CS 387 - Lab 4 - Understanding Postgres Performance

The goal of this lab is to experiment with postgres performance and gain some insights as to best practices when it comes to inserting and querying large datasets. You may do this assignment in groups of 2.

Exercise-1:

In this exercise, you'll operate on a large dataset by inserting it into a postgres database using multiple methods and measure the time taken to insert the dataset for each of these methods. The aim here is to make you familiar with the different methods available to insert data into postgres and the performance of those methods.

Download this dataset available in CSV format from **kaggle**. Remove tuples from the dataset which has empty column entries (the dataset has 8M+ rows, create another csv file from it by removing rows containing empty column values). Now, use this dataset as the base and form multiple smaller datasets with a different no. of rows (say 100000 - 500000 in increments of 100000) to perform the tasks mentioned below:

1. Create a simple schema (a single table is fine) to store this information in Postgres (this step is done only once in the exercise)
2. You should insert the data in the following modes:
    a. **Bulk load** each csv using psql (https://www.postgresql.org/docs/8.4/populate.html) and measure the time it takes in each instance.
    b. Generate a .sql file containing **`insert`** statements for each tuple (you can programmatically generate .sql file from the csv file containing insert statements) and execute it to insert the data **using psql.** Measure the time it takes to run the generated file in PSQL for each data set.
    c. Do the Bulk load using available Python drivers for PostgreSQL and measure the time it takes for each dataset.
    d. Inserting each tuple one by one using Python (by connecting to postgres and reusing the connection across inserts).
    e. Inserting each tuple one by one using Python (with closing connection after every insert and reopening it for the next insert).
    In all cases, time your programs using any appropriate strategy.
3. Report the following information:
    a. configuration of the machine on which the above tasks were performed
    b. The time to load the data in all five methods for different sizes of CSV files as a histogram with time on the Y axis and file sizes on the X axis. Plot a single graph for 2a-2d and a separate graph for 2e.

c. Give a brief explanation of how you performed the five kinds of data loading operations, in particular, how did you do bulk loading.

Exercise-2:

We know **fetching** all rows at once is not very efficient, especially in case of a very large dataset. So we fetch data of limited size at once. Most websites and apps use pagination or "load more" techniques. In this question, we will explore a general mistake that might be done while fetching rows using OFFSET, LIMIT, a loop structure of some sort. If you use this method, everything will be fine for the first few thousand rows but after some point, you will observe that the same query is taking more time than the first few iterations. So write a script in python(**2_a.py**) which fetches data (all columns in a select * from <table>.....) in the aforementioned method (LIMIT and OFFSET) - 100 rows at a time (parameterize this number so that you can vary it if needed) for a 1 million row dataset (use the same dataset from Kaggle you had downloaded and cleaned up). Run and time it for each iteration. Graph the per iteration time to fetch the 100 rows against the iteration number.

Write another script in python(**2_b.py**) to fix this problem (look up the use of **CURSORS**) and plot the same graph with 2_b.py as well to compare against the 2_a.py's graph. Feel free to try out any other strategies to improve the performance of a paginated query like this.

Report the time to fetch complete dataset in both methods, attach plots and give a brief explanation of how why fetching data using the OFFSET method is not performing well and why your fix is performing well

Submission Instructions:

- Make a folder named **<rollnumber1>-<rollnumber2>-a4**
- Submit **1_b.py**, **1_c.py**, **1_d.py**, **1_e.py** files for the corresponding parts of exercise-1 and **2_a.py**, **2_b.py** for exercise-2
- Add a **config.py** (same for both exercises) file for postgres configuration (similar to config.py file given for inlab). At all places in code whenever you are connecting to postgres, you **must use config.py** to **create a connection**
- Add a **report.pdf** file that has a report and plots for each of exercise 1 and 2.
- Your submission should not contain any other files
- **Zip** the folder and upload it to the **moodle**
- Your code **must work** on the **docker image that we had put out.**
- Before submitting, check all the file names and run your code one more time to ensure that everything is working fine as we'll only run the python files described above.

Grading Rubric

Exercise	Marks
1	**10(for each method) * 5 + 5(report) = 55**
2	**7.5(for each script file) * 2 + 5(report) = 20**
Total	**75**

Updates

Ex-1

1. For machine configuration, you have to report the number of CPU cores and ram
2. For parts **2a, 2c, 2d**, you will create 5 smaller datasets (with 100000 - 500000 rows in increments of 100000), name them data1.csv, data2.csv,..., data5.csv
   For parts **2b, 2e**, you will create 5 much smaller datasets (with 5000 - 13000 rows in increments of 2000), name them data6.csv, data7.csv,..., data10.csv
   You can assume these CSV files in the same working directory
3. Plot a single combined graph for 2a, 2c, 2d and another combined graph for 2b, 2e

Ex-2

1. You may run the experiment for 100,000 runs instead of a million, if the execution is taking too much time. I suspect you should be able to see a climb in response time by then, if it doesn't, then you should continue the loop