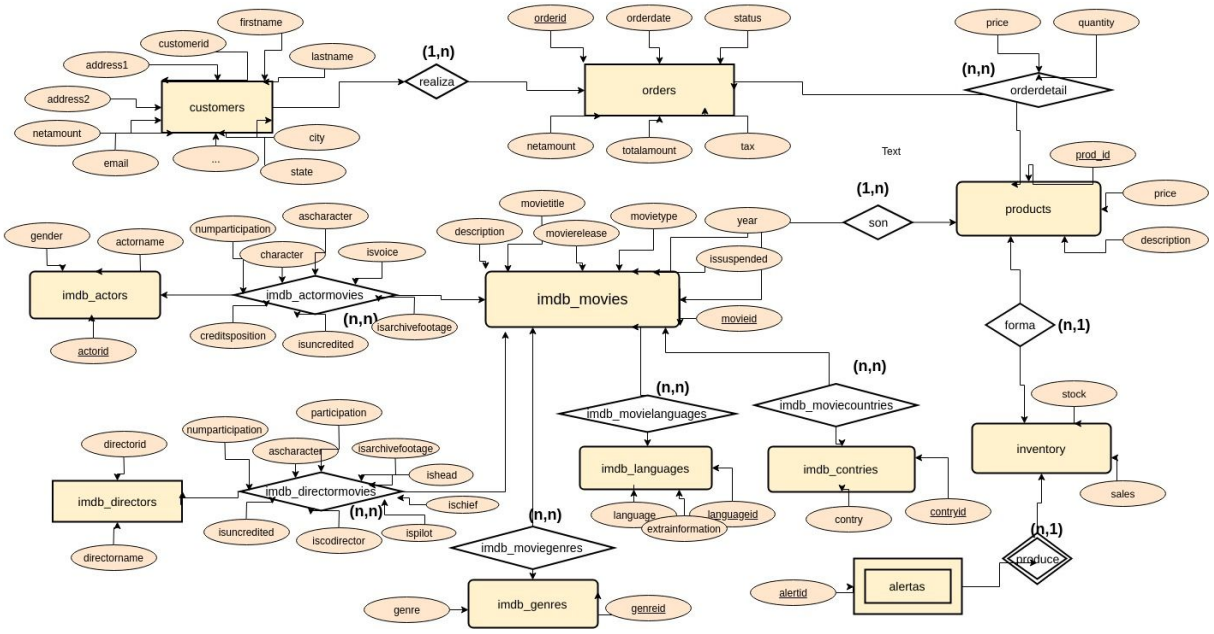
Práctica 3. Programación Web y Bases de Datos.

Blanca Abella Miravet
Maria Barroso Honrubia
Grupo 1401

Descripción Esquema Entidad Relación y Base de datos

El diagrama entidad-relación de la base de datos resultante tras aplicar el script 'actualiza.sql' es el siguiente:



Todos los atributos se han mantenido respecto a la base de datos de partida y se ha añadido un atributo description a la clase imdb_movies.

La clase ‘imdb_movies’ hace referencia a las películas, que están relacionadas con la clase ‘products’ con una relación 1 a n, ya que hay varios productos para una misma película. Los productos forman la clase ‘inventory’ que nos da información del stock de cada producto y su número de ventas. Es decir, cada producto tiene un inventario asociado. Debido a la importancia de gestionar el atributo de stock surge la clase ‘alertas’.
Las películas están representadas por actores (relación n-n, ya que una película está representada por varios actores y un actor actúa en varias películas), están dirigidas por directores (relación n-n por el mismo motivo) y están en varios idiomas, se encuentran en varios países y tienen varios géneros temáticos. La relación entre películas y actores posee atributos que especifican detalles sobre la participación de un actor en una película. De la misma manera, existe lo mismo para relacionar directores y películas.

Para guardar la información relativa a los diferentes clientes y detallar las compras concretas que han ido realizando estos a través de la web contamos con 2 clases ‘customers’ y ‘orders’. Un cliente puede realizar varios pedidos pero un pedido sólo puede haber sido realizado por un cliente en concreto. Por otro lado los pedidos están formados por determinados productos diferentes o iguales, gestionado por la relación ‘orderdetail’.

Blanca Abella Miravet y María Barroso Honrubia

Descripción actualiza.sql

Para garantizar la integridad de los datos hemos añadido Foreign Keys a aquellas tablas que que no estaban relacionadas permitiendo a su vez cambios en cascada.

Tal y como se pedía en el enunciado, hemos creado las tablas 'imdb_languajes', 'imdb_genres' y 'imdb_contries' dotandolas de un identificador y los atributos 'moviecountries', 'movie-genres' y 'movielanguages' respectivos. De esta forma se han modificado las tablas originales donde en vez de almacenar varchar, almacenamos índices que se corresponden con la información anterior.

Como se ha visto en el diagrama ER, hemos creado la tabla 'alertas' que antes no existía. Se irán creando nuevas alertas al activarse el trigger updInventory cuando el stock de un producto sea 0.

Por otro lado, hemos optado por aceptar el valor NULL en ciertos campos de 'customer' para mantener acciones como registrar un usuario tal y como las planteamos en la práctica anterior.

Para poder mostrar la descripción relativa a una película, hemos creado un atributo 'description', dotándolo de un texto Lorem ipsum.

Por último, hemos actualizado los id de las clases customers y orders para que cuando se creen nuevas instancias de cada uno se asigne el valor siguiente respecto al último creado.

Descripción setPrice.sql

La consulta realizada en setPrice.sql da un valor al atributo ‘price’ presente en la clase ‘orderdetails’. Como el precio de un producto se incrementa un 2% por cada año desde su venta (atributo ‘orderdate’ en ‘orders’), y el precio inicial está guardado en el atributo ‘price’ de ‘products’, para calcular dicho valor basta con multiplicar el precio inicial por 0.98 tantas veces como años hayan pasado desde la fecha de compra hasta la fecha actual.

```
si1=# select * from orderdetail where orderid=1050;
 orderid | prod_id |    price     | quantity
---------+---------+--------------+----------
    1050 |     466 |   13.8355224 |        1
    1050 |    2480 |  22.13683584 |        1
    1050 |    3140 |  22.13683584 |        1
    1050 |    4571 |  15.68025872 |        1
    1050 |     188 |   13.8355224 |        1
    1050 |    6346 | 14.388943296 |        1
    1050 |    3672 |  17.98617912 |        1
    1050 |    3730 | 13.282101504 |        1
    1050 |    1009 |  10.14604976 |        1
    1050 |     899 |  14.75789056 |        1
(10 rows)
```

Prueba del atributo ‘price’ de ‘orderdetail’ con orderid=1050

Descripción setOrderAmount.sql

En setOrderAmount.sql se implementa un procedimiento almacenado llamado setOrderAmount(), que se encarga de completar las columnas 'netamount' y 'totalamount' de la tabla 'orders’.

El valor 'netamount' se actualiza con un update y se corresponde con el valor devuelto por el sumatorio de producto del precio ‘price’ por la cantidad ‘quantity’ de los orderdetail cuyo orderid sea el mismo que el de orders.

El valor 'totalamount' se actualiza con un update y se corresponde con incrementar a dicho valor la multiplicación de 'netamount' por el porcentaje de los impuestos ‘tax’.

```
si1=# select * from orders where orderid=1050;
 orderid | orderdate  | customerid |   netamount    | tax |     totalamount      |  status
---------+------------+------------+----------------+-----+----------------------+-----------
    1050 | 2014-08-02 |         72 | 158.186139440 |  15 | 181.9140603560000000 | Processed
(1 row)
```

Prueba de los atributos 'netamount' y 'totalamount' de ‘orders’ con orderid=1050

Descripción getTopVentas.sql

Con esta consulta queremos conseguir la película más vendida de cada año a partir de cierto año. Estos datos los conseguimos creando una función a la que se le pasa dicho año y nos devuelve una tabla con los años, los títulos y la ventas correspondientes. Para la implementación de esta función, primero sumamos las ventas de cada película cada año. Después, cogemos el máximo de esas sumas para cada año desde el año introducido como parámetro, consiguiendo así el resultado deseado ordenandolo por año.

```
maria@maria-UX303UA:~/Escritorio/SI1/practica3/SQL$ psql si1 < getTopVentas.sql
DROP FUNCTION
CREATE FUNCTION
 annio |            titulo            | sales
-------+------------------------------+-------
  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
(7 rows)
```

Descripción getTopMonths.sql

Queremos hallar los meses en los que las ventas superan ciertos umbrales de número de productos y de importe ('totalamount') acumulados. Creamos una función que recibirá dichos umbrales y devolverá una tabla con los resultados. Realizamos una consulta en las que obtenemos de cada mes de cada año, el totalamount y el número total de productos vendidos. Después filtramos por los umbrales establecidos y mostramos el resultado en forma de tabla.

```
maria@maria-UX303UA:~/Escritorio/SI1/practica3/SQL$ psql si1 < getTopMonths.sql
DROP FUNCTION
CREATE FUNCTION
 annio | mes | import  | cantidad
-------+-----+---------+----------
  2013 |   1 |  462474 |     3881
  2013 |   2 |  550243 |     4629
  2013 |   3 |  864630 |     7262
  2013 |   4 |  935437 |     7897
  2013 |   5 | 1189232 |    10146
  2013 |   6 | 1313429 |    11078
  2013 |   7 | 1479052 |    12547
  2013 |   8 | 1759112 |    14916
  2013 |   9 | 1853862 |    15639
  2013 |  10 | 2131597 |    18178
  2013 |  11 | 2140979 |    17917
  2013 |  12 | 2071184 |    17671
  2014 |   1 | 2274248 |    18867
  2014 |   2 | 2041780 |    16942
  2014 |   3 | 2239911 |    18302
  2014 |   4 | 2128223 |    17642
  2014 |   5 | 2185314 |    18194
  2014 |   6 | 2099330 |    17366
  2014 |   7 | 2299889 |    18760
  2014 |   8 | 2263912 |    18663
  2014 |   9 | 2325145 |    19133
  2014 |  10 | 2292351 |    19086
  2014 |  11 | 2245448 |    18329
  2014 |  12 | 2300558 |    18789
  2015 |   1 | 2243565 |    18252
  2015 |   2 | 2101395 |    16918
  2015 |   3 | 2314225 |    18605
  2015 |   4 | 2126721 |    17454
  2015 |   5 | 2325167 |    19001
  2015 |   6 | 2279248 |    18417
  2015 |   7 | 2238920 |    18147
  2015 |   8 | 2274642 |    18463
  2015 |   9 | 2254954 |    18273
  2015 |  10 | 2291445 |    18698
  2015 |  11 | 2271347 |    18363
  2015 |  12 | 2312699 |    18511
  2016 |   1 | 2300443 |    18399
  2016 |   2 | 2263158 |    17854
  2016 |   3 | 2333589 |    18683
  2016 |   4 | 2309395 |    18541
  2016 |   5 | 2322616 |    18322
  2016 |   6 | 2268739 |    18136
  2016 |   7 | 2437185 |    19280
  2016 |   8 | 2289970 |    18058
  2016 |   9 | 2259732 |    17971
  2016 |  10 | 2320650 |    18256
  2016 |  11 | 2229258 |    17756
  2016 |  12 | 2305888 |    18275
  2017 |   1 | 2374347 |    18598
  2017 |   2 | 2118598 |    16472
  2017 |   3 | 2264505 |    17807
  2017 |   4 | 2264182 |    17641
  2017 |   5 | 2311825 |    18068
  2017 |   6 | 2267993 |    17708
  2017 |   7 | 2285909 |    17755
  2017 |   8 | 2342666 |    18522
  2017 |   9 | 2403448 |    18387
  2017 |  10 | 2440190 |    18576
  2017 |  11 | 2215198 |    16948
  2017 |  12 | 2056373 |    15634
  2018 |   1 | 1976402 |    14671
  2018 |   2 | 1623013 |    12133
  2018 |   3 | 1504514 |    11089
  2018 |   4 | 1262623 |     9396
  2018 |   5 | 1178165 |     8762
  2018 |   6 |  827551 |     6240
  2018 |   7 |  769368 |     5649
  2018 |   8 |  532308 |     3906
(68 rows)
```

Descripción updOrders.sql

En updOrders.sql se implementa un trigger llamado updOrders encargado de actualizar la información de la tabla 'orders' cuando se añada o elimine un artículo del carrito. El trigger llama a la función f_updOrders() cuando se realiza un procedimiento DELETE, INSERT o UPDATE sobre la tabla orderdetails. En cada caso se actualizará el atributo netamount y el totalamount según el valor de netamount. Por tanto, sólo comentaremos como se ha calculado el valor de netamount. Al referirnos al precio de un orderdetail hacemos referencia a su precio segun el numero de cantidad que exista de este.

Si el procedimiento utilizado es INSERT INTO ⇒

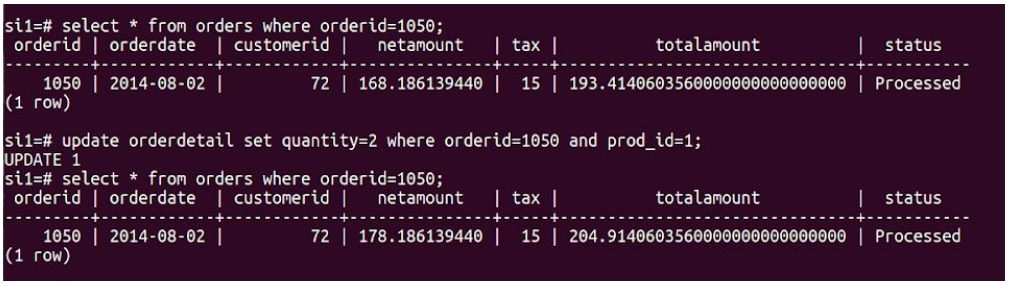
Se desea añadir un orderdetail actualizando el atributo netamount del orders con mismo orderid según el valor del nuevo o los nuevos productos añadidos.

```
si1=# select * from orders where orderid=1050;
 orderid | orderdate  | customerid |   netamount    | tax |     totalamount      |  status
---------+------------+------------+----------------+-----+----------------------+-----------
    1050 | 2014-08-02 |         72 | 158.186139440 |  15 | 181.9140603560000000 | Processed
(1 row)

si1=# insert into orderdetail(orderid, prod_id, price, quantity) values(1050, 1, 10, 1);
INSERT 0 1
si1=# select * from orders where orderid=1050;
 orderid | orderdate  | customerid |   netamount    | tax |          totalamount          |  status
---------+------------+------------+----------------+-----+-------------------------------+-----------
    1050 | 2014-08-02 |         72 | 168.186139440 |  15 | 193.41406035600000000000000000 | Processed
(1 row)
```

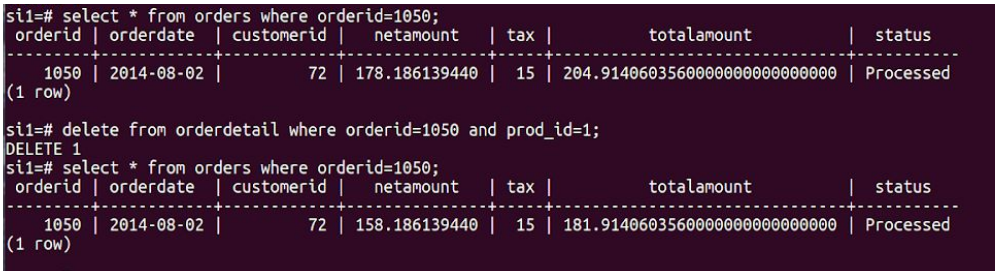
Si el procedimiento utilizado es UPDATE ⇒

Se desea modificar la cantidad de un orderdetail y se actualiza netamount restando el precio de la/las pelicula/s que habia previamente a la actualización y sumando el precio de la/las pelicula/s actualizadas.



Si el procedimiento utilizado es DELETE ⇒

Se desea eliminar un orderdetail y se modifica por consiguiente el atributo netamount de orders restando a su precio actual el precio del orderdetail que ha sido eliminado.



Descripción updInventory.sql

Estamos en el caso en el que se ha realizado una compra y por tanto, se ha cambiado el status de orders a Paid. Hay que actualizar el inventario de manera que el nuevo stock sea el anterior menos la cantidad comprada y se añada a las ventas ese mismo número.
Como se ha comentado anteriormente en la memoria, hemos creado una nueva tabla 'alerta' que se activa cuando el stock de algún producto llega a 0. Para realizar esta funcionalidad, cada vez que se realiza una compra y se actualiza el inventario, comprobamos si el stock resultante es 0. En caso de esto ocurra, se añadirá una nueva alerta a la tabla con el id del producto y un id que se genera automáticamente.

En la siguiente captura, se muestra como se actualiza el inventario de un producto cuando se compra:

```
si1=# select * from orders where orderid=1051;
 orderid | orderdate  | customerid | netamount | tax |     totalamount      | status
---------+------------+------------+-----------+-----+----------------------+---------
    1051 | 2017-06-20 |         72 |   174.048 |  15 | 200.1552000000000000 | Shipped
(1 row)

si1=# select * from orderdetail where orderid=1051;
 orderid | prod_id | price  | quantity
---------+---------+--------+----------
    1051 |    2951 | 14.112 |        1
    1051 |    1717 |  16.17 |        1
    1051 |    3885 |   14.7 |        1
    1051 |    1520 |  15.68 |        2
    1051 |    4603 | 14.112 |        1
    1051 |     424 |  15.68 |        1
    1051 |     884 | 15.288 |        1
    1051 |    3988 | 12.936 |        1
    1051 |    5753 |  16.17 |        1
    1051 |    2480 |  23.52 |        1
(10 rows)

si1=# select * from inventory where prod_id=2951;
 prod_id | stock | sales
---------+-------+-------
    2951 |   292 |   185
(1 row)

si1=# update orders set status='Paid' where orderid=1051;
UPDATE 1
si1=# select * from inventory where prod_id=2951;
 prod_id | stock | sales
---------+-------+-------
    2951 |   291 |   186
(1 row)
```

En esta captura, se ve cómo se crea una alerta tras comprar un producto con stock=1:

```
si1=# select * from inventory where stock=1;
 prod_id | stock | sales
---------+-------+-------
    2508 |     1 |   173
     220 |     1 |   161
    2064 |     1 |   143
    3972 |     1 |   165
    5615 |     1 |   171
    5689 |     1 |   152
(6 rows)

si1=# select * from orderdetail where prod_id=220;
si1=# select * from orders where orderid=15861;
 orderid | orderdate  | customerid | netamount | tax |     totalamount      | status
---------+------------+------------+-----------+-----+----------------------+---------
   15861 | 2016-10-27 |       1194 | 144.63624 |  15 | 166.3316760000000000 | Shipped
(1 row)

si1=# update orders set status='Paid' where orderid=15861;
UPDATE 1
si1=# select * from alertas;
ERROR:  relation "alertas" does not exist
LINE 1: select * from alertas;
                      ^
si1=# select * from alerta;
 alertid | prod_id
---------+---------
       1 |     220
(1 row)
```