Devin Perdue

Advanced-Data Management- D191

6/19/23

Professor Sewell

# **Section A**

# The business question we aim to answer is: What are our top five most profitable categories? This will provide insight into what categories are providing us with the most profit. This will help us have these categories of movies more readily available to meet consumer demand. We can create this report from the category, film, and film\_category tables, which provide us with the necessary information such as film\_id, film\_name, category\_id, category\_name, rental\_duration, and rental\_rate.

## **A1**

The fields included in the detailed table are film\_id, title, category\_id, and profit (the profit will come from the rental duration multiplied by the rental rate. These two values come from the film table). This information will come from the category and film tables. The data types are as follows, film\_id will be an integer, title will be varchar, category\_id will be an integer, and profit will be a numeric data type. Integer data types are limited to numbers and cannot be letters or other characters. Varchar is a variable-length data type that can hold letters and numbers. Finally, numeric is similar to an integer data type in which it stores a number value; however, these can also store decimal values.

Next up is our summary table. This table will be significantly shorter as it will only include two variables. These are category\_name and profit. These will also come from the category and film tables. The datatypes for each are category\_name a Varchar, and profit is numeric. As stated above, varchar is a variable-length data type that can hold letters and numbers, and numeric is similar to an integer data type but can hold decimal values.

## **A2**

The different data fields we will be using are integer, varchar, and numeric. Integer values are limited to numbers; these can be negative or positive but must be numbered. Varchar is a variable-length data type with a variable amount of characters based on the limit you set. These can be alpha-numeric values and are not limited to one another. Finally, numeric values are similar to integer value types in that they store numbers, but numeric data types can also store decimal values. For example, if you say numeric(4,2), this number is four digits with two digits past the decimal place.

| Source Table | Column Name | Data Type and Precision | Destination | Definition |
| --- | --- | --- | --- | --- |
| film | film\_id | INT | Detailed | Film ID/Unique Identifier |
| film | title | VARCHAR(45) | Detailed | Movie Name |
| category | category\_id | INT | Detailed | Category ID for each category |
| film | Rental duration \* rental rate as profit | NUMERIC (8,2) | Detailed and Summary | Profit |
| category | category\_name | VARCHAR(20) | Summary | Category/Genre |
| Derived Field | category\_id | VARCHAR(15) | Summary | Provides category name from category ID |

## **A3**

The tables that will supply the detailed table with the necessary information are the category and film tables. We will also use the film\_category table to connect the two tables with JOINS. Regarding the summary table, we will get this information also from the category and film tables.

## **A4**

As stated in our detailed table, we have a column named category\_id which is an integer. We will create a custom transformation that will take in this integer and return the category name associated with it. We will call this function when making our summary table so we will receive the category names instead of numbers.

## **A5**

With our detailed table, we will have fim\_id, title, category\_id, and profit for all films. This table will contain lots of data with hundreds of rows. However, the stakeholder could use this information to look at specific movies and see the sales for those movies. This would break down the information per movie, which can provide the stakeholder with which movies are selling the best and are most popular.

Our summary table is much smaller, only containing category\_name and profit. This will break down all the information from the detailed table into categories and how much profit was made from each. This will give the stakeholder a simple table showing which categories consumers are buying the most. This will provide the company with insight and be able to provide more movies in the most popular categories.

## **A6**

This report should be updated on a biweekly base. This will provide relevant and recent data showing what the consumers are buying. This can help the company prepare to have more movies in the most popular categories ready for consumers. This can also prepare the DVD store around holidays to have categories such as animation, children, and classics more readily available.

# **Section B**

CREATE OR REPLACE FUNCTION category\_func(cat integer) returns VARCHAR (15)

language plpgsql as

$$

begin

if cat = 1 then return ‘Action’;

elseif cat = 2 then return ‘Animation’;

elseif cat = 3 then return ‘Children’;

elseif cat = 4 then return ‘Classics’;

elseif cat = 5 then return ‘Comedy’;

elseif cat = 6 then return ‘Documentary’;

elseif cat = 7 then return ‘Drama’;

elseif cat = 8 then return ‘Family’;

elseif cat = 9 then return ‘Foreign’;

elseif cat = 10 then return ‘Games’;

elseif cat = 11 then return ‘Horror’;

elseif cat = 12 then return ‘Music’;

elseif cat = 13 then return ‘New’;

elseif cat = 14 then return ‘Sci-Fi’;

elseif cat = 15 then return ‘Sports’;

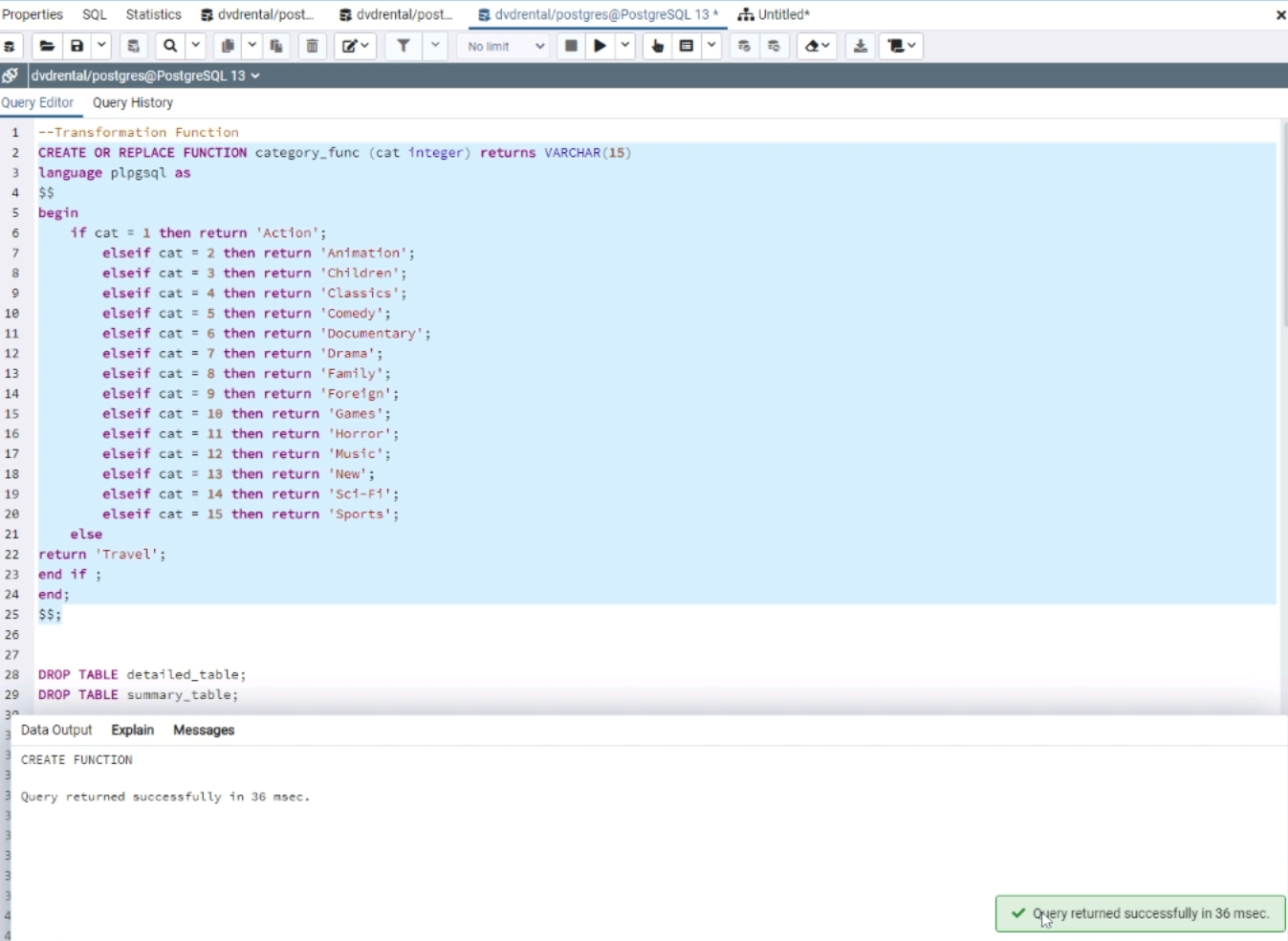
else

return ‘Travel’;

end if ;

end;

$$;



# **Section C**

CREATE TABLE detailed\_table (

film\_id INT,

category\_id INT,

title VARCHAR(45),

profit NUMERIC(8,2)

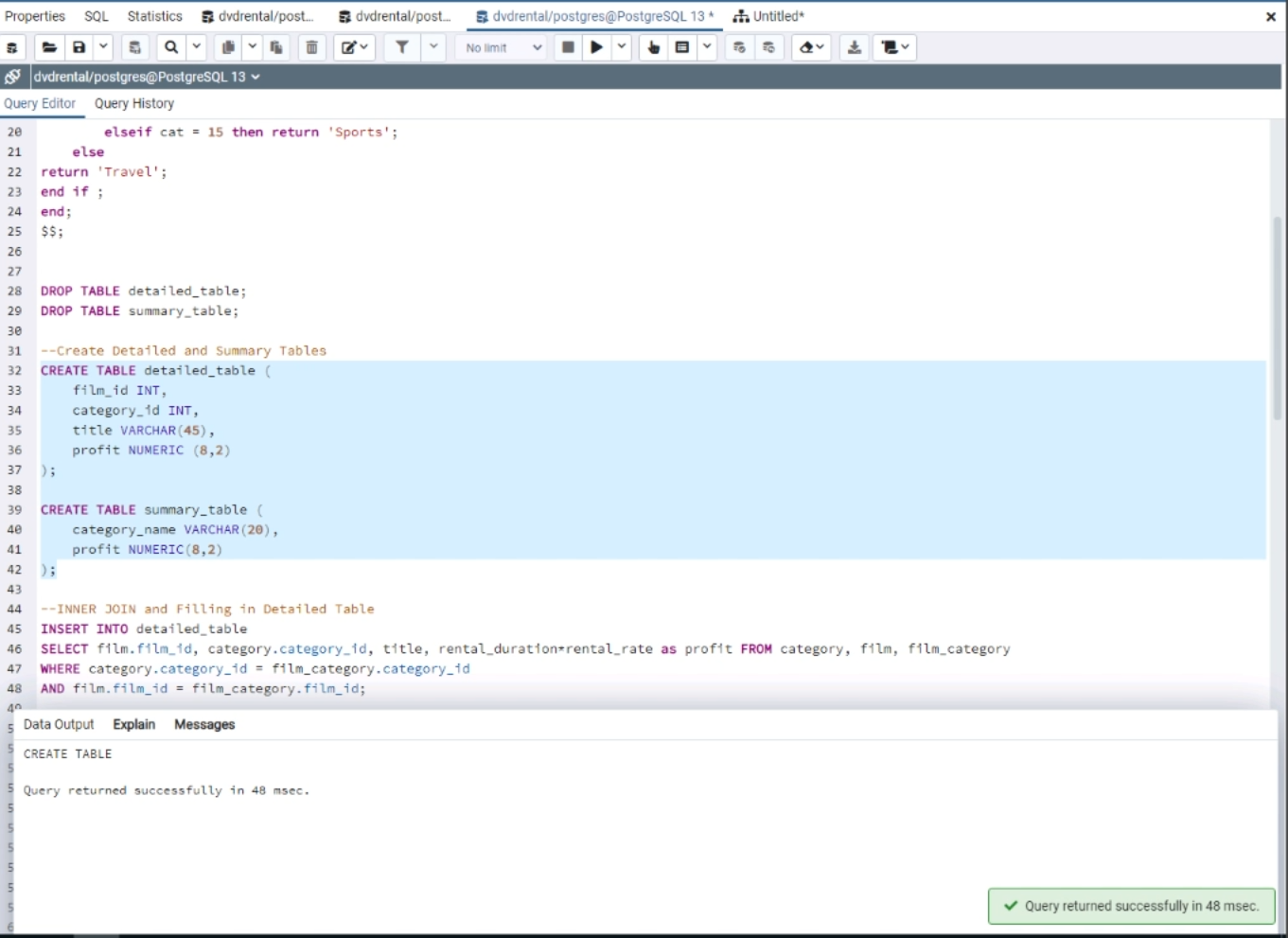
);

CREATE TABLE summary\_table (

category\_name VARCHAR(20),

profit NUMERIC(8,2)

);



# **Section D**

This inner join will be inserted into the detailed\_table to give us all the data we need.

INSERT INTO detailed\_table

SELECT film.film\_id, category.category\_id, title, rental\_duration\*rental\_rate as profit

FROM category, film, film\_category

WHERE category.category\_id = film\_category.category\_id

AND film.film\_id = film\_category.film\_id;

This is the code we utilize to fill our summary\_table.

INSERT INTO summary\_table

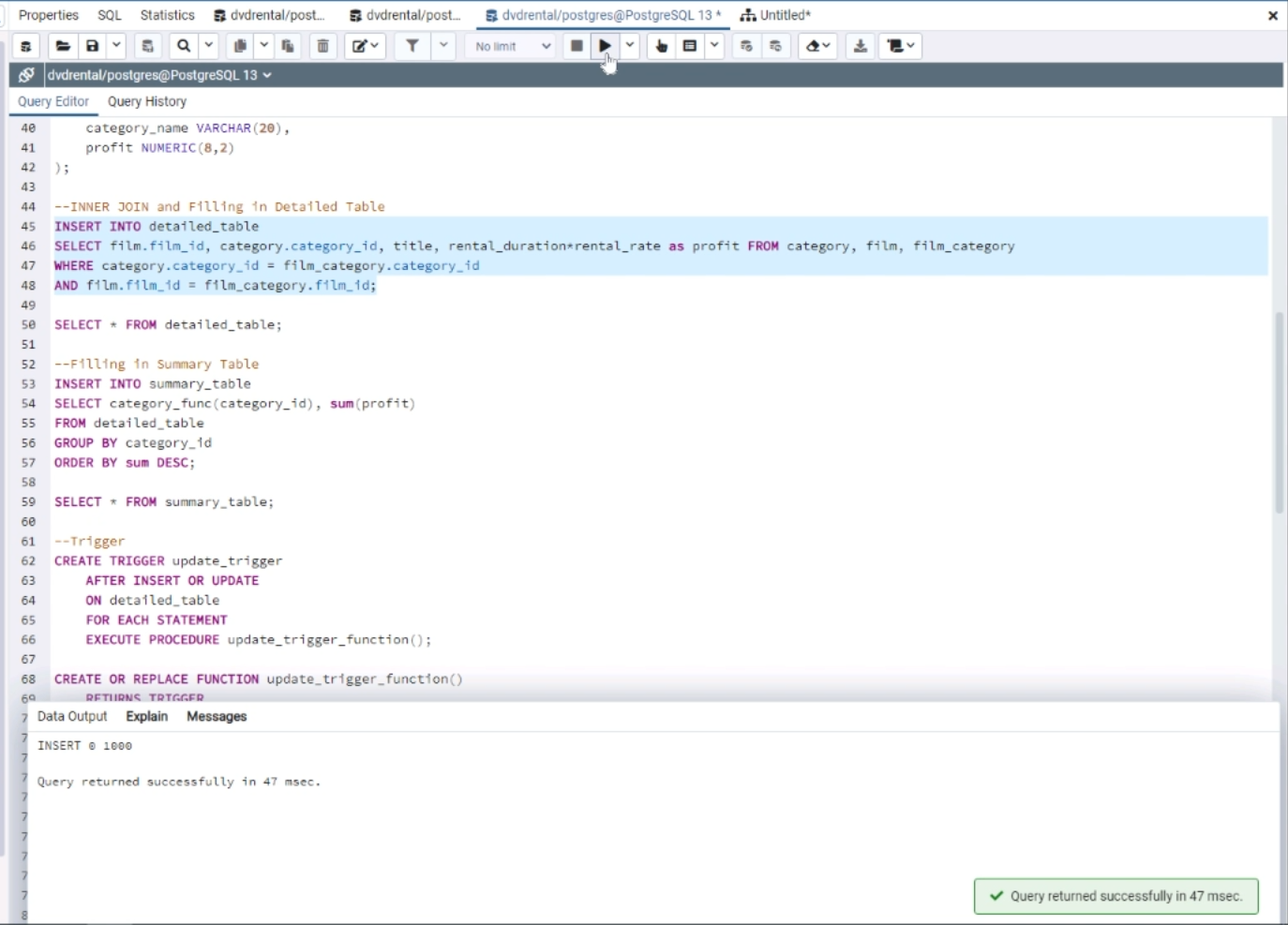
SELECT category\_func(category\_id), sum(profit)

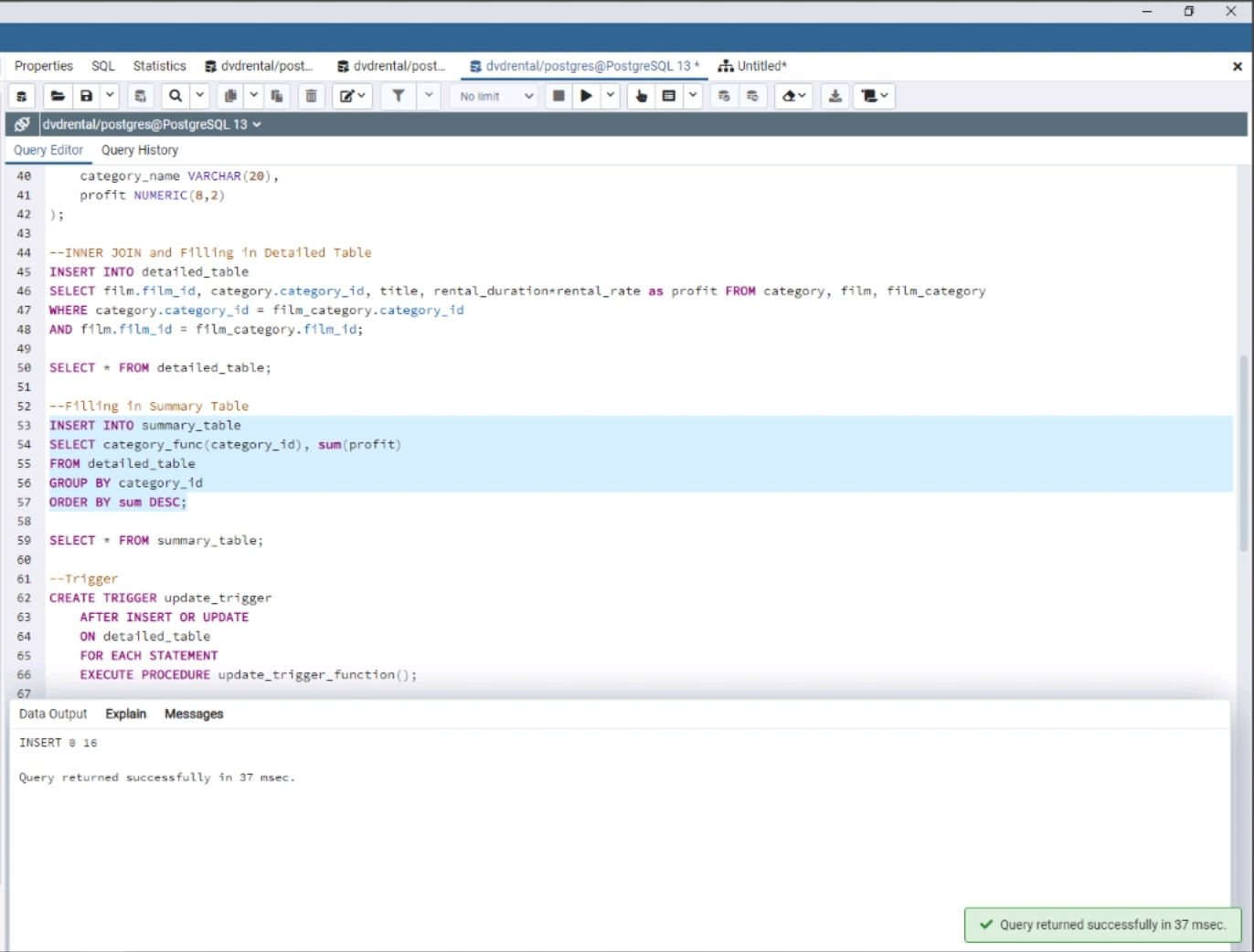
FROM detailed\_table

GROUP BY category\_id

ORDER BY sum DESC;

(Malik, Goldwasser, & Johnston, 2019, p.74-83)





# **Section E**

CREATE TRIGGER update\_trigger

AFTER INSERT OR UPDATE

ON detailed\_table

FOR EACH STATEMENT

EXECUTE PROCEDURE update\_trigger\_function();

CREATE OR REPLACE FUNCTION update\_trigger\_function()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM sumamry\_table

INSERT INTO sumamry\_table

SELECT category\_func(category\_id), sum(profit)

FROM detailed\_table

GROUP BY category\_id

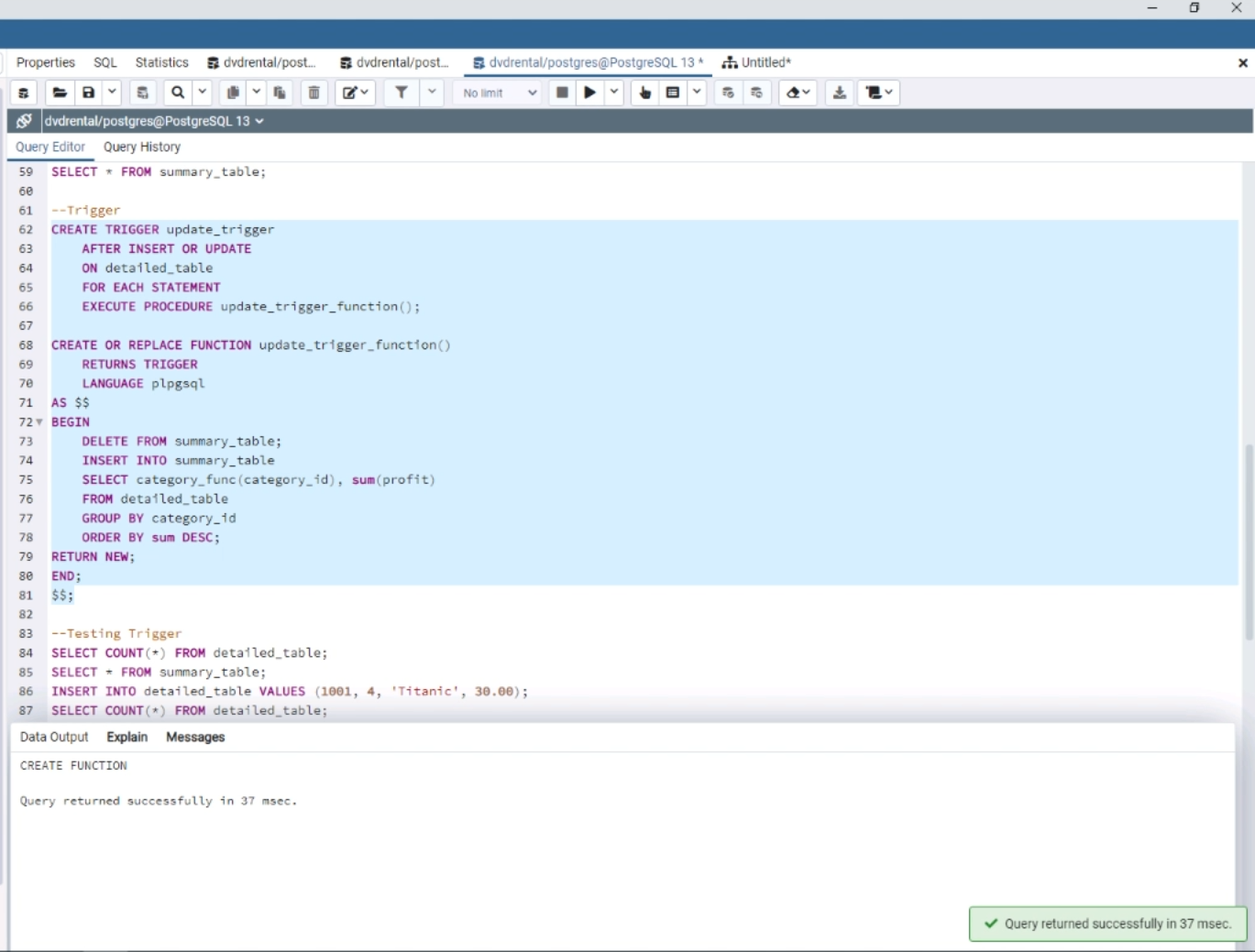
ORDER BY sum DESC;

RETURN NEW;

END;

$$;

(PostgreSQL Tutorial, n.d.-b)



# **Section F**

CREATE OR REPLACE PROCEDURE create\_refresh()

LANGUAGE plpgsql

AS $$

BEGIN

DROP TABLE IF EXISTS detailed\_table;

DROP TABLE IF EXISTS summary\_table;

CREATE TABLE detailed\_table AS

SELECT film.film\_id, category.category\_id, title, rental\_duration\*rental\_rate as profit

FROM category, film, film\_category

WHERE category.category\_id = film\_category.category\_id

AND film.film\_id = film\_category.film\_id;

CREATE TABLE AS

SELECT category\_func(category\_id), sum(profit)

FROM detailed\_table

GROUP BY category\_id

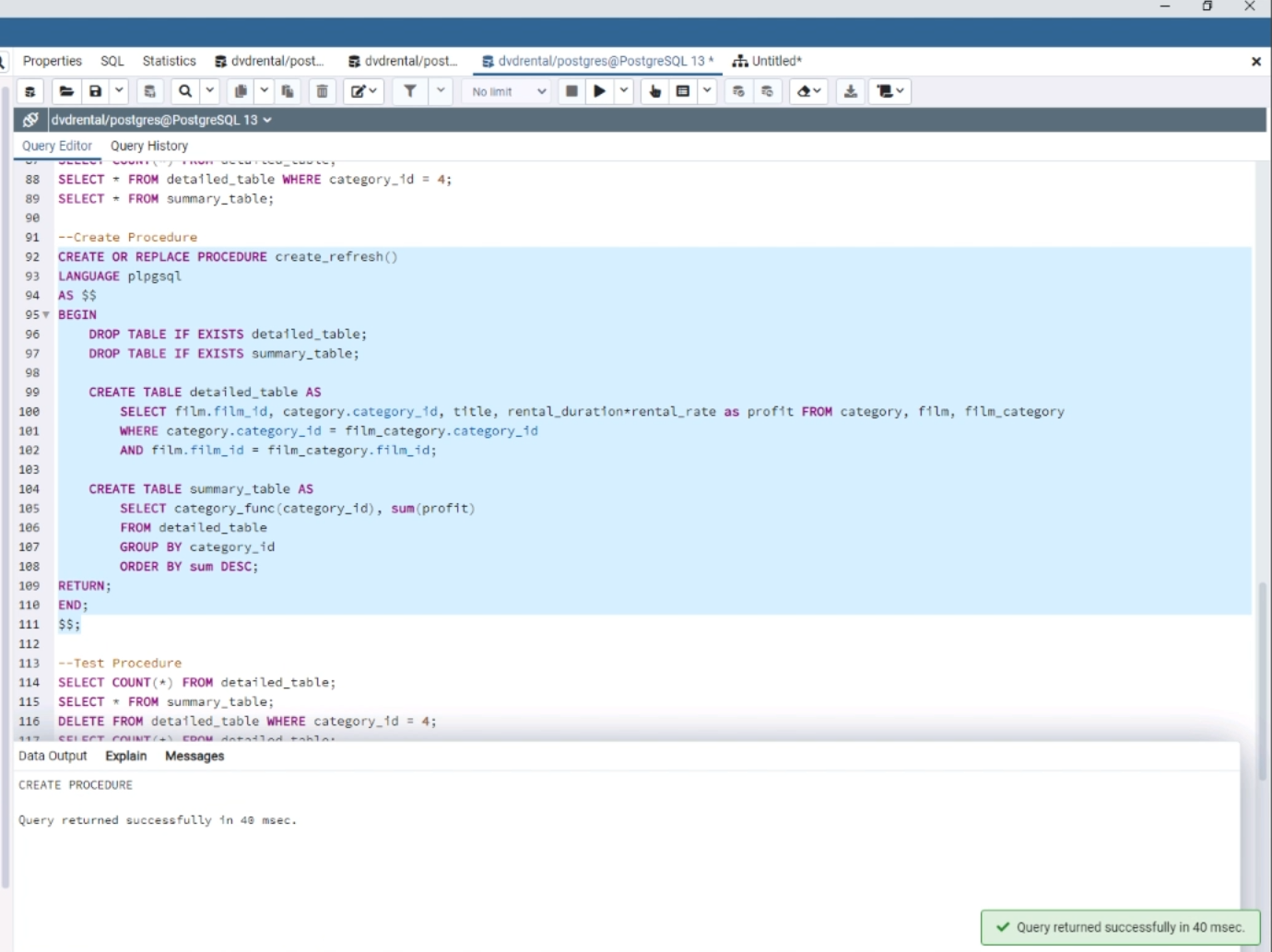
ORDER BY sum DESC;

RETURN;

END;

$$;

(PostgreSQL Tutorial, n.d.-a)



## **F1**

Linux Crontab can be used as a job scheduling tool. This will allow our stored procedure to run periodically in the background. This will refresh our data however frequently we set the scheduling tool to run. This will make sure we have fresh and reliable data in the future (Dias, 2022).

# 

# 

# **Section H**

Dias, H. (2022, May 4). *An overview of job scheduling tools for postgresql*. Severalnines. https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/

Malik, U., Goldwasser, M., & Johnston, B. (2019). SQL for Data Preparation. In *SQL for data analytics: Perform fast and efficient data analysis with the power of SQL* (1st ed., pp. 71–99). essay, Packt publishing.

PostgreSQL Tutorial. (n.d.-a). *PostgreSQL create procedure*. PostgreSQL Tutorial. https://www.postgresqltutorial.com/postgresql-plpgsql/postgresql-create-procedure/

PostgreSQL Tutorial. (n.d.-b). *PostgreSQL create trigger*. PostgreSQL Tutorial. https://www.postgresqltutorial.com/postgresql-triggers/creating-first-trigger-postgresql/