Jordan Johnson

WGU ID# 007198614

Advanced Data Management

Because new movies are released constantly it is necessary for the DVD rental store to change their inventory to keep it up to date. Currently, the store manager is looking for a way to determine how many films of each category needs to be replaced. To do this he has asked for a report that will provide this information. The report that will satisfy this purpose will have the categories listed in one column and in the other column it will have the number of times a film of that category has been rented. This will allow the store manager to see which categories are most popular and least popular. If further information is needed the numbers can be converted to a percentage of the total movies rented so that the store manager has an exact number to calculate how many of each type of film should be bought based on the size of the inventory change.

A1.

The data used for the report is from a DVD rental store. It contains film, rental, actor, payment, staff, customer, inventory, and address information. The data used for my detailed and summary is about the rental, and film data.

A2.

The two tables that will provide the necessary data for my report are the rental and the category table.

A3.

The specific fields that will be included in the detailed report are the rental\_id, category, and rental\_date. The specific fields that will be included in the summary report are category, and times\_rented.

A4.

The field in the detailed section that will require a custom transformation is the rental\_date field which should be changed to not include the time. The reason it should be transformed is because knowing the exact time that the customer rented the film is unnecessary and will distract from the date.

A5.

The business use of the detailed report is to show what data is being aggregated in the summary report and to see what dates the summary report is pulling data from. The business use of the summary report is to show the most rented films of each category of film so that when it comes to change out movies with new ones the more popular movies can be prioritized.

A6.

To remain relevant to the stakeholders the report needs to be refreshed every month. This will keep the numbers accurate and up to date so that if there is any change in popularity between categories the stakeholders will be aware and can act accordingly. The reports should also be refreshed every time the DVD rental store needs to be restocked with films or any other time there is an inventory change to make sure that the most up to date information is being used to determine how the categories should be restocked.

F1.

The stored procedure can be run automatically on a schedule every month by using an external tool like Linux crontab, Agent pgAgent, or Extension pg\_cron. “Linux crontab is a program that allows tasks to automatically run in the background.” (Dias) “Agent pgAgent is a job scheduling agent available for PostgreSQL that allows the execution of stored procedures, SQL statements, and shell scripts.” (Dias) “The pg\_cron extension performs the same function as the other two but runs inside the database as an extension.” (Dias)

Sources:

[PostgreSQL Tutorial - Learn PostgreSQL from Scratch](https://www.postgresqltutorial.com/)

[SQL Tutorial (w3schools.com)](https://www.w3schools.com/sql/default.asp)

Malik, Upom, et al. *SQL for Data Analytics : Perform Fast and Efficient Data Analysis with the Power of SQL*, Packt Publishing, Limited, 2019.*ProQuest Ebook Central*, https://ebookcentral.proquest.com/lib/westerngovernors-ebooks/detail.action?docID=5888693.

Dias, Hugo. “An Overview of Job Scheduling Tools for PostgreSQL.” *Severalnines*, 3 Feb. 2020, severalnines.com/database-blog/overview-job-scheduling-tools-postgresql.

-- Section B starts here

CREATE TABLE summary(

genre char(30),

times\_rented int

);

CREATE TABLE detailed(

rental\_id int,

genre char(30),

rental\_date timestamp

);

-- Section C starts here

INSERT INTO detailed

SELECT rental\_id, category.name, rental\_date

FROM rental

INNER JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category ON inventory.film\_id = film\_category.film\_id

INNER JOIN category ON film\_category.category\_id = category.category\_id

INSERT INTO summary

SELECT genre, count(genre)

FROM detailed

GROUP BY genre

SELECT \* FROM detailed

SELECT \* FROM rental\_id, rental\_date, FROM rental

SELECT \* FROM category

SELECT name

FROM rental

INNER JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category ON inventory.film\_id = film\_category.film\_id

INNER JOIN category ON film\_category.category\_id = category.category\_id

WHERE rental\_id = 2

-- Section D starts here

CREATE OR REPLACE FUNCTION fixdate()

RETURNS void

LANGUAGE plpgsql AS

$$

BEGIN

UPDATE detailed

SET rental\_date = DATE\_TRUNC('DAY', rental\_date);

END;

$$

SELECT fixdate()

SELECT \* FROM detailed

-- Section E starts here

CREATE FUNCTION summary\_trigger()

RETURNS trigger

LANGUAGE plpgsql

AS

$$

BEGIN

UPDATE summary

SET times\_rented = times\_rented + 1

WHERE genre = NEW.genre;

RETURN NEW;

END;

$$

CREATE TRIGGER summary\_trigger

AFTER INSERT

ON detailed

FOR EACH ROW

EXECUTE PROCEDURE summary\_trigger();

SELECT \* FROM summary

SELECT \* FROM detailed

INSERT INTO detailed

SELECT rental\_id, category.name, rental\_date

FROM rental

INNER JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category ON inventory.film\_id = film\_category.film\_id

INNER JOIN category ON film\_category.category\_id = category.category\_id

TRUNCATE TABLE summary

TRUNCATE TABLE detailed

INSERT INTO summary

SELECT genre, count(genre)

FROM detailed

GROUP BY genre

-- Section F starts here

CREATE PROCEDURE update\_report()

LANGUAGE plpgsql AS

$$

BEGIN

TRUNCATE TABLE detailed;

TRUNCATE TABLE summary;

INSERT INTO detailed

SELECT rental\_id, name, rental\_date

FROM rental

INNER JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

INNER JOIN film\_category ON inventory.film\_id = film\_category.film\_id

INNER JOIN category ON film\_category.category\_id = category.category\_id;

INSERT INTO summary

SELECT genre, count(genre)

FROM detailed

GROUP BY genre;

COMMIT;

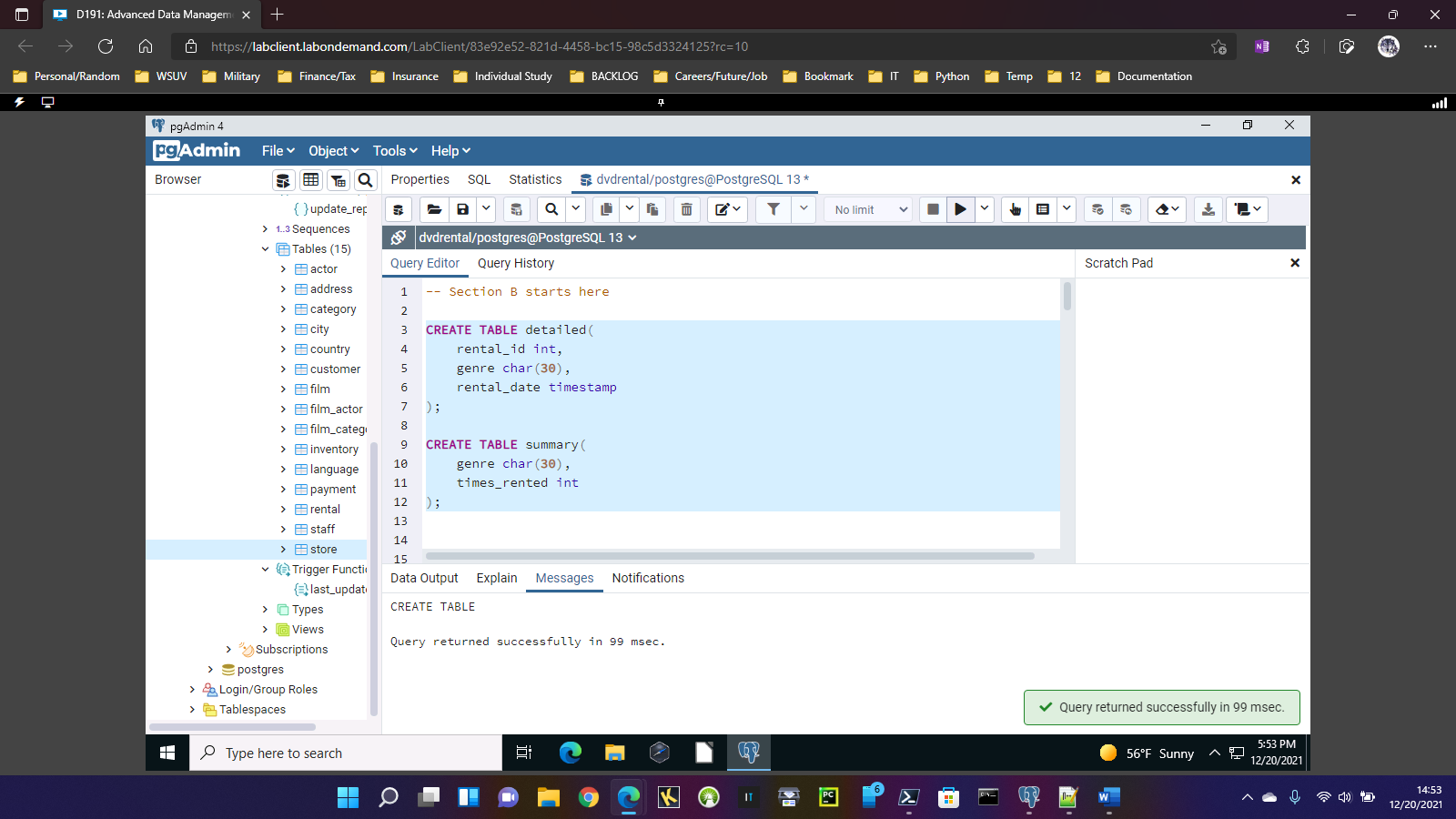
END;

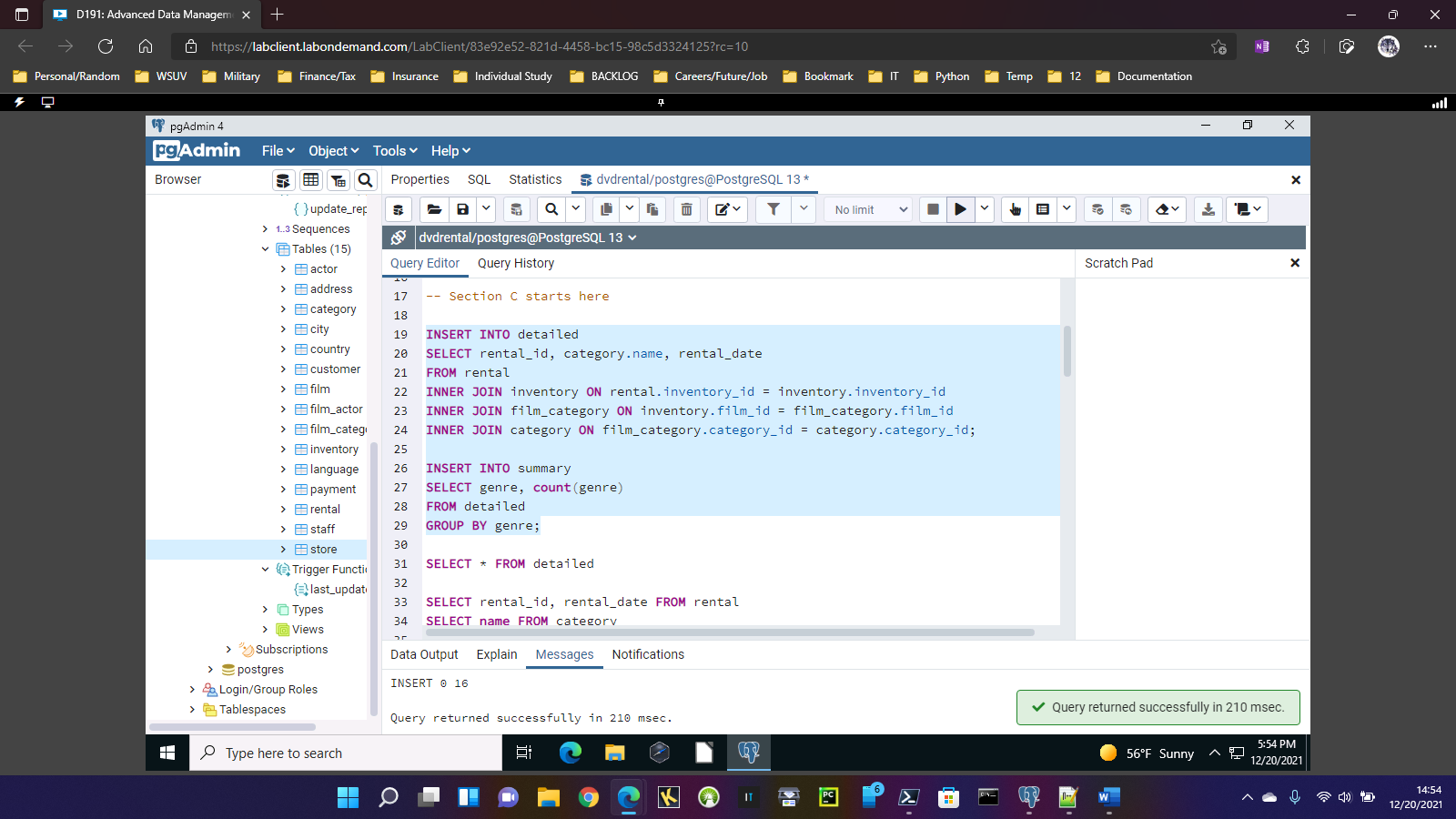
$$

-- The stored procedure should be run every month or anytime that the DVD rental store

-- is restocking their inventory before and after.

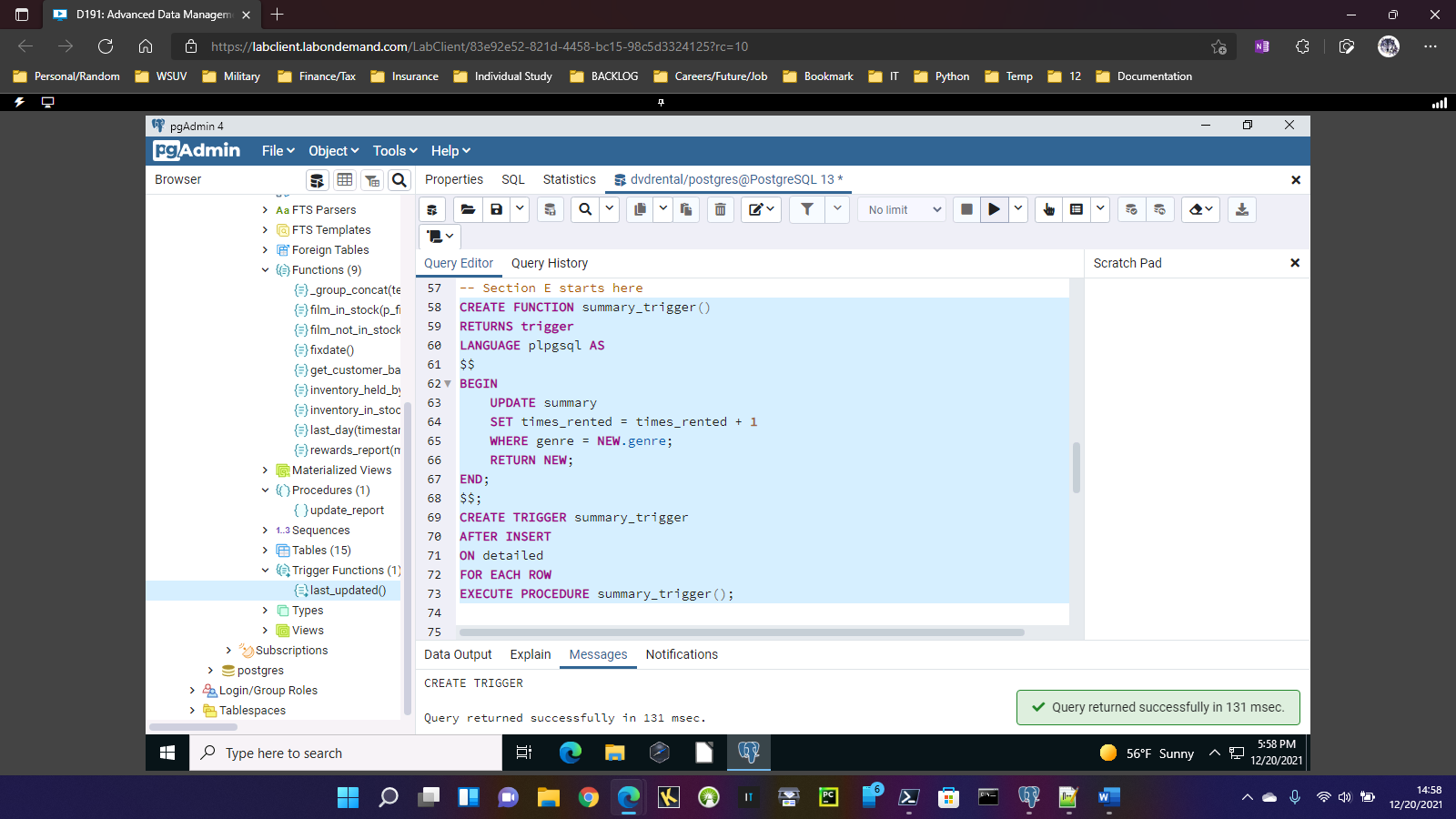
CALL update\_report()

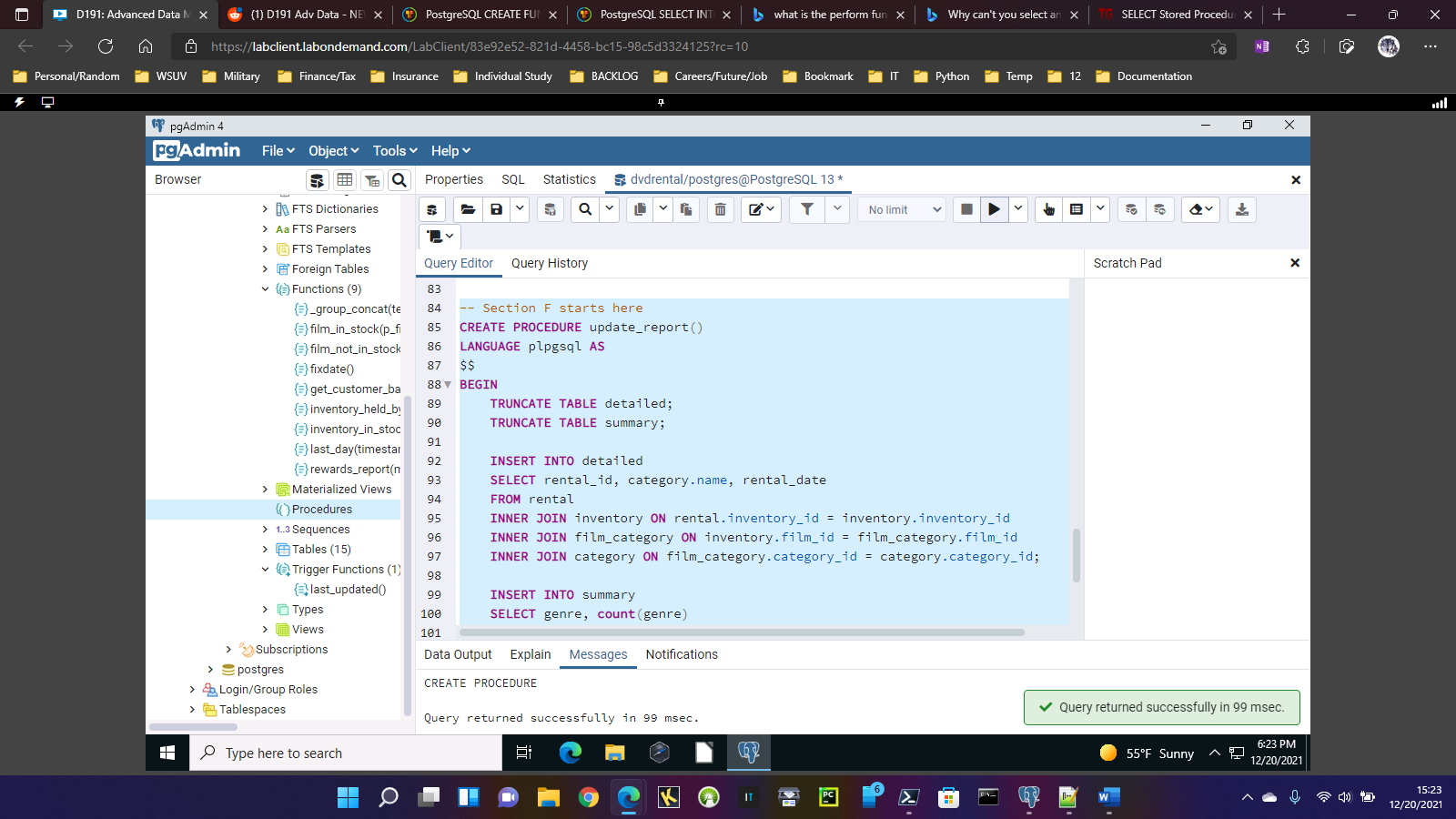




A screenshot of a computer

Description automatically generated





A screenshot of a computer

Description automatically generated