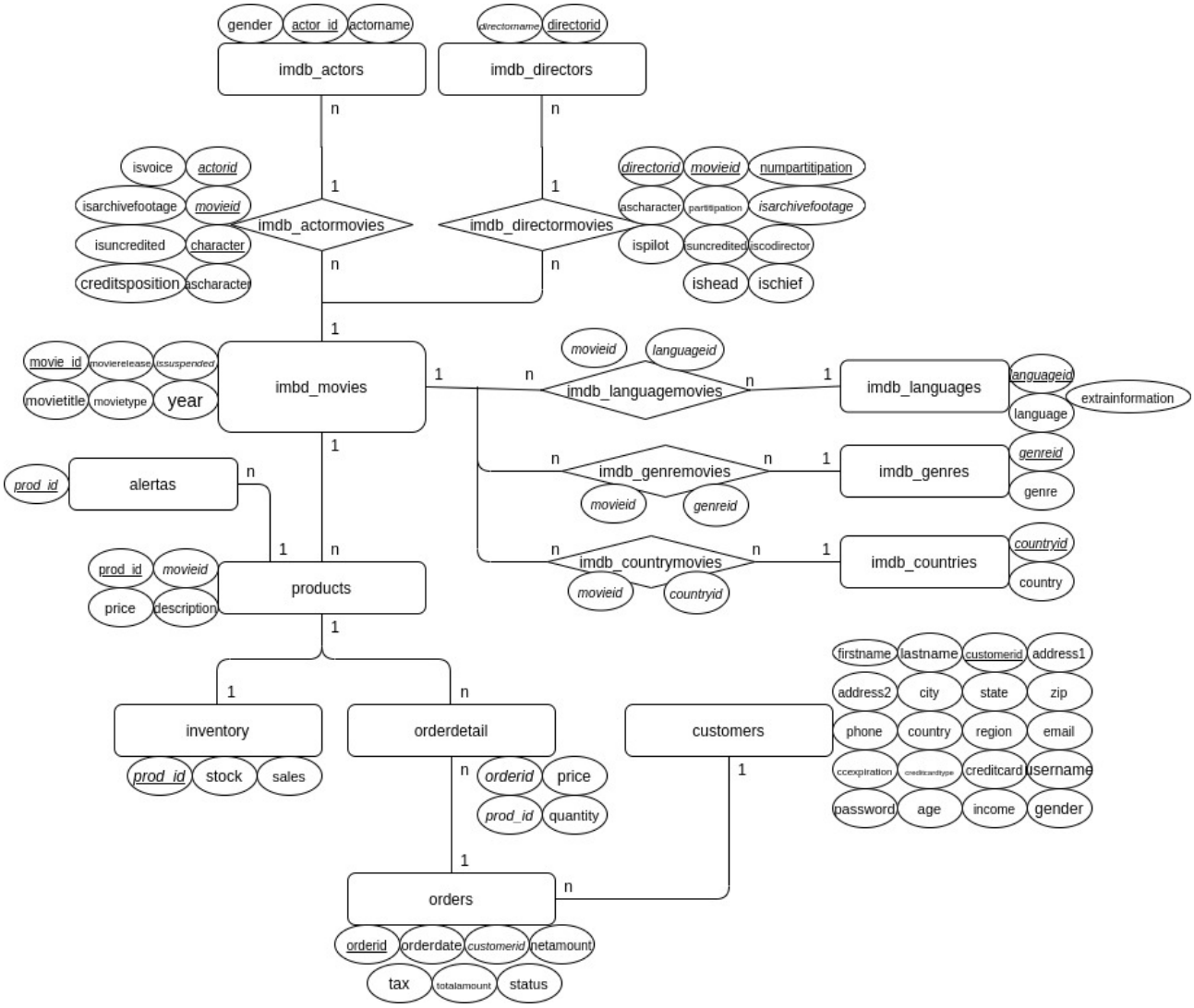
# Práctica 2
# Sistemas Informáticos I

Tercer curso de Ingeniería Informática
Escuela Politéctina Superior
Universidad Autónoma de Madrid

Rodrigo Juez Hernandez
Pablo Ernesto Soëtard García

Tras haber obtenido el diagrama original de la base de datos entregada como punto de partida, hemos aplicado la mejoras de los apartados a) y f), obteniendo el diagrama siguiente:




La base de datos que se nos proporcionaba inicialmente carecía de muchas claves primarias y foráneas, lo que producía un alto grado de ineficiencia en la manera en la que se guardan y buscan los datos en ella. Esto se debe a la falta de relación entre tablas, que puede producir almacenamiento de datos redundantes lo que aumenta el tamaño de la base de datos innecesariamente, y mayor tiempo de búsqueda, debido a la inexistencia de relaciones cruciales entre datos.

Los cambios realizados para mejorar el diseño de la base de datos han sido los siguientes:

- Añadir clave primaria compuesta por actorid, movieid y character a imdb_actormovies, así como una claves foráneas actorid a la tabla imdb_actors y movieid a la tabla imdb_movies. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos. Además de eliminar datos redundantes.

- Reemplazar las claves foráneas existentes en imdb_directormovies por las claves foráneas directorid a la tabla imdb_directors y movieid a la tabla imdb_movies. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos.

- Reemplazar la clave foránea existente en imdb_movielanguages por la clave foránea movieid a la tabla imdb_movies. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos.

- Reemplazar la clave foránea existente en imdb_moviegenres por la clave foránea movieid a la tabla imdb_movies. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos.

- Reemplazar la clave foránea existente en imdb_moviecountries por la clave foránea movieid a la tabla imdb_movies. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos.

- Reemplazar la clave foránea existentes en products por la clave foránea movieid a la tabla imdb_movies. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos.

- Añadir a orderdetail las claves foráneas orderid a la tabla orders y prod_id a la tabla products. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos. Además de eliminar datos redundantes.

- Añadir a inventory la clave foránea prod_id a la tabla products. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos.

- Añadir a orders la clave foránea customerid a la tabla customers. Esto aumenta la flexibilidad y eficiencia tanto en espacio y tiempo de la base de datos.

En el apartado f) se nos pedía convertir los atributos multivaluados de ‘moviecountries’, ‘moviegenres’ y ‘movielanguages’ en relaciones entre tablas, todo ello para garantizar la integridad de los datos. Para ello en las tres tablas mencionadas anteriormente hemos realizado lo siguiente:

- Hemos renombrado la tabla original para que siga el patrón de nombre de las demás tablas (por ejemplo imdb_moviecountries pasa a ser imdb_countrymovies).

- Hemos creado la tabla donde se almacenarán los atributos multivaluados (por ejemplo imdb_countries), que está compuesta por un id serial y el atributo multivaluado.

- Tras haber creado la tabla anterior hemos insertado en ella todos los atributos multivaluados distintos de la tabla original.

- En la tabla original añadimos una columna que hará referencia al id serial, mencionado anteriormente, de la otra tabla.

- Rellenamos la columna id serial que acabamos de crear con el id correspondiente que haga referencia al atributo multivaluado de la otra tabla.

- Por último borramos la columna de los atributos multivaluados de la tabla original, dejando la del id serial que hemos creado en pasos anteriores y creamos una relación de clave foránea con el id serial de la otra tabla.

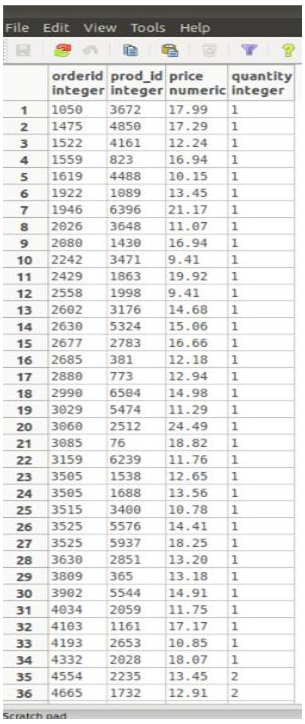
- Además, como pedía el apartado h), hemos creado una tabla alertas con un solo campo 'product_id' que es una clave foránea a la tabla products.

b) La consulta que proponemos para este apartado es la siguiente:

```sql
/*Actualiza el precio de los orderdetail diminuyendo 2% el precio por año desde la orderdate*/
UPDATE orderdetail
SET price = Round(Cast(temporary.price * Pow(0.98, Extract(YEAR
                                                           FROM Now()) - Extract(YEAR
                                                                                 FROM temporary.orderdate)) AS NUMERIC), 2)
FROM
  /*Selecciona todos los orderid con su correspondientes prod_id, orderdate y price*/
  (SELECT orderdetail.orderid,
          orderdetail.prod_id,
          orderdate,
          products.price
   FROM orderdetail
   INNER JOIN orders ON orderdetail.orderid = orders.orderid
   INNER JOIN products ON orderdetail.prod_id = products.prod_id) AS
temporary
WHERE orderdetail.orderid = temporary.orderid
  AND orderdetail.prod_id = temporary.prod_id;
```

Esta consulta realiza un update de la tabla orderdetail, para ello actualiza su precio. Primero obtenemos una tabla intermedia con las columnas orderid, prod_id, orderdate y price resultante de hacer los joins correspondientes de la tabla orderdetail con orders y products.
Tras ello actualizamos el precio de orderdetail igualándolo al resultado de multiplicar su precio por 0.98 elevado a la diferencia entre el año actual y el orderdate (esto anulará el 2% que ha aumentado por cada año pasado, como dice el enunciado).

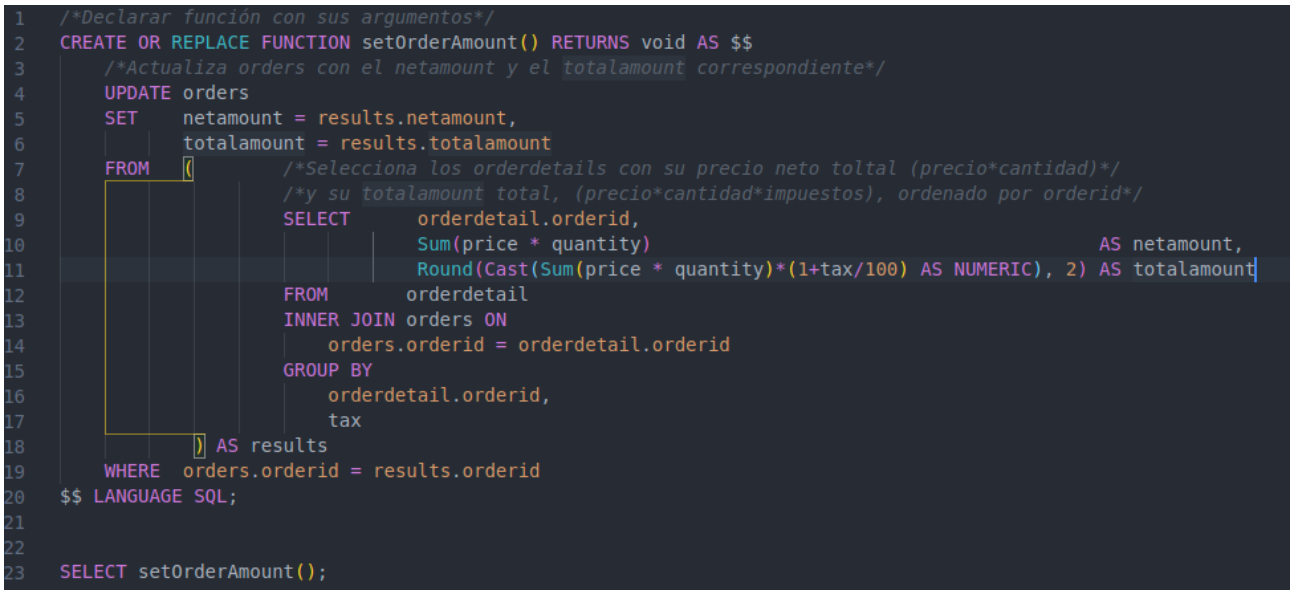
Como se puede observar, la columna de price se rellena correctamente, hemos hecho los cálculos correspondientes con una muestra razonablemente grande de películas y el resultado del campo price que actualiza nuestra consulta y el de los cálculos numéricos coincide.


File Edit View Tools Help

| | orderid integer | prod_id integer | price numeric | quantity integer |
|---|---|---|---|---|
| 1 | 1050 | 3672 | 17.99 | 1 |
| 2 | 1475 | 4850 | 17.29 | 1 |
| 3 | 1522 | 4161 | 12.24 | 1 |
| 4 | 1559 | 823 | 16.94 | 1 |
| 5 | 1619 | 4488 | 10.15 | 1 |
| 6 | 1922 | 1089 | 13.45 | 1 |
| 7 | 1946 | 6396 | 21.17 | 1 |
| 8 | 2026 | 3648 | 11.07 | 1 |
| 9 | 2080 | 1430 | 16.94 | 1 |
| 10 | 2242 | 3471 | 9.41 | 1 |
| 11 | 2429 | 1863 | 19.92 | 1 |
| 12 | 2558 | 1998 | 9.41 | 1 |
| 13 | 2602 | 3176 | 14.68 | 1 |
| 14 | 2630 | 5324 | 15.06 | 1 |
| 15 | 2677 | 2783 | 16.66 | 1 |
| 16 | 2685 | 381 | 12.18 | 1 |
| 17 | 2880 | 773 | 12.94 | 1 |
| 18 | 2990 | 6504 | 14.98 | 1 |
| 19 | 3029 | 5474 | 11.29 | 1 |
| 20 | 3060 | 2512 | 24.49 | 1 |
| 21 | 3085 | 76 | 18.82 | 1 |
| 22 | 3159 | 6239 | 11.76 | 1 |
| 23 | 3505 | 1538 | 12.65 | 1 |
| 24 | 3505 | 1688 | 13.56 | 1 |
| 25 | 3515 | 3400 | 10.78 | 1 |
| 26 | 3525 | 5576 | 14.41 | 1 |
| 27 | 3525 | 5937 | 18.25 | 1 |
| 28 | 3630 | 2851 | 13.20 | 1 |
| 29 | 3809 | 365 | 13.18 | 1 |
| 30 | 3902 | 5544 | 14.91 | 1 |
| 31 | 4034 | 2059 | 11.75 | 1 |
| 32 | 4103 | 1161 | 17.17 | 1 |
| 33 | 4193 | 2653 | 10.85 | 1 |
| 34 | 4332 | 2028 | 18.07 | 1 |
| 35 | 4554 | 2235 | 13.45 | 2 |
| 36 | 4665 | 1732 | 12.91 | 2 |

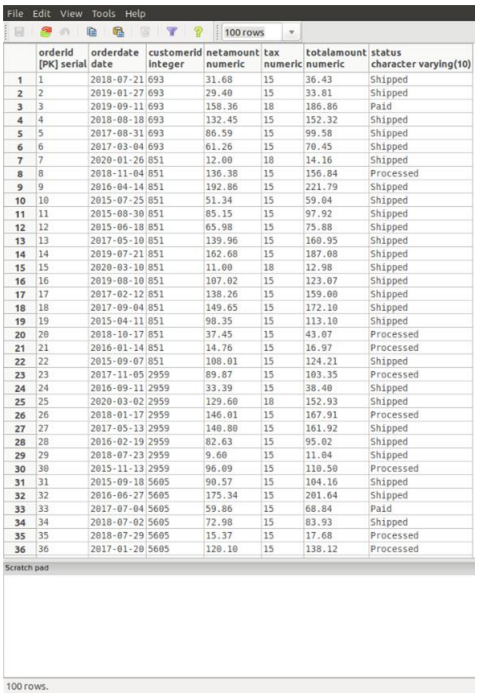
Scratch pad

c) El procedimiento almacenado que proponemos para este apartado es la siguiente:



```
/*Declarar función con sus argumentos*/
CREATE OR REPLACE FUNCTION setOrderAmount() RETURNS void AS $$
    /*Actualiza orders con el netamount y el totalamount correspondiente*/
    UPDATE orders
    SET    netamount = results.netamount,
           totalamount = results.totalamount
    FROM   (        /*Selecciona los orderdetails con su precio neto toltal (precio*cantidad)*/
                    /*y su totalamount total, (precio*cantidad*impuestos), ordenado por orderid*/
                    SELECT      orderdetail.orderid,
                                Sum(price * quantity)                                   AS netamount,
                                Round(Cast(Sum(price * quantity)*(1+tax/100) AS NUMERIC), 2) AS totalamount
                    FROM       orderdetail
                    INNER JOIN orders ON
                        orders.orderid = orderdetail.orderid
                    GROUP BY
                        orderdetail.orderid,
                        tax
           ) AS results
    WHERE  orders.orderid = results.orderid
$$ LANGUAGE SQL;


SELECT setOrderAmount();
```

Esta función actualiza los campos de netamount y totalamount de la tabla orders. Primero obtenemos una tabla intermedia con las columnas orderid, netamount (es la suma total del precio de cada producto por la cantidad del mismo), totalamount (es la suma total del precio de cada producto por la cantidad y los impuestos del mismo redondeado a 2 decimales) resultante de hacer el join correspondiente de la tabla orderdetail con orders. Estos datos se devuelven agrupados por orderid y tax, tras lo cual sirven para actualizar los campos netamount y totalamount del order correspondiente.




| | orderid [PK] serial | orderdate date | customerid integer | netamount numeric | tax numeric | totalamount numeric | status character varying(10) |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2018-07-21 | 693 | 31.68 | 15 | 36.43 | Shipped |
| 2 | 2 | 2019-01-27 | 693 | 29.40 | 15 | 33.81 | Shipped |
| 3 | 3 | 2019-09-11 | 693 | 158.36 | 18 | 186.86 | Paid |
| 4 | 4 | 2018-08-18 | 693 | 132.45 | 15 | 152.32 | Shipped |
| 5 | 5 | 2017-08-31 | 693 | 86.59 | 15 | 99.58 | Shipped |
| 6 | 6 | 2017-03-04 | 693 | 61.26 | 15 | 70.45 | Shipped |
| 7 | 7 | 2020-01-26 | 851 | 12.00 | 18 | 14.16 | Shipped |
| 8 | 8 | 2018-11-04 | 851 | 136.38 | 15 | 156.84 | Processed |
| 9 | 9 | 2016-04-14 | 851 | 192.86 | 15 | 221.79 | Shipped |
| 10 | 10 | 2015-07-25 | 851 | 51.34 | 15 | 59.04 | Shipped |
| 11 | 11 | 2015-08-30 | 851 | 85.15 | 15 | 97.92 | Shipped |
| 12 | 12 | 2015-06-18 | 851 | 65.98 | 15 | 75.88 | Shipped |
| 13 | 13 | 2017-05-10 | 851 | 139.96 | 15 | 160.95 | Shipped |
| 14 | 14 | 2019-07-21 | 851 | 162.68 | 15 | 187.08 | Shipped |
| 15 | 15 | 2020-03-10 | 851 | 11.00 | 18 | 12.98 | Shipped |
| 16 | 16 | 2019-08-10 | 851 | 107.02 | 15 | 123.07 | Shipped |
| 17 | 17 | 2017-02-12 | 851 | 138.26 | 15 | 159.00 | Shipped |
| 18 | 18 | 2017-09-04 | 851 | 149.65 | 15 | 172.10 | Shipped |
| 19 | 19 | 2015-04-11 | 851 | 98.35 | 15 | 113.10 | Shipped |
| 20 | 20 | 2018-10-17 | 851 | 37.45 | 15 | 43.07 | Processed |
| 21 | 21 | 2016-01-14 | 851 | 14.76 | 15 | 16.97 | Processed |
| 22 | 22 | 2015-09-07 | 851 | 108.01 | 15 | 124.21 | Shipped |
| 23 | 23 | 2017-11-05 | 2959 | 89.87 | 15 | 103.35 | Processed |
| 24 | 24 | 2016-09-11 | 2959 | 33.39 | 15 | 38.40 | Shipped |
| 25 | 25 | 2020-03-02 | 2959 | 129.60 | 18 | 152.93 | Shipped |
| 26 | 26 | 2018-01-17 | 2959 | 146.01 | 15 | 167.91 | Processed |
| 27 | 27 | 2017-05-13 | 2959 | 140.80 | 15 | 161.92 | Shipped |
| 28 | 28 | 2016-02-19 | 2959 | 82.63 | 15 | 95.02 | Shipped |
| 29 | 29 | 2018-07-23 | 2959 | 9.60 | 15 | 11.04 | Shipped |
| 30 | 30 | 2015-11-13 | 2959 | 96.09 | 15 | 110.50 | Processed |
| 31 | 31 | 2015-09-18 | 5605 | 90.57 | 15 | 104.16 | Shipped |
| 32 | 32 | 2016-06-27 | 5605 | 175.34 | 15 | 201.64 | Shipped |
| 33 | 33 | 2017-07-04 | 5605 | 59.86 | 15 | 68.84 | Paid |
| 34 | 34 | 2018-07-02 | 5605 | 72.98 | 15 | 83.93 | Shipped |
| 35 | 35 | 2018-07-29 | 5605 | 15.37 | 15 | 17.68 | Processed |
| 36 | 36 | 2017-01-20 | 5605 | 120.10 | 15 | 138.12 | Processed |

Como se puede observar, las columnas de netamount y totalamount se rellenan correctamente, hemos hecho los cálculos correspondientes con una muestra razonablemente grande de orders y el resultado de los campos netamount y totalamount que actualiza nuestra función y el de los cálculos numéricos coinciden.

d) La función que proponemos para este apartado es la siguiente:

```sql
/*Declarar función con sus argumentos*/
CREATE OR REPLACE FUNCTION getTopVentas(año1 double precision, año2 double precision) RETURNS TABLE (Año DOUBLE PRECISION, Pelicula CHARACTER varying(255),
                                                                                                        Ventas BIGINT) AS $$

    SELECT
        /*Selecccionar año, título y número de ventas*/
        result2.year,
        imdb_movies.movietitle,
        sales
    FROM(
            /*Seleccionar todas las pelícluas con distinto año y mayor número de ventas*/
            SELECT DISTINCT ON(year)
                    year, movie, sales
            FROM (
                    SELECT
                        /*Seleccionar año, película y ventas de esa película por año, ordenado por año y película*/
                        EXTRACT(year from orders.orderdate) as year,
                        products.movieid as movie,
                        sum(orderdetail.quantity) as sales

                    FROM orderdetail
                    INNER JOIN orders ON
                        orderdetail.orderid = orders.orderid
                    INNER JOIN products ON
                        products.prod_id = orderdetail.prod_id
                    /*Filtramos los orders que son del carrito*/
                    WHERE orders.status <> ''
                    GROUP BY year, products.movieid
                    ) as result
            WHERE
                /*Filtrar y ordenar por años pasados como argumento*/
                year >= $1
                and year <= $2
            ORDER BY year, sales DESC) as result2

    INNER JOIN imdb_movies on
        imdb_movies.movieid = result2.movie


$$ LANGUAGE SQL;
```

Esta función devuelve el año, título y número de ventas de las películas más vendidas entre dos años pasados como argumentos a la función. Primero obtenemos una tabla intermedia con las columnas year (año de la orderdate), movie (movieid del producto), sales (cantidad de ese producto) resultante de hacer los joins correspondientes de la tabla orderdetail con orders y products, además aprovechamos para filtrar los orders que pertenecen al carrito (status = ”). Estos datos se devuelven agrupados por year y movie, tras lo cual se ordenan por year y sales de forma descendente y se filtran para obtener aquellos cuyo year se encuentra entre los años pasados como argumentos. A continuación se escogen solo los resultados con años diferentes (que, al estar ordenados de forma descendente, son aquellos que más sales tienen por año) y se hace un join de estos datos con la tabla imdb_movies para obtener el título de la película a partir de su movieid.

```
SELECT *
FROM getTopVentas(2018, 2020)
```

Output pane

Data Output | Explain | Messages | History

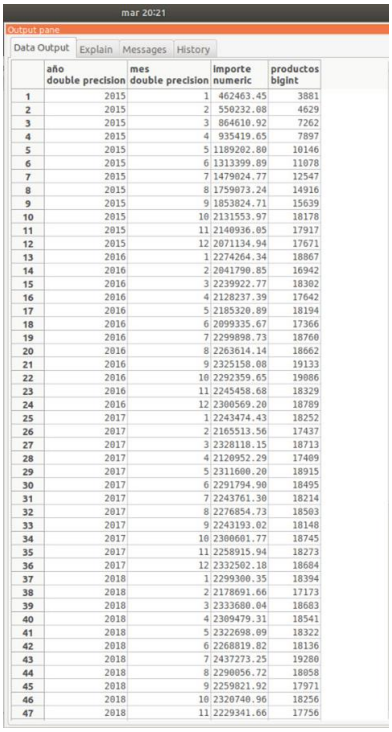
| | año<br>double precision | película<br>character varying | ventas<br>bigint |
|---|---|---|---|
| 1 | 2018 | Wizard of Oz, The (1939) | 134 |
| 2 | 2019 | Life Less Ordinary, A (1997) | 134 |
| 3 | 2020 | Life with Mikey (1993) | 57 |

Como se puede observar, el resultado de ejecutar nuestra función con los argumentos 2018, 2020, es decir, las películas más vendidas del 2018, 2019 y 2020. Se obtiene que en 2018 la más vendida fue 'Wizard of Oz, The (1939)' con 134 ventas, en 2019 'Life Less Ordinary, A (1997)' con 134 ventas y en 2020 es 'Life with Mikey (1993)' con 57 ventas.
Estos resultados los hemos comprobado ayudándonos de los resultados parciales de las sub-consultas que conforman nuestra función.

e) La función que proponemos para este apartado es la siguiente:

```sql
CREATE OR REPLACE FUNCTION getTopMonths(num_products int, importe int) RETURNS TABLE (Año DOUBLE PRECISION, Mes DOUBLE PRECISION, Importe NUMERIC, Productos BIGINT) AS $$

  SELECT *
  FROM (  SELECT
            /*Devuelve el año, mes, importe acumulado y productos acumulados, agrupados por año y mes*/
            EXTRACT(year from orders.orderdate) as Año,
            EXTRACT(month from orders.orderdate) as Mes,
            sum(orders.totalamount) as Importe,
            sum(orderdetail.quantity) as Productos
          FROM orderdetail
          INNER JOIN orders ON
            orderdetail.orderid = orders.orderid
          WHERE orders.status <> ''
          GROUP BY
              Año,
              Mes
        ) as result

  WHERE
      /*Filtrar por productos e importe de los por argumentos*/
      result.Productos >= $1
      OR result.Importe >= $2;

$$ LANGUAGE SQL;
```

Esta función devuelve el año, el mes, el importe obtenido y los productos vendidos en ese mes de los meses en los que se ha superado el ingreso y/o número de productos vendidos especificado en los argumentos de la función. Primero obtenemos una tabla intermedia con las columnas Año (año de la orderdate), Mes (mes de la orderdate), Importe (suma total de los totalamount de las orders de ese mes) y Productos (suma total de las quantity de los orderdetail de ese mes) resultante de hacer el join correspondiente de la tabla orderdetail con orders, además aprovechamos para filtrar los orders que pertenecen al carrito (status=’’). Estos datos se devuelven agrupados por Año y Mes, tras lo cual se filtran para obtener aquellos cuyo Productos sea mayor o igual que el argumento num_products y/o Importe sea mayor o igual que el argumento importe de la función.

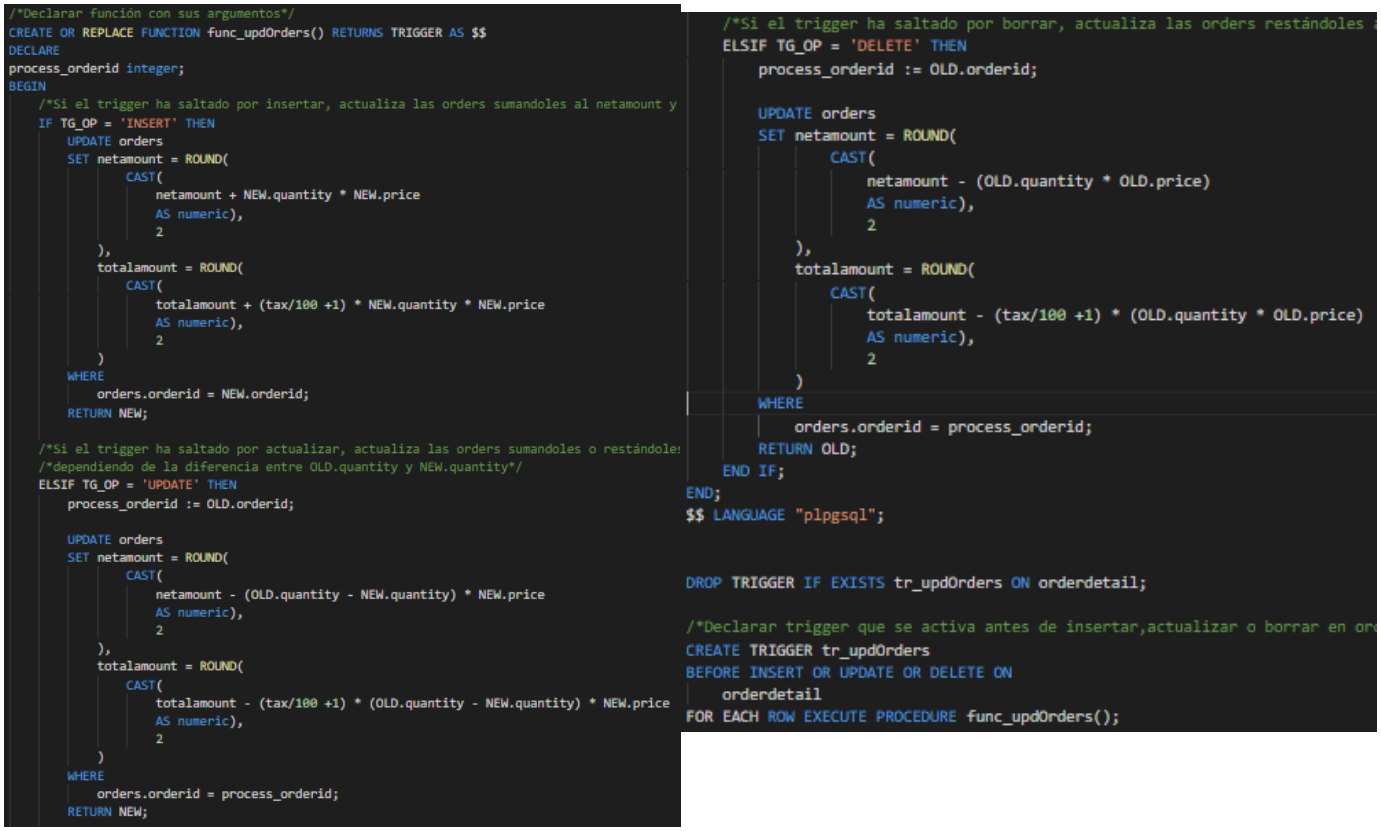



| | año double precision | mes double precision | importe numeric | productos bigint |
|---|---|---|---|---|
| 1 | 2015 | 1 | 462463.45 | 3881 |
| 2 | 2015 | 2 | 550232.08 | 4629 |
| 3 | 2015 | 3 | 864610.92 | 7262 |
| 4 | 2015 | 4 | 935419.65 | 7897 |
| 5 | 2015 | 5 | 1189202.80 | 10146 |
| 6 | 2015 | 6 | 1313399.89 | 11078 |
| 7 | 2015 | 7 | 1479024.77 | 12547 |
| 8 | 2015 | 8 | 1759073.24 | 14916 |
| 9 | 2015 | 9 | 1853824.71 | 15639 |
| 10 | 2015 | 10 | 2131553.97 | 18178 |
| 11 | 2015 | 11 | 2140936.05 | 17917 |
| 12 | 2015 | 12 | 2071134.94 | 17671 |
| 13 | 2016 | 1 | 2274264.34 | 18867 |
| 14 | 2016 | 2 | 2041790.85 | 16942 |
| 15 | 2016 | 3 | 2239922.77 | 18302 |
| 16 | 2016 | 4 | 2128237.39 | 17642 |
| 17 | 2016 | 5 | 2185320.89 | 18194 |
| 18 | 2016 | 6 | 2099335.67 | 17366 |
| 19 | 2016 | 7 | 2299898.73 | 18760 |
| 20 | 2016 | 8 | 2263614.14 | 18662 |
| 21 | 2016 | 9 | 2325158.08 | 19133 |
| 22 | 2016 | 10 | 2292359.65 | 19086 |
| 23 | 2016 | 11 | 2245458.68 | 18329 |
| 24 | 2016 | 12 | 2300569.20 | 18789 |
| 25 | 2017 | 1 | 2243474.43 | 18252 |
| 26 | 2017 | 2 | 2165513.56 | 17437 |
| 27 | 2017 | 3 | 2328118.15 | 18713 |
| 28 | 2017 | 4 | 2120952.29 | 17409 |
| 29 | 2017 | 5 | 2311600.20 | 18915 |
| 30 | 2017 | 6 | 2291794.90 | 18495 |
| 31 | 2017 | 7 | 2243761.30 | 18214 |
| 32 | 2017 | 8 | 2276854.73 | 18503 |
| 33 | 2017 | 9 | 2243193.02 | 18148 |
| 34 | 2017 | 10 | 2300601.77 | 18745 |
| 35 | 2017 | 11 | 2258915.94 | 18273 |
| 36 | 2017 | 12 | 2332502.18 | 18684 |
| 37 | 2018 | 1 | 2299300.35 | 18394 |
| 38 | 2018 | 2 | 2178691.66 | 17173 |
| 39 | 2018 | 3 | 2333680.04 | 18683 |
| 40 | 2018 | 4 | 2309479.31 | 18541 |
| 41 | 2018 | 5 | 2322698.09 | 18322 |
| 42 | 2018 | 6 | 2268819.82 | 18136 |
| 43 | 2018 | 7 | 2437273.25 | 19280 |
| 44 | 2018 | 8 | 2290056.72 | 18058 |
| 45 | 2018 | 9 | 2259821.92 | 17971 |
| 46 | 2018 | 10 | 2320740.96 | 18256 |
| 47 | 2018 | 11 | 2229341.66 | 17756 |

Como se puede observar, el resultado de ejecutar nuestra función con los argumentos 19000, 320000, es decir, los meses en los que se han superado los 320000 euros de ingresos y/o los 19000 productos vendidos. Se obtiene una lista de meses ordenados por año y mes en los que se cumple la condición antes mencionada.

Estos resultados los hemos comprobado ayudándonos de los resultados parciales de las sub-consultas que conforman nuestra función.

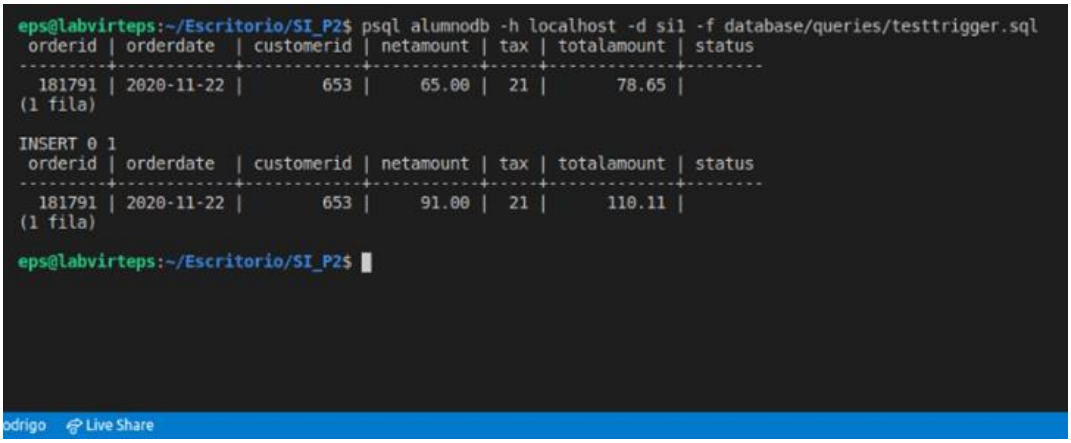
g) El trigger que proponemos para este apartado es el siguiente:



```
/*Declarar función con sus argumentos*/
CREATE OR REPLACE FUNCTION func_updOrders() RETURNS TRIGGER AS $$
DECLARE
process_orderid integer;
BEGIN
    /*Si el trigger ha saltado por insertar, actualiza las orders sumandoles al netamount y
    IF TG_OP = 'INSERT' THEN
        UPDATE orders
        SET netamount = ROUND(
                CAST(
                    netamount + NEW.quantity * NEW.price
                    AS numeric),
                    2
            ),
            totalamount = ROUND(
                CAST(
                    totalamount + (tax/100 +1) * NEW.quantity * NEW.price
                    AS numeric),
                    2
            )
        WHERE
            orders.orderid = NEW.orderid;
        RETURN NEW;

    /*Si el trigger ha saltado por actualizar, actualiza las orders sumandoles o restándole
    /*dependiendo de la diferencia entre OLD.quantity y NEW.quantity*/
    ELSIF TG_OP = 'UPDATE' THEN
        process_orderid := OLD.orderid;

        UPDATE orders
        SET netamount = ROUND(
                CAST(
                    netamount - (OLD.quantity - NEW.quantity) * NEW.price
                    AS numeric),
                    2
            ),
            totalamount = ROUND(
                CAST(
                    totalamount - (tax/100 +1) * (OLD.quantity - NEW.quantity) * NEW.price
                    AS numeric),
                    2
            )
        WHERE
            orders.orderid = process_orderid;
        RETURN NEW;
    /*Si el trigger ha saltado por borrar, actualiza las orders restándoles
    ELSIF TG_OP = 'DELETE' THEN
        process_orderid := OLD.orderid;

        UPDATE orders
        SET netamount = ROUND(
                CAST(
                    netamount - (OLD.quantity * OLD.price)
                    AS numeric),
                    2
            ),
            totalamount = ROUND(
                CAST(
                    totalamount - (tax/100 +1) * (OLD.quantity * OLD.price)
                    AS numeric),
                    2
            )
        WHERE
            orders.orderid = process_orderid;
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE "plpgsql";


DROP TRIGGER IF EXISTS tr_updOrders ON orderdetail;

/*Declarar trigger que se activa antes de insertar,actualizar o borrar en or
CREATE TRIGGER tr_updOrders
BEFORE INSERT OR UPDATE OR DELETE ON
    orderdetail
FOR EACH ROW EXECUTE PROCEDURE func_updOrders();
```

Este trigger actúa antes de insertar, actualizar o borrar alguna entrada de la tabla orderdetail. Básicamente, dependiendo el tipo de operación por la que se invoque el trigger (TG_OP) hará una cosa u otra. Si la operación es por inserción, aumenta, con la cantidad adecuada (valor anterior + valor nuevo) el netamount y el totalamount de la order con el mismo id que la orderid modificada. Si es update, hace lo mismo pero esta vez puede aumentar o disminuir los campos netamount y totalamount, dependiendoo de el resultado de la diferencia de OLD.quantity - NEW.quantity. Si es delete, solamente puede disminuir los campos netamount y totalamount por la cantidad correspondiente al orderdetail eliminado.

Para

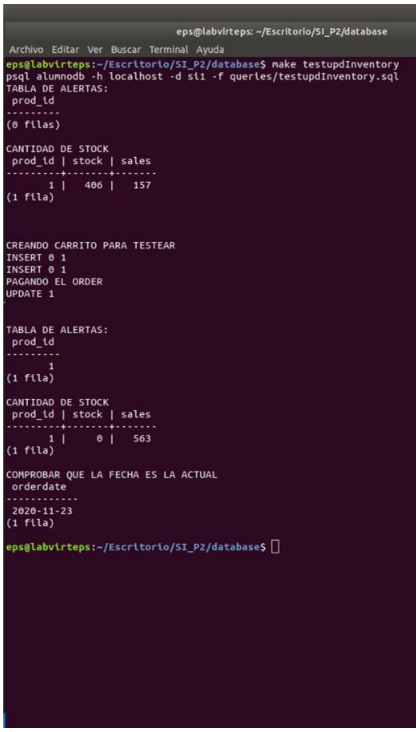



comprobar el correcto funcionamiento del trigger, hemos creado un script 'testupdOrder.sql' con algún test de prueba, el resto de tests los hemos realizado a mano. Como se ve en la imagen superior, al ejecutar el script y añadir un nuevo orderdetail a la order 181791, el netamount y el totalamount se ven incrementados.

h) El trigger que proponemos para este apartado es el siguiente:

```
/*Declarar función con sus argumentos*/
CREATE OR REPLACE FUNCTION func_updInventory() RETURNS TRIGGER AS $$
DECLARE
BEGIN
    IF NEW.STATUS <> 'Paid' OR OLD.STATUS <> '' THEN
        RETURN NEW;
    END IF;

    /*Resto y sumo las cantidades del pedido*/
    UPDATE inventory
    SET
        sales = sales + products.quantity,
        stock = stock - products.quantity
    FROM ( /*obtengo las cantidades del orderdetails*/
        SELECT prod_id, quantity
        FROM orderdetail
        WHERE orderid = NEW.orderid
    ) as products
    WHERE products.prod_id = inventory.prod_id;

    /*Actualizamos la fecha a la de hoy*/
    NEW.orderdate = NOW();

    /*Cuando ya tenemos los stock actualizado solo tenemos que obtener los que
    tienen stock 0 y es del order que se ha actualizado*/
    INSERT INTO alertas(prod_id)
        SELECT orderdetail.prod_id
        FROM inventory
        INNER JOIN orderdetail ON
            inventory.prod_id = orderdetail.prod_id
        WHERE
            inventory.stock = 0
            AND orderdetail.orderid = NEW.orderid;

    RETURN NEW;


END;
$$ LANGUAGE "plpgsql";
```

Este trigger actúa antes de actualizar alguna entrada de la tabla orders. Lo primero que hace es comprobar si la actualización se ha debido, entre otros, a un cambio del estado del pedido a 'Paid', en caso contrario retornará sin hacer nada. Si lo anterior se cumple, se actualiza en el inventario los campos de sales y stock con la correspondiente cantidad del producto de cada uno de los productos de las orderdetail de el order modificado. A continuación, se actualiza el orderdate a la fecha actual y se inserta en la tabla alertas todos aquellos prod_id cuyo stock se haya quedado a cero tras la compra.




Para comprobar el correcto funcionamiento del trigger, hemos creado un script 'testupdInventory.sql' con algunos tests de prueba, el resto de tests los hemos realizado a mano. Como se ve en la imagen superior, al ejecutar el script y actualizar el estado del order 999999 a 'Paid' a la tabla de alertas se le añade el producto 1 y en la tabla de inventario el stock disminuye y el sales aumenta, así como se actualiza la fecha de la order.