrarlo también con la base de datos. El único inconveniente es que, obviamente, no podemos tener todas las imágenes de todas las películas en nuestro json, por lo que añadirle portadas a las películas obtenidas al buscar era una opción inviable.