**PosgreSQL DVD Rental Inventory**

Summary:

This project uses the DVD Rental sample database included in PostgreSQL. The scenario is that of the DVD Rental Company inventory manager who wants to determine how many copies of each DVD should be stocked on the shelves. To make this decision, the manager needs to know which DVDs are rented most frequently, and how many days they are rented for. This information needs to be store-specific because each store has their own stock of DVDs on the shelves.

The data used for the report includes rental information about the DVDs, such as the dates rented and returned. It also includes information identifying which store the DVDs are from, and what films they are.

There are two source tables needed to provide the data necessary for the detailed and summary sections of the report. These tables are the rental table and the inventory table. The two new tables created are the rental\_detail, and rental\_summary tables.

The fields that are included in the detailed and summary reports from the source data rental table are the rental \_id, rental\_date, and return\_date fields. The fields included from the source data inventory table are the the store\_id field, the film\_id field.

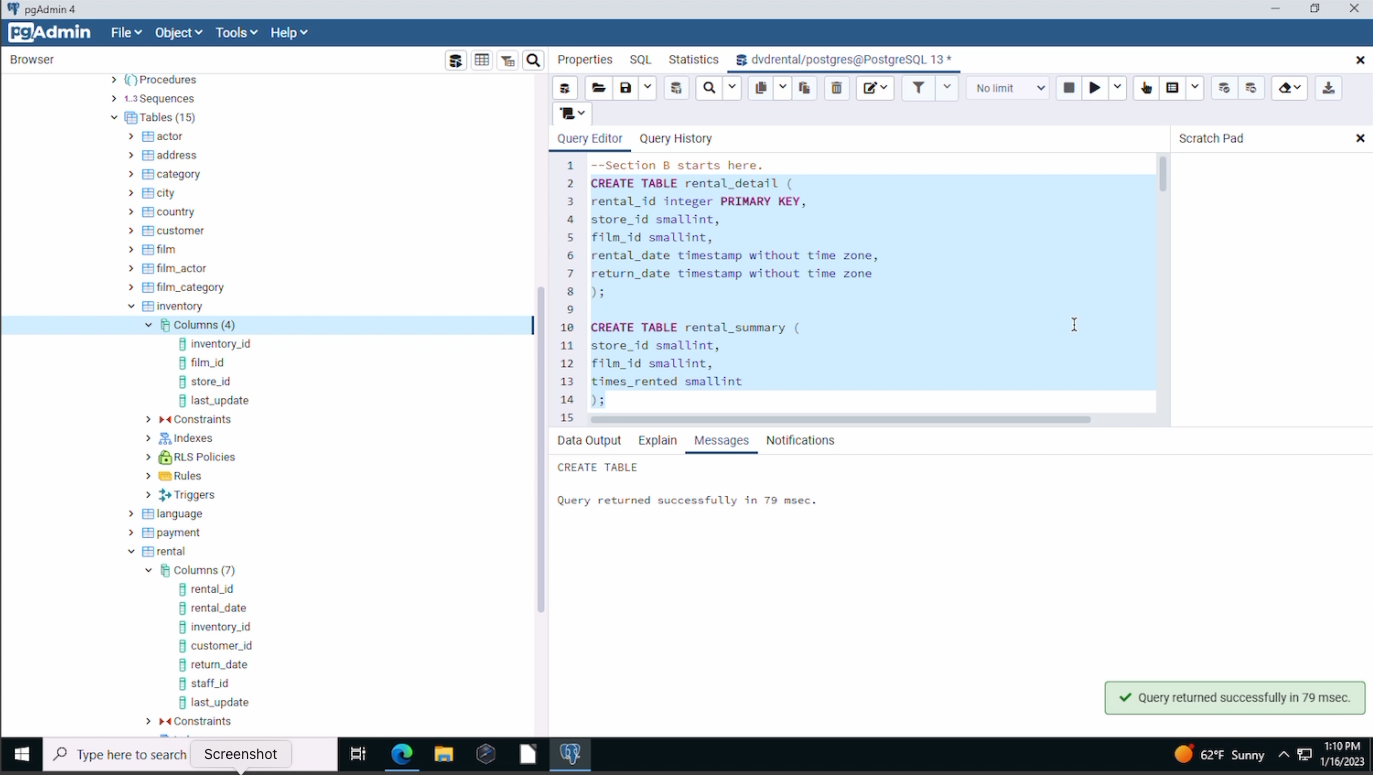
A data transformation included is the days\_rented\_out field in the detailed report which calculates the difference, in days and time, between the rental\_date, and the return\_date fields from the rental table. The days\_rented\_out field allows the inventory manager to quickly identify DVDs which were kept for an especially long period of time, and which were therefore unavailable for rental during that time.

The detail table provides a record of every rental record in the report. This will be useful because it will indicate how many total rentals have taken place. It will also show the number of days that a DVD was rented out for each rental. The summary table shows the total number of times rented for each DVD (film\_id) at each store.

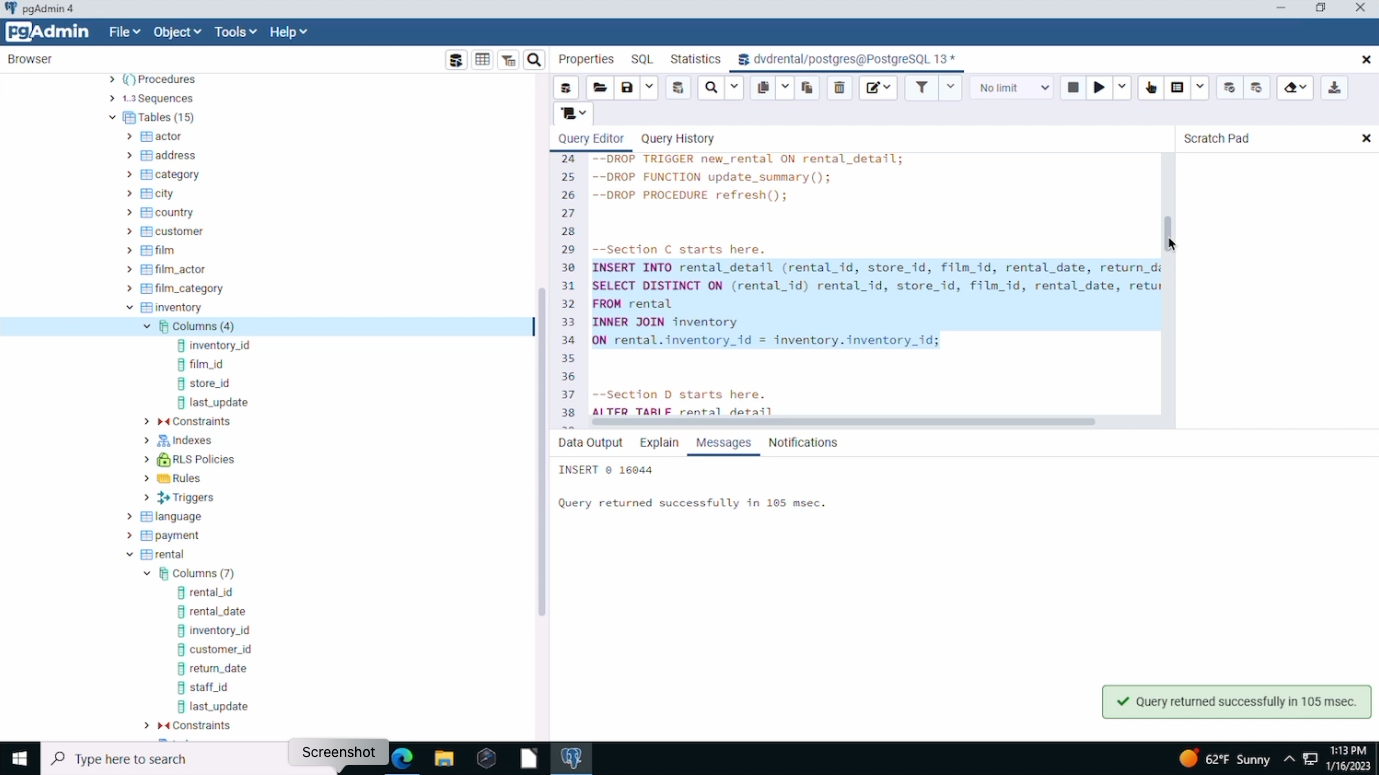
The report should be refreshed bi-weekly. The inventory manager needs to decide how often to evaluate the relative popularity of DVD titles to adjust and optimize the stock levels accordingly. If he does it too often though, weekly for example, it will take too much time away from other duties while providing relatively little benefit. However, monthly wouldn’t be frequent enough because the popularity of movie titles is likely to fluctuate significantly over a one-month span.

*Project execution screenshots:*

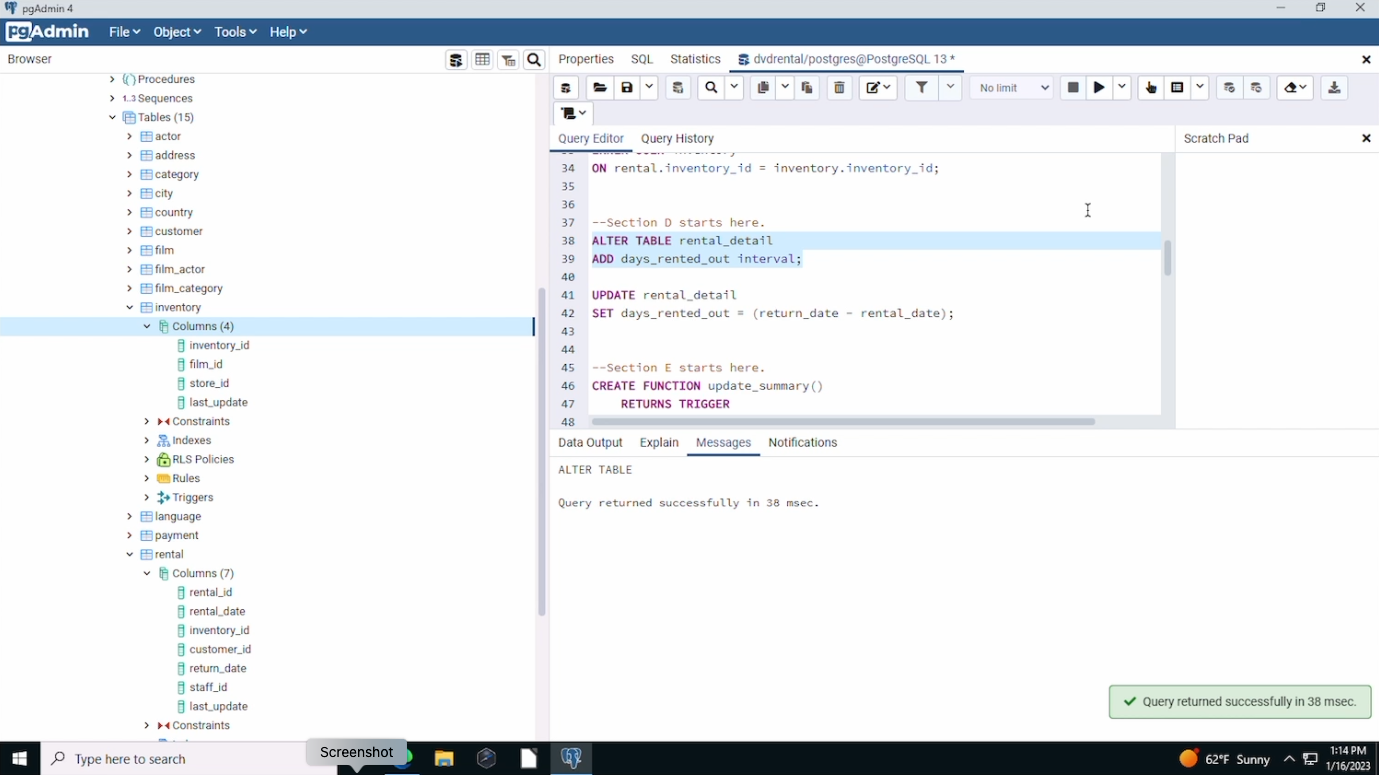
Create rental\_detail and rental\_summary tables:

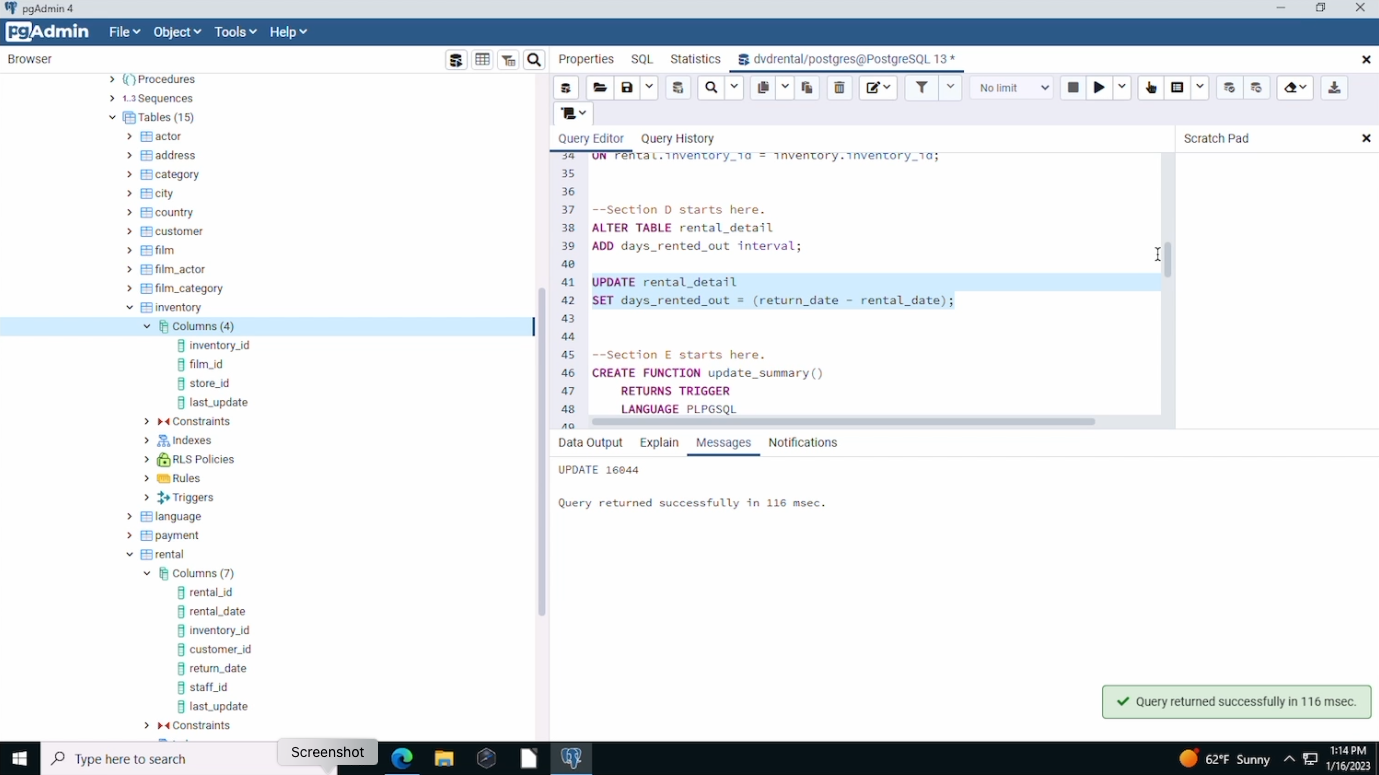


Extract data and insert into rental\_detail table:

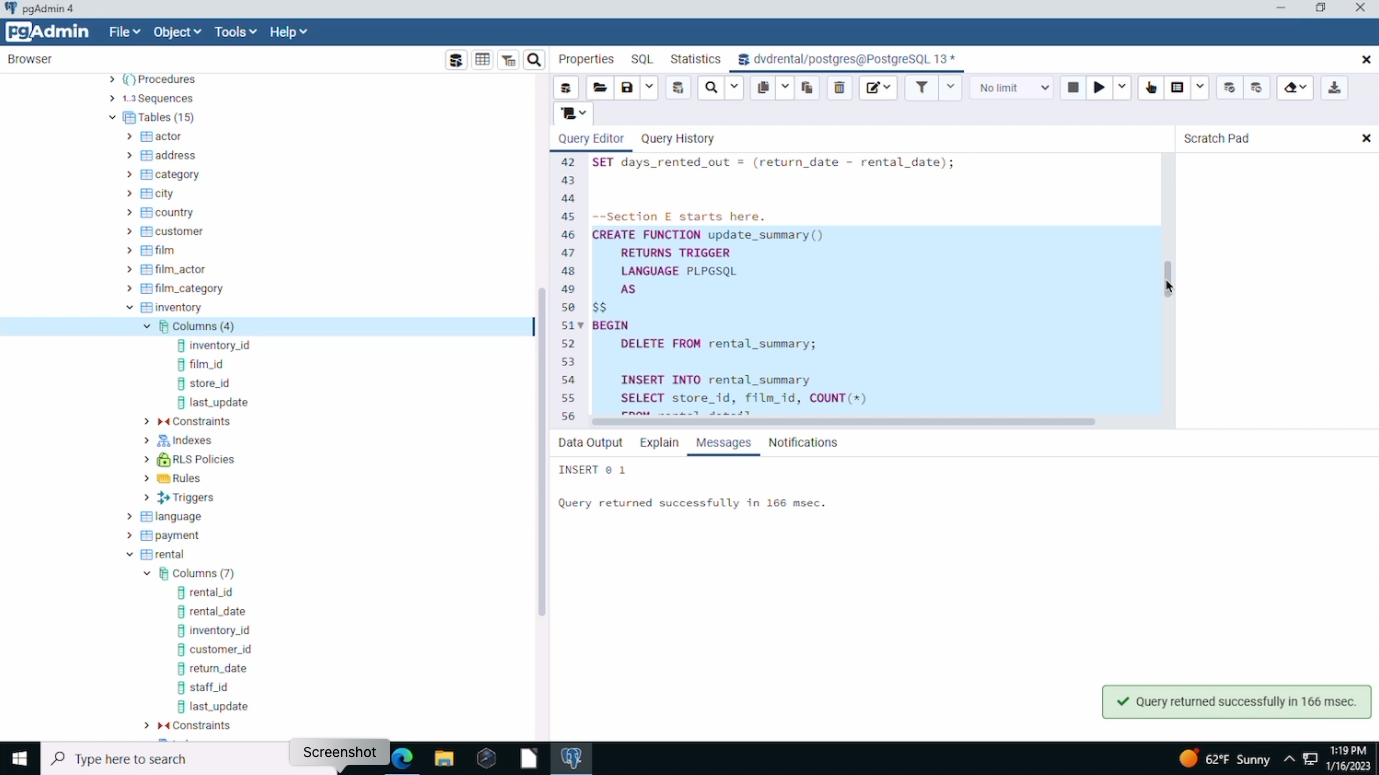


Data transformation. Add column days\_rented\_out, and set the timestamp interval:

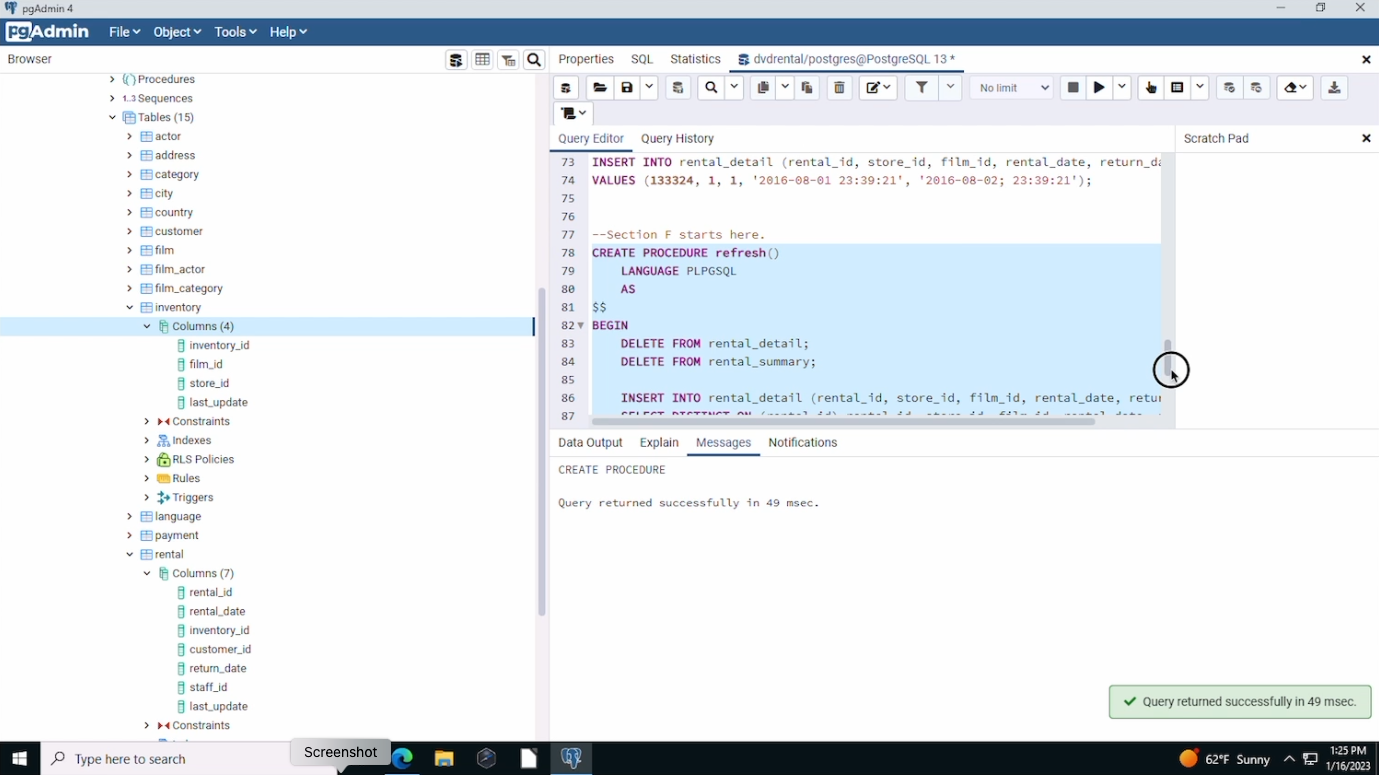




Create function and trigger to update the rental\_summary table when a new record is added:



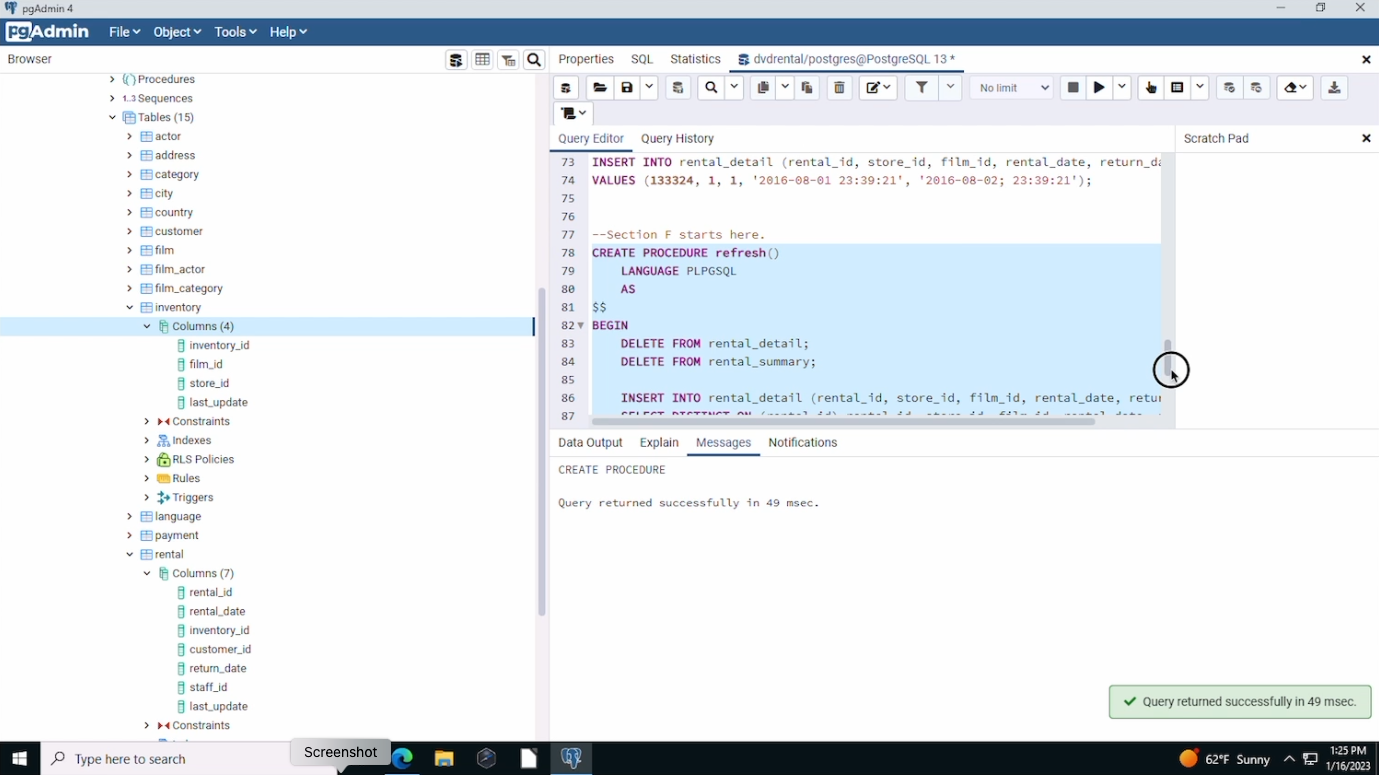
Create a stored procedure to refresh the data in the rental\_detail and rental\_summary tables, and then call the procedure:

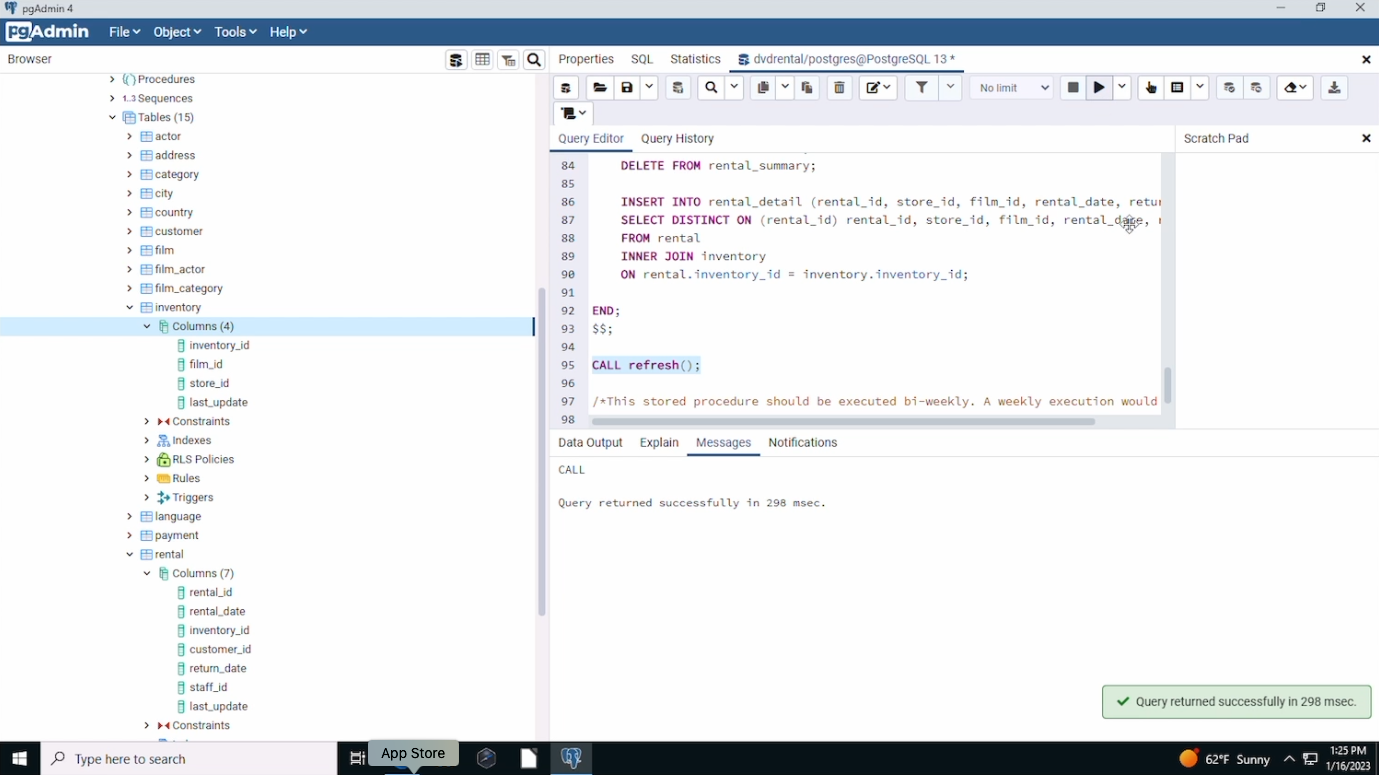


This stored procedure should be executed bi-weekly. A weekly execution would not  
provide enough value to justify the time it takes to perform and review, and the  
information from a monthly execution wouldn't be timely enough.

The stored procedure could be run on a schedule using a Python program (PostgreSQL Python: Call PostgreSQL Stored Procedures, 2022). As explained on PostgreSQLtutorial.com, this could be accomplished by first creating a new database connection to the database server. Then, a cursor should be created by calling the cursor() method. The the name of the stored procedure, refresh() in this case, would be passed to the execute() method of the cursor object that was just created. Then, the commit() method would be called to commit the transaction. Finally, the close() method would be called to close the connection to the database server.

The Python program could then be scheduled to run on a bi-weekly schedule with a Chrontab (How to Schedule Python Scripts With Chron – The Only Guide You’ll Ever Need, 2021).





References:

* Postgresqltutorial.com. 2022. *PostgreSQL Create Function Statement.*[online] Available at: <https://www.postgresqltutorial.com/postgresql-triggers/creating-first-trigger-postgresql/>. (accessed 2023).
* Postgresqltutorial.com. 2022. *PostgreSQL CREATE TRIGGER.*[online] Available at: <https://www.postgresqltutorial.com/postgresql-triggers/creating-first-trigger-postgresql/>. (accessed 2023).
* Postgresqltutorial.com. 2022. *PostgreSQL Python: Call PostgreSQL Stored Procedures.*[online] Available at: <https://www.postgresqltutorial.com/postgresql-python/call-stored-procedures/>. (accessed 2023).
* Radecic, D. (2021). *How to Schedule Python Scripts With Chron – The Only Guide You’ll Ever Need.*[online] Available at: <https://towardsdatascience.com/how-to-schedule-python-scripts-with-cron-the-only-guide-youll-ever-need-deea2df63b4e>(accessed 2023).
* W3schools.com. 2023. *SQL*.[online] Available at: <https://www.w3schools.com/sql/sql_insert_into_select.asp>. (accessed 2023).
* Winsberg, P. (2022). *Introduction to Databases with SQL*. zyBooks, a Wiley brand. <https://learn.zybooks.com/zybook/WGUC170v6>. (accessed 2023).