Homework 3

(20 points)

Overview

Objectives:
- Working with PostgreSQL
- Nested queries, grouping, aggregation, joins
- Timing of queries and indexing

Postgres Setup Steps: PostgreSQL & pgAdmin
Database Files: chinook_postgres.sql and flights.zip
Submission File: hw3.yaml
What to Submit: Submit the .yaml file you downloaded above, with your answers to Gradescope.
Due Date: Check Gradescope

> ⚠ **Alert!**
>
> Begin working on this homework only after you have completed the required PostgreSQL setup *(linked above)*. If you are doing the homework in a group, you each **must** set up PostgresSQL as you will need it for the final exam.

> ⚡ **Warning**
>
> To encourage local testing of commands, and to reduce the load on the Gradescope autograder, you are allowed a **MAXIMUM of 5 submissions**. If you exceed the number of submissions, your submission will not be graded. The submission with the highest score will be your final grade.

Chinook dataset

> 🖊 **Instructions**
>
> Make sure to order all your queries in ascending order

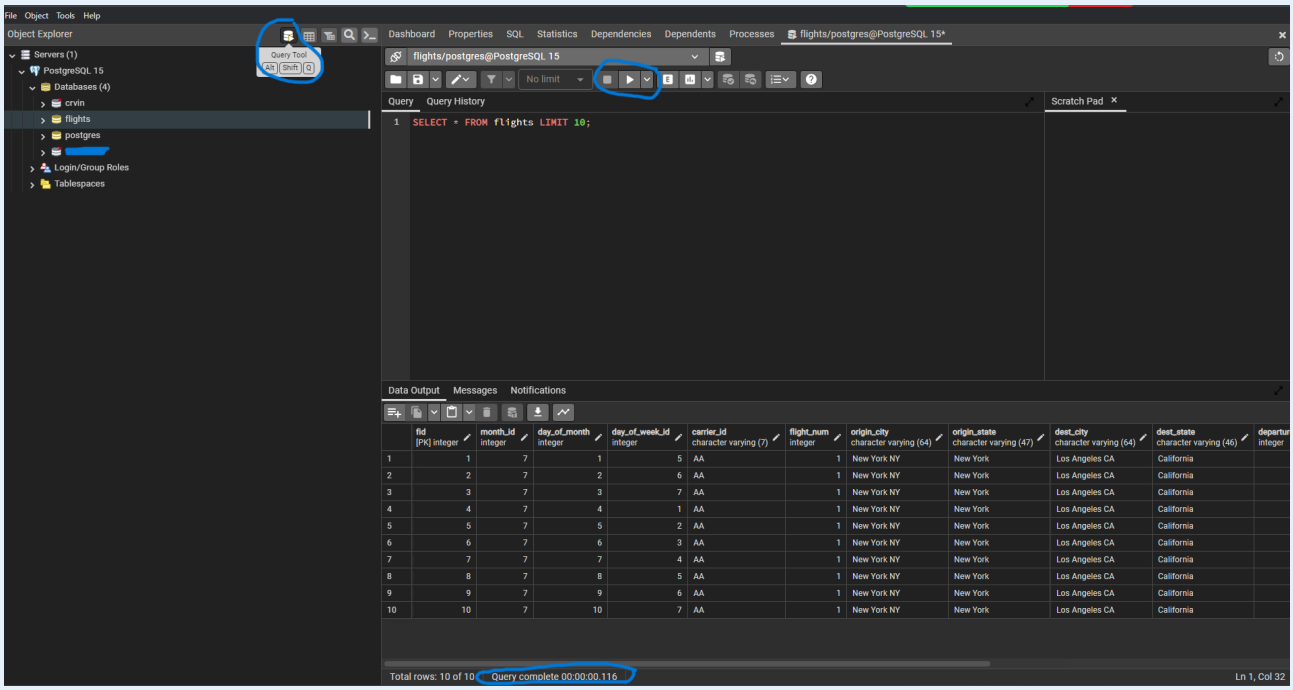
1. (1 point) Write the PostgreSQL statements (**NOT** shell commands!!) needed to import the dataset. Start by creating the database with name as `chinook`, connecting to it and importing the data. Take a look at the `\` commands in `psql`. Try `\?` for help.

2. (1 point) List all tracks that were never purchased by any customers. Return distinct track names only. *Output relation cardinality: 1458*
3. (1 point) List the names of all songs that do not belong to the 90’s Music (with a typographic apostrophe ’ not a straight apostrophe ') playlist. Return distinct track names only. *Output relation cardinality: 1943*
4. (1 point) List the artists who did not record any tracks of the Blues genre. Return distinct artist names only. *Output relation cardinality: 270*
5. (1 point) List all the playlists that do not have any track in the Rock or Blues genres. Return distinct playlist names only. *Output relation cardinality: 10*
6. (1 point) Find the list of artists that record in at least 3 different genres. Return artist names only. *Output relation cardinality: 7*

Flights dataset

Instructions

Make sure to time your queries! This is important for your final question. If you are running your queries on the terminal using `psql`, you can use the `\timing on` command. pgAdmin and BeeKeeper also allow you to view the timing of your query at the bottom of the screen.



Important

No query takes longer than 2 seconds. If your query takes longer, there is probably a better way to write it.

7. (2 points) Write the PostgreSQL statements (**NOT** shell commands!!) needed to import the dataset. Remember that you have to create the database with name as `flights`, connect to it, create the tables, and copy the data. This [stackoverflow post] will help with copying the data. Of course, the autograder doesn't have superuser access so make sure to read the post carefully :)

   ### Note

   Based on your method of import, it is possible that your database might import the `''` (double quotes) as part of your string. Be careful as we do not want the quotes in any of our outputs

8. (2 points) For each origin city, find the destination city (or cities) with the longest direct flight. By direct flight, we mean a flight with no intermediate stops. Judge the longest flight in time, not distance. Name the output columns `origin_city`, `dest_city`, and `time` representing the flight time between them. Do not include duplicates of the same origin and destination city pair. Order the result by `origin_city` and then `dest_city` in ascending order. *Output relation cardinality: 334*
9. (2 points) Find all origin cities that only serve flights shorter than 3 hours. Missing data implies that the flights are shorter than 3 hours. Name the output column `city` and sort them. List each city only once in the result. *Output relation cardinality: 109*

> **Hint**
> Using subqueries will take too long. Check out the `FILTER` keyword from PostgreSQL: Documentation: 16: 4.2. Value Expressions

10. (2 points) For each origin city, consider flights that haven't been canceled. Now, find the percentage of departed flights shorter than 3 hours. For this question, treat flights with missing time data as longer than 3 hours. Name the output columns `origin_city` and `percentage`. Order by descending `percentage` value and then ascending `origin_city`. Be careful to handle cities without any flights shorter than 3 hours. Report `percentage` rounded to 2 decimal places and as percentages (75.25 rather than 0.7525). *Output relation cardinality: 327*
11. (2 points) List all cities that cannot be reached from San Diego through a direct flight but can be reached with one stop (i.e., with any two flights that go through an intermediate city). Do not include San Diego as one of these destinations (even though you could get back with two flights). Name the output column `city`. Order the output ascending by city. *Output relation cardinality: 258*

> **Hint**
> Think about joining two small tables that have only the required data instead of the very big `FLIGHTS` table

12. 1. (1 point) List the names of carriers that operate flights from San Diego to San Francisco. Return each carrier's name only once. Use a nested query to answer this question. Name the output column `carrier`. Order the output ascending by carrier. *Output relation cardinality: 4*
    2. (1 point) Express the same query as above, but do so without using a nested query. Again, name the output column `carrier` and order ascending. *Output relation cardinality: 4*
13. (2 points) Now that you have completed each query without indexes. Go ahead and add the following indices/indexes. Feel free to add more indexes based on the queries that you have written. Get an intuition on which queries would work better if certain columns were ordered.
    `sql CREATE INDEX idx_flights_origin_actual ON FLIGHTS(origin_city, actual_time);`
    `CREATE INDEX idx_flights_origin_dest ON FLIGHTS(origin_city, dest_city);`
    Rerun queries 8-12.2 and note down the time it took to execute those queries. Replace the `0.00`

values in the yaml submission file with the execution time in seconds. Provide a 1-2 sentence explanation on why the queries ran faster with indexes.

> **Note**
> This question will be graded manually.