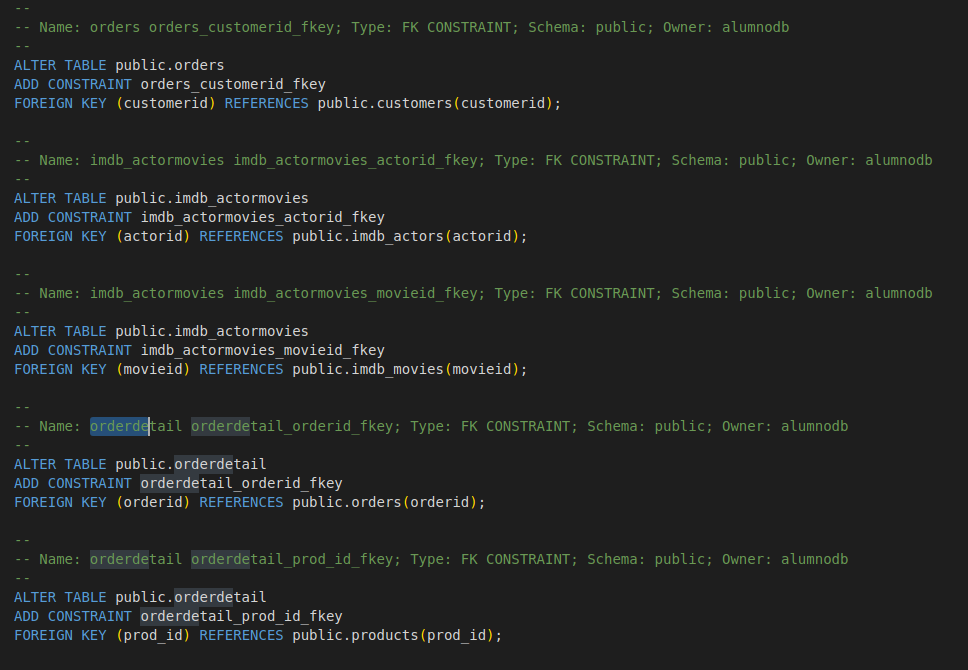
Práctica 2. Conceptos Avanzados sobre Bases de Datos Relacionales y Optimización de Consultas

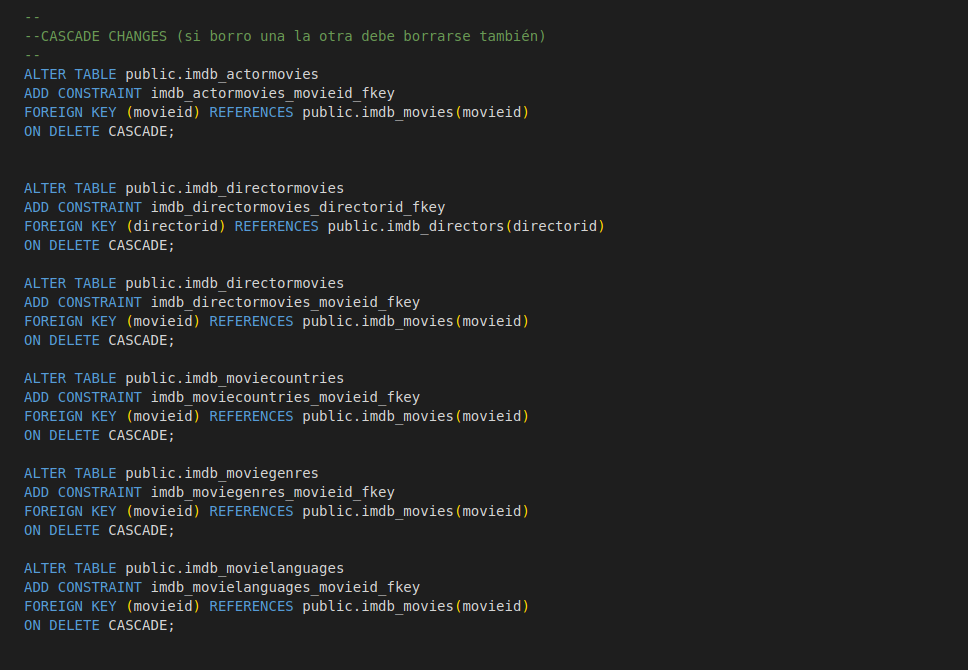
**Diseño de la BD**

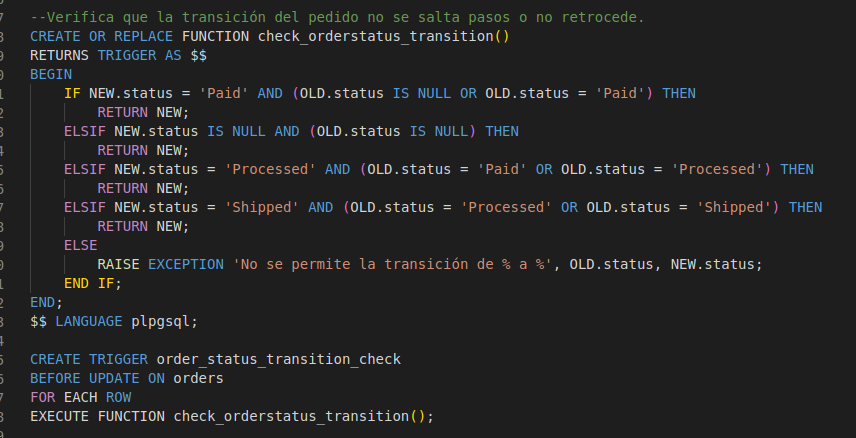
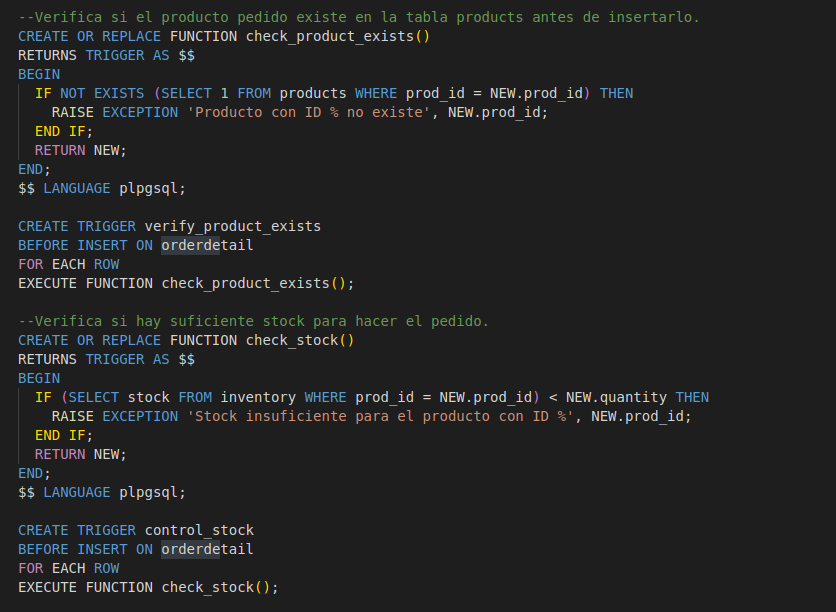
**a)**-Obtener el Diagrama Entidad-Relación correspondiente.

-Completar aquellos aspectos que se consideren necesarios

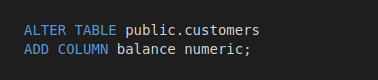
Los cambios realizados han sido los siguientes:

* Añadir todas las claves foráneas necesarias:
* Añadir cambios en cascada:

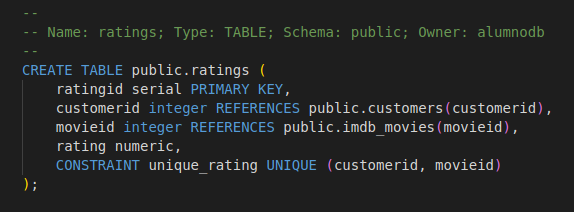


-Los triggers necesarios:  


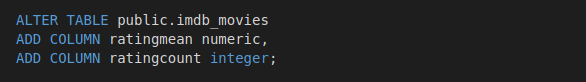
-Un campo balance en la tabla customers, para guardar el saldo de los clientes.



-Una nueva tabla ratings para guardar las valoraciones que ha dado cada usuario a cada película.



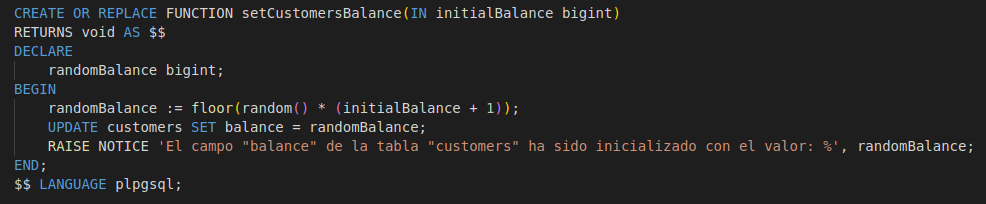
-Añadir dos campos a la tabla imdb\_movies, para contener la valoración media ratingmean y el número de valoraciones ratingcount, de cada película.



-Aumentar el tamaño del campo password en la tabla customers.



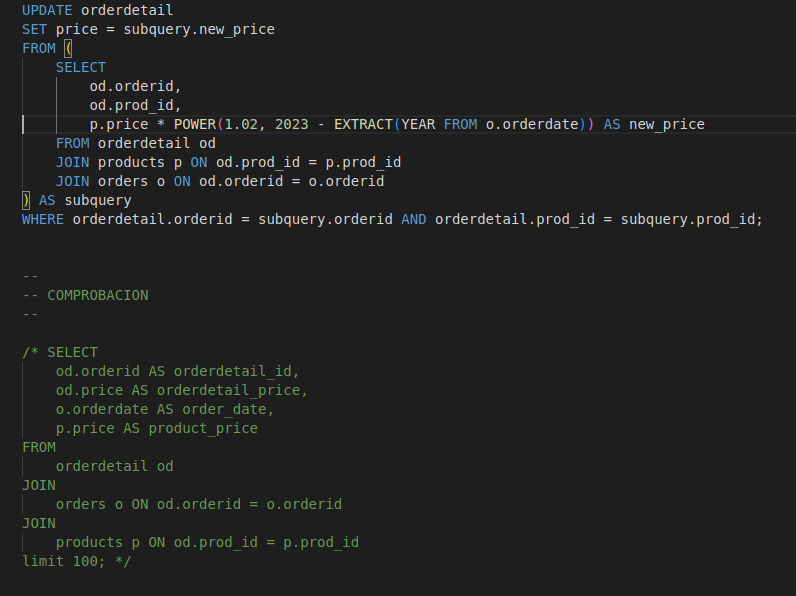
-Crear un procedimiento que inicialice el campo balance de la tabla customers a un número aleatorio entre 0 y N.



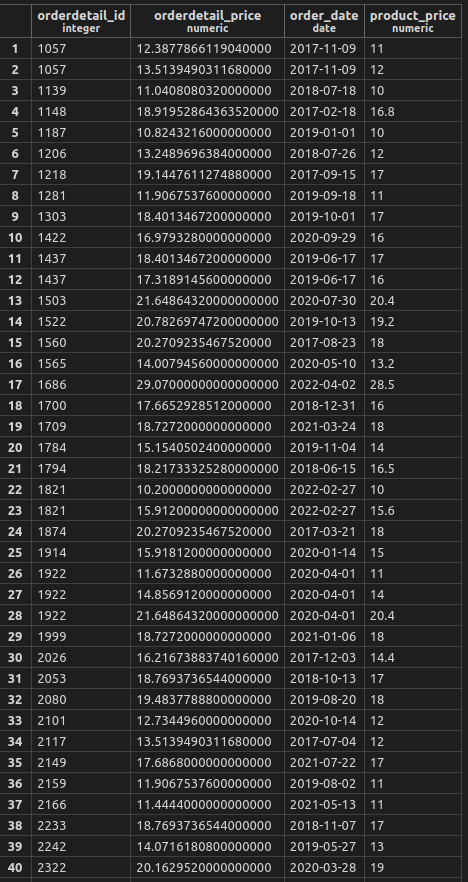
-Añadir a actualiza.sql una llamada a dicho procedimiento, con N = 200.



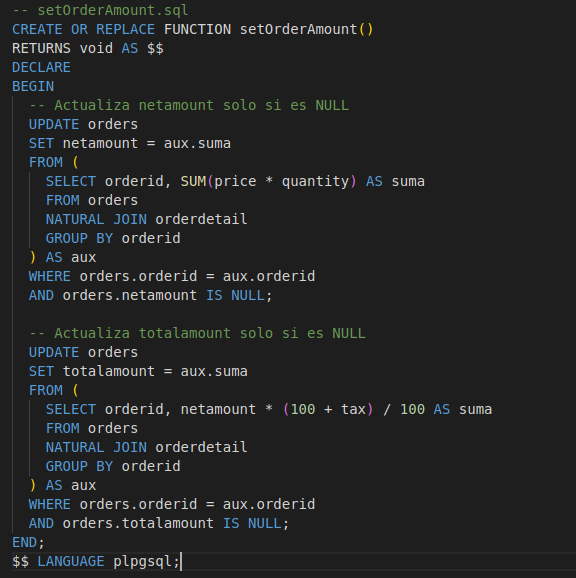
b) Sabiendo que los precios de las películas se han ido incrementando un 2% anualmente, elaborar la consulta setPrice.sql que complete la columna price de la tabla orderdetail, sabiendo que el precio actual es el de la tabla products.



La subconsulta calcula el nuevo precio multiplicando el precio actual del producto por la potencia de 1.02 elevada a la diferencia entre el año 2023 y el año de la fecha de pedido, lo cual lo hallamos con la función EXTRACT. Luego, la cláusula WHERE vincula las filas de la tabla "orderdetail" con las correspondientes de la subconsulta mediante las claves de orden y producto, permitiendo así la actualización de los precios.

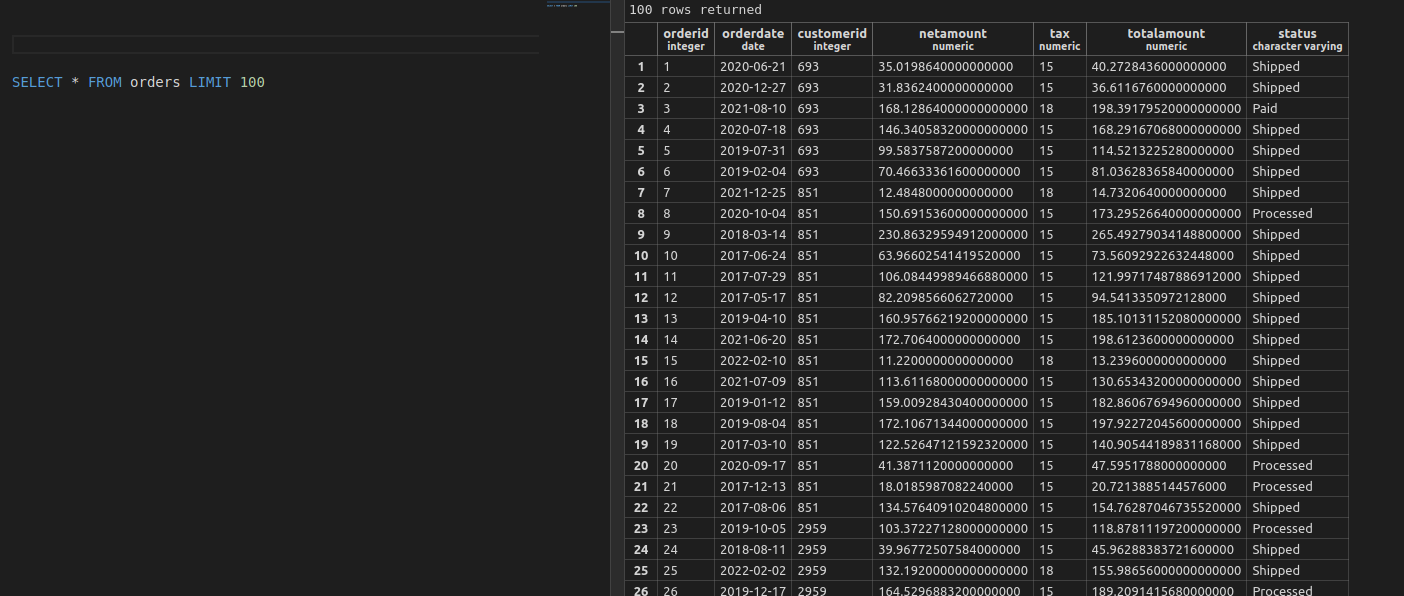
Además hicimos la comprobación para asegurarnos de que se hacía correctamente. Este fue el resultado:  


c)realizar un procedimiento almacenado setOrderAmount.



Primero, realizamos una actualización de "netamount", calculando la suma del producto de precio y cantidad agrupado por "orderid" en la tabla "orderdetail". Luego, actualizamos "totalamount" multiplicando "netamount" por la suma del impuesto más 100, dividido por 100, por ejemplo si el impuesto es del 21%, debemos multiplicar su valor por 1.21. Ambas actualizaciones solo ocurren si los valores actuales de "netamount" o "totalamount" son NULL, que se comprueba en en el WHERE.

Tras llamar al procedimiento y que se hicieran los cambios, hicimos la siguiente comprobación con la cual nos aseguramos de que se cambiaban correctamente:

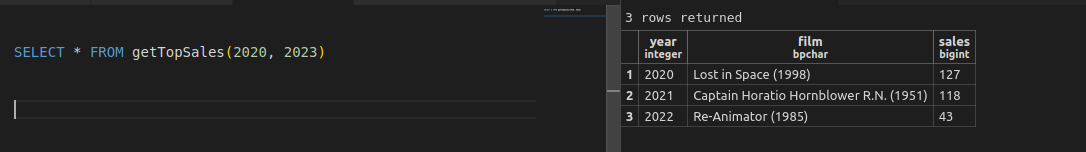
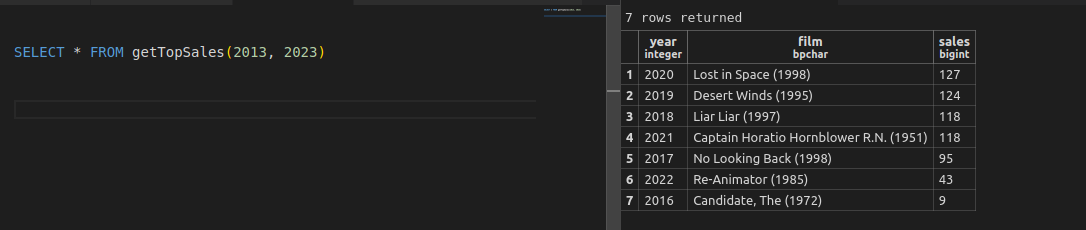


d)Realizar la función getTopSales.



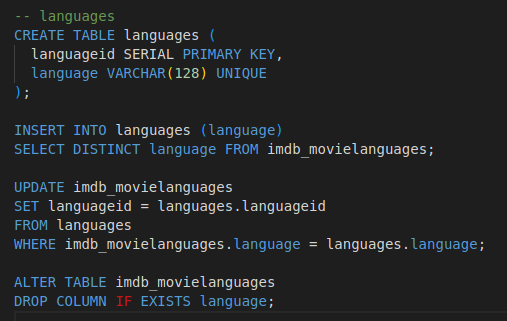
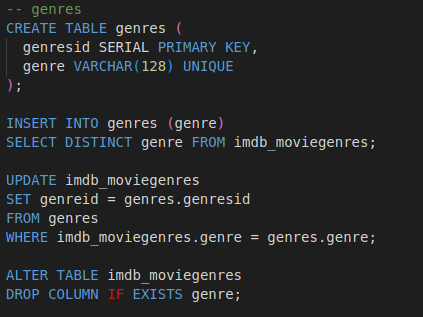
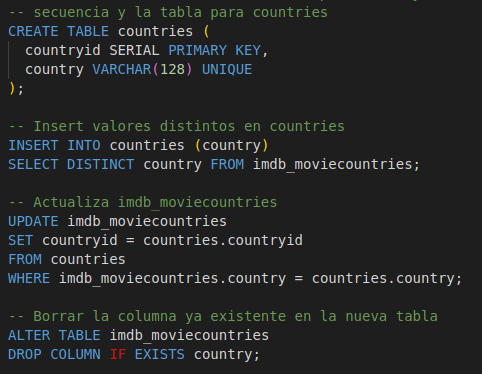
creando una tabla temporal llamada "temp\_results" para almacenar los resultados. Luego, utiliza bucles para iterar sobre los máximos de ventas para cada año y, dentro de cada año, encuentra la película con las ventas más altas. Los resultados se insertan en la tabla temporal. Finalmente, la función devuelve los datos año por año ordenados por ventas y elimina la tabla temporal al final.

Las pruebas realizadas fueron:

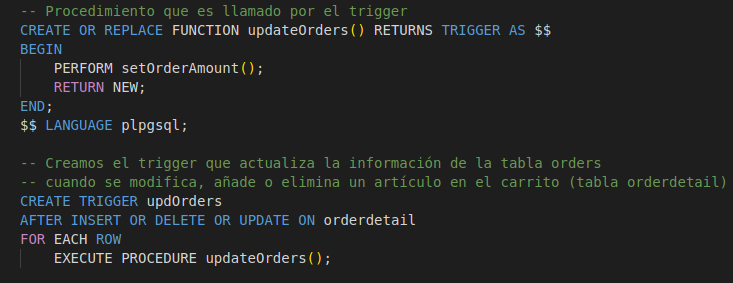


e)

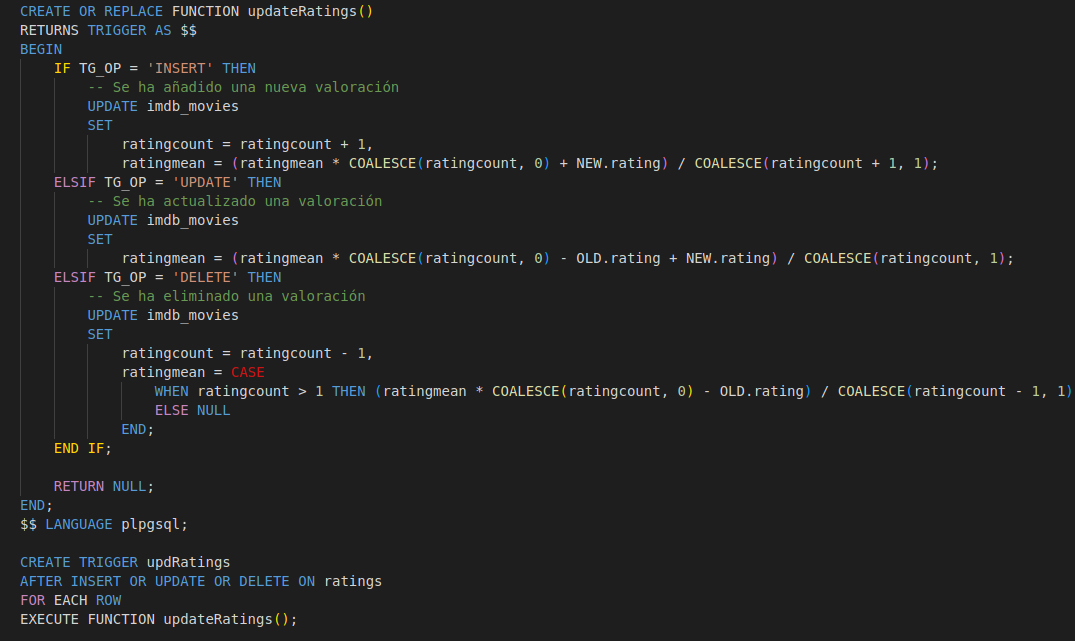
f)Crear las tablas correspondientes y convertir los atributos multivaluados moviecountries, moviegenres y movielanguages en relaciones entre la tabla movies y las tablas creadas.



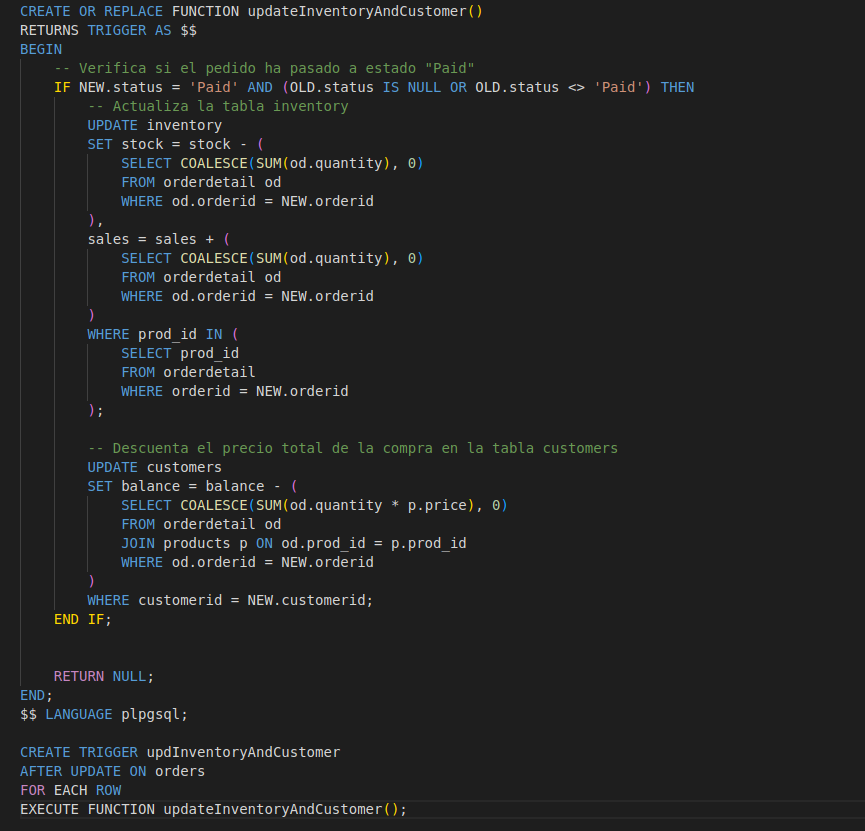
g)Realizar el trigger updOrders.



h)Realizar el trigger updRatings,

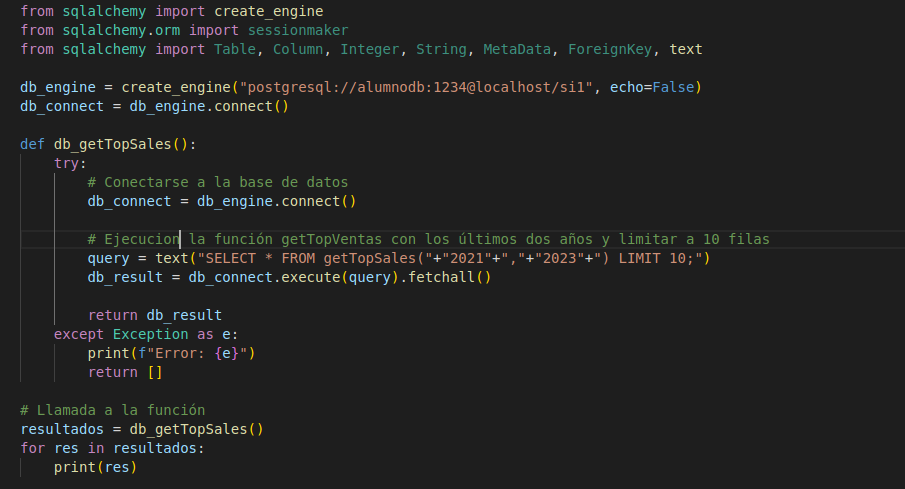


i)Crear el trigger updInventoryAndCustomer



**Integración con Python**

j) Utiliza SQLAlchemy para mostrar la tabla resultante en el apartado d) anterior, para los dos últimos años, y limitado a 10 filas.





**Optimización**

k)