

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.

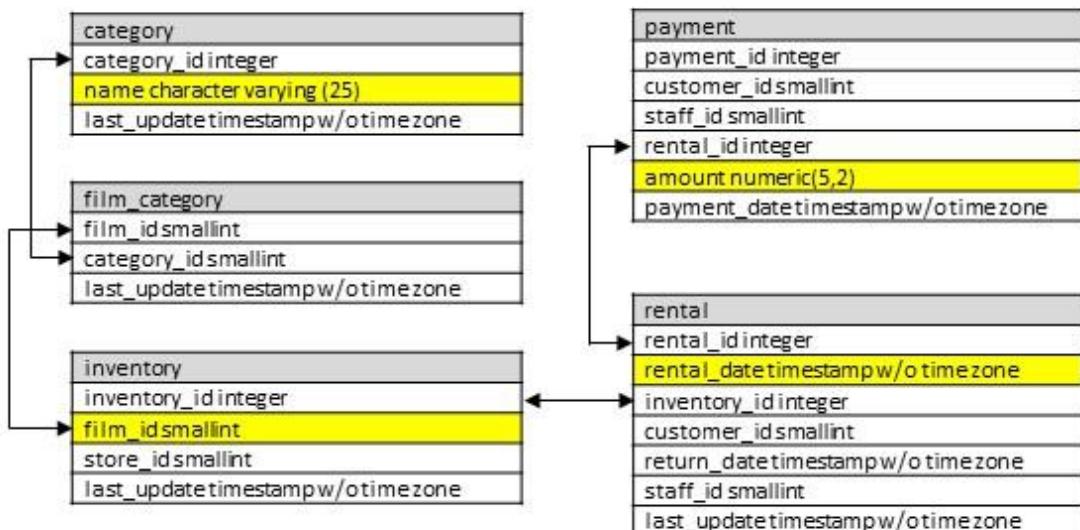
category_id is an integer (also shown as smallint) – it will be used to join the `film_category` and `category` tables.

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.

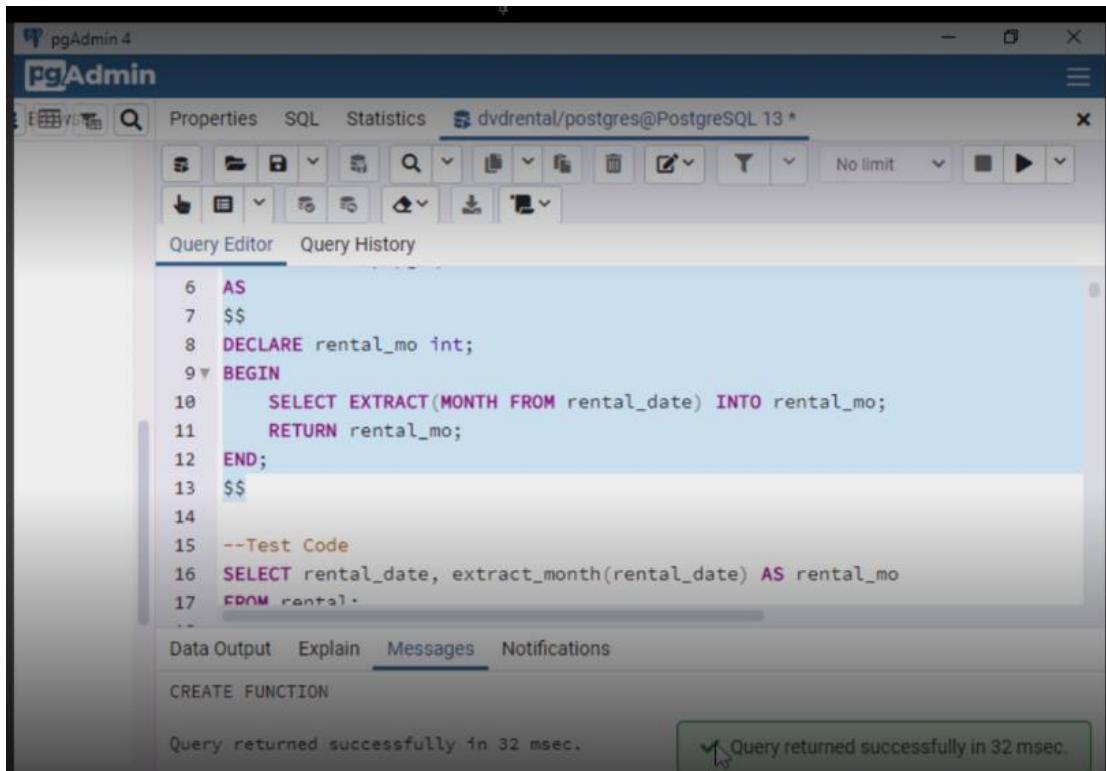
6. Explain how frequently your report should be refreshed to remain relevant to stakeholders.

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
$$
DECLARE rental_mo int;
BEGIN
    SELECT EXTRACT(MONTH FROM rental_date) INTO rental_mo;
    RETURN rental_mo;
END;
$$
```



The screenshot shows the pgAdmin 4 interface with a query editor window. The code in the editor is:

```

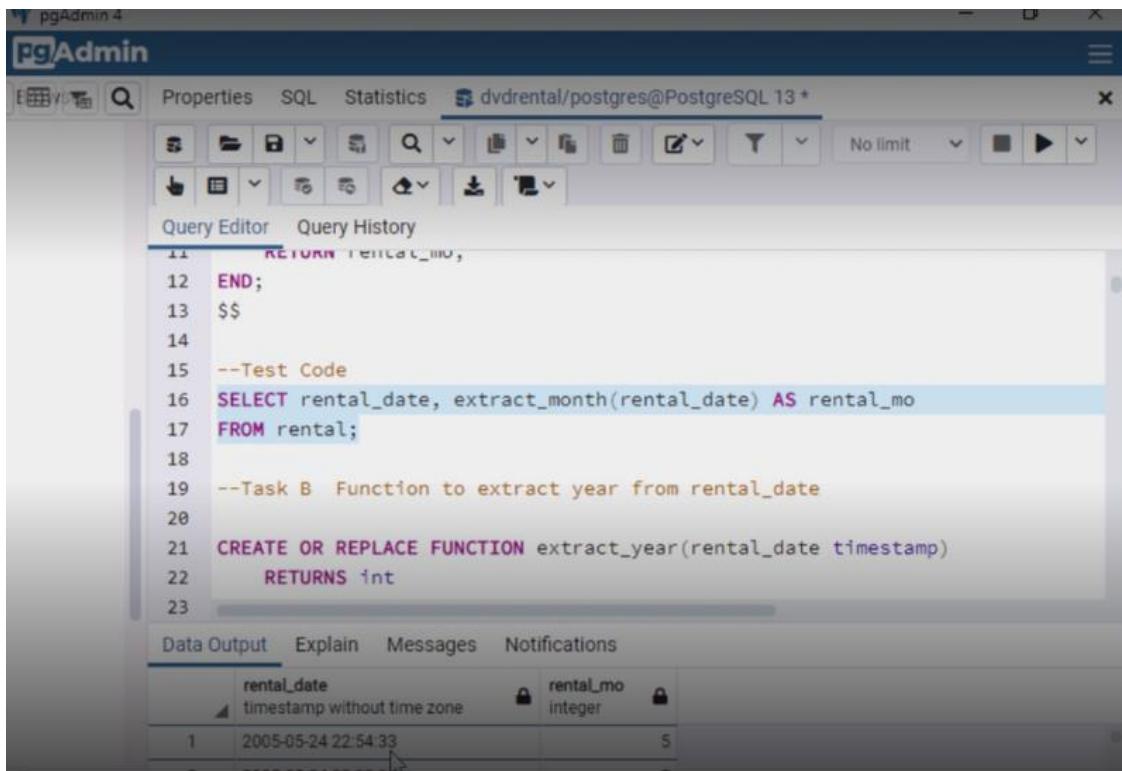
6 AS
7 $$
8 DECLARE rental_mo int;
9 BEGIN
10   SELECT EXTRACT(MONTH FROM rental_date) INTO rental_mo;
11   RETURN rental_mo;
12 END;
$$
14
15 --Test Code
16 SELECT rental_date, extract_month(rental_date) AS rental_mo
17 FROM rental;

```

The status bar at the bottom indicates "Query returned successfully in 32 msec.".

--Test Code

```
SELECT rental_date, extract_month(rental_date) AS rental_mo
FROM rental;
```



The screenshot shows the pgAdmin 4 interface with a query editor window. The code in the editor is:

```

11   RETURN rental_mo,
12 END;
$$
14
15 --Test Code
16 SELECT rental_date, extract_month(rental_date) AS rental_mo
17 FROM rental;
18
19 --Task B Function to extract year from rental_date
20
21 CREATE OR REPLACE FUNCTION extract_year(rental_date timestamp)
22 RETURNS int
23

```

The status bar at the bottom indicates "Query returned successfully in 32 msec.".

rental_date	rental_mo
timestamp without time zone	integer
2005-05-24 22:54:33	5

--Task B Function to extract year from rental_date

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

The screenshot shows the pgAdmin 4 interface with a query editor window. The query editor contains the SQL code for creating a function named 'extract_year'. The code defines the function to take a timestamp parameter, return an integer, and use the 'plpgsql' language. It declares a variable 'rental_yr' of type integer, begins a block, selects the year from the input timestamp into 'rental_yr', returns 'rental_yr', ends the block, and ends the function definition with '\$\$'. The pgAdmin interface includes a toolbar, a menu bar with 'Properties', 'SQL', 'Statistics', and a connection tab for 'dvdrental/postgres@PostgreSQL 13 *'. Below the toolbar are various icons for file operations like open, save, and delete. The main area has tabs for 'Query Editor' (which is selected) and 'Query History'. At the bottom, there are tabs for 'Data Output', 'Explain', 'Messages' (which is selected), and 'Notifications'. A message bar at the bottom indicates that the query was executed successfully in 40 msec.

```
21 CREATE OR REPLACE FUNCTION extract_year(rental_date timestamp)
22 RETURNS int
23 LANGUAGE plpgsql
24 AS
25 $$
26 DECLARE rental_yr int;
27 BEGIN
28     SELECT EXTRACT(YEAR FROM rental_date) INTO rental_yr;
29     RETURN rental_yr;
30 END;
31 $$
```

CREATE FUNCTION

Query returned successfully in 40 msec.

--Test Code

```
SELECT rental_date, extract_year(rental_date) AS rental_yr  
FROM rental;
```

The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin 4" and "pgAdmin". The menu bar includes Properties, SQL, Statistics, and a connection to "dvdrental/postgres@PostgreSQL 13 *". The toolbar has various icons for database management. The main area is divided into "Query Editor" and "Query History". The Query Editor contains the following code:

```
30  END;  
31  $$  
32  
33  --Test Code  
34  SELECT rental_date, extract_year(rental_date) AS rental_yr  
35  FROM rental;  
36  
37  --Task C  Create detail table  
38  
39  DROP TABLE IF EXISTS dvd2u_detail;  
40
```

Below the code, there are tabs for Data Output, Explain, Messages, and Notifications. The Data Output tab shows a table with two columns: "rental_date" (timestamp without time zone) and "rental_yr" (integer). The data is as follows:

	rental_date	rental_yr
1	2005-05-24 22:54:33	2005
2	2005-05-24 23:03:39	2005
3	2005-05-24 23:04:41	2005

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));
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The title bar says 'pgAdmin 4' and 'Properties SQL Statistics dvdrental/postgres@PostgreSQL 13 *'. The main area is a 'Query Editor' tab with the following SQL code:

```
34 SELECT rental_date, extract_year(rental_date) AS rental_yr
35 FROM rental;
36
37 --Task C Create detail table
38
39 DROP TABLE IF EXISTS dvd2u_detail;
40 CREATE TABLE dvd2u_detail(rental_mo int, rental_yr int, film_genre varchar(
41
42 --Test Code
43 SELECT *
```

Below the code, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing the message 'CREATE TABLE'. At the bottom, a status bar says 'Query returned successfully in 45 msec.' with a green checkmark icon.

```
--Test Code  
SELECT *  
FROM dvd2u_detail;
```

The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin 4" and "PgAdmin". The main window is titled "dvdrental/postgres@PostgreSQL 13 *". The toolbar has various icons for database management. Below the toolbar is a menu bar with "Properties", "SQL", "Statistics", and "No limit". The main area is a "Query Editor" tab, which is active. It contains the following PostgreSQL code:

```
39  DROP TABLE IF EXISTS dvd2u_detail;  
40  CREATE TABLE dvd2u_detail(rental_mo int, rental_yr int, film_genre varchar(  
41  
42  --Test Code  
43  SELECT *  
44  FROM dvd2u_detail;  
45  
46  --Task C Create summary table  
47  
48  DROP TABLE IF EXISTS dvd2u_summary;  
49
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". A preview of the table structure is shown:

rental_mo	rental_yr	film_genre	mo_revenue
integer	integer	character varying (25)	numeric (7,2)

--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);
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The title bar reads "pgAdmin 4" and "Properties SQL Statistics dvdrental/postgres@PostgreSQL 13 *". The main area is a "Query Editor" tab containing the following SQL code:

```
45
46 --Task C Create summary table
47
48 DROP TABLE IF EXISTS dvd2u_summary;
49 CREATE TABLE dvd2u_summary(film_genre varchar(25), total_revenue numeric(8,
50
51 --Test Code
52 SELECT *
53 FROM dvd2u_summary;
54
55
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the message "CREATE TABLE". At the bottom of the screen, a status bar displays "Query returned successfully in 43 msec.".

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

```

45
46  --Task C  Create summary table
47
48  DROP TABLE IF EXISTS dvd2u_summary;
49  CREATE TABLE dvd2u_summary(film_genre varchar(25), total_revenue numeric(8,
50
51  --Test Code
52  SELECT *
53  FROM dvd2u_summary;
54
55

```

film_genre	total_revenue	total_inventory
character varying (25)	numeric (8,2)	integer

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;
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The query is a complex SQL statement involving multiple joins and aggregate functions. The status bar at the bottom indicates "Query returned successfully in 162 msec." with a green checkmark icon.

```

62     SUM(p.amount) AS mo_revenue
63   FROM rental AS r
64   JOIN payment AS p ON r.rental_id = p.rental_id
65   JOIN inventory AS i ON r.inventory_id = i.inventory_id
66   JOIN film_category AS f ON i.film_id = f.film_id
67   JOIN category AS c ON f.category_id = c.category_id
68 WHERE extract_month(r.rental_date) IN (6,7,8)
69 GROUP BY rental_yr, rental_mo, film_genre
70 ORDER BY rental_yr, rental_mo, film_genre;
71
72
73
74
75
76
77
78
79

```

INSERT 0 48

Query returned successfully in 162 msec.

✓ Query returned successfully in 162 msec.

--Test Code

```

SELECT *
FROM dvd2u_detail;

```

The screenshot shows the pgAdmin 4 interface with a query editor window containing a test code section. The results pane below displays a table with three rows of data, corresponding to the results of the test code query.

```

69     GROUP BY rental_yr, rental_mo, film_genre
70     ORDER BY rental_yr, rental_mo, film_genre;
71
72 --Test Code
73 SELECT *
74 FROM dvd2u_detail;
75
76
77 --Task E Create trigger function
78
79

```

Data Output Explain Messages Notifications

	rental_mo	rental_yr	film_genre	mo_revenue
1	6	2005	Action	628.52
2	6	2005	Animation	569.53
3	6	2005	Children	437.83

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

The screenshot shows the pgAdmin 4 interface with a query editor window. The title bar reads "pgAdmin 4" and "dvdrental/postgres@PostgreSQL 13 *". The main area is a "Query Editor" containing the following SQL code:

```
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 END;
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the message "Query returned successfully in 34 msec." with a green checkmark icon.

--Task E Create trigger

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

The screenshot shows the pgAdmin 4 interface with a query editor window. The title bar says "pgAdmin 4" and "PgAdmin". The menu bar includes "Properties", "SQL", "Statistics", and a connection tab "dvdrental/postgres@PostgreSQL 13 *". The toolbar has various icons for database management. The query editor tab is selected, showing the following code:

```
105 --Task E Create trigger
106
107 CREATE TRIGGER update_summary
108   AFTER INSERT
109   ON dvd2u_detail
110   FOR EACH STATEMENT
111   EXECUTE PROCEDURE update_trigger_function();
112
113 --Test code
114 --Add test row to detail table
115
```

Below the code, the "Messages" tab is selected, displaying the output:

```
CREATE TRIGGER
```

Query returned successfully in 34 msec.

A green message box at the bottom right indicates: ✓ Query returned successfully in 34 msec.

```
--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;
```

The screenshot shows the pgAdmin 4 interface with a query editor containing the following SQL code:

```
117
118 --View test row in detail table
119 SELECT *
120 FROM dvd2u_detail;
121
122 --View update in summary table
123 SELECT *
```

Below the code, the Data Output tab displays the results of the query:

	rental_mo	rental_yr	film_genre	mo_revenue
44	integer	integer	character varying (25)	numeric (7,2)
45	8	2005	New	1626.54
46	8	2005	Sci-Fi	1707.15
47	8	2005	Sports	1962.68
48	8	2005	Travel	1244.02
49	6	2006	Test_Genre	999.00

--View update in summary table

```
SELECT *  
FROM dvd2u_summary;
```

The screenshot shows the pgAdmin 4 interface. The top bar displays 'pgAdmin 4' and the connection details 'dvdrental/postgres@PostgreSQL 13 *'. The main area has two tabs: 'Query Editor' (selected) and 'Query History'. In the Query Editor, the following SQL code is visible:

```
1 --Task B Function to extract month from rental_date  
2  
3 CREATE OR REPLACE FUNCTION extract_month(rental_date timestamp)  
4     RETURNS int  
5     LANGUAGE plpgsql  
6 AS
```

Below the code, the 'Data Output' tab is selected, showing a table with the following data:

	film_genre	total_revenue	total_inventory
11	character varying (25) Horror	330 / .39	248
12	Music	3046.61	232
13	New	3935.49	275
14	Sci-Fi	4305.10	312
15	Sports	4851.34	344
16	Test_Genre	999.00	[null]
17	Travel	3196.45	235

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;
$$;
```

The screenshot shows the pgAdmin 4 interface with a query editor window. The title bar reads "pgAdmin 4" and "Properties SQL Statistics". The connection tab says "dvdrental/postgres@PostgreSQL 13 *". The toolbar has various icons for file operations like Open, Save, and Print. Below the toolbar is a set of small icons for navigating between queries, including back, forward, and search. The main area is the "Query Editor" tab, which contains the following SQL code:

```
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
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the message "CREATE PROCEDURE". At the bottom of the screen, a green status bar indicates "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;
```

The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin 4" and "PgAdmin". The menu bar includes "File", "Edit", "View", "Properties", "SQL", "Statistics", and "dvdrental/postgres@PostgreSQL 13 *". The toolbar has various icons for database management. The main area is divided into "Query Editor" and "Query History". The Query Editor contains the following PostgreSQL code:

```
161 --Review detail table to confirm action genre removed
162 SELECT *
163 FROM dvd2u_detail;
164
165 --Call procedure to restore action genre
166 CALL refresh_dvd2u_tables();
167
168 --Review detail table to confirm action genre restored
169 SELECT *
170 FROM dvd2u_detail.
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Data Output" tab is selected and displays a table with the following data:

	rental_mo	rental_yr	film_genre	mo_revenue
	integer	integer	character varying (25)	numeric (7,2)
1		6	2005 Animation	569.53
2		6	2005 Children	437.83
3		6	2005 Classics	459.81
4		6	2005 Comedy	550.00

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

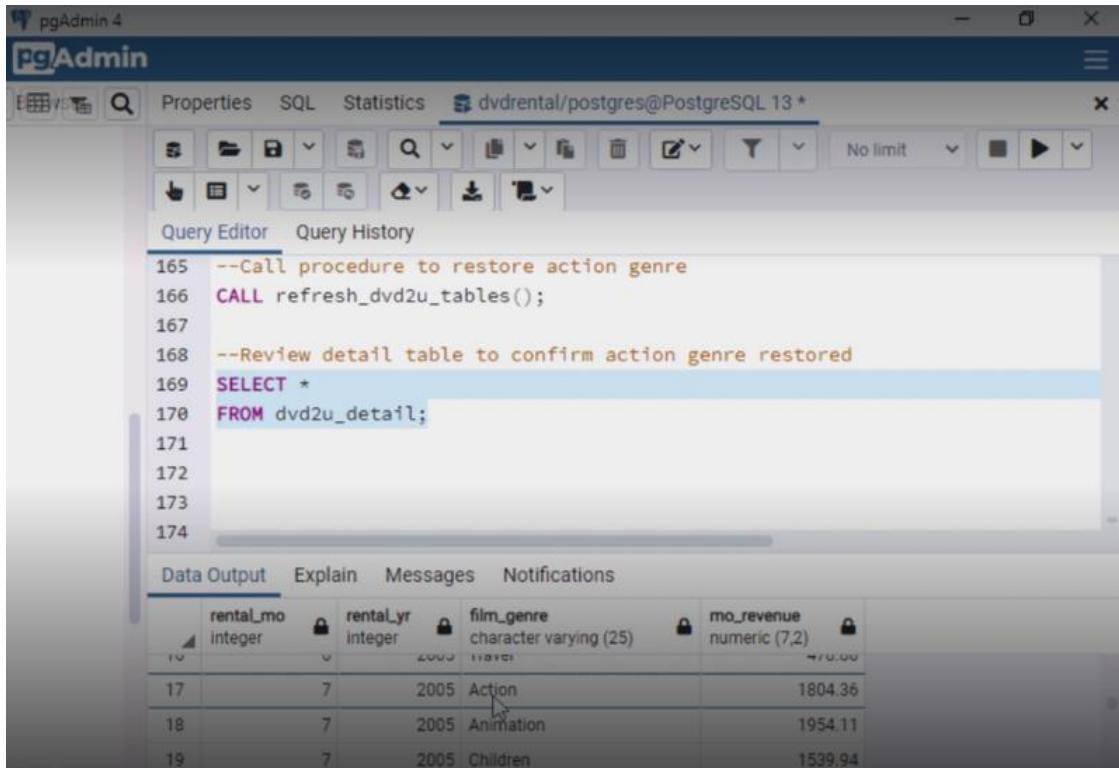
The screenshot shows the pgAdmin 4 interface with a query editor window. The connection is set to `dvdrental/postgres@PostgreSQL 13*`. The query editor contains the following SQL code:

```
--Review detail table to confirm action genre removed  
162 SELECT *  
163 FROM dvd2u_detail;  
164  
165 --Call procedure to restore action genre  
166 CALL refresh_dvd2u_tables();  
167  
168 --Review detail table to confirm action genre restored  
169 SELECT *  
170 FROM dvd2u_detail.
```

The message bar at the bottom right indicates: **✓ Query returned successfully in 155 msec.**

--Review detail table to confirm action genre restored

```
SELECT *
FROM dvd2u_detail;
```



The screenshot shows the pgAdmin 4 interface. The title bar says "pgAdmin". The toolbar has various icons for database management. The menu bar includes "Properties", "SQL", "Statistics", and "dvdrental/postgres@PostgreSQL 13 *". The main area has two tabs: "Query Editor" (selected) and "Query History". The Query Editor contains the following SQL code:

```

165 --Call procedure to restore action genre
166 CALL refresh_dvd2u_tables();
167
168 --Review detail table to confirm action genre restored
169 SELECT *
170 FROM dvd2u_detail;
171
172
173
174

```

Below the code, there are four tabs: "Data Output", "Explain", "Messages", and "Notifications". The "Data Output" tab is selected and shows a table with the following data:

	rental_mo	rentalYr	film_genre	mo_revenue
17	7	2005	Action	1804.36
18	7	2005	Animation	1954.11
19	7	2005	Children	1539.94

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 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/>