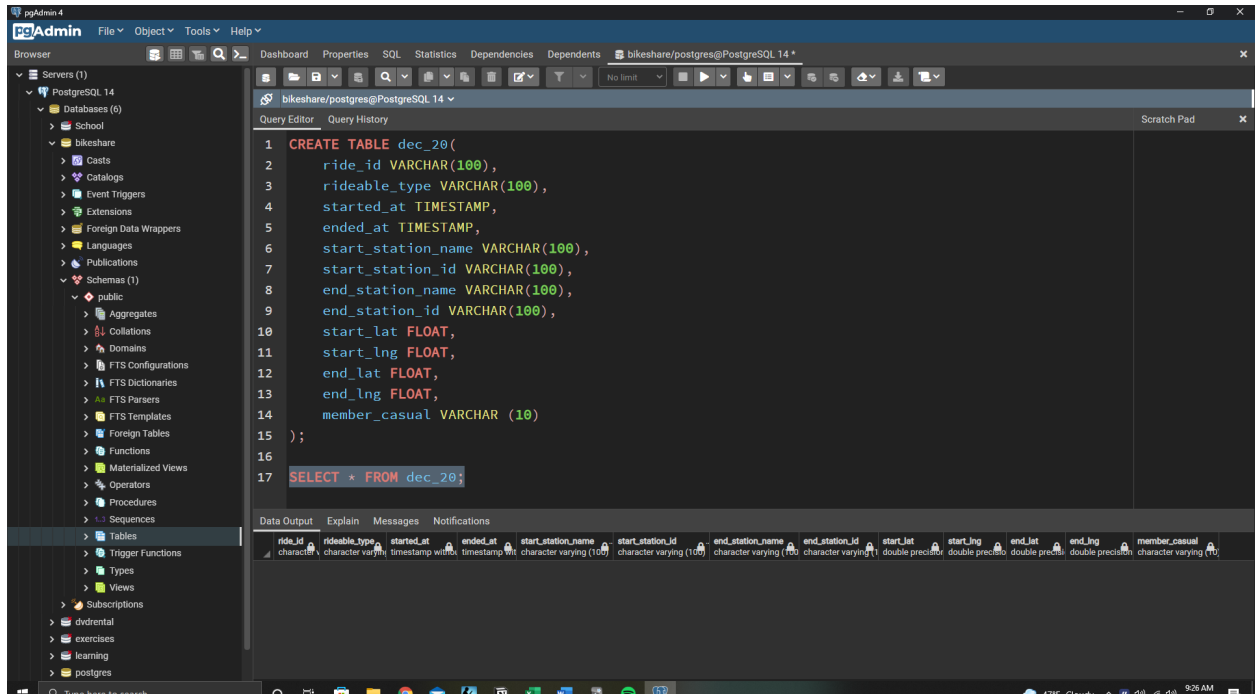
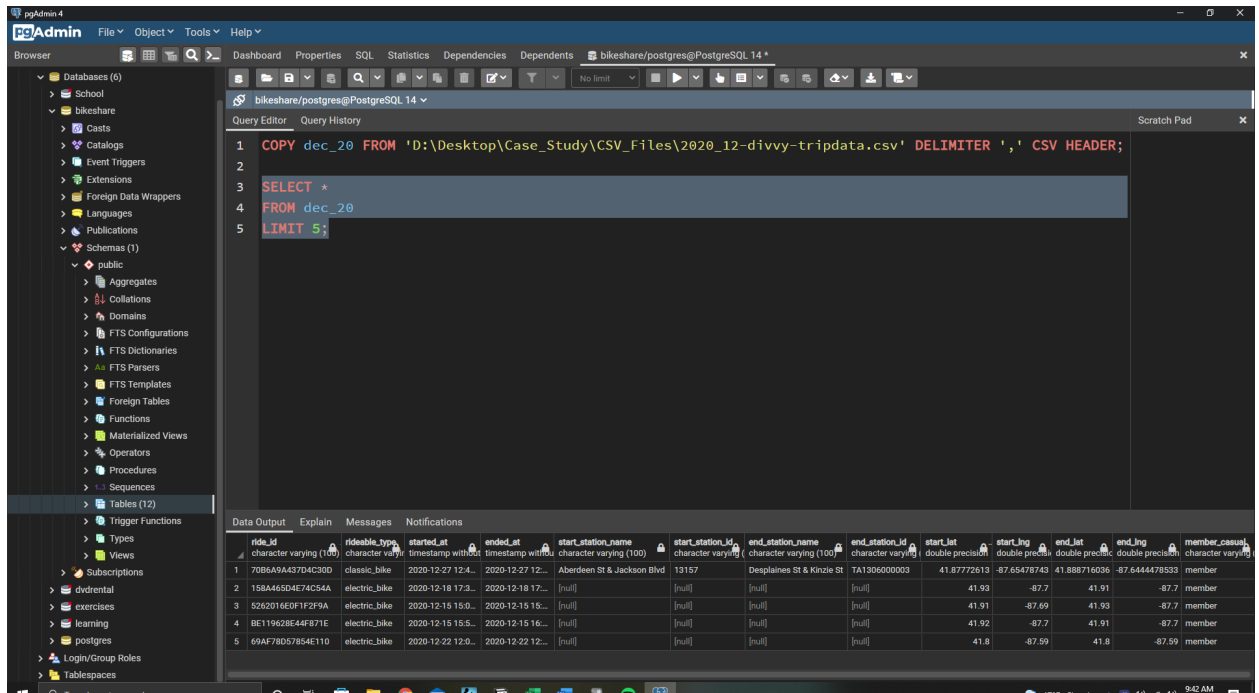


Cyclistic Bike-Share Data Cleaning Documentation

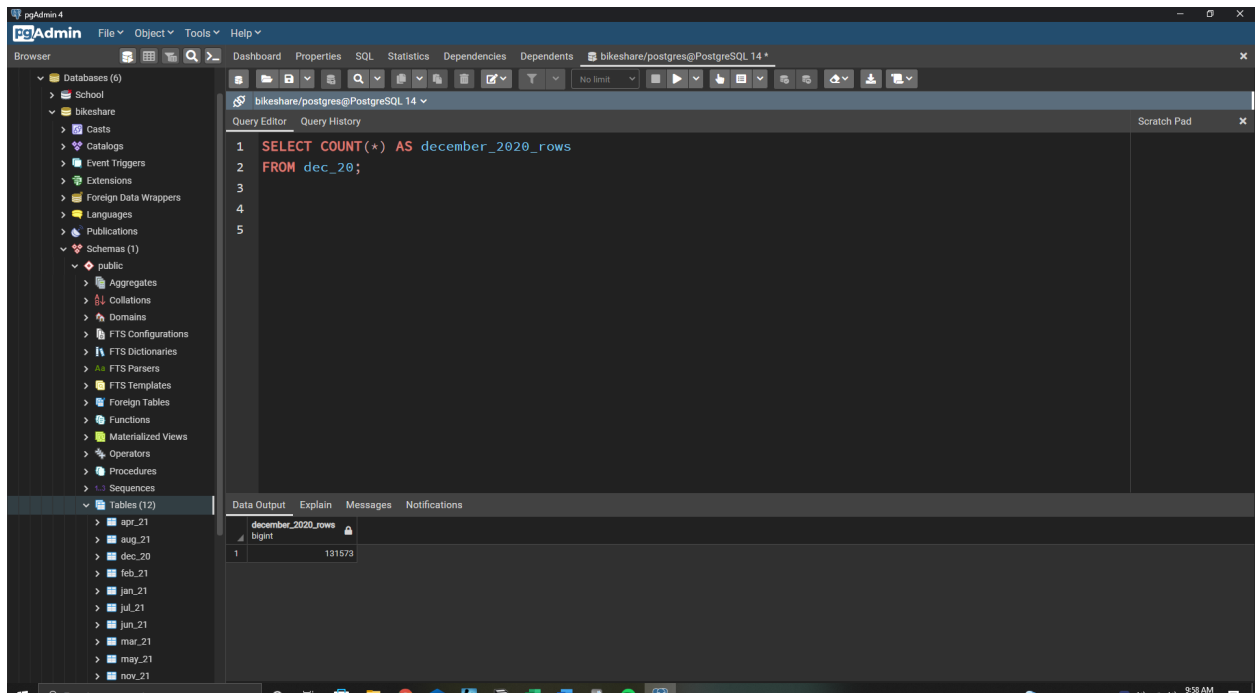
1. The 12 files, one for each month from December 2020 to November 2021, were too large to merge into one Excel file. So, I tried to merge them into one file on Google's BigQuery. However, I use the unpaid version, so I couldn't merge all the files into one for the same reason as Excel. Therefore, used PostgreSQL to merge all the files.
2. To copy a file into PostgreSQL I needed to create a table with the appropriate column name and data type. The Excel files had the same 13 columns, so I created the following table for each month from December 2020 to November 2021 by changing the table name.



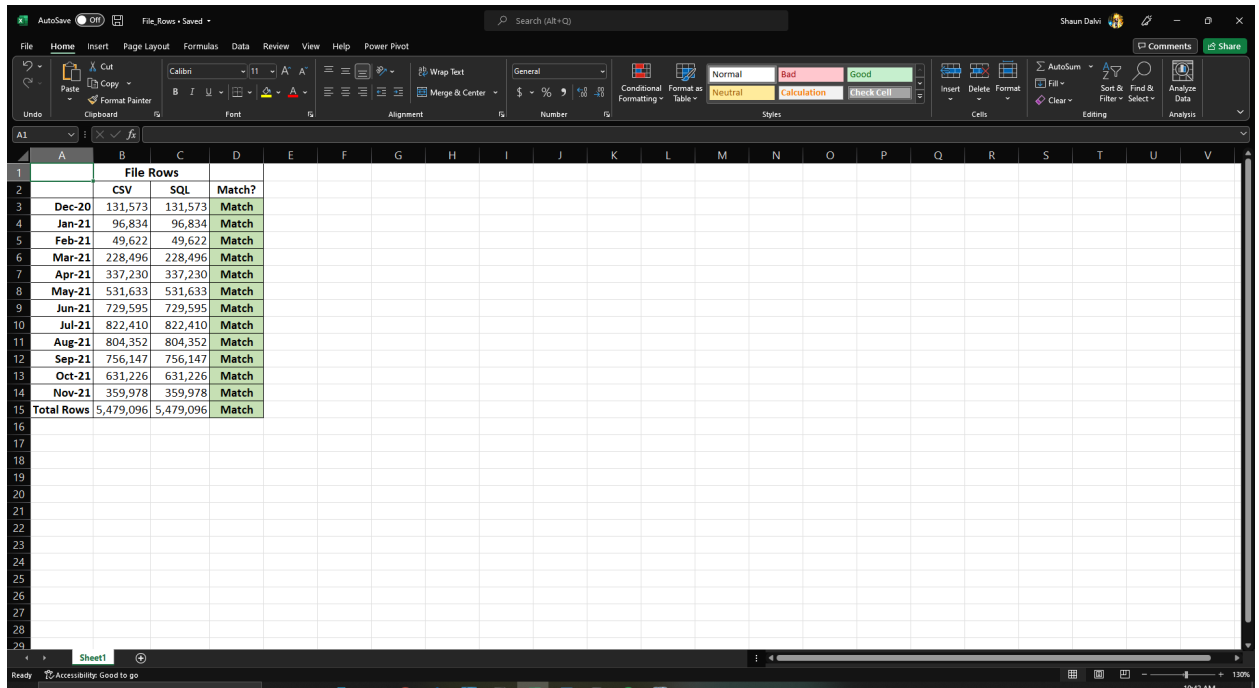
- The next thing I did was to convert all the Excel files into CSV files to import into PostgreSQL. After doing this, I was able to use the COPY FROM command to import each CSV file into its respective table. I also printed the first few rows to verify that they matched the CSV file.



- To confirm that every row was imported correctly I used the COUNT function to return the number of rows in the SQL table for all 12 months.

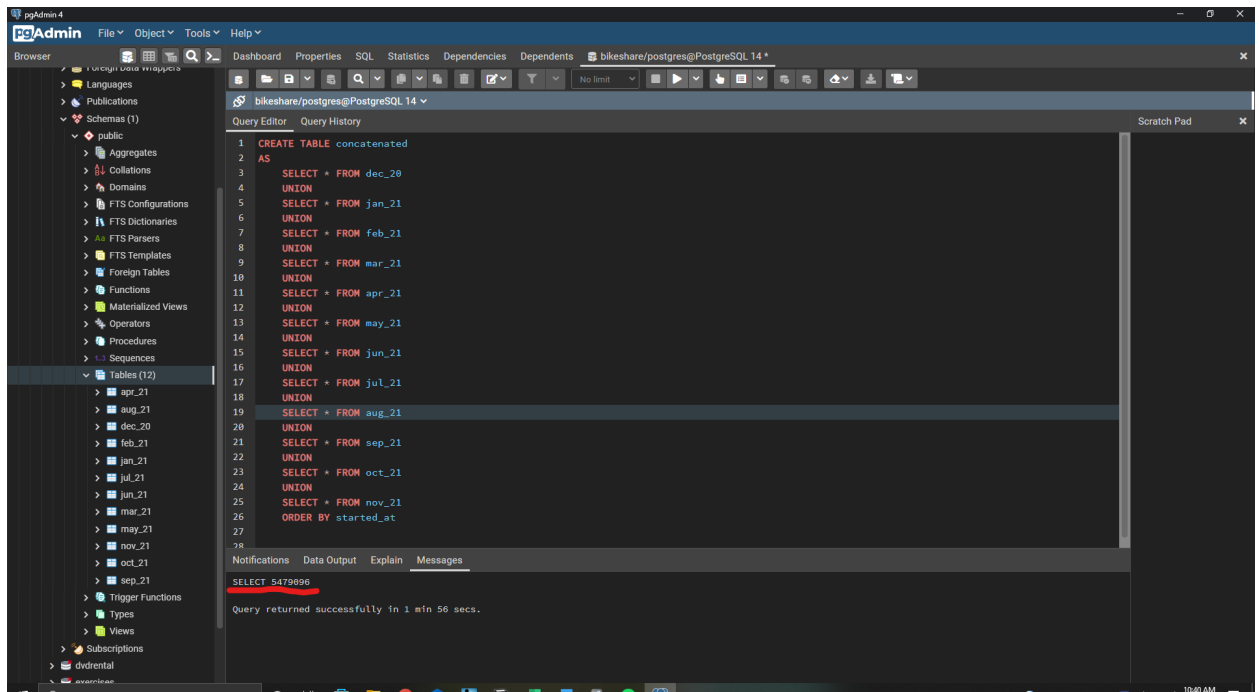


5. I created a table in an Excel file to confirm that all 12 files were imported correctly.



| | File Rows | | |
|------------|-----------|-----------|--------|
| | CSV | SQL | Match? |
| Dec-20 | 131,573 | 131,573 | Match |
| Jan-21 | 96,834 | 96,834 | Match |
| Feb-21 | 49,622 | 49,622 | Match |
| Mar-21 | 228,496 | 228,496 | Match |
| Apr-21 | 337,230 | 337,230 | Match |
| May-21 | 531,633 | 531,633 | Match |
| Jun-21 | 729,595 | 729,595 | Match |
| Jul-21 | 822,410 | 822,410 | Match |
| Aug-21 | 804,352 | 804,352 | Match |
| Sep-21 | 756,147 | 756,147 | Match |
| Oct-21 | 631,226 | 631,226 | Match |
| Nov-21 | 359,978 | 359,978 | Match |
| Total Rows | 5,479,096 | 5,479,096 | Match |

6. Once I confirmed that all the CSV files were properly imported into their respective tables I created a new table that combined all the files into one. I used UNION instead of JOIN because I wanted one file to be pasted directly under the other.



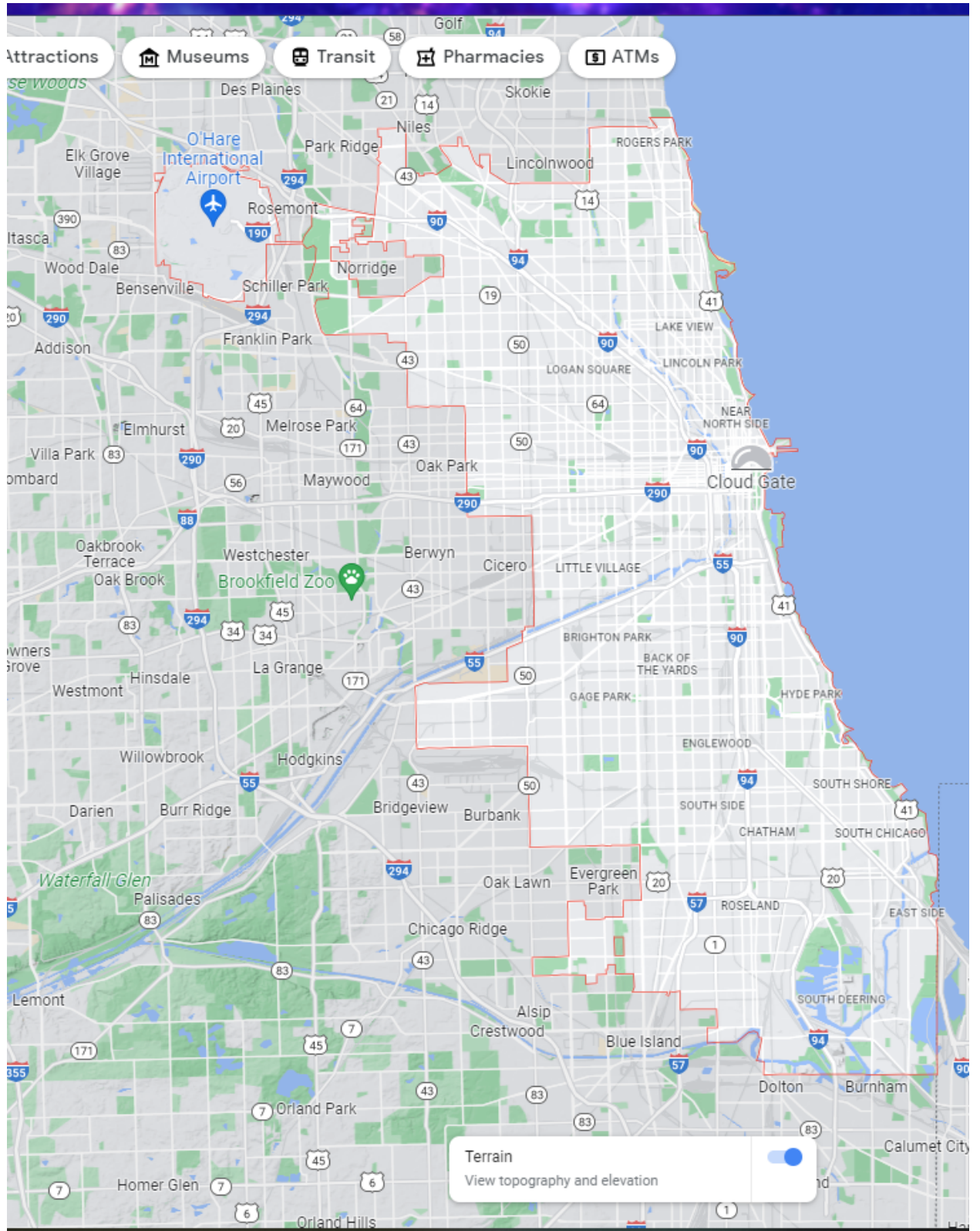
```
1 CREATE TABLE concatenated
2 AS
3 SELECT * FROM dec_20
4 UNION
5 SELECT * FROM jan_21
6 UNION
7 SELECT * FROM feb_21
8 UNION
9 SELECT * FROM mar_21
10 UNION
11 SELECT * FROM apr_21
12 UNION
13 SELECT * FROM may_21
14 UNION
15 SELECT * FROM jun_21
16 UNION
17 SELECT * FROM jul_21
18 UNION
19 SELECT * FROM aug_21
20 UNION
21 SELECT * FROM sep_21
22 UNION
23 SELECT * FROM oct_21
24 UNION
25 SELECT * FROM nov_21
26 ORDER BY started_at
27
```

SELECT 5479096

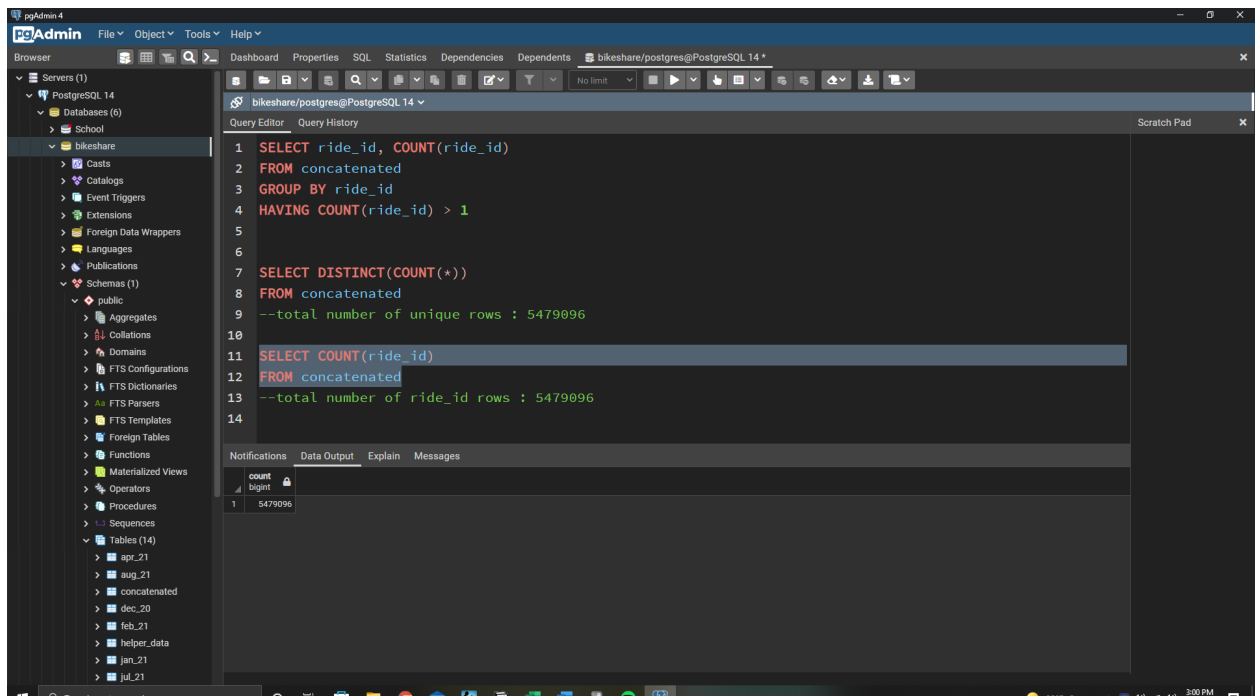
Query returned successfully in 1 min 56 secs.

The total number of rows in the merged file is underlined in the picture. This matches the total number of rows in the previous picture as well.

7. Now that all the rows were in one table I wanted to confirm that the latitude and longitude of each entry were accurate to the city of Chicago.

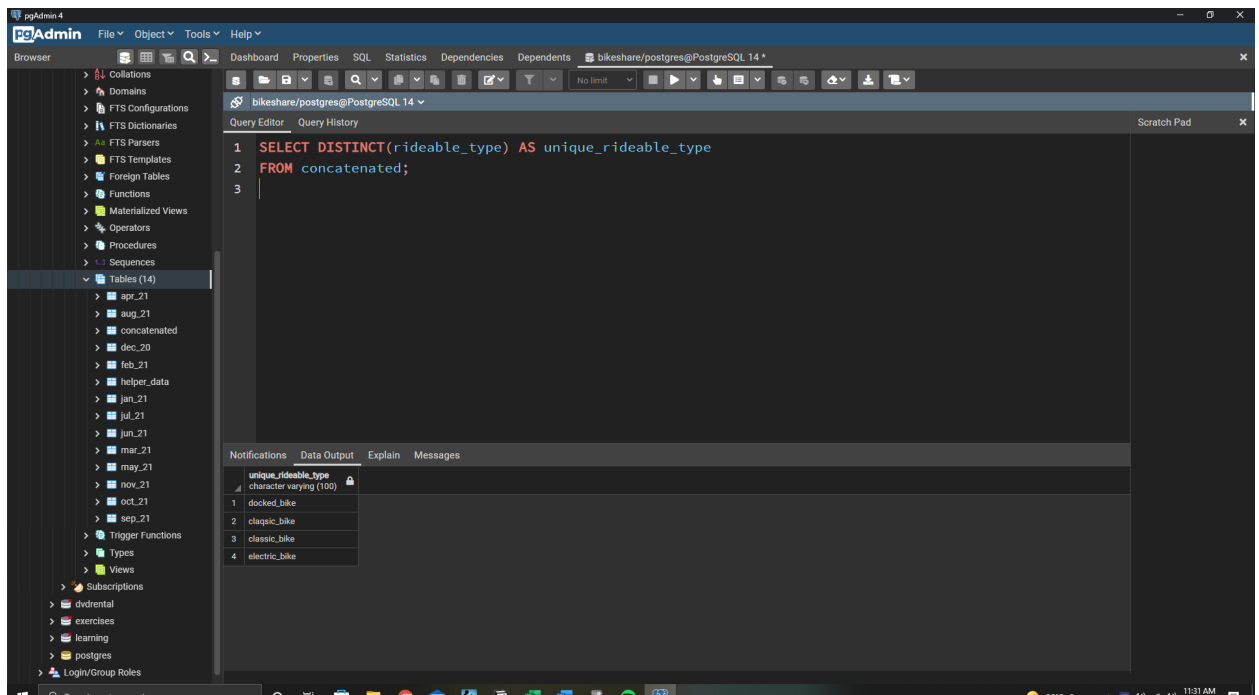


8. Then I checked to see if there were any duplicate entries. I did this by comparing the total number of unique rows to the total number of rows.

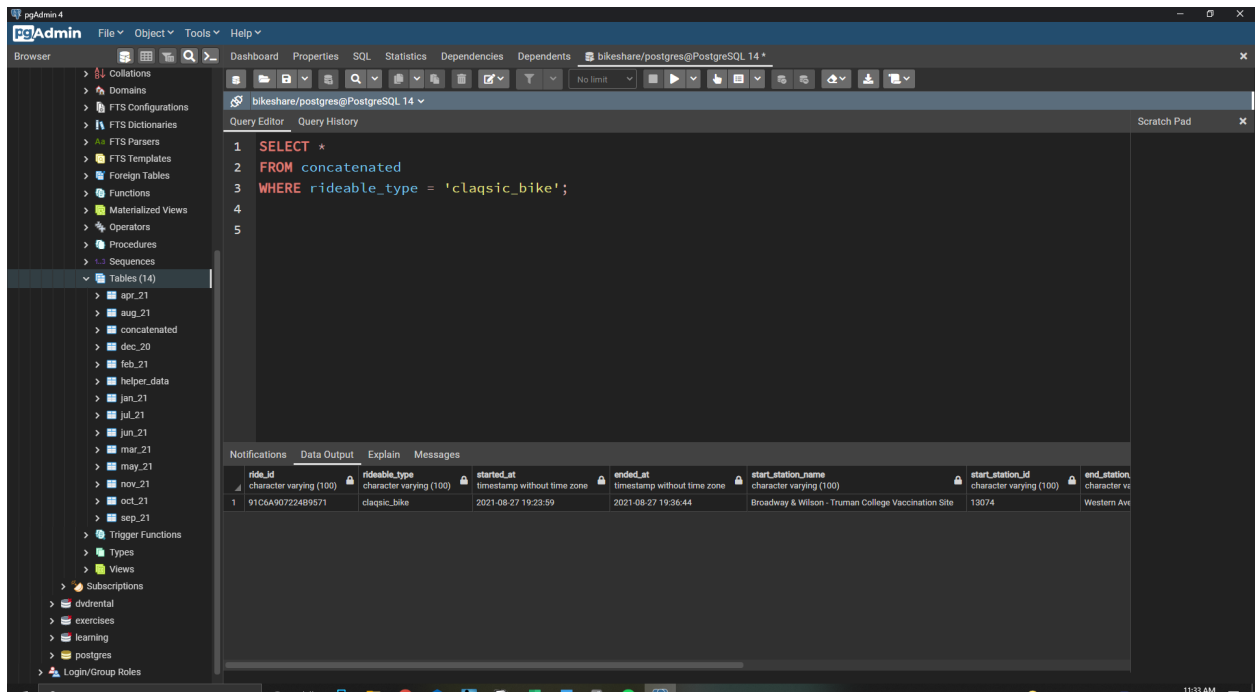


They both yielded the same number of rows so there weren't any duplicate entries.

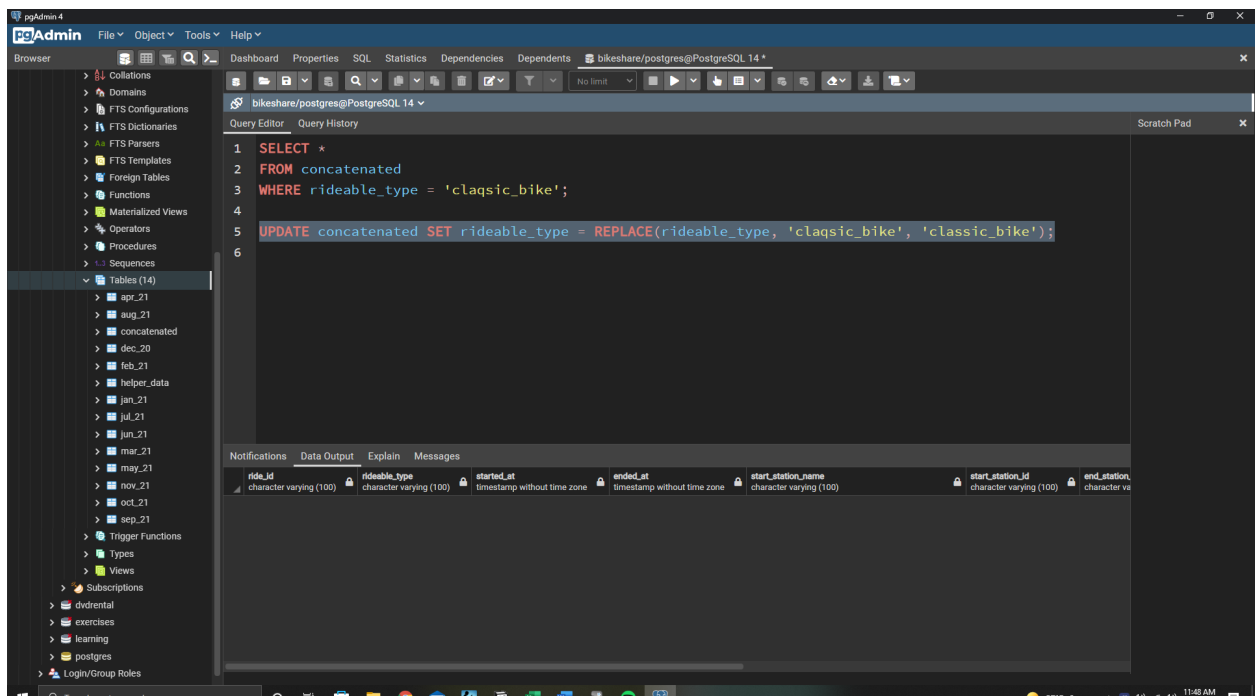
9. Based on the company information, there should be three rideable types and two member types. I printed all the unique member types and there were only two, however, when I did this with the rideable type there were four instead of three. So, I investigated further.



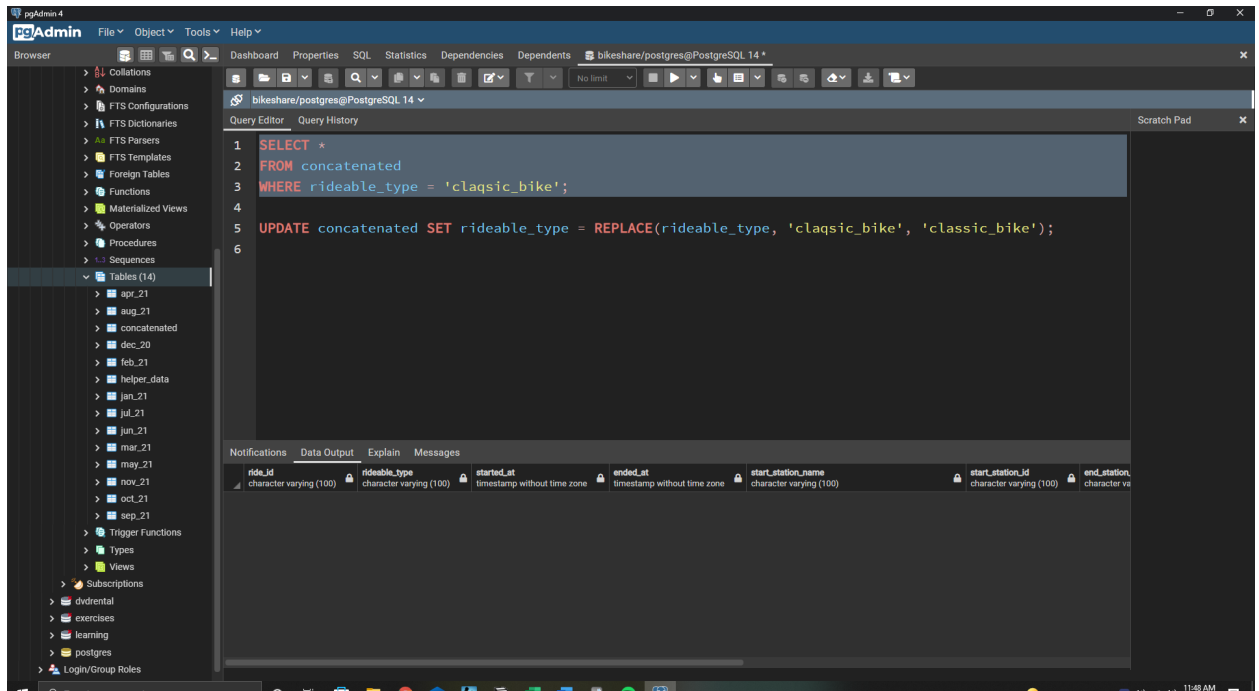
I noticed that there was a misspelling when the data was entered. I wanted to see how many rows had this misspelling.



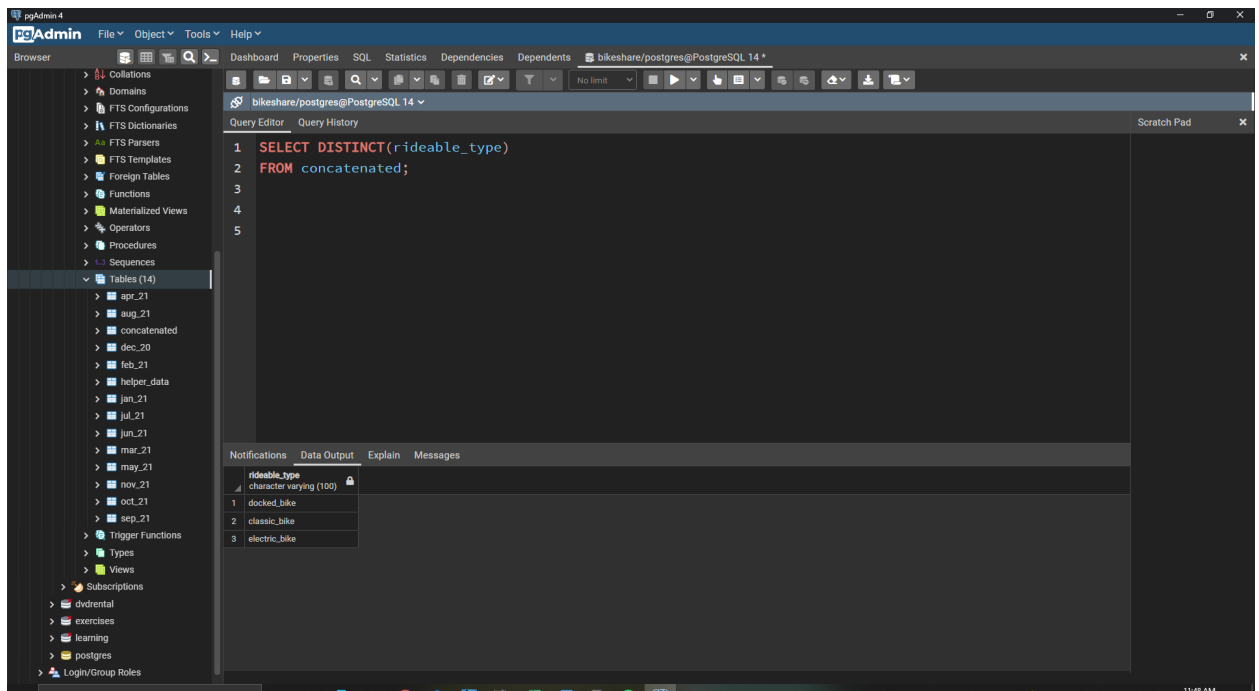
Only one row was entered incorrectly. So, I updated the rideable type row to fix this issue.



I wanted to make sure this error was fixed, so I ran the previous code to check.

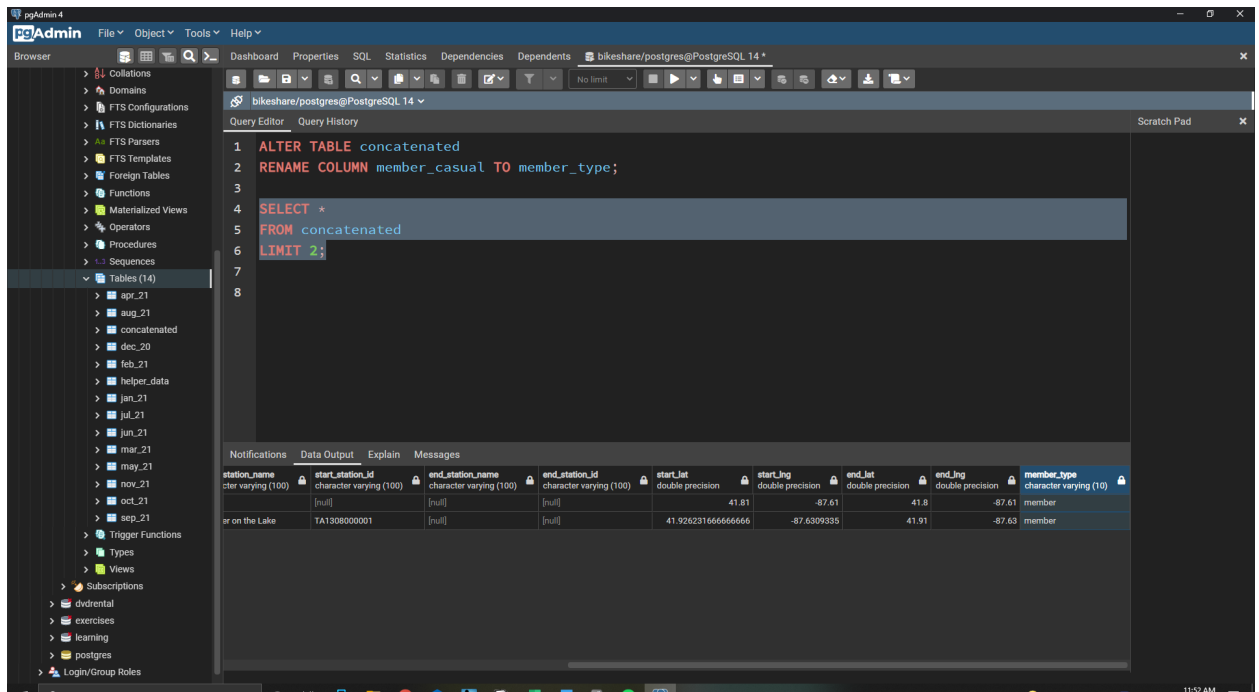


It looks like the misspelling was fixed. To make sure the rideable types had only three options I checked to see all of the unique rideable types again.

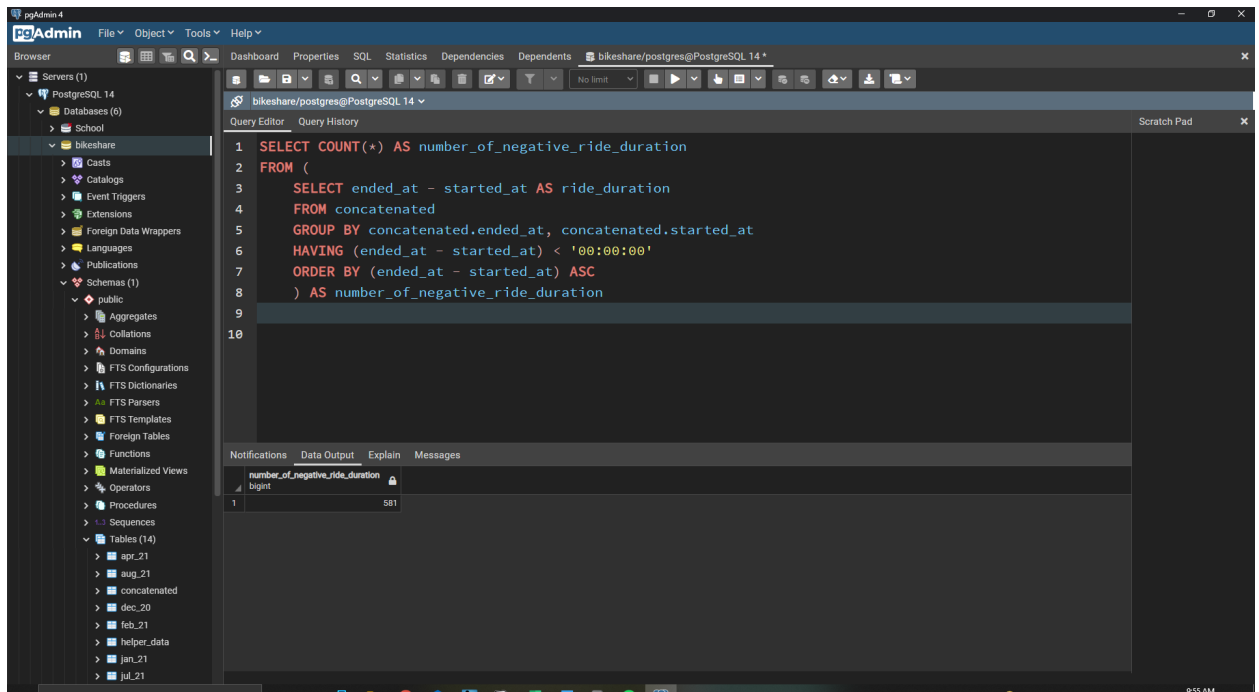


Now there are only three rideable options like there should be.

10. To make the member type row easier to understand I changed the name of the column from *member_casual* to *member_type*.

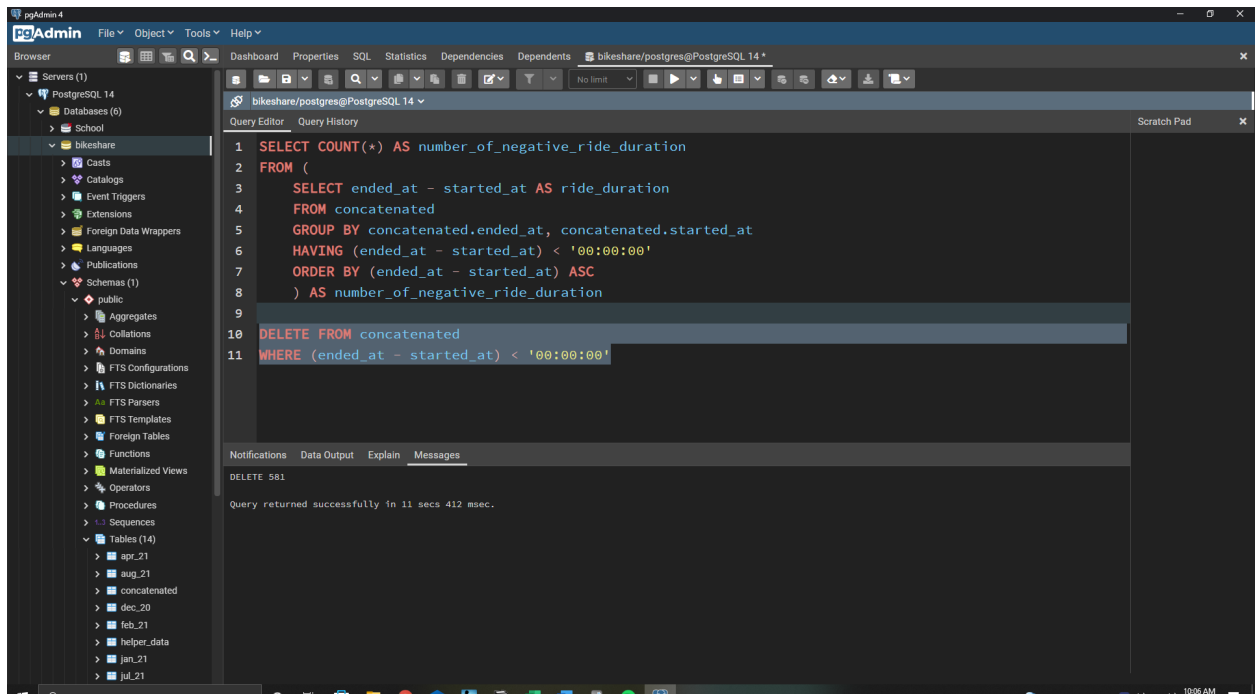


11. Another possible issue could be negative ride duration. This would be an issue when it comes to analyzing the data so, I wanted to count all of the rows that had a negative ride duration.

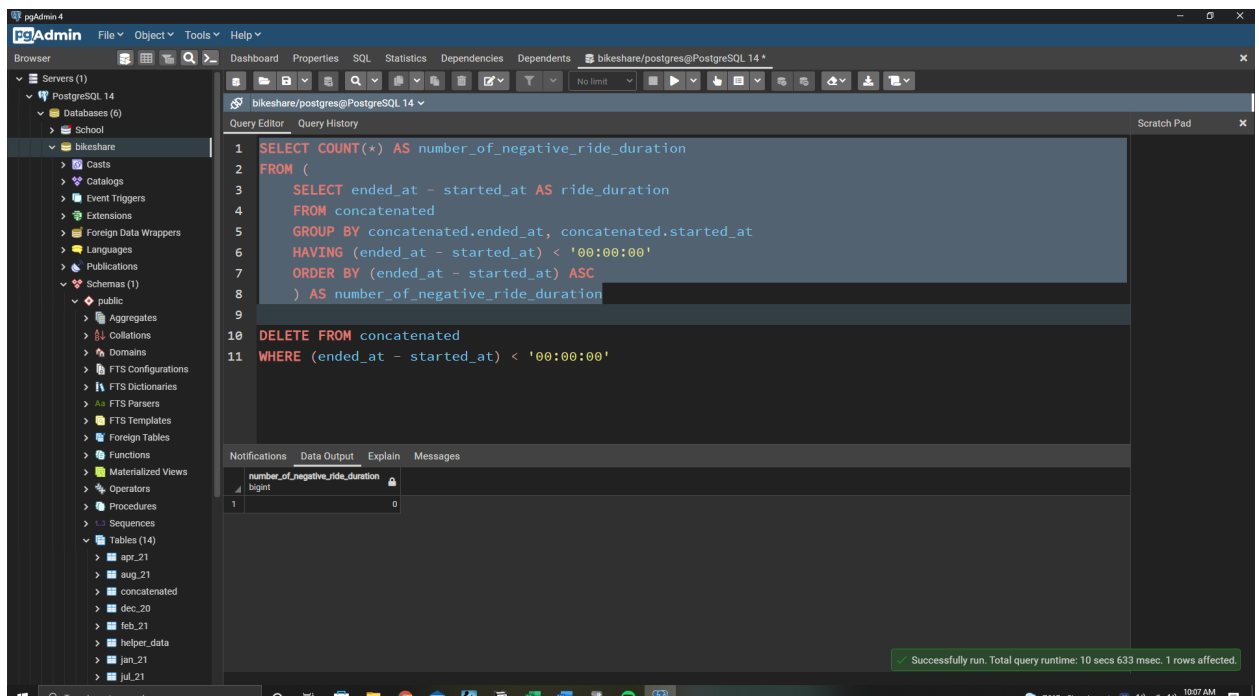


There were some rows with a negative ride duration. We could investigate deeper into why this happened but for time sensitivity I decided to delete these rows. Another reason why I decided to delete them is that this was only a small percentage of the total number of rows in our data

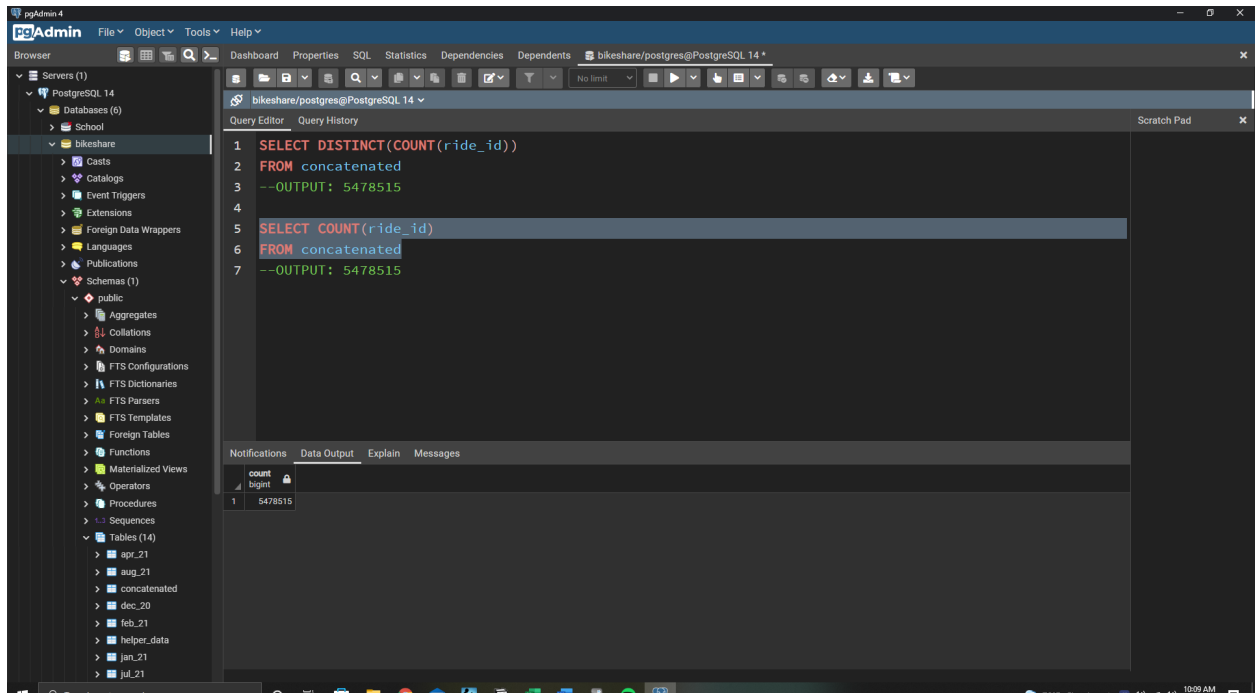
set and the remaining number of rows would be a sufficient data set sample. $[(581 \text{ negative duration} / 5,479,096 \text{ total rows}) * 100 = 0.01\%]$



I wanted to make use of all the rows that were deleted properly so I ran the previous code again.

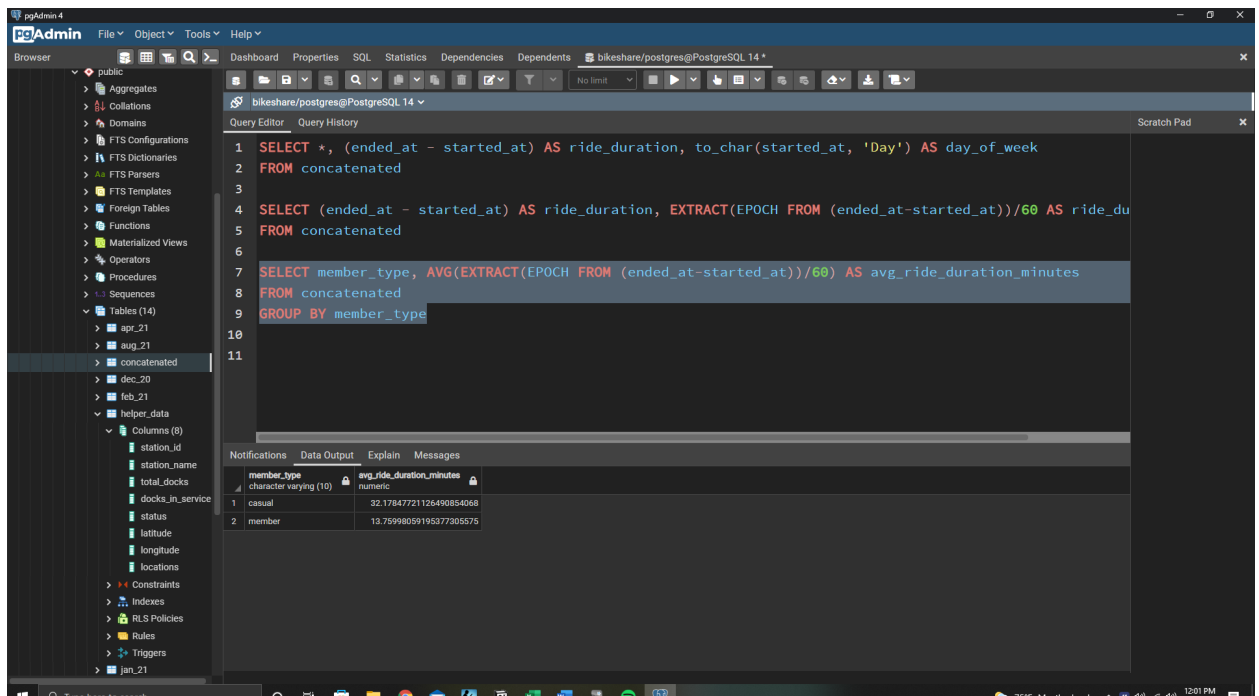


To confirm the total number has been updated I used the COUNT function again.



The total number of rows is correct. [$5,479,096 - 581 = 5,478,515$]

12. Now that the data is clean, I wanted to analyze it. I started by calculating the average ride duration based on member type.



The screenshot shows the pgAdmin 4 interface with the Query Editor open. The left sidebar displays the database structure, including tables like 'station_id', 'station_name', 'total_docks', 'docks_in_service', 'status', 'latitude', 'longitude', 'locations', 'Constraints', 'Indexes', 'RLS Policies', 'Rules', 'Triggers', and 'jan_21'. The main pane shows the Query Editor with the following SQL queries:

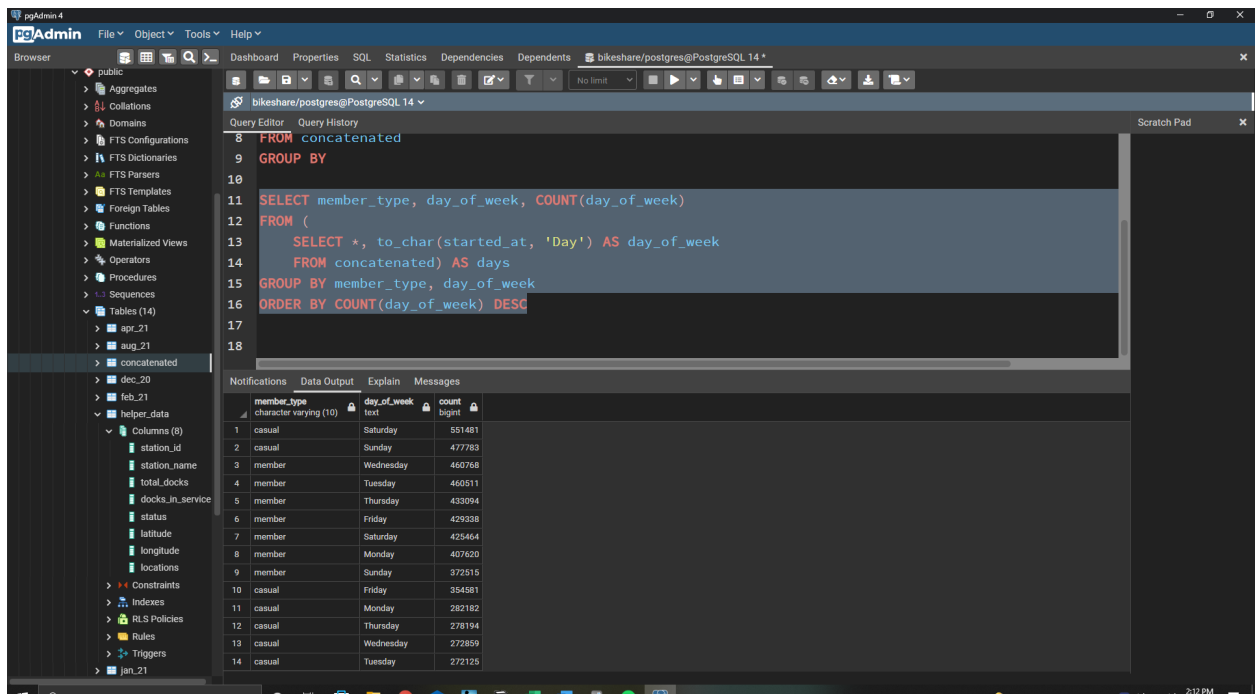
```
1 SELECT *, (ended_at - started_at) AS ride_duration, to_char(started_at, 'Day') AS day_of_week
2 FROM concatenated
3
4 SELECT (ended_at - started_at) AS ride_duration, EXTRACT(EPOCH FROM (ended_at-started_at))/60 AS ride_du
5 FROM concatenated
6
7 SELECT member_type, AVG(EXTRACT(EPOCH FROM (ended_at-started_at))/60) AS avg_ride_duration_minutes
8 FROM concatenated
9 GROUP BY member_type
10
11
```

The bottom pane shows the 'Data Output' tab with the following results:

| member_type | avg_ride_duration_minutes |
|-------------|---------------------------|
| casual | 22.17847721126490854068 |
| member | 13.75998059195377305575 |

Casual riders have an average ride duration that is 2.3 times longer than members.

13. Since my task is to compare how members and casual riders differ, I checked which day of the week was the busiest based on member type.

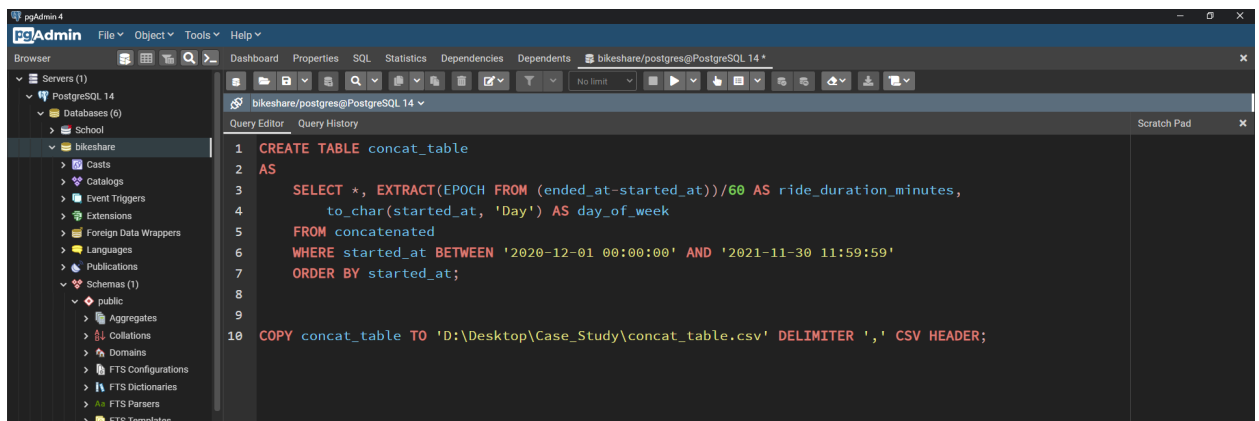


The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'bikeshare' database and its tables. The main window shows a SQL query in the Query Editor, which is executed. The results are displayed in a table with columns: member_type, day_of_week, and count. The data shows that Saturday and Sunday are the busiest days for casual riders, while weekdays are busier for members.

| member_type | day_of_week | count |
|-------------|-------------|--------|
| casual | Saturday | 551481 |
| casual | Sunday | 477783 |
| member | Wednesday | 460768 |
| member | Tuesday | 460511 |
| member | Thursday | 433094 |
| member | Friday | 429338 |
| member | Saturday | 425464 |
| member | Monday | 407620 |
| member | Sunday | 372515 |
| casual | Friday | 354581 |
| casual | Monday | 282182 |
| casual | Thursday | 278194 |
| casual | Wednesday | 272859 |
| casual | Tuesday | 272125 |

This shows that the busiest days are Saturday and Sunday because of casual riders. Then members use Cyclistic bikes the most every day after that. This is followed by the weekday for casual riders.

14. Now that the analysis was complete I created a new table with the additional rows. One for the ride duration in minutes and one for the day of the week the ride took place. Then I copied the table to a CSV file so I could use it in Tableau for my visualization.



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database structure, including the 'bikeshare' database and its tables. The main window shows a SQL query in the Query Editor, which is executed. The query creates a new table named 'concat_table' and copies its data to a CSV file.

```
1 CREATE TABLE concat_table
2 AS
3 SELECT *, EXTRACT(EPOCH FROM (ended_at-started_at))/60 AS ride_duration_minutes,
4 to_char(started_at, 'Day') AS day_of_week
5 FROM concatenated
6 WHERE started_at BETWEEN '2020-12-01 00:00:00' AND '2021-11-30 11:59:59'
7 ORDER BY started_at;
8
9
10 COPY concat_table TO 'D:\Desktop\Case_Study\concat_table.csv' DELIMITER ',' CSV HEADER;
```