D191 - Advanced Management Performance Assessment

HometownDVD would like to have a breakdown of rental volumes throughout the week, by day, at the various locations in order to make key decisions for a variety of issues. The primary key issues they would like to address with this information is minimal staffing requirements for each location on a given weekday as well as analyzing the viability of new marketing campaigns or special deals that could be implemented on weekdays with smaller rental volumes in an effort to increase sales on those particular days. They would also like to track the effectiveness of any implemented sales specials to determine their effectiveness and viability.

Part A:

A1. The data used for the reports will primarily be based on the transaction details of how many rentals occur on any given weekday at the various locations. The dates of rentals and the store where they were initiated at are particularly important for this report. The primary concern for the generation of these reports is the rental_date field of the rental table. It is a timestamp that describes the date and time of when a rental has occurred. These reports are designed to coalesce data for each of the stores within HometownDVD's company. In order to create this link between the rental_date and the particular store that it occurred at, the customer table is utilized as a link because it has a primary/foreign key link with both the store table and the rental table via store_id and customer_id respectively.

A2. In order to generate the report, information from the Rental, Store, and Customer tables are needed. There are no direct links between the Rental and Store tables. However, the Customer and Store tables share a common field: store_id. Furthermore, the Customer and Rental tables share a common field: customer_id. Merging these three tables using these two fields would create a table that links each rental to the particular store it occurred at, and the rental table includes the second key piece of data for our analysis: rental_date.

A3. The Detailed section will include the customer_id field from the Customer table, the rental_date from the Rental table, and the store_id field from the Store table. The Summary section will contain a field for store location as well as fields for the weekdays Monday - Sunday.

A4. In the store table, store_id is only stored as a basic integer. However, HometownDVD has a North location and a South location. To better summarize the data, the store_id tags for each store will be transformed into its proper "North" or "South" location to provide clear communication to anyone which particular location a rental was initiated at.

A5. As stated before, this report provides a couple of key uses for HometownDVD. On one hand, a breakdown of rental volumes on any given day of the week provides insight into which days require more staffing considerations vs days where staffing should be less due to sales. On the other hand, it also provides an opportunity to create incentives like promotions or specials on "slower" days of the week with the aim to increase sales on those particular weekdays. The detailed table generated has the sole purpose of gathering the data to be better formalized in the summary table. The summary table is the primary area of interest in this report. The summary table provides information on the total volume of rentals (sales) that occur on a given day of the week. This information will allow management to improve the business in two specific directions. Firstly, it provides a basis on how they should schedule employees in the future. For example, if Tuesday has a recurring number of lower rentals (sales), then it does not require the same amount of staff currently and the schedule created could reflect this. Secondly, the goal is to understand how current business trends are in order to determine the possibility of incorporating promotional or sale incentives to customers to drive more rentals on days that show lower numbers of rentals. For example, a buy one rental get a second one half of on Tuesdays if Tuesdays rental volumes are consecutively low. The report would also provide data in the future as to the effectiveness of such promotions…..if Tuesdays rental volume goes up then the promotion was successful, if not it wasn't successful and a new strategy to increase rentals on slower days should be implemented.

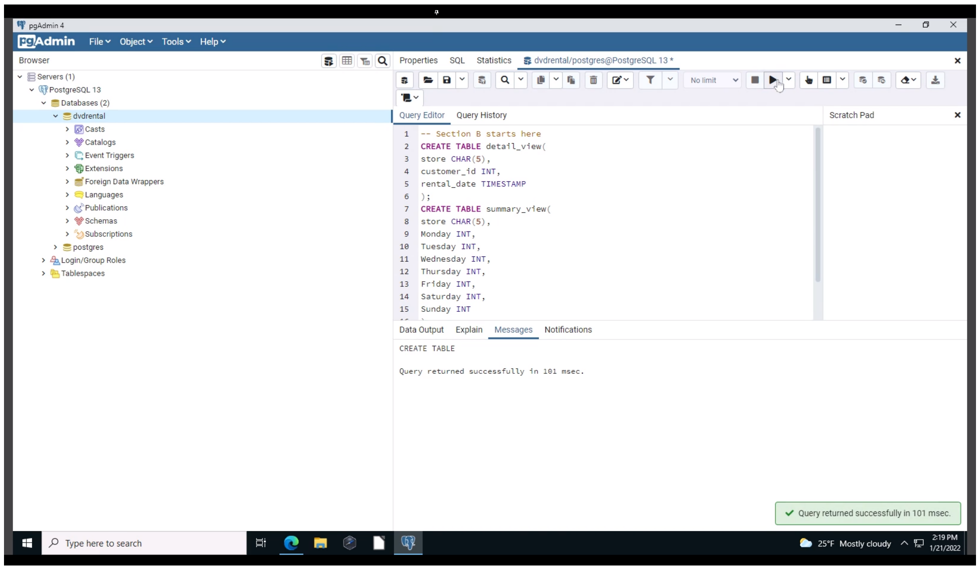
A6. This report should be run daily for two reasons. Firstly, it gives an initial impression how effective a promotion or special sale aspect that might have been implemented for that weekday was. Secondly it provides information on whether staffing considerations in the upcoming week need to be reevaluated…..a particularly popular special or promotion that increased rental volumes for a this weeks day in question would provide insight into staffing the next weeks similar weekday to avoid customer frustration due to long lines, wait times, or anything else association with an establishment being understaffed.

Part B:

```
CREATE TABLE detail_view(
store CHAR(5),
customer_id INT,
rental_date TIMESTAMP
);
CREATE TABLE summary_view(
store CHAR(5),
Monday INT,
Tuesday INT,
Wednesday INT,
Thursday INT,
Friday INT,
Saturday INT,
Sunday INT
);
```

The detail view will contain all the individual transactions of rentals by joining the store, customer, and rental tables. There is no direct link between the store and rental table so customer is used because it contains a relationship with both tables.

The summary view creates a breakdown on how many rentals occurred for each day of the previous week.



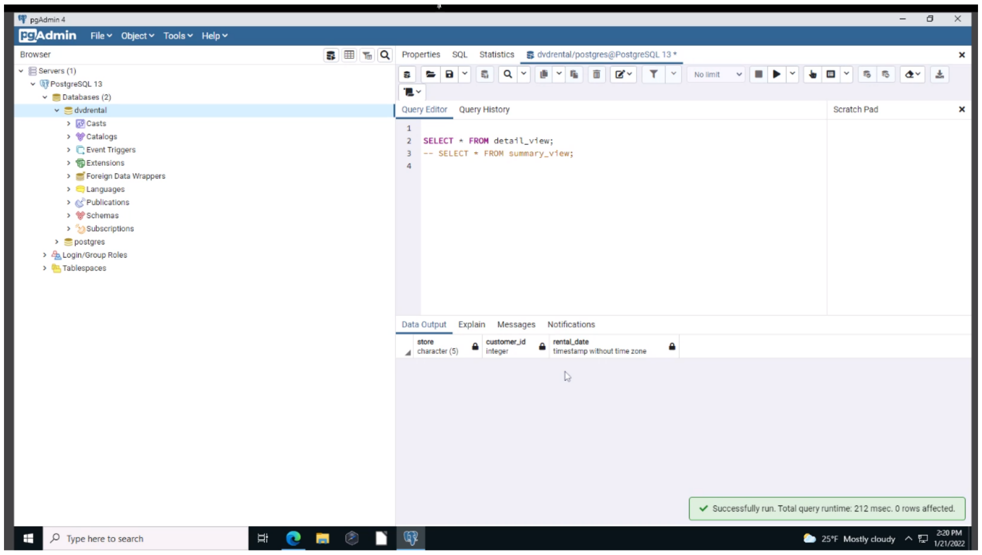
```
-- Section B starts here
CREATE TABLE detail_view(
store CHAR(5),
customer_id INT,
rental_date TIMESTAMP
);
CREATE TABLE summary_view(
store CHAR(5),
Monday INT,
Tuesday INT,
Wednesday INT,
Thursday INT,
Friday INT,
Saturday INT,
Sunday INT
```

```
CREATE TABLE

Query returned successfully in 101 msec.
```

Query returned successfully in 101 msec.

Screen capture showing the detail table was created:

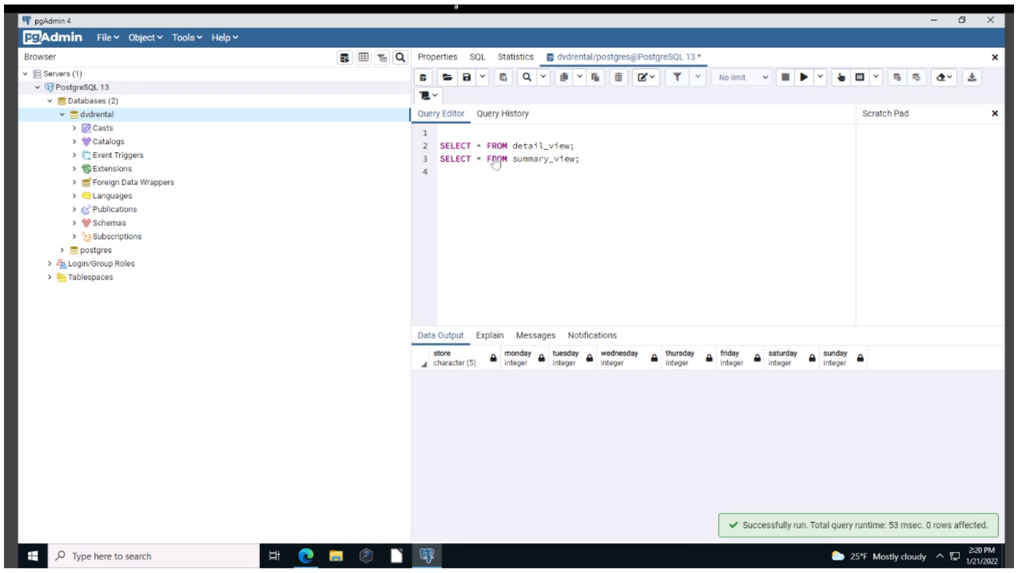


```
SELECT * FROM detail_view;
-- SELECT * FROM summary_view;
```

```
store           customer_id   rental_date
character (5)   integer       timestamp without time zone
```

Successfully run. Total query runtime: 212 msec. 0 rows affected.

Screen capture showing the summary table was created:


```
SELECT * FROM detail_view;
SELECT * FROM summary_view;
```

```
store          monday   tuesday  wednesday  thursday  friday   saturday  sunday
character (5)  integer  integer  integer    integer   integer  integer   integer
```

Successfully run. Total query runtime: 53 msec. 0 rows affected.

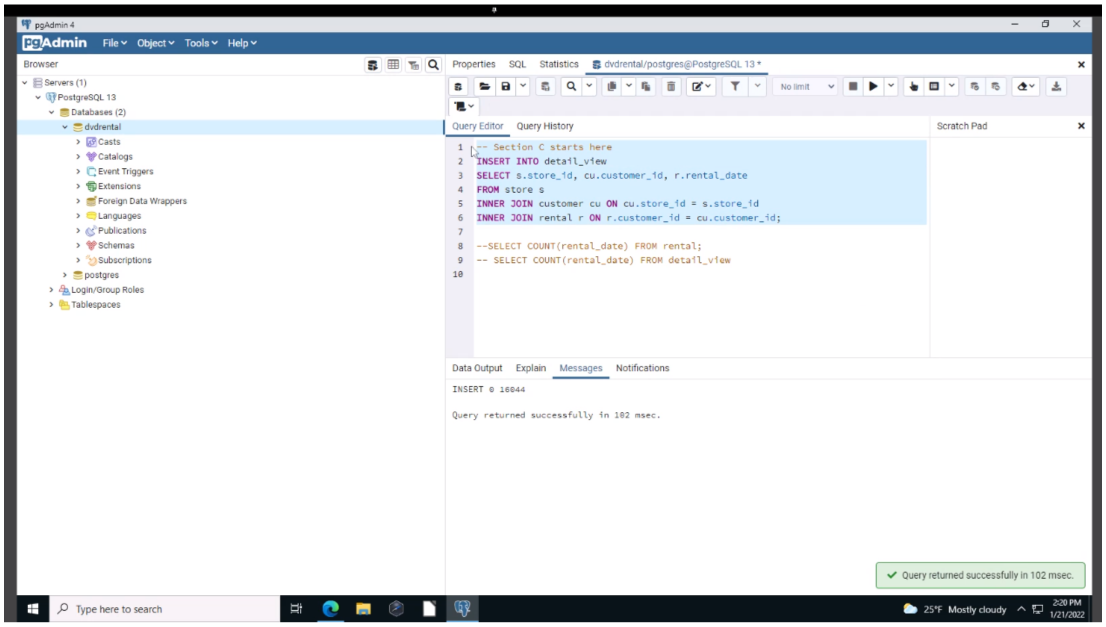
Part C:

```
INSERT INTO detail_view
SELECT s.store_id, cu.customer_id, r.rental_date
FROM store s
INNER JOIN customer cu ON cu.store_id = s.store_id
INNER JOIN rental r ON r.customer_id = cu.customer_id;


--SELECT COUNT(rental_date) FROM rental;
-- SELECT COUNT(rental_date) FROM detail_view
```

Part C loads all of the rental records into the detail table by joining the store, customer, and rental tables using the store_id, customer_id, and rental_date fields.

Screen capture showing the code was successfully implemented:



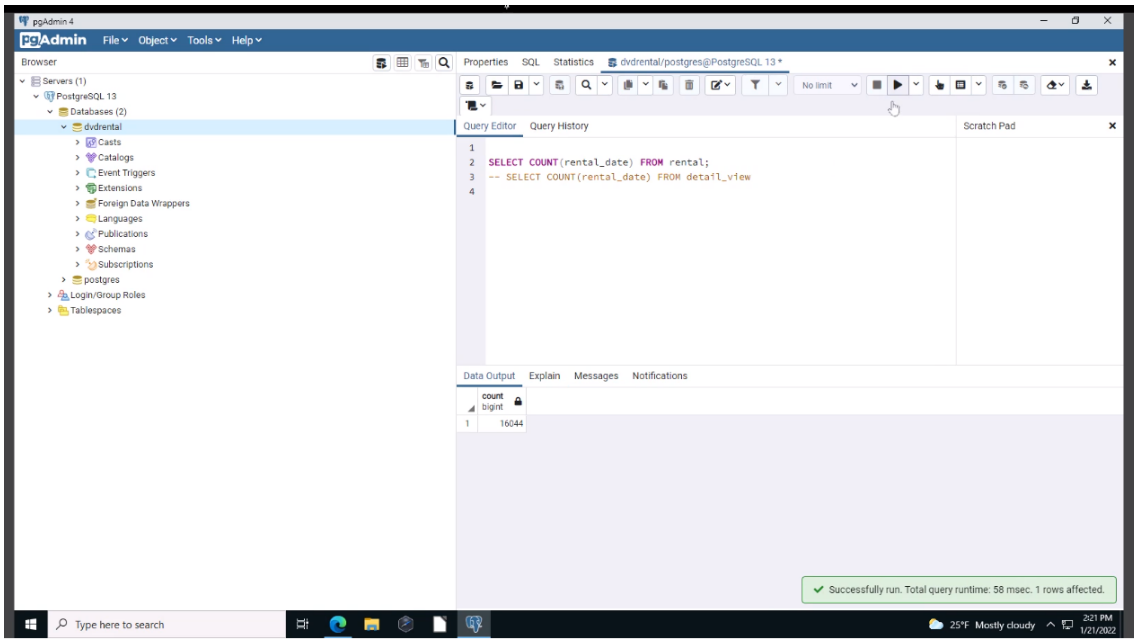
```
-- Section C starts here
INSERT INTO detail_view
SELECT s.store_id, cu.customer_id, r.rental_date
FROM store s
INNER JOIN customer cu ON cu.store_id = s.store_id
INNER JOIN rental r ON r.customer_id = cu.customer_id;

--SELECT COUNT(rental_date) FROM rental;
-- SELECT COUNT(rental_date) FROM detail_view
```

```
INSERT 0 16044

Query returned successfully in 102 msec.
```

To verify that all the appropriate records were copied to the detail view the count of rental_date records:



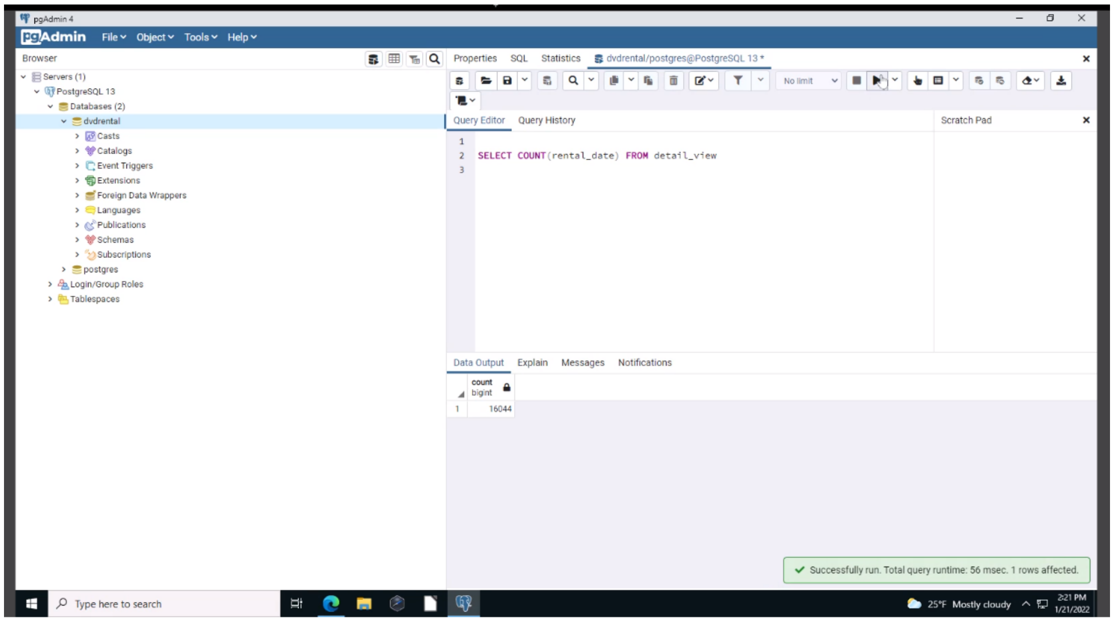
```

SELECT COUNT(rental_date) FROM rental;
-- SELECT COUNT(rental_date) FROM detail_view
```

```
  count
  bigint
1  16044
```

Successfully run. Total query runtime: 58 msec. 1 rows affected.

Is compared to the count of records contained in the detailed table:



```
SELECT COUNT(rental_date) FROM detail_view
```

```
count
bigint
1    16044
```

Successfully run. Total query runtime: 56 msec. 1 rows affected.

Both counts came out to 16044, so all the data was properly carried over to the detail table.

Part D:

```
CREATE FUNCTION summary_function() RETURNS TRIGGER AS $$
BEGIN
DELETE FROM summary_view;
INSERT INTO summary_view
VALUES('1',
        (SELECT COUNT(store) FROM detail_view d
        WHERE d.store = '1' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 1
        AND DATE_PART('day', CURRENT_TIMESTAMP - d.rental_date) <7),
       (SELECT COUNT(rental_date) FROM detail_view d
        WHERE d.store = '1' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 2 ),
        (SELECT COUNT(rental_date) FROM detail_view d
        WHERE d.store = '1' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 3 ),
        (SELECT COUNT(rental_date) FROM detail_view d
        WHERE d.store = '1' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 4 ),
        (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '1' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 5 ),
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '1' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 6 ),
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '1' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 0 ));
INSERT INTO summary_view
VALUES('2',
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '2' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 1
            AND DATE_PART('day', CURRENT_TIMESTAMP - d.rental_date) <7),
           (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '2' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 2 ),
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '2' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 3 ),
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '2' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 4 ),
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '2' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 5 ),
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '2' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 6 ),
            (SELECT COUNT(rental_date) FROM detail_view d
            WHERE d.store = '2' AND MOD(CAST(DATE_PART('day',
d.rental_date-'2000-01-03') AS INT),7) = 0 ));

UPDATE summary_view
SET store = 'North'
WHERE store = '1';
UPDATE summary_view
```

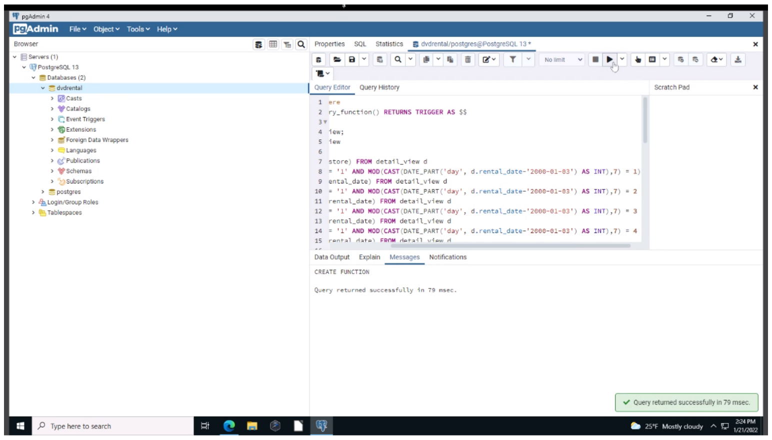
```
SET store = 'South'
WHERE store = '2';
RETURN NEW;
END; $$
LANGUAGE PLPGSQL;
```

This part creates a function (Malik, U., Goldwasser, M., & Johnston, B., 2019, p. 253) that returns a trigger that will be utilized later each time a record is inserted into detail view. The function counts the number of records that occur on a given day of the week for each store and inserts it into the appropriate column in the summary table. It does this by calculating the remainder of the count divided by the seven days of the week using the Modulo function. (PostgreSQL Tutorial, 2017) To calculate the count the DATE_PART (PostgreSQL Tutorial, 2017) function is used to find the total number of days between the rental_date and a known date, in this case January 3, 2000 which is a Monday.

DATE_PART('day', CURRENT_TIMESTAMP - d.rental_date) <7 is used to find the records for only past seven days by finding the number of days between the rental date and and the current date. If the difference is less than seven then the record is included in the count. This was only included in one of the parameters due to the fact that the database has no recent data. In order to show that the function works properly it needs to be removed for the meantime for testing purposes, which gives a summary of the total number of rentals that occur on each day of the week. However final implementation of the code would have it included in each part of the function.

Once the data has been uploaded into the table the table is then updated to transform the store field. There are only two stores currently owned by HometownDVD, a North store and a South store. To provide a more descriptive entry in the summary table the store numbers are transformed into the North/South store designation within the summary table.

Screen capture showing the function was successfully created in the database:



```
ere
ry_function() RETURNS TRIGGER AS $$

iew;
iew

store) FROM detail_view d
= '1' AND MOD(CAST(DATE_PART('day', d.rental_date-'2000-01-03') AS INT),7) = 1)
ental_date) FROM detail_view d
= '1' AND MOD(CAST(DATE_PART('day', d.rental_date-'2000-01-03') AS INT),7) = 2
rental_date) FROM detail_view d
= '1' AND MOD(CAST(DATE_PART('day', d.rental_date-'2000-01-03') AS INT),7) = 3
rental_date) FROM detail_view d
= '1' AND MOD(CAST(DATE_PART('day', d.rental_date-'2000-01-03') AS INT),7) = 4
rental_date) FROM detail_view d
```

```
CREATE FUNCTION

Query returned successfully in 79 msec.
```

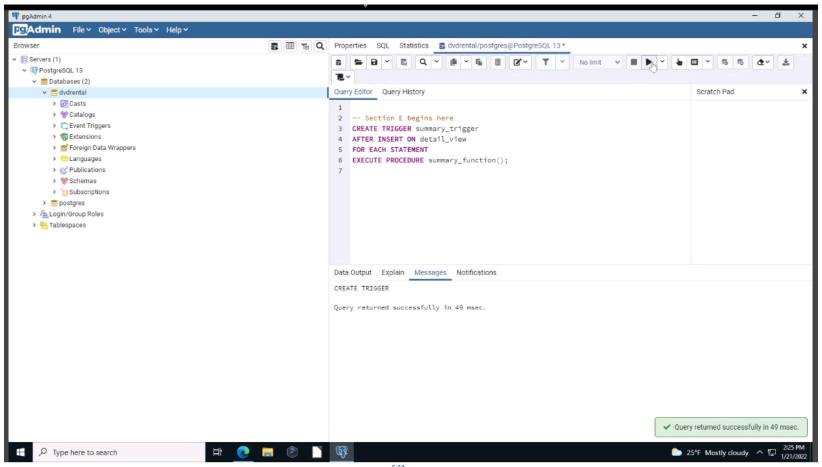
Query returned successfully in 79 msec.

Part E:

CREATE TRIGGER summary_trigger
AFTER INSERT ON detail_view
FOR EACH STATEMENT
EXECUTE PROCEDURE summary_function();

A trigger (Malik, U., Goldwasser, M., & Johnston, B., 2019, p. 262) called summary_trigger was created to initiate the summary_function() that inserts data into the summary table from the detailed table after each insertion of data into the detailed table.

Screen capture showing the creation of the trigger was successful:



```

-- Section E begins here
CREATE TRIGGER summary_trigger
AFTER INSERT ON detail_view
FOR EACH STATEMENT
EXECUTE PROCEDURE summary_function();

```

```
CREATE TRIGGER

Query returned successfully in 49 msec.
```

Query returned successfully in 49 msec.

Part F:

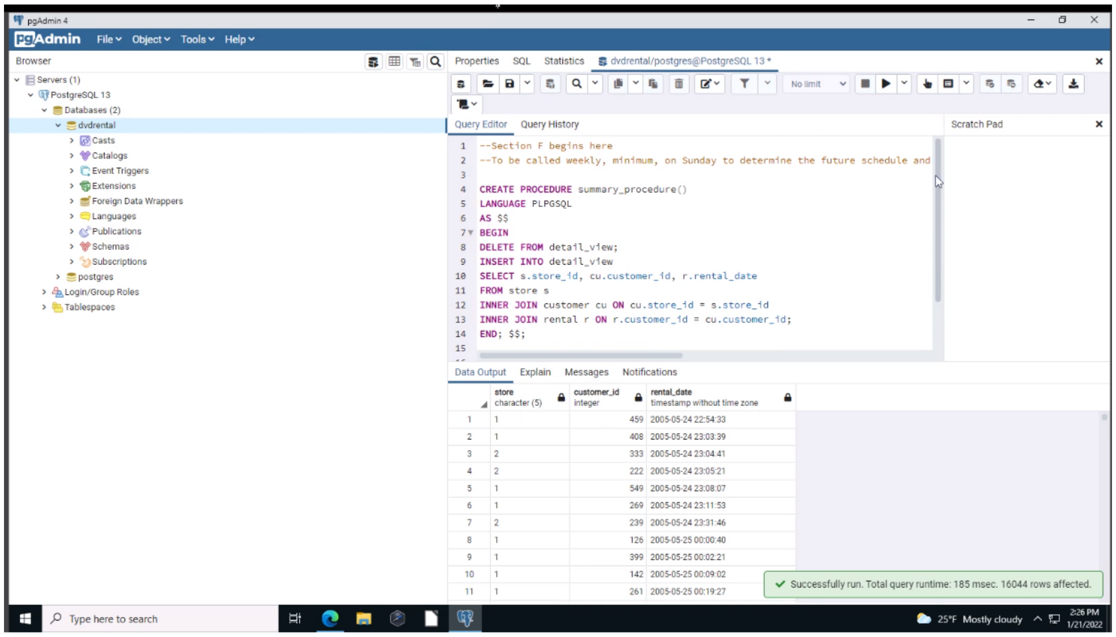
```
--To be called weekly, minimum, on Sunday to determine the future schedule and
effectiveness of the promotional deals put in place for that week using Agent pgAgent
as a job scheduler.

CREATE PROCEDURE summary_procedure()
LANGUAGE PLPGSQL
AS $$
BEGIN
DELETE FROM detail_view;
INSERT INTO detail_view
SELECT s.store_id, cu.customer_id, r.rental_date
FROM store s
INNER JOIN customer cu ON cu.store_id = s.store_id
INNER JOIN rental r ON r.customer_id = cu.customer_id;
END; $$;

CALL summary_procedure();
SELECT * FROM detail_view;
--SELECT * FROM summary_view;
```

A procedure (Bennett, 2020) was created so that this report can be generated on a weekly basis using pgAgent as a job scheduler. The summary_procedure() when called deletes all of the records that are in the detailed table so no duplicates are created. It then inserts the information from the store, customer, and rental tables using the code created in part C. This insert then causes the previously created trigger to use the summary_function() and upload the count of appropriate records into the summary tables fields.

Screen capture to show that the procedure was successfully created and shows that the detail table was uploaded with the appropriate data:



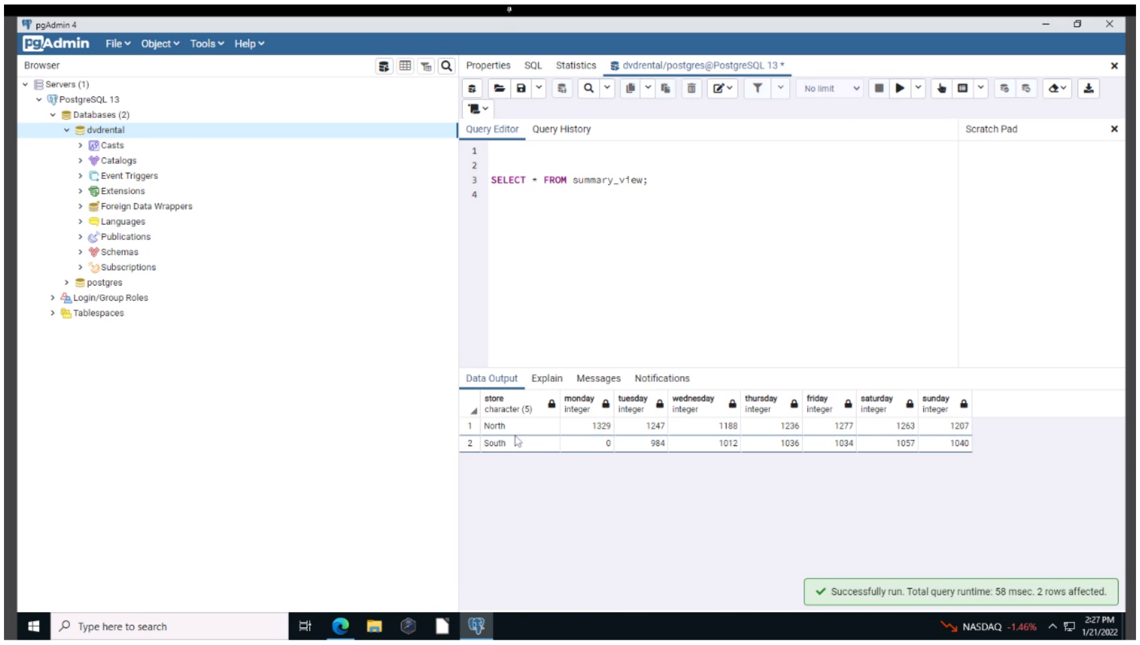
```
--Section F begins here
--To be called weekly, minimum, on Sunday to determine the future schedule and

CREATE PROCEDURE summary_procedure()
LANGUAGE PLPGSQL
AS $$
BEGIN
DELETE FROM detail_view;
INSERT INTO detail_view
SELECT s.store_id, cu.customer_id, r.rental_date
FROM store s
INNER JOIN customer cu ON cu.store_id = s.store_id
INNER JOIN rental r ON r.customer_id = cu.customer_id;
END; $$;
```

	store character (5)	customer_id integer	rental_date timestamp without time zone
1	1	459	2005-05-24 22:54:33
2	1	408	2005-05-24 23:03:39
3	2	333	2005-05-24 23:04:41
4	2	222	2005-05-24 23:05:21
5	1	549	2005-05-24 23:08:07
6	1	269	2005-05-24 23:11:53
7	2	239	2005-05-24 23:31:46
8	1	126	2005-05-25 00:00:40
9	1	399	2005-05-25 00:02:21
10	1	142	2005-05-25 00:09:02
11	1	261	2005-05-25 00:19:27

```
Successfully run. Total query runtime: 185 msec. 16044 rows affected.
```

Screen capture to show that the procedure was called correctly and that the data was successfully transformed into the summary table:



```
SELECT * FROM summary_view;
```

	store character (5)	monday integer	tuesday integer	wednesday integer	thursday integer	friday integer	saturday integer	sunday integer
1	North	1329	1247	1188	1236	1277	1263	1207
2	South	0	984	1012	1036	1034	1057	1040

```
Successfully run. Total query runtime: 58 msec. 2 rows affected.
```

Monday for the South store is 0 because it has the portion of code that gathers only records from the past 7 days to be counted for the summary view. It is zero because there are no records for the past 7 days so this portion of code wasn't implemented in the overall function code in order to be able to show that function works correctly based on the data that is currently contained in the database. At final implementation this bit of code will be introduced into the appropriate portions of the function to create the summary view of the number of rental transactions that occur for each day of the week for the past week. This currently is a summary of the total of transaction for each day of the week for all data in the database.

Web Sources:
https://postgresqltutorial.com/ was utilized utilized as a reference in this project's code development Specifically, the website's portions covering the topics of modulo and date_part functions.
https://www.pluralsight.com/guides/using-stored-procedures-to-create-custom-workflows-in-postgresql was used as a reference when creating the stored procedure portion of code.

Sources

Malik, U., Goldwasser, M., & Johnston, B. (2019). *SQL for data analytics: Perform fast and efficient data analysis with the power of SQL. (new tab)* Packt Publishing. ISBN: 978-1-78980-735-6

PostgreSQL Tutorial. (2017, August 17). *PostgreSQL mod function - postgresql tutorial*. PostgreSQL Tutorial. Retrieved January 20, 2022, from https://www.postgresqltutorial.com/postgresql-mod/

PostgreSQL Tutorial. (2017, August 17). *PostgreSQL Date_Part function - postgresql tutorial*. PostgreSQL Tutorial. Retrieved January 20, 2022, from https://www.postgresqltutorial.com/postgresql-date_part/

Bennett, Z. (2020, August 12). *Zachary Bennett*. Pluralsight. Retrieved January 20, 2022, from https://www.pluralsight.com/guides/using-stored-procedures-to-create-custom-workflows-in-postgresql