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

SQL Tutorial (w3schools.com)

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.

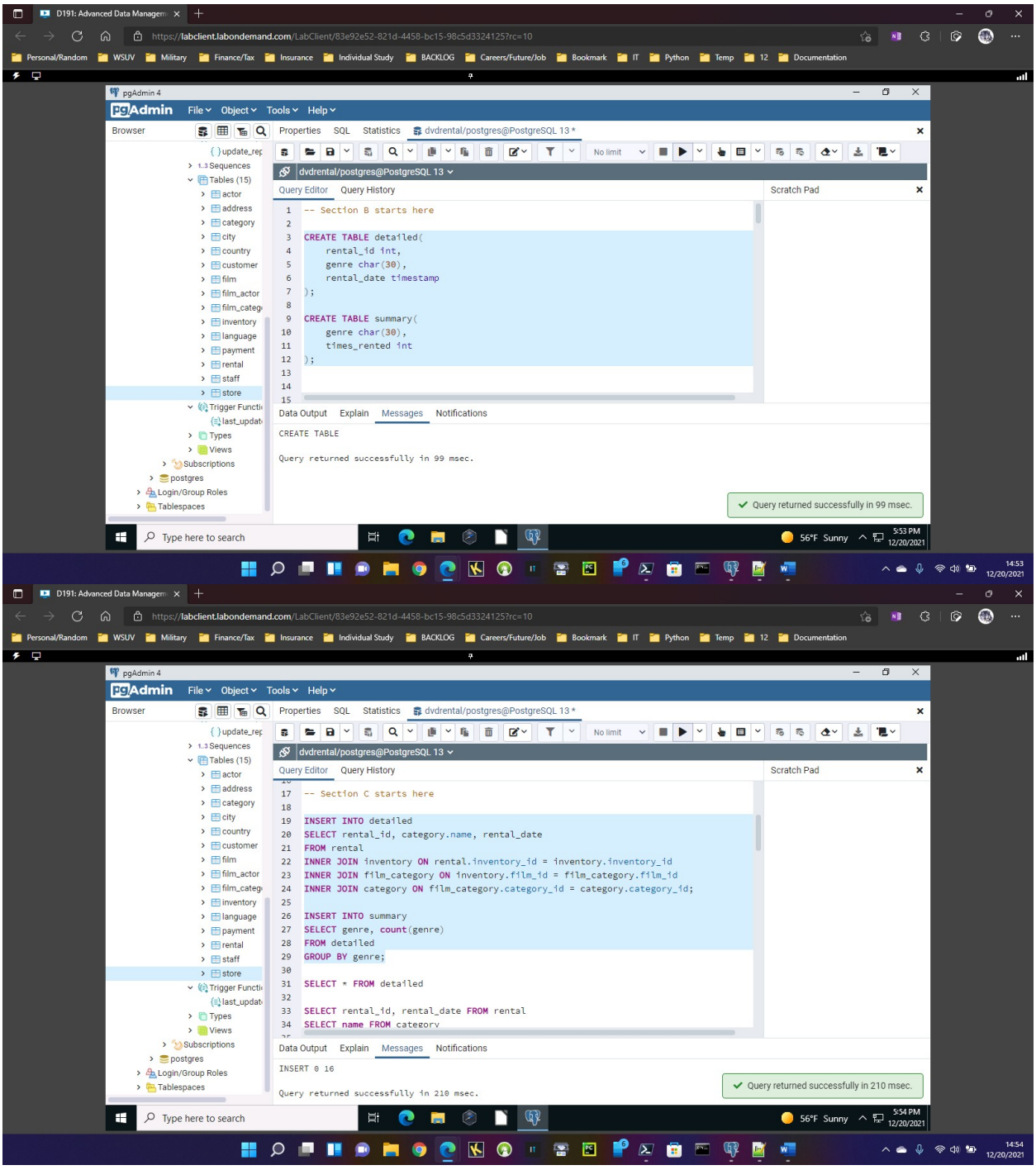
Dias, Hugo. “An Overview of Job Scheduling Tools for PostgreSQL.” *Severalnines*, 3 Feb. 2020, severalnines.com/database-blog/overview-job-scheduling-tools-postgresql.

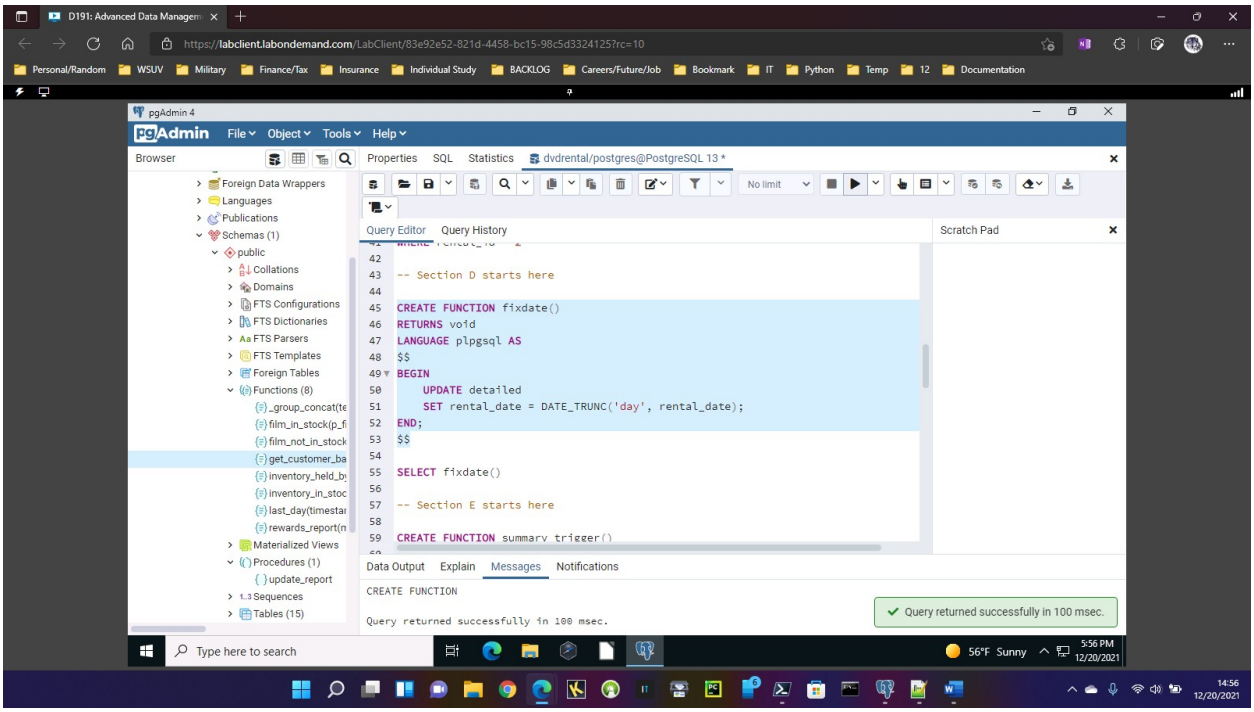

```
-- Section B starts here

CREATE TABLE summary(
        genre char(30),
        times_rented int
);

CREATE TABLE detailed(
```

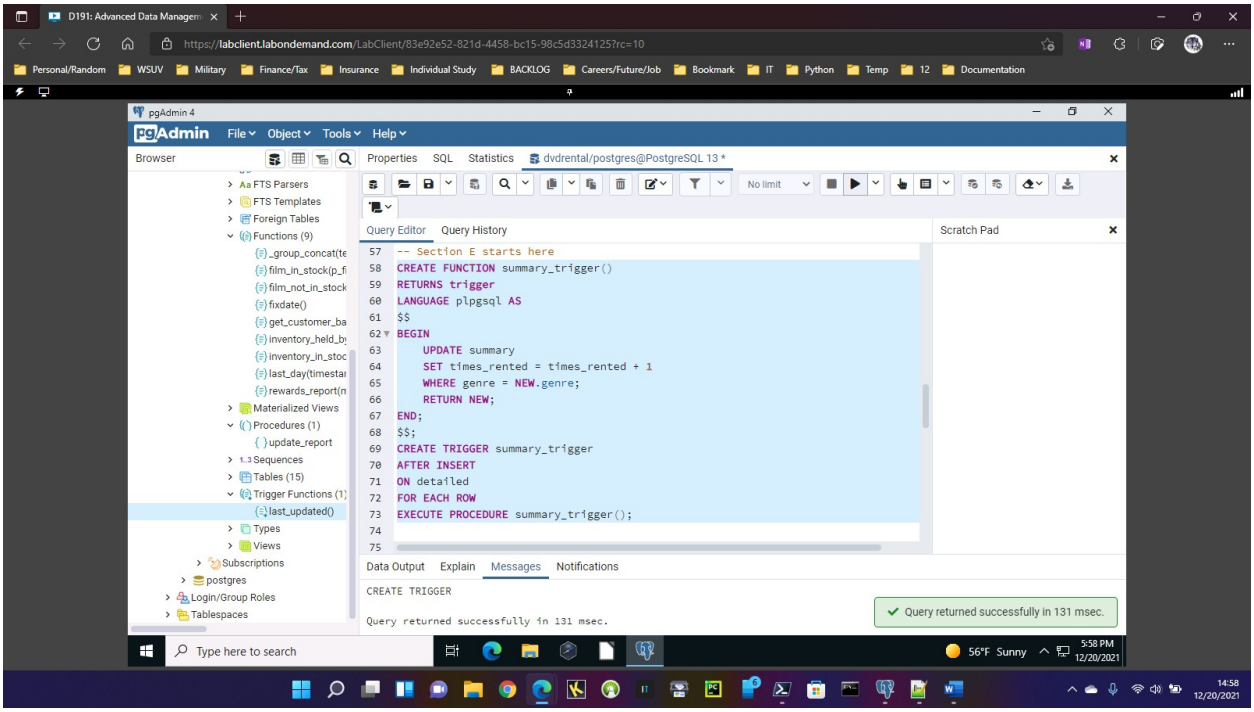
```
	rental_id int,
	genre char(30),
	rental_date timestamp
);

-- 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()
```

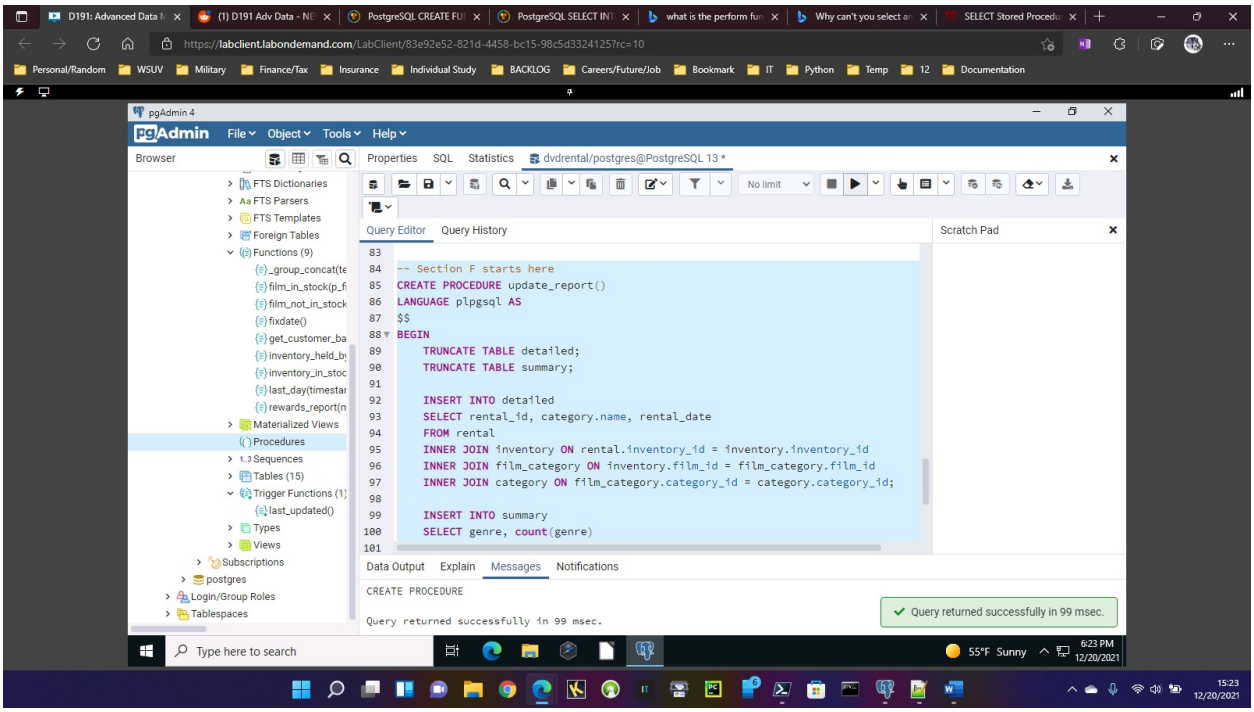
```
SELECT * FROM detailed

-- 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
```

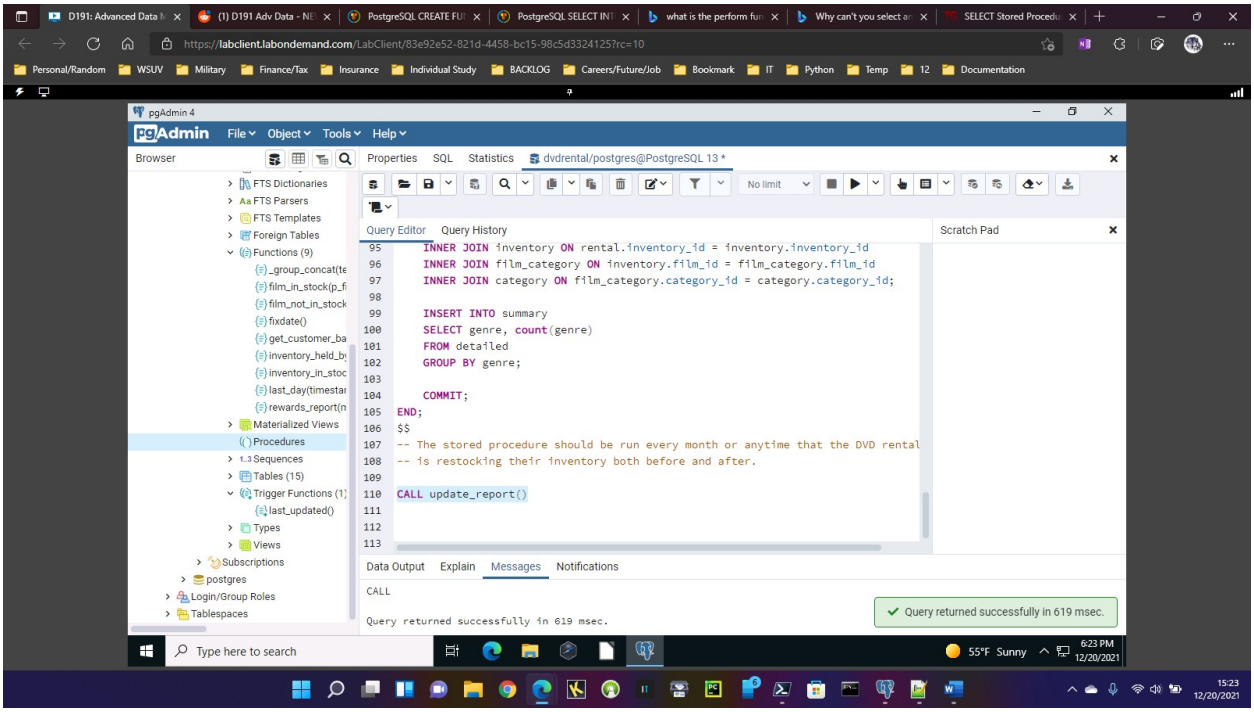
```
CREATE PROCEDURE update_report()
LANGUAGE plpgsql AS
$$
BEGIN
	TRUNCATE TABLE detailed;
	TRUNCATE TABLE summary;

	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()
```


D191: Advanced Data Managem
https://labclient.labondemand.com/LabClient/83e92e52-821d-4458-bc15-98c5d3324125?rc=10
pgAdmin 4
File
Object
Tools
Help
Browser
Properties
SQL
Statistics
dvdrental/postgres@PostgreSQL 13 *
Query Editor
Query History
Scratch Pad
-- Section B starts here
CREATE TABLE detailed(
rental_id int,
genre char(30),
rental_date timestamp
);
CREATE TABLE summary(
genre char(30),
times_rented int
);
Data Output
Explain
Messages
Notifications
CREATE TABLE
Query returned successfully in 99 msec.
Type here to search
56°F Sunny
5:53 PM
12/20/2021
14:53
12/20/2021
-- 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 rental_id, rental_date FROM rental
SELECT name FROM category
INSERT 0 16
Query returned successfully in 210 msec.
5:54 PM
12/20/2021
14:54
12/20/2021


D191: Advanced Data Managem
https://labclient.labondemand.com/LabClient/83e92e52-821d-4458-bc15-98c5d3324125?rc=10
pgAdmin 4
File Object Tools Help
Browser
Properties SQL Statistics dvdrental/postgres@PostgreSQL 13 *
Query Editor Query History
Scratch Pad
-- Section D starts here
CREATE FUNCTION fixdate()
RETURNS void
LANGUAGE plpgsql AS
$$
BEGIN
UPDATE detailed
SET rental_date = DATE_TRUNC('day', rental_date);
END;
$$
SELECT fixdate()
-- Section E starts here
CREATE FUNCTION summary_trigger()
Data Output Explain Messages Notifications
CREATE FUNCTION
Query returned successfully in 100 msec.
Query returned successfully in 100 msec.
Type here to search
56°F Sunny
5:56 PM
12/20/2021


D191: Advanced Data Managem
https://labclient.labondemand.com/LabClient/83e92e52-821d-4458-bc15-98c5d3324125?rc=10
pgAdmin 4
File Object Tools Help
Browser
Properties SQL Statistics dvdrental/postgres@PostgreSQL 13 *
Query Editor Query History
Scratch Pad
-- 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();
Data Output Explain Messages Notifications
CREATE TRIGGER
Query returned successfully in 131 msec.
Query returned successfully in 131 msec.
Type here to search
56°F Sunny
5:58 PM
12/20/2021


pgAdmin 4
File Object Tools Help
Browser
Properties SQL Statistics dvdrental/postgres@PostgreSQL 13 *
Query Editor Query History
Scratch Pad
-- Section F starts here
CREATE PROCEDURE update_report()
LANGUAGE plpgsql AS
$$
BEGIN
TRUNCATE TABLE detailed;
TRUNCATE TABLE summary;
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)
Data Output Explain Messages Notifications
CREATE PROCEDURE
Query returned successfully in 99 msec.
Query returned successfully in 99 msec.


pgAdmin 4
File Object Tools Help
Browser
Properties SQL Statistics dvdrental/postgres@PostgreSQL 13 *
Query Editor Query History
Scratch Pad
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
-- is restocking their inventory both before and after.
CALL update_report()
Data Output Explain Messages Notifications
CALL
Query returned successfully in 619 msec.
Query returned successfully in 619 msec.