Advanced Data Management – D191

Jennifer Codemo-Thomas

College of Information Technology, Western Governors University

April 8, 2023

Advanced Data Management – D191

Task A

Summarize one real-world written business report that can be created from the DVD Dataset from the "Labs on Demand Assessment Environment and DVD Database" attachment.

For this assignment, I will provide reporting for the fictitious DVD rental company DVD2U. DVD2U is looking to grow their market share and to ensure that they are investing in the most profitable films. To assist them in this process, I will be providing them a report with both detail and summary tables. This report will address the following questions:

- What is DVD2U's most profitable film genre?
- Does DVD2U's inventory align with the most profitable film genres?

1. Identify the specific fields that will be included in the detailed table and the summary table of the report.

For the detail table, I will be including the following fields:

rental_mo
rental_yr
film_genre
mo_revenue

For the summary table, I will be including the following fields:

film_genre
total_revenue
total_inventory

2. Describe the types of data fields used for the report.

film_id is a smallint – it will be used to join the inventory table to the film_category table; in addition, it will be used to calculate the total_inventory.

name is a varchar – it is the film_genre field in both the detail and summary tables.

rental_date is a timestamp - it will be used to extract the month and year fields within functions.

rental_mo is an integer field created using a user-defined function.

rental_yr is an integer field created using a user-defined function.

amount is a numeric with 5 digits and 2 decimals – it will be used to calculate the mo_revenue and total_revenue.

mo_revenue is a numeric with 7 digits and 2 decimals – it will be calculated as part of the detail table.

total_revenue is a numeric with 8 digits and 2 decimals – it will be calculated as part of the summary table.

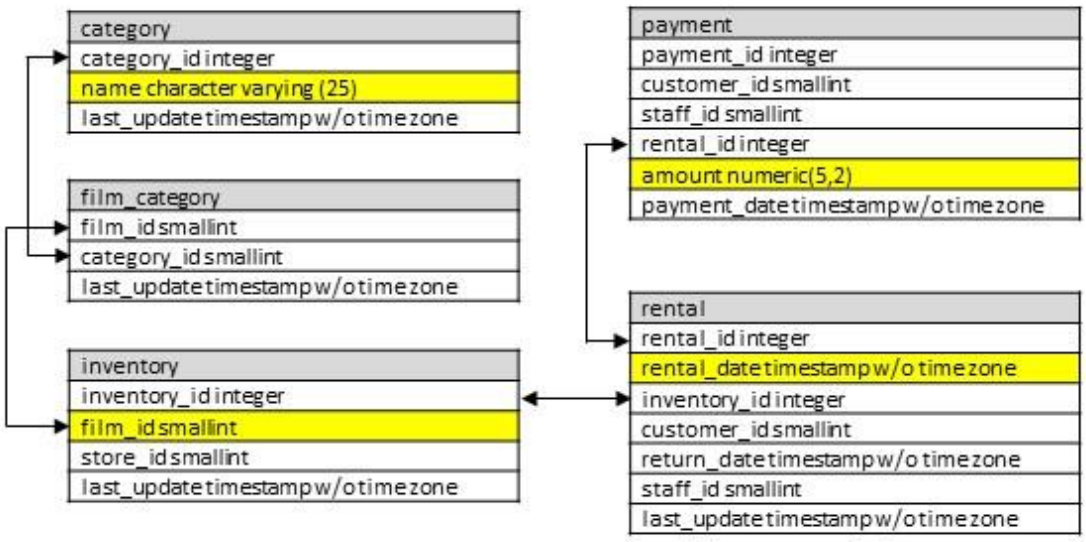
category_id is an integer (also shown as smallint) – it will be used to join the film_category and category tables.

rental_id is an integer – it will be used to join the payment and rental tables.

inventory_id is an integer – it will be used to join the rental and inventory tables.

3. Identify at least two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report.

The following shows the five tables that will be used to extract the data necessary for DVD2U's report. The field names highlighted in yellow will be used in the final report or to calculate fields used in the final report, and there are arrows indicating the fields that will be used to join the tables.



4. Identify at least one field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of N to No and Y to Yes).

I will be transforming the rental_date field which is currently a timestamp type. A function will be created to extract both the month and the year into separate fields. This will allow me to group the detail information by month and year, providing DVD2U a monthly snapshot of revenue by genre.

5. Explain the different business uses of the detailed table section and the summary table section of the report.

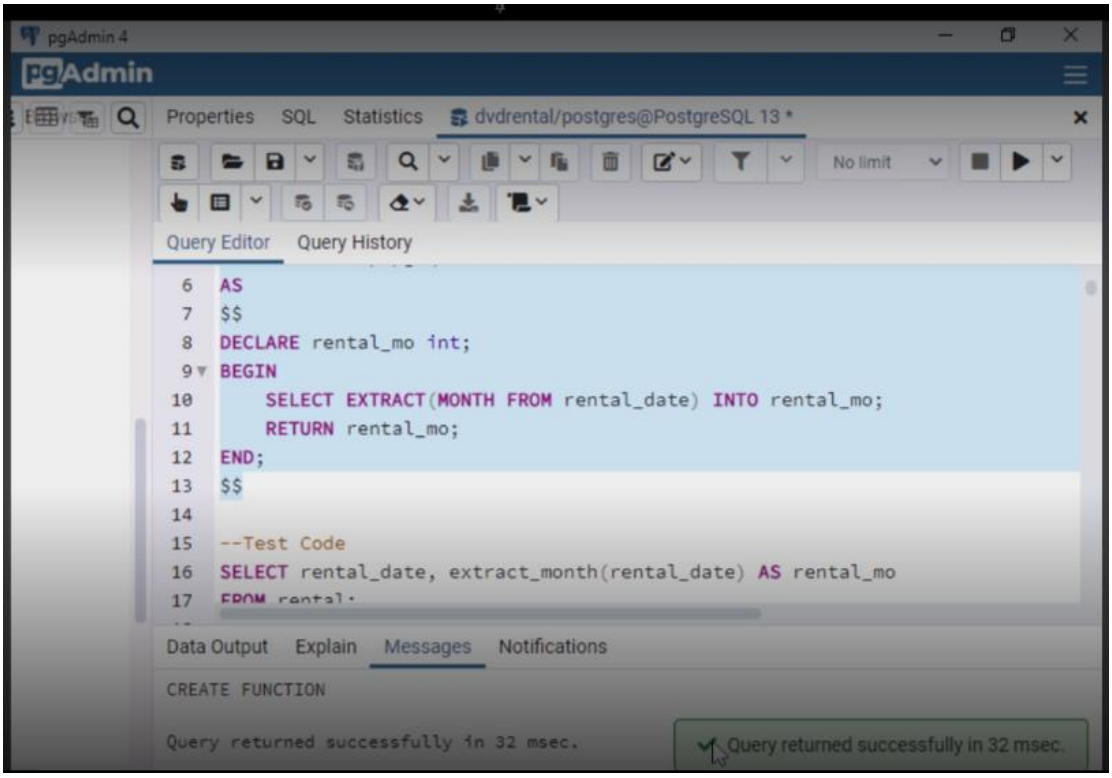
DVD2U wants to ensure that they are investing in the correct film genres to maximize their revenue. The detail table will provide them with a monthly revenue snapshot which can be utilized to see the top grossing genre on a monthly basis. The summary table will provide DVD2U with the total revenue by genre and will include an inventory count so they can monitor inventory levels to ensure revenue is maximized.

6. Explain how frequently your report should be refreshed to remain relevant to stakeholders.

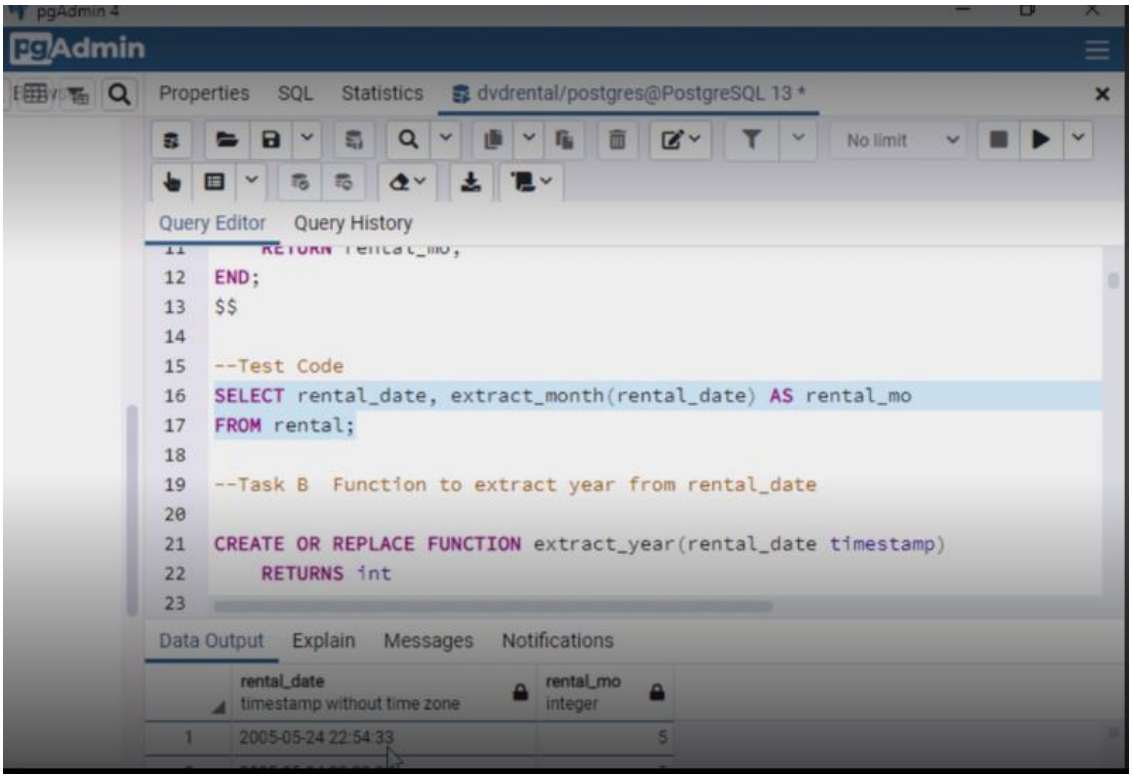
This report should be refreshed monthly on the first of the month to capture the entirety of the previous month's rental transactions. For example, on May 1st, the report would be refreshed to capture all rental transactions through June 30th.

B. Provide original code for function(s) in text format that perform the transformation(s) you identified in part A4.

```
--Task B  Function to extract month from rental_date

CREATE OR REPLACE FUNCTION extract_month(rental_date timestamp)
   RETURNS int
   LANGUAGE plpgsql
AS
$$
DECLARE rental_mo int;
BEGIN
   SELECT EXTRACT(MONTH FROM rental_date) INTO rental_mo;
   RETURN rental_mo;
END;
$$
```

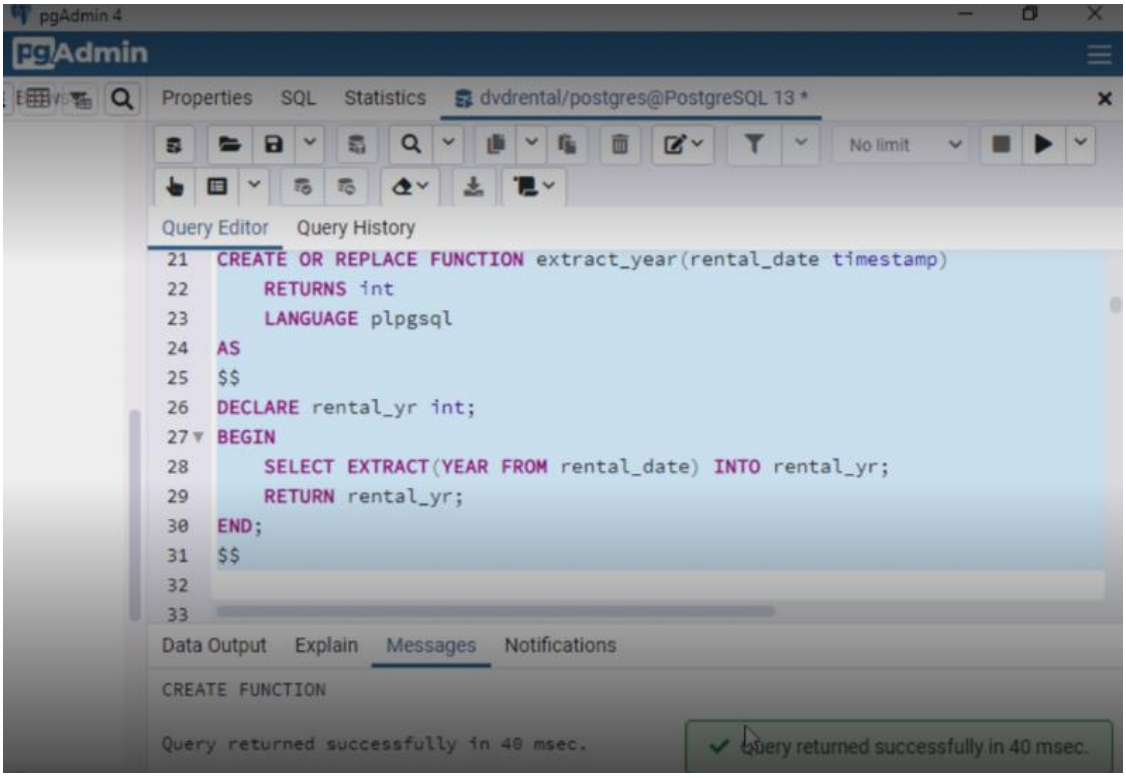


--Test Code
SELECT rental_date, extract_month(rental_date) AS rental_mo
FROM rental;

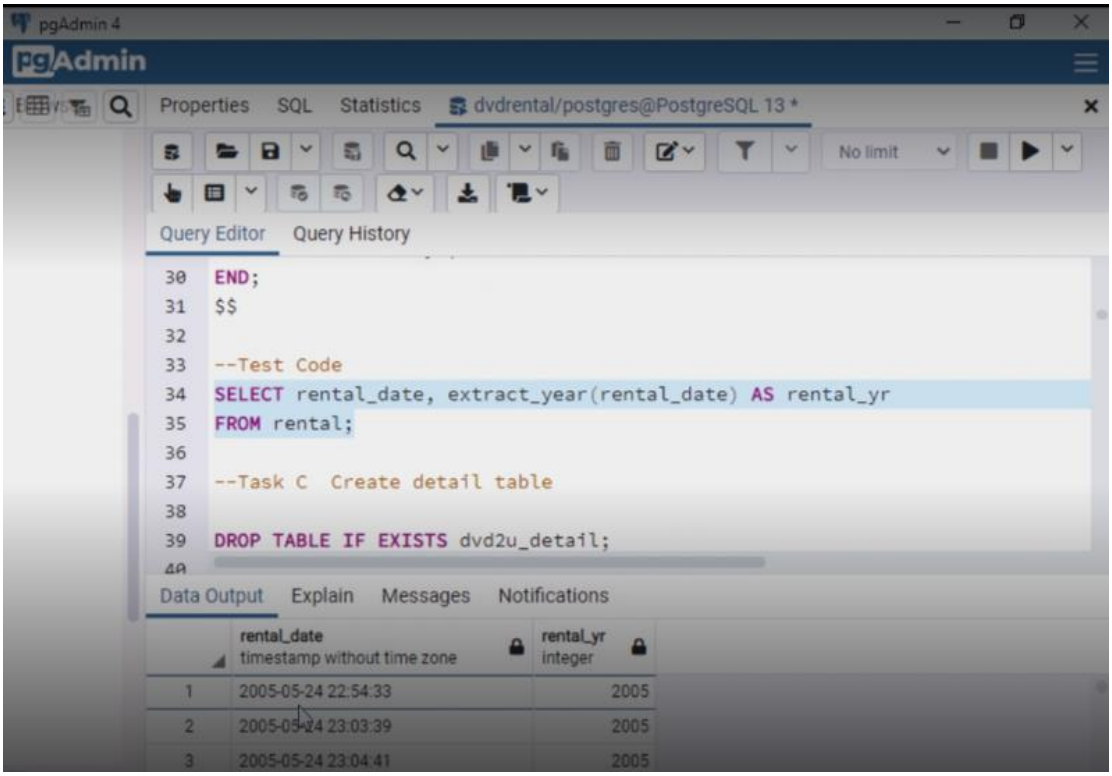


--Task B Function to extract year from rental_date

```
CREATE OR REPLACE FUNCTION extract_year(rental_date timestamp)
   RETURNS int
    LANGUAGE plpgsql
AS
$$
DECLARE rental_yr int;
BEGIN
   SELECT EXTRACT(YEAR FROM rental_date) INTO rental_yr;
    RETURN rental_yr;
END;
$$
```



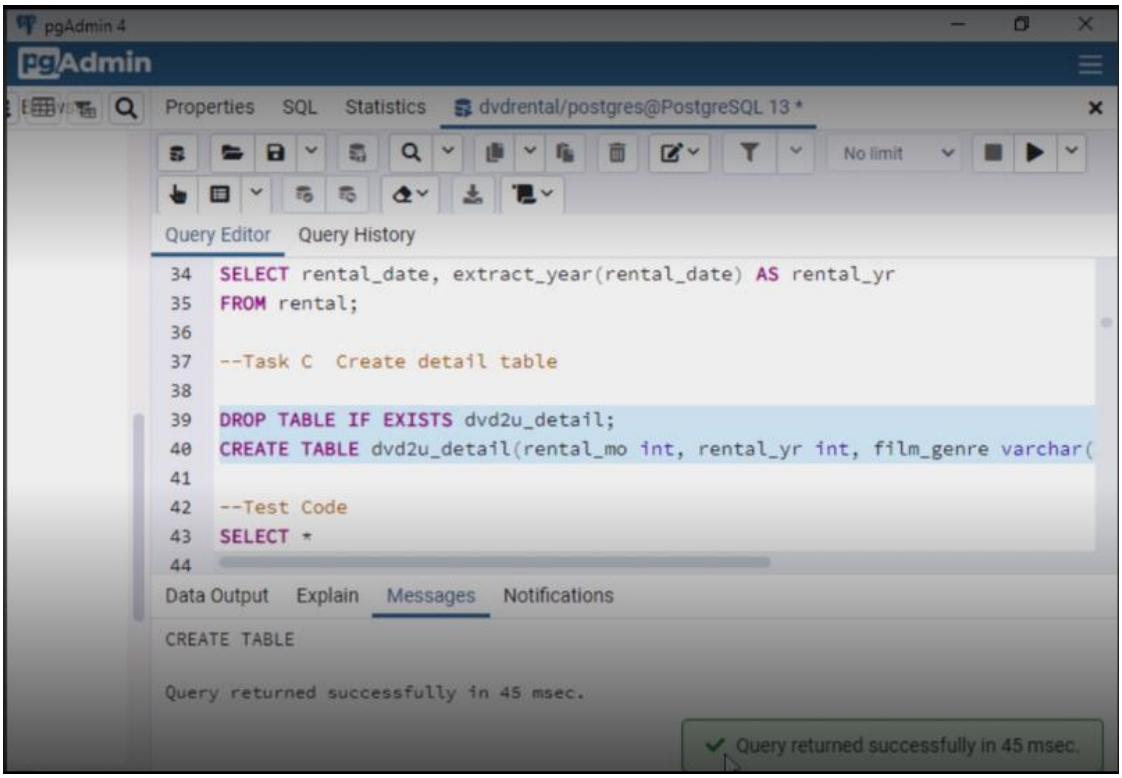
--Test Code
SELECT rental_date, extract_year(rental_date) AS rental_yr
FROM rental;



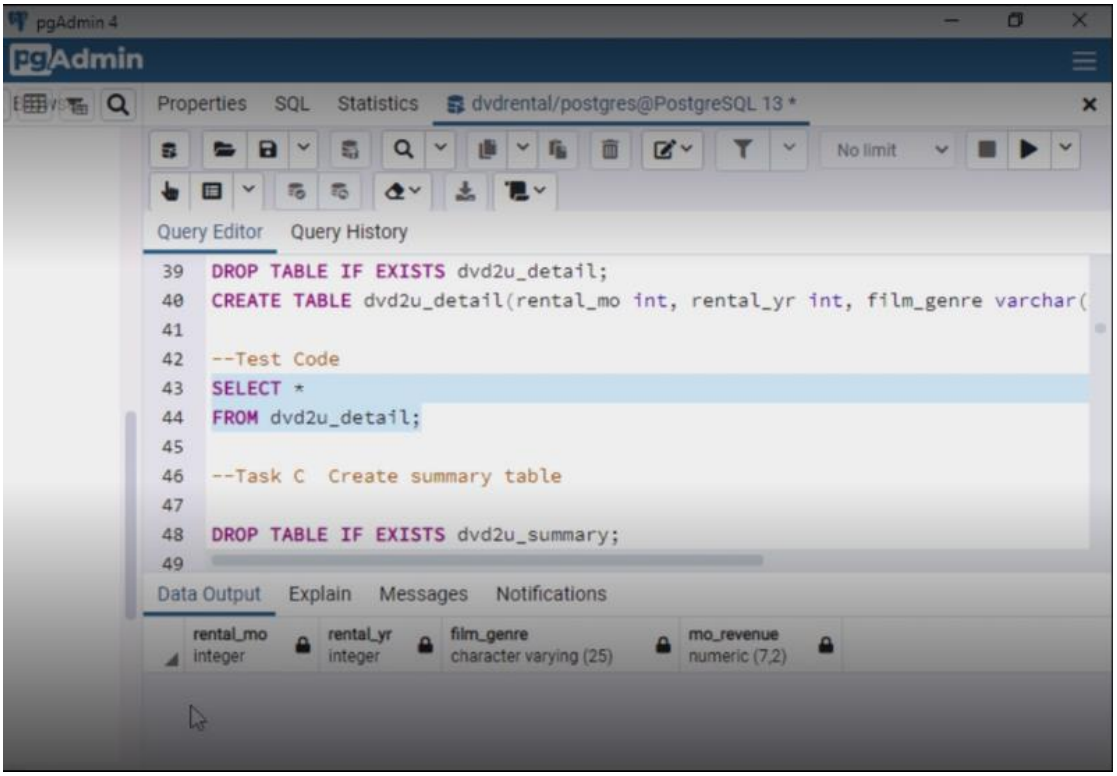
C. Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections.

--Task C Create detail table

DROP TABLE IF EXISTS dvd2u_detail;
CREATE TABLE dvd2u_detail(rental_mo int, rental_yr int, film_genre varchar(25), mo_revenue numeric(7,2));

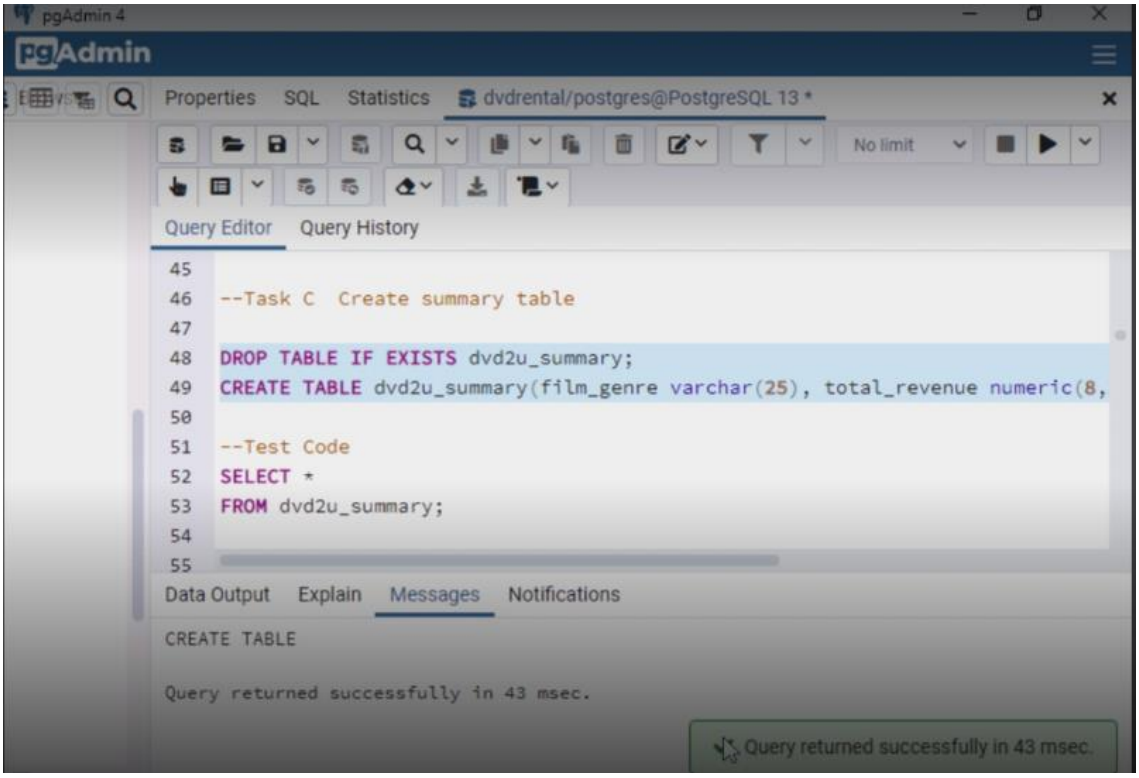


--Test Code
SELECT *
FROM dvd2u_detail;

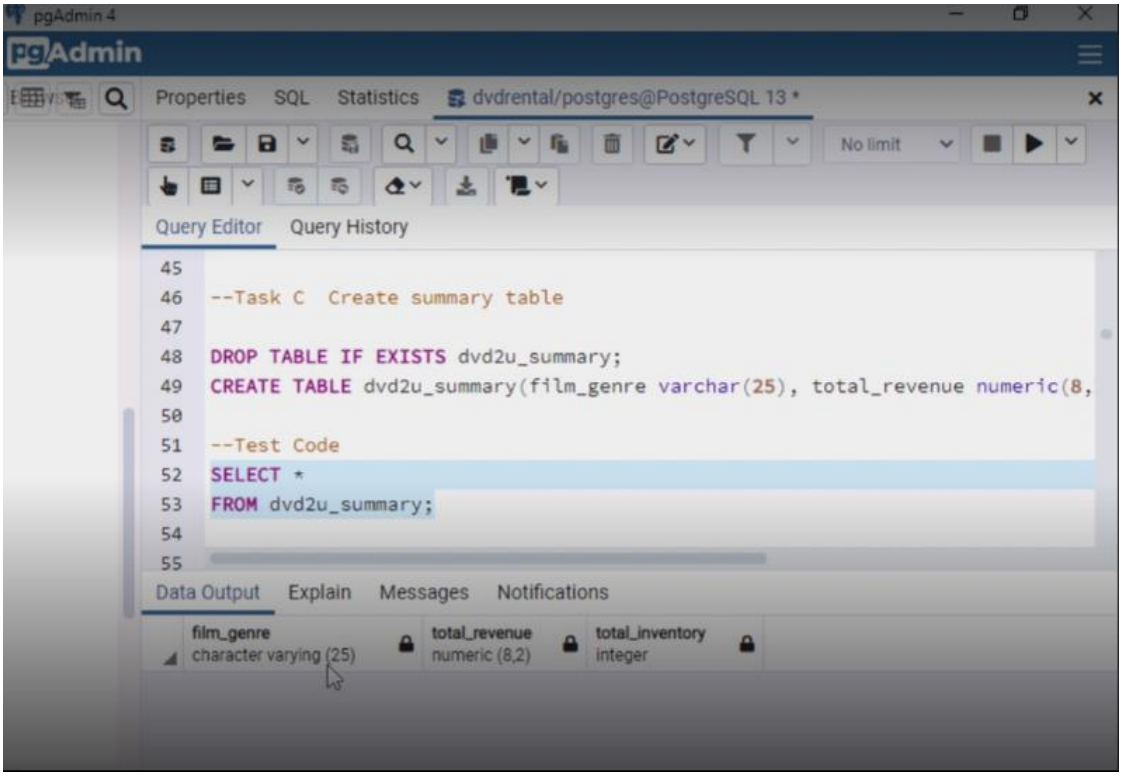


--Task C Create summary table

DROP TABLE IF EXISTS dvd2u_summary;
CREATE TABLE dvd2u_summary(film_genre varchar(25), total_revenue numeric(8,2), total_inventory int);

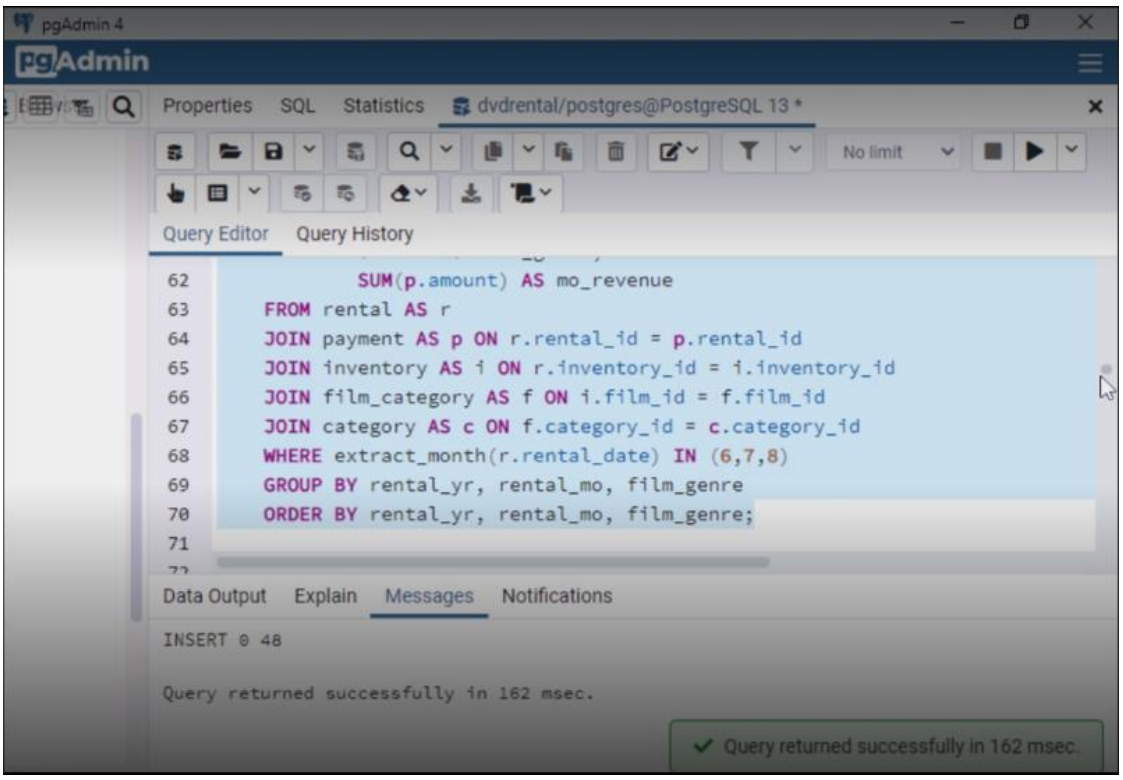


```
--Test Code
SELECT *
FROM dvd2u_summary;
```

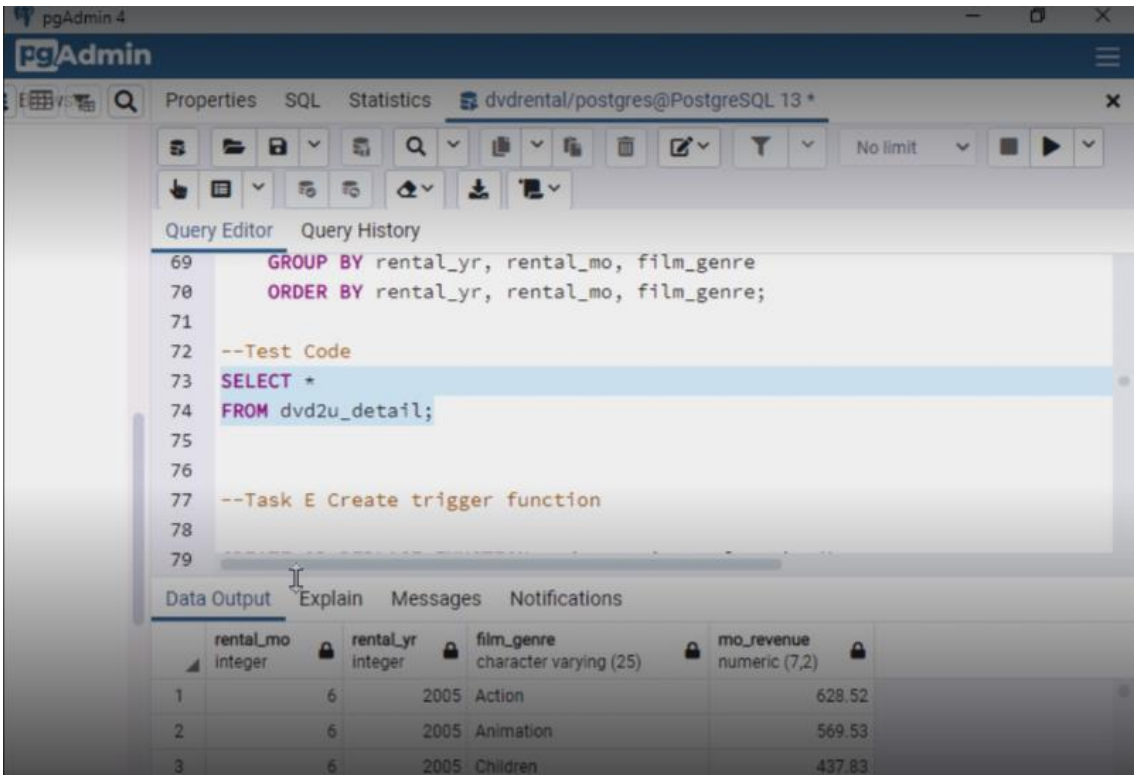


D. Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database.

```
--Task D Input Detail info into Detail Table

INSERT INTO dvd2u_detail
        SELECT      extract_month(r.rental_date) AS rental_mo,
                    extract_year(r.rental_date) AS rental_yr,
                    c.name AS film_genre,
                    SUM(p.amount) AS mo_revenue
        FROM rental AS r
        JOIN payment AS p ON r.rental_id = p.rental_id
        JOIN inventory AS i ON r.inventory_id = i.inventory_id
        JOIN film_category AS f ON i.film_id = f.film_id
        JOIN category AS c ON f.category_id = c.category_id
        WHERE extract_month(r.rental_date) IN (6,7,8)
        GROUP BY rental_yr, rental_mo, film_genre
        ORDER BY rental_yr, rental_mo, film_genre;
```

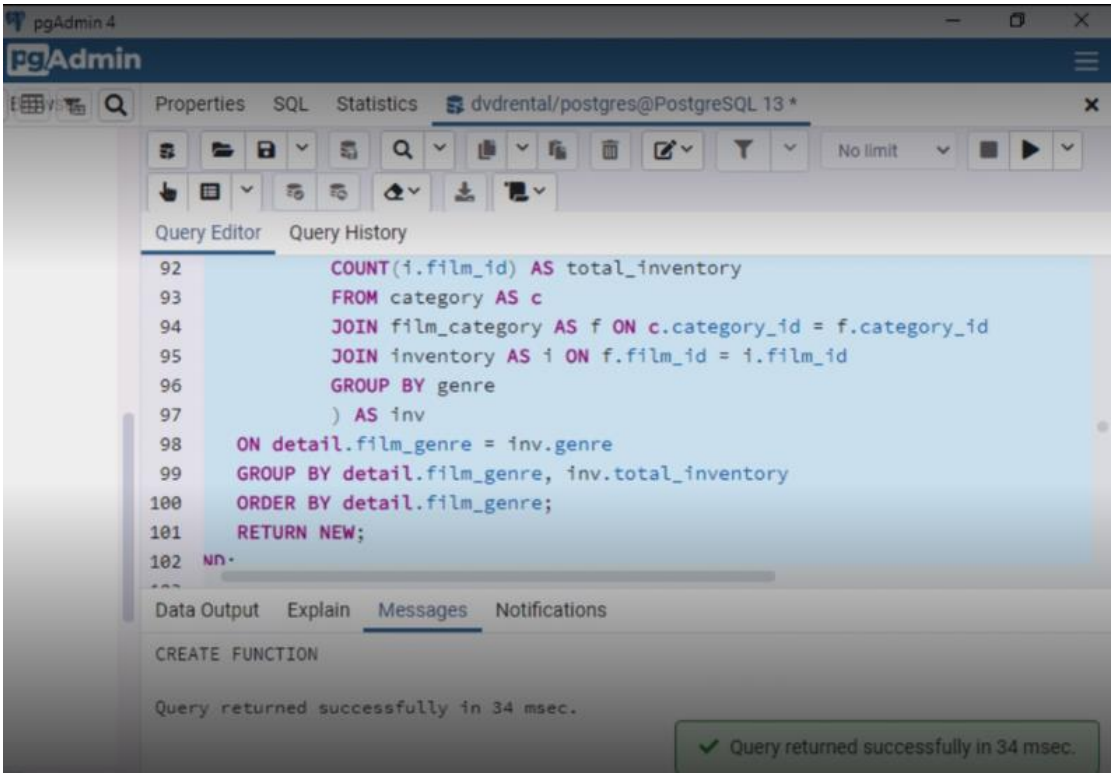


--Test Code
SELECT *
FROM dvd2u_detail;



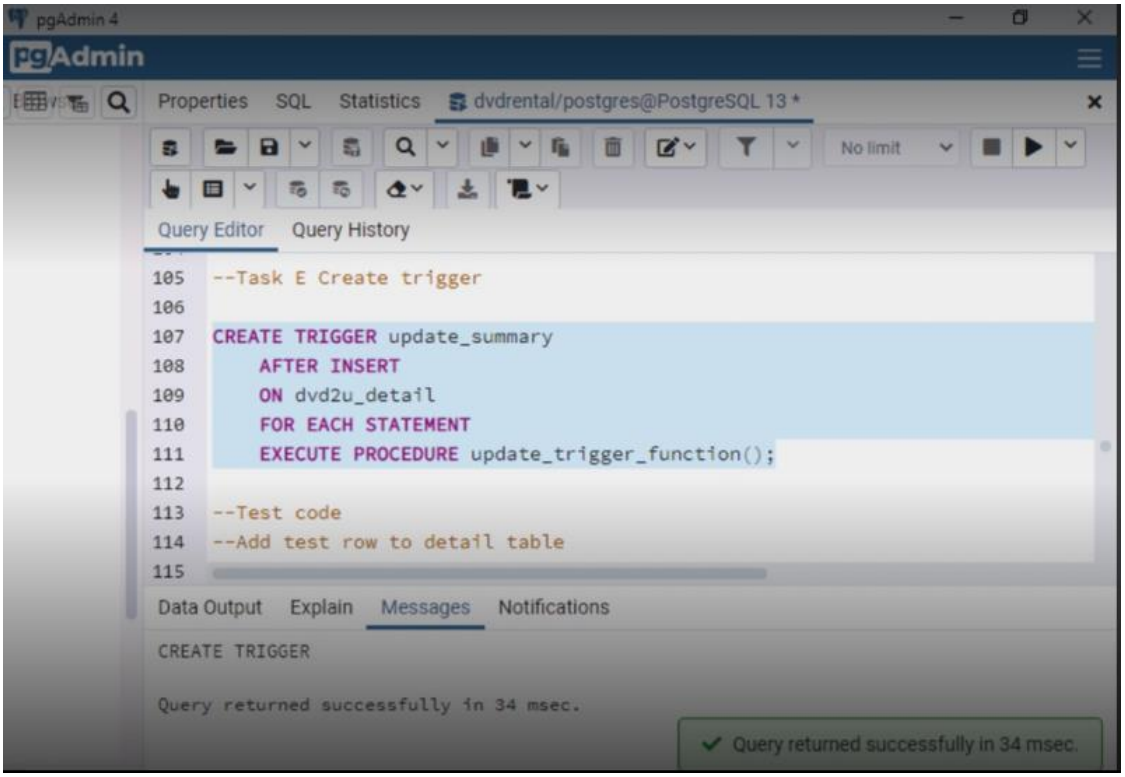
E. Provide original SQL code in a text format that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table

```
--Task E Create trigger function

CREATE OR REPLACE FUNCTION update_trigger_function()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS $$
BEGIN
    DELETE FROM dvd2u_summary;
    INSERT INTO dvd2u_summary
    SELECT  detail.film_genre,
            SUM(detail.mo_revenue) AS total_revenue,
            inv.total_inventory
    FROM dvd2u_detail AS detail
    JOIN (
            SELECT    c.name as genre,
                      COUNT(i.film_id) AS total_inventory
            FROM category AS c
            JOIN film_category AS f ON c.category_id = f.category_id
            JOIN inventory AS i ON f.film_id = i.film_id
            GROUP BY genre
            ) AS inv
    ON detail.film_genre = inv.genre
    GROUP BY detail.film_genre, inv.total_inventory
    ORDER BY detail.film_genre;
    RETURN NEW;
END;
$$;
```

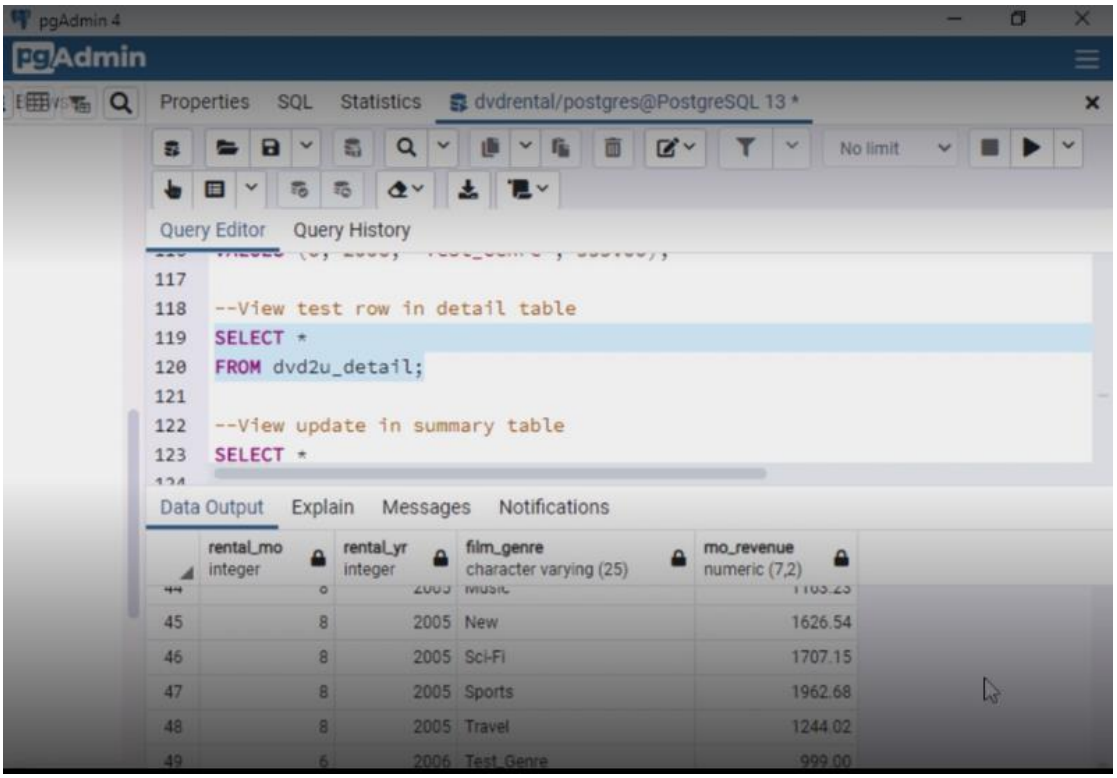

pgAdmin 4
pgAdmin
Properties
SQL
Statistics
dvdrental/postgres@PostgreSQL 13 *
No limit
Query Editor
Query History
92 COUNT(i.film_id) AS total_inventory
93 FROM category AS c
94 JOIN film_category AS f ON c.category_id = f.category_id
95 JOIN inventory AS i ON f.film_id = i.film_id
96 GROUP BY genre
97) AS inv
98 ON detail.film_genre = inv.genre
99 GROUP BY detail.film_genre, inv.total_inventory
100 ORDER BY detail.film_genre;
101 RETURN NEW;
102 ND;
Data Output
Explain
Messages
Notifications
CREATE FUNCTION
Query returned successfully in 34 msec.
Query returned successfully in 34 msec.

--Task E Create trigger

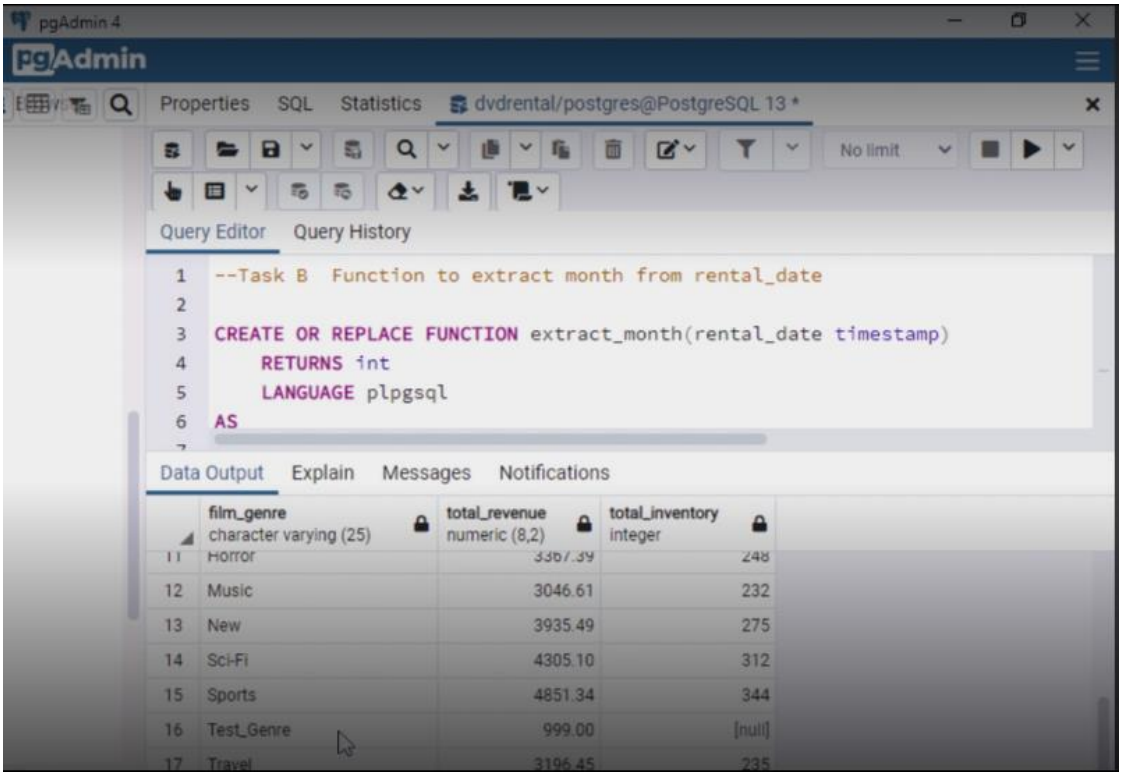
```
CREATE TRIGGER update_summary
        AFTER INSERT
        ON dvd2u_detail
        FOR EACH STATEMENT
        EXECUTE PROCEDURE update_trigger_function();
```



```
--Test code
--Add test row to detail table
INSERT INTO dvd2u_detail(rental_mo, rental_yr, film_genre, mo_revenue)
VALUES (6, 2006, Test_Genre, 999.00);

--View test row in detail table
SELECT *
FROM dvd2u_detail;
```

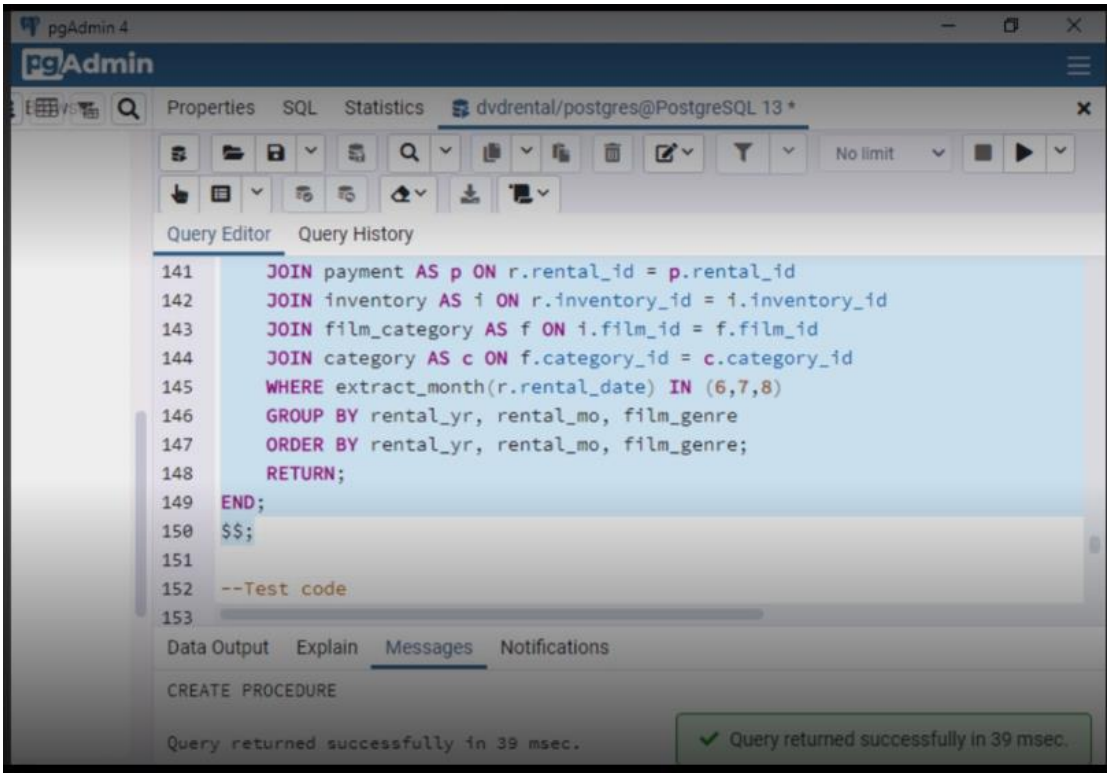


--View update in summary table
SELECT *
FROM dvd2u_summary;

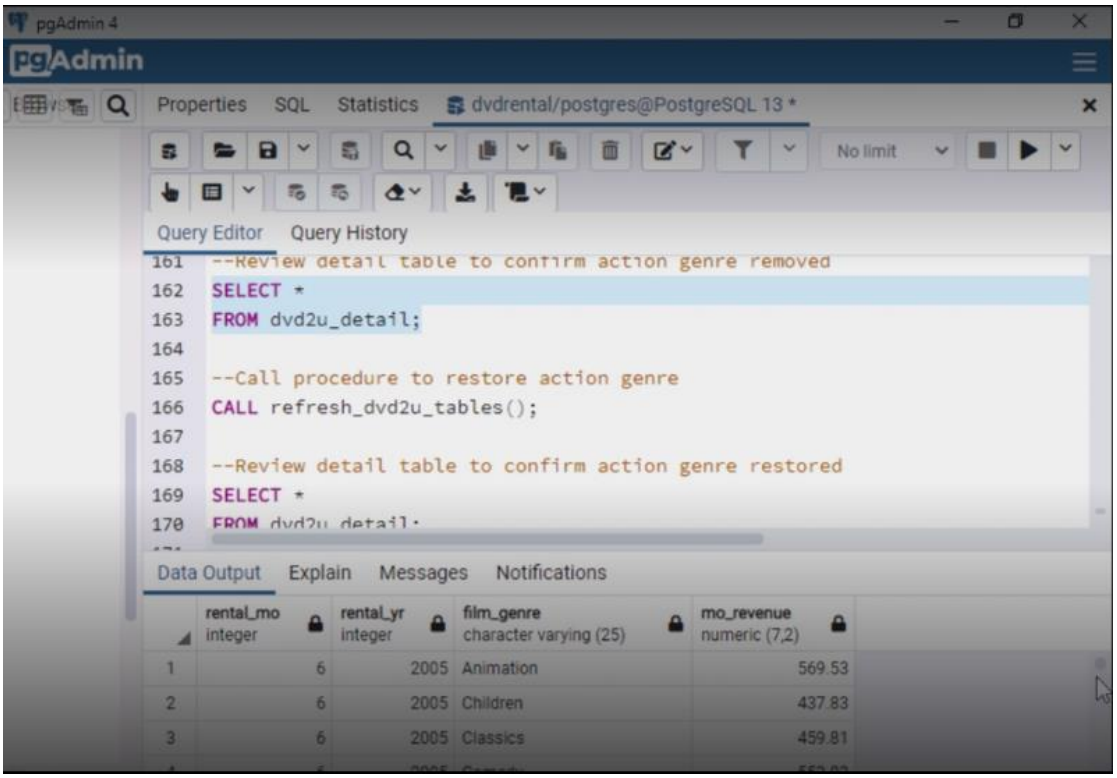


F. Provide an original stored procedure in a text format that can be used to refresh the data in both the detailed table and summary table. The procedure should clear the contents of the detailed table and summary table and perform the raw data extraction from part D.

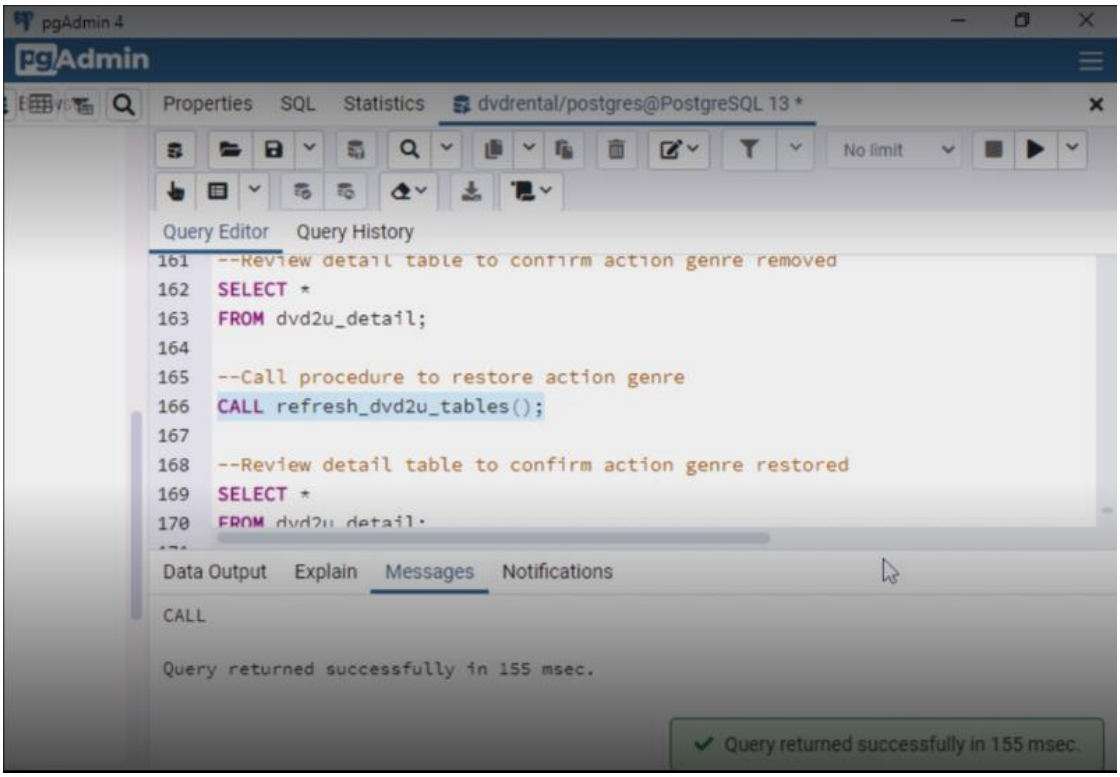
```
/*Task F Create a stored procedure to clear contents of detail & summary table and perform
data extraction from part D*/

CREATE OR REPLACE PROCEDURE refresh_dvd2u_tables()
LANGUAGE plpgsql
AS $$
BEGIN
        DELETE FROM dvd2u_detail;
        DELETE FROM dvd2u_summary;
        INSERT INTO dvd2u_detail
        SELECT          extract_month(r.rental_date) AS rental_mo,
                        extract_year(r.rental_date) AS rental_yr,
                        c.name AS film_genre,
                        SUM(p.amount) AS mo_revenue
        FROM rental AS r
        JOIN payment AS p ON r.rental_id = p.rental_id
        JOIN inventory AS i ON r.inventory_id = i.inventory_id
        JOIN film_category AS f ON i.film_id = f.film_id
        JOIN category AS c ON f.category_id = c.category_id
        WHERE extract_month(r.rental_date) IN (6,7,8)
        GROUP BY rental_yr, rental_mo, film_genre
        ORDER BY rental_yr, rental_mo, film_genre;
RETURN;
END;
$$;
```


pgAdmin 4
pgAdmin
Properties
SQL
Statistics
dvdrental/postgres@PostgreSQL 13 *
No limit
Query Editor
Query History
141 JOIN payment AS p ON r.rental_id = p.rental_id
142 JOIN inventory AS i ON r.inventory_id = i.inventory_id
143 JOIN film_category AS f ON i.film_id = f.film_id
144 JOIN category AS c ON f.category_id = c.category_id
145 WHERE extract_month(r.rental_date) IN (6,7,8)
146 GROUP BY rental_yr, rental_mo, film_genre
147 ORDER BY rental_yr, rental_mo, film_genre;
148 RETURN;
149 END;
150 $$;
151
152 --Test code
153
Data Output
Explain
Messages
Notifications
CREATE PROCEDURE
Query returned successfully in 39 msec.
Query returned successfully in 39 msec.

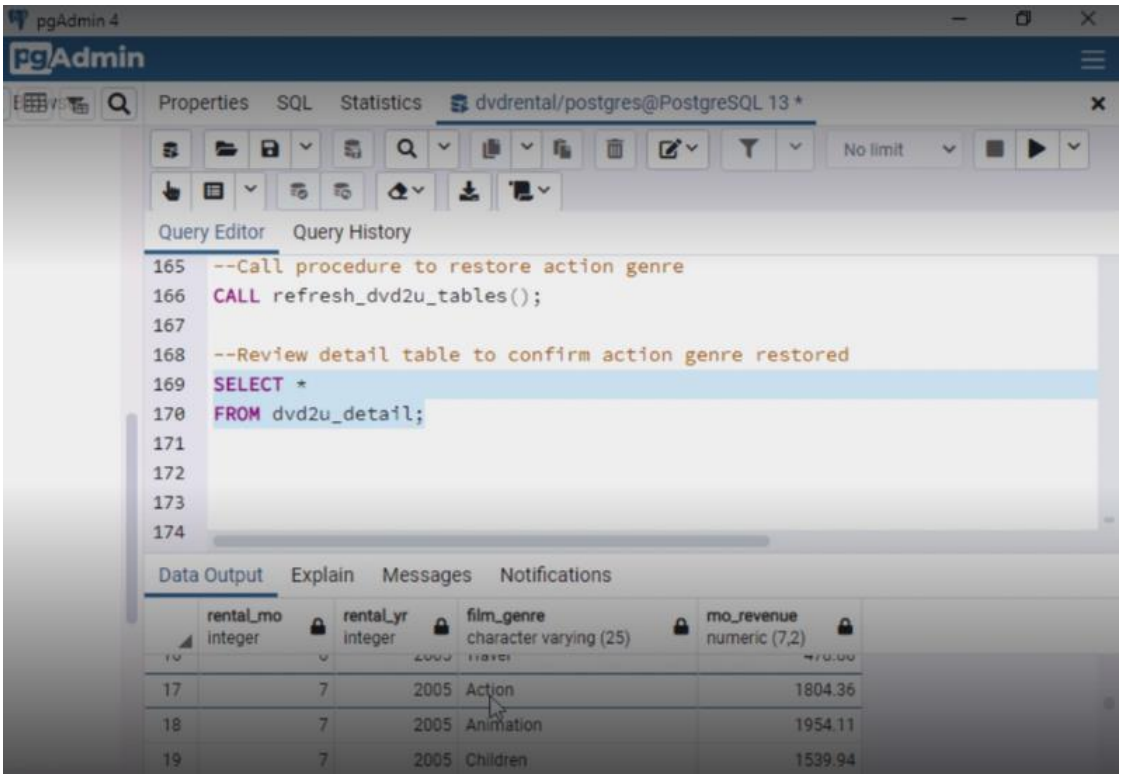
```
--Test code
--Review detail table
SELECT *
FROM dvd2u_detail;

--Remove action genre
DELETE FROM dvd2u_detail
WHERE film_genre = 'Action';

--Review detail table to confirm action genre removed
SELECT *
FROM dvd2u_detail;
```



--Call procedure to restore action genre
CALL refresh_dvd2u_tables();



```
--Review detail table to confirm action genre restored
SELECT *
FROM dvd2u_detail;
```



1. Identify a relevant job scheduling tool that can be used to automate the stored procedure.

Because PostgreSQL does not have its own built-in scheduler, pgAgent can be used to to execute the stored procedure. Scheduling can be managed by pgAdmin 4 but must be manually installed, as it is not part of the default installation. Once installed, a user would right click on pgAgent Jobs to create a new job that would automatically execute the stored procedure per user-defined execution steps (Dias, H., 2020).

References

Dias, H. (2020). *An Overview of Job Scheduling Tools for PostgreSQL.* https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/