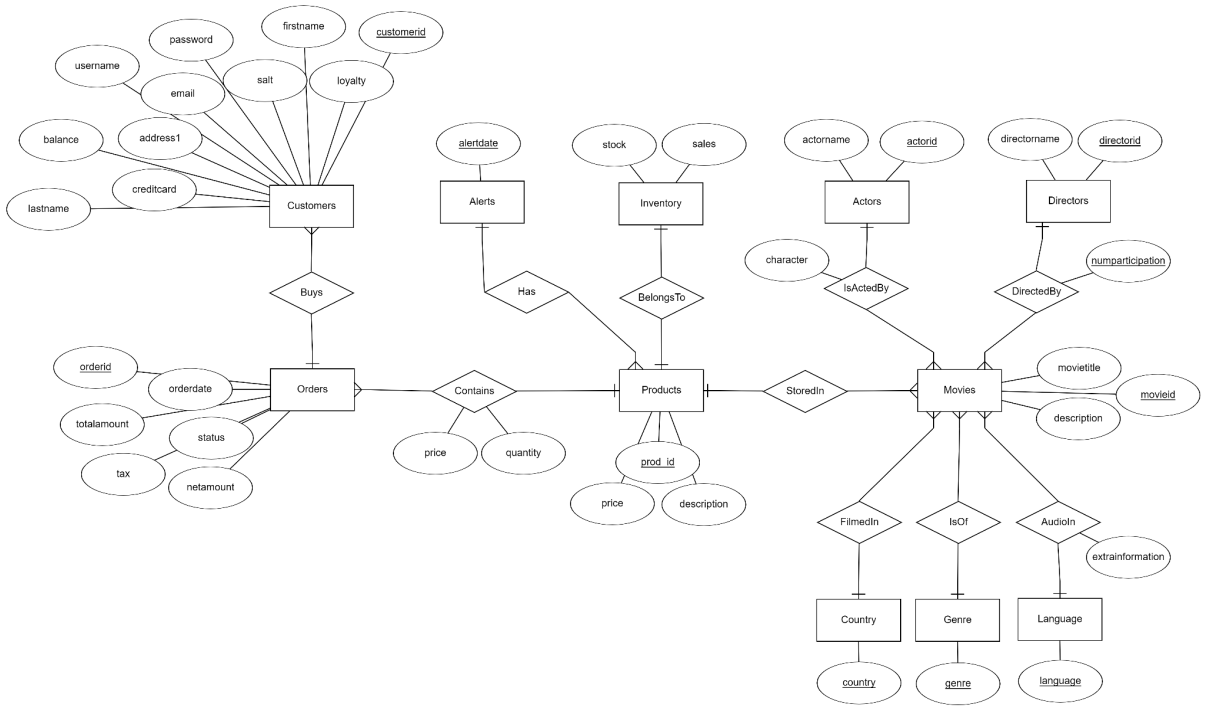
Memoria Práctica 2 - SI

Webflix

Pedro Sánchez de la Muela
Pablo Quintanilla Berriochoa

Base de datos utilizada:

Para realizar la práctica se nos proporcionó una base de datos, a la cual le hicimos una serie de modificaciones para que sirviese para los propósitos de la práctica. Tras las modificaciones el diagrama entidad-relación de la base de datos es el siguiente:



(Este diagrama está incluído en la práctica como *ER_Diagram.png* para mejor visualización)

Cambios realizados:

Para mejorar y adaptar la base de datos hemos realizado una serie de cambios incluídos en *actualiza.sql*, estos cambios consisten en:

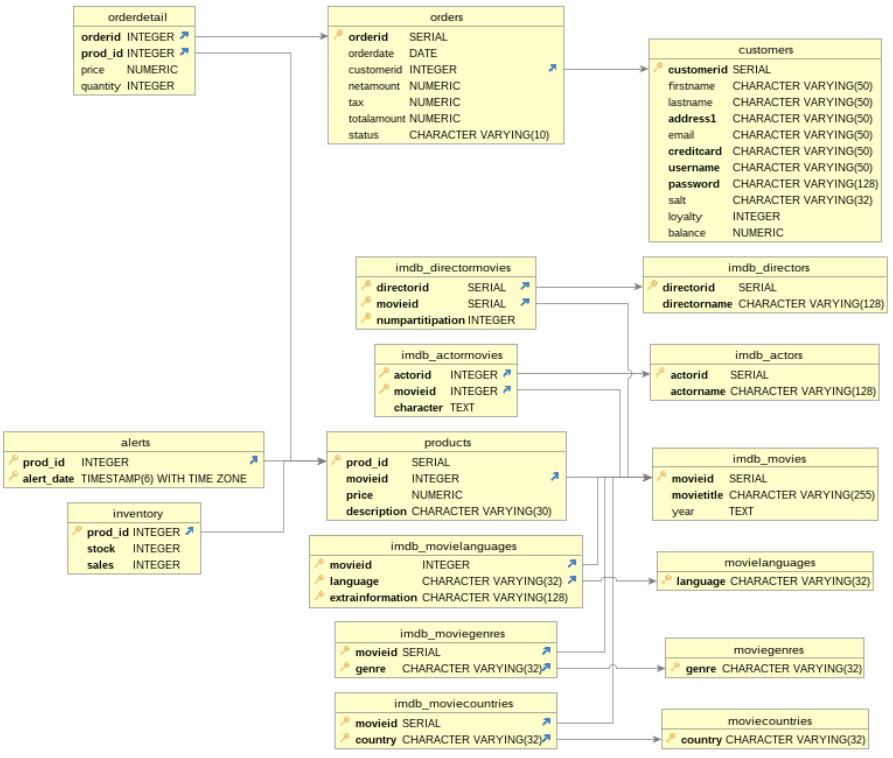
- Añadir a la tabla customers los campos *salt, loyalty* y *balance*. *Salt* es un campo de tipo *varchar(32)* para almacenar el *salt* usado en las contraseñas de los usuarios *Loyalty* es un entero en el que se guardan los puntos acumulados por el usuario, y tiene por defecto el valor 0. Y *balance* es el dinero que tiene el usuario.
- Modificar los campos *firstname* y *lastname* de customers para que puedan tomar valores *NULL*. Y el campo *password* para que sea de longitud 128 caracteres, y así poder almacenar el hash de su contraseña.
- Modificar el año de la película que era del año '1998-1999' a '1998', ya que el año será posteriormente convertido a entero.
- Crear las tablas *moviegenres, moviecountries* y *movielanguages*, para almacenar los posibles valores que pueden tomar los géneros, países e idiomas correspondientes a las películas. Y dar a estas tablas los distintos valores que había ya almacenados en la base de datos.
- Crear la tabla *alerts* para almacenar alertas cuando el *stock* de un producto llega a 0.

- Modificar las columnas *inventory.prod_id, ordedetail.prod_id, orderdetail.orderid, orders.customerid, imdb_actormovies.movieid, imdb_actormovies.actorid, imdb_moviegenres.genre, imdb_moviecountries.country, imdb_movielanguages.language* y *alerts.prod_id* a ser claves extranjeras a los campos a los que hacen referencia. Esto lo hemos hecho para evitar inconsistencias y simplificar el trabajo con sqlalchemy.
- Eliminar los campos de las tablas que hemos considerado innecesarios para nuestra práctica. Esto lo hemos hecho así por simplicidad y por reducir el tamaño de la base de datos.
- Reiniciar los valores de las secuencias *customers_customerid_seq* y *orders_orderid_seq* al máximo de los ids de sus tablas para evitar problemas al insertar nuevos valores con sqlalchemy.
- Dar valores por defecto a las columnas *orderdetail.quantity* y *orders.tax* de 1 y 21 respectivamente, además de eliminar su restricción *NOT NULL*. También para añadir simplicidad a los inserts en estas tablas.
- Utilizar los valores *actorid* y *movieid* de *imdb_actormovies* como la clave primaria de esta tabla. Para asegurar su unicidad y evitar inconsistencias.

Finalmente hemos definido una función *setCustomersBalance* que da un valor aleatorio a la columna *balance* de la tabla customers. Esta función la hemos llamado para asignar a todos los usuarios un crédito entre 0€ y 100€.

Nota: con los cambios del *password* y *salt* de la tabla *customers* hemos podido implementar el cifrado de contraseñas para mayor seguridad.

Tras ejecutar estos cambios el diagrama de clases de nuestra base de datos resultante es el siguiente (este diagrama también se incluye en la práctica como: *Class_Diagram.png*):



Análisis de la solución dada a las consultas, triggers y funciones solicitados:

- ## Consulta setPrice

Esta función se encarga de dar un valor a la columna *price* de la tabla *orderprice*, calculando este valor a partir del valor del precio del producto correspondiente, sabiendo que el precio de las películas ha ido aumentando un 2% por cada año. El nuevo valor del precio lo hemos calculado como precio_producto * (1 - 1/51*(año_actual - año_compra)). Esto no es exacto ya que no tiene en cuenta que el precio de cada año es un 2% menor, sino que simplemente resta un 2% del precio actual por cada año que ha pasado, pero es una aproximación lo suficientemente buena. Cabe destacar también que usamos 1/51, que viene de que el precio de la película este año es el 102% del precio del año pasado y 2%/102% = 1/51, esto es más preciso que usar 1/50.

Podemos comprobar esta función si imprimimos el nuevo precio , los años que hace desde que se compró la película y el precio antiguo (en este orden) para algunas películas:

```
   order_price    | year_diff | product_price
------------------+-----------+---------------
 17.2941176470588 |         2 |            18
               10 |         0 |            10
               16 |         3 |            17
             19.6 |         2 |          20.4
               24 |         3 |          25.5
```

- ## Función setOrderAmount

Esta función se encarga de rellenar las filas *netamount* y *totalamount* de la tabla *orders*. Para rellenar el *netamount* suma los precios de los *orderdetails* asociados a esta *order* multiplicados por sus cantidades (el valor *quantity*). Y para calcular el *totalamount* simplemente aplica al *netamount* los impuestos aplicados a esa *order*.

Comprobamos el comportamiento de esta función al extraer estos datos:

```
     netamount      | tax |      totalamount     |    suma_precios
--------------------+-----+----------------------+--------------------
   31.7058823529412 |  15 |   36.461764705882380 |   31.7058823529412
   29.4117647058823 |  15 |   33.823529411764645 |   29.4117647058823
 158.43137254901960 |  18 | 186.9490196078431280 | 158.43137254901960
  132.4921568627451 |  15 |  152.365980392156865 |  132.4921568627451
   86.5882352941176 |  15 |   99.576470588235240 |   86.5882352941176
```

- ## Función getTopSales

Esta función recibe por parámetros 2 años, y devuelve, para cada año entre estos dos años, las películas que más se han vendido ese año. Nótese que devolvemos las películas más vendidas en dicho año, no las películas más vendidas producidas en dicho año. Además, consideramos como película más vendida la que tiene mayor cantidad total en orders en dicho año, es decir, sumamos la columna *quantity* para cada película y cada año. Esto lo hemos implementado utilizando la función *rank()* sobre una partición por años de una query que suma las cantidades vendidas de cada película en cada año. Esta query está ordenada por la cantidad de ventas de forma descendente, de tal manera que las entradas con rank = 1 son las películas más vendidas para cada año.

El resultado de llamar a esta función para los años 2015 y 2021 obtenemos:

```
 year |             film              | sales
------+-------------------------------+-------
 2018 | Illtown (1996)                |   142
 2017 | Love and a .45 (1994)         |   136
 2019 | Wizard of Oz, The (1939)      |   134
 2020 | Life Less Ordinary, A (1997)  |   134
 2016 | No Looking Back (1998)        |   101
 2021 | Mary Reilly (1996)            |    59
 2015 | Male and Female (1919)        |     9
```

- ## Función getTopActors

Esta función recibe un género dado por parámetros y devuelve la lista de actores o actrices que más veces han actuado en películas de este género (siempre que hayan actuado más de 4 veces). Devolviendo el nombre de cada uno de estos actores, el número de actuaciones en el género, el año en el que debutaron para este género, la película en la que lo hicieron y el director/es de esta película.

Para implementar esta función hemos utilizado una query con dos subqueries. La primera de estas subqueries se encarga de extraer el número de actuaciones de cada actor y filtrar los que tienen menos de 4. La segunda subquery extrae la película del género en la que debutó cada actor, utilizando también la función rank() sobre una partición ordenada por años, para seleccionar el primero de los años. Finalmente la query utilizada enlaza estas dos subqueries y enlaza también con los directores de estas películas.

El resultado de ejecutar esta función para el género ‘Drama’ y limitarlo a 5 filas es:

```
       actor        | num | debut |           film           |     director
--------------------+-----+-------+--------------------------+------------------
 Jackson, Samuel L. |  26 |  1988 | School Daze (1988)       | Lee, Spike
 Duvall, Robert (I) |  26 |  1962 | To Kill a Mockingbird... | Mulligan, Robert
 De Niro, Robert    |  24 |  1971 | Born to Win (1971)       | Passer, Ivan
 Walsh, M. Emmet    |  23 |  1969 | Midnight Cowboy (1969)   | Schlesinger, John
 Keitel, Harvey     |  22 |  1976 | Taxi Driver (1976)       | Scorsese, Martin
```

- ## Trigger updOrders

Este trigger salta cuando hay un *INSERT/DELETE/UPDATE (en la columna quantity)* en la tabla orderdetails. Y se encarga de actualizar los campos de *netamount* y *totalamount* de la tabla *orders*. Esto lo hemos implementado de la misma manera que la función *setOrderAmount*, pero limitándonos a actualizar las *orders* correspondientes a la *orderdetail* que se ha insertado/borrado/actualizado.

Podemos comprobar el comportamiento de este trigger observando los cambios que se producen en una entrada de la tabla *orders* al hacer algún cambio en alguna de sus *orderdetails*. Antes de hacer ningún cambio el estado de la *order* con *orderid* = 151582

```
 orderid |    netamount     | tax |    totalamount
---------+------------------+-----+-------------------
  151582 | 63.7058823529412 |  15 | 73.26176470588238
```

La *orderdetail* que vamos a modificar es:

```
 orderid | prod_id |      price       | quantity
---------+---------+------------------+----------
  151582 |    4432 | 12.3529411764706 |        1
```

Si cambiamos su *quantity* a 3 la *order* pasa a tener estos valores:

```
 orderid |    netamount     | tax |    totalamount
---------+------------------+-----+--------------------
  151582 | 88.4117647058824 |  15 | 101.67352941176476
```

Si borramos esta orderdetail obtenemos:

```
 orderid |    netamount     | tax |    totalamount
---------+------------------+-----+-------------------
  151582 | 51.3529411764706 |  15 | 59.05588235294119
```

Finalmente, si la volvemos a añadir volvemos a los valores originales

- ## Trigger updInventoryAndCustomer

Este trigger se dispara cuando un usuario realiza una compra, esto es, cuando una *order* deja de tener *status* = NULL. Cuando esto ocurre, este trigger se encarga de actualizar la *orderdate* de la *order* modificada, de actualizar el *stock* y los *sales* de las entradas en *inventory* para las películas correspondientes a esta *order*. Además crea una entrada en la tabla *alerts* para cada producto cuyo *stock* ha llegado a 0 tras esta compra. Finalmente, actualiza los campos de *loyalty* y *balance* del usuario que ha realizado la compra.

Podemos comprobar el comportamiento de este trigger analizando los contenidos de las tablas *orders, orderdetail, inventory, customers* y *alerts* antes y después de hacer una compra. Además modificamos una entrada para que tenga cantidad = 30 y así el stock baje por debajo

de 0 y se cree una entrada en la tabla alerts.Antes de hacer la compra el contenido de estas tablas correspondientes a la *order* con id = 146760 es:

```
 orderid | customerid | netamount | tax |  totalamount  | status
---------+------------+-----------+-----+---------------+--------
  146760 |      11387 |     622.8 |  21 | 753.588000000 |

orderdetails + inventory:
 orderid | prod_id | price | quantity | prod_id | stock | sales
---------+---------+-------+----------+---------+-------+-------
  146760 |    4416 |  13.2 |        1 |    4416 |   910 |   149
  146760 |    2486 |    19 |        1 |    2486 |   864 |   162
  146760 |    6567 |    18 |        1 |    6567 |   221 |   151
  146760 |    2272 |    16 |        1 |    2272 |   466 |   180
  146760 |    5585 |  14.4 |        1 |    5585 |   962 |   172
  146760 |    2574 |    13 |        1 |    2574 |   922 |   176
  146760 |    3053 |  20.4 |        1 |    3053 |   581 |   168
  146760 |     936 |  21.6 |        1 |     936 |   294 |   163
  146760 |    5118 |  19.2 |        1 |         |       |
  146760 |    3758 |  15.6 |       30 |    3758 |    22 |   159

customer with customerid = 11387:
 firstname | balance | loyalty
-----------+---------+---------
 janell    |      68 |       0

alerts:
 prod_id | alert_date
---------+------------
(0 filas)
```

Si realizamos esta compra el contenido de estas tablas pasa a ser:

```
orders:
 orderid | customerid | netamount | tax |  totalamount  | status
---------+------------+-----------+-----+---------------+--------
  146760 |      11387 |     622.8 |  21 | 753.588000000 | Paid

orderdetails + inventory:
 orderid | prod_id | price | quantity | prod_id | stock | sales
---------+---------+-------+----------+---------+-------+-------
  146760 |    4416 |  13.2 |        1 |    4416 |   909 |   150
  146760 |    3758 |  15.6 |       30 |    3758 |    -8 |   189
  146760 |    2486 |    19 |        1 |    2486 |   863 |   163
  146760 |    6567 |    18 |        1 |    6567 |   220 |   152
  146760 |    2272 |    16 |        1 |    2272 |   465 |   181
  146760 |    5585 |  14.4 |        1 |    5585 |   961 |   173
  146760 |    2574 |    13 |        1 |    2574 |   921 |   177
  146760 |    3053 |  20.4 |        1 |    3053 |   580 |   169
  146760 |     936 |  21.6 |        1 |     936 |   293 |   164
  146760 |    5118 |  19.2 |        1 |         |       |
```

```
customer with customerid = 11387:
 firstname |          balance           | loyalty
-----------+----------------------------+---------
 janell    | -685.588000000000000000000 |    3768

alerts:
 prod_id |          alert_date
---------+-------------------------------
    3758 | 2021-11-19 11:46:33.212353+01
```

NOTA: las consultas sql para obtener estos datos de prueba están incluídas en el archivo *test.sql*.