Outlab 4:

Anish Deshpande 180100013
Rajat Jain 180100091

Exercise 1:

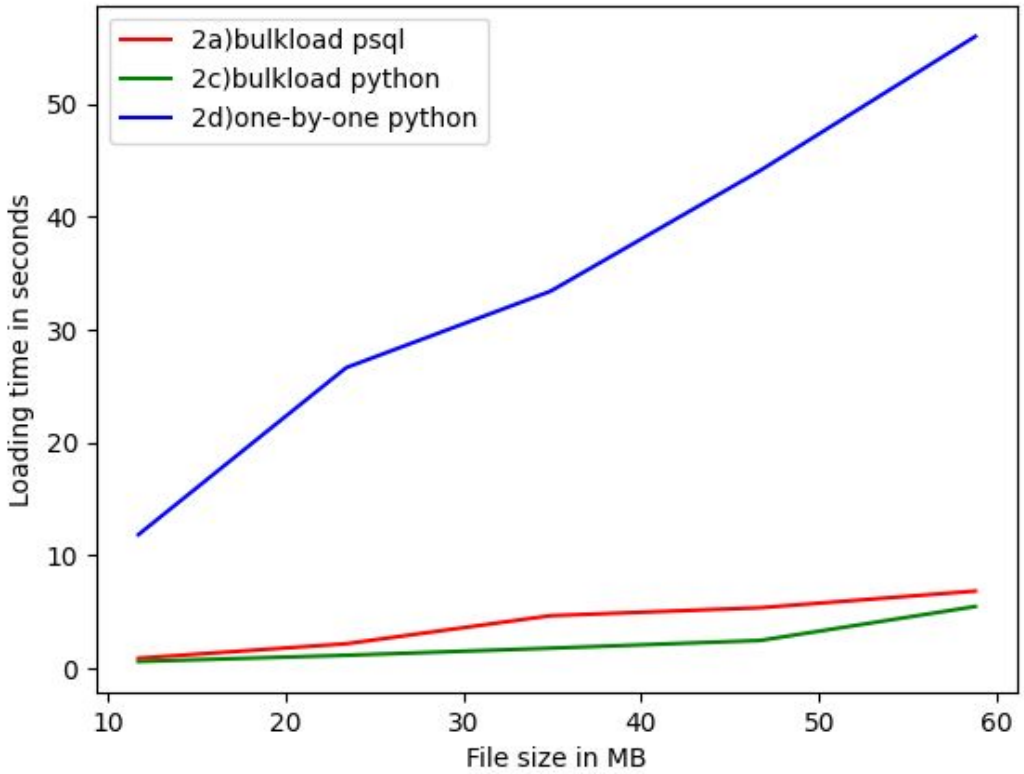
1) Machine details: (intel i3 processor, Ubuntu 18.04)
    a) 5.7 GiB RAM
    b) 4 CPUs
2)
    a) Bulk loading-
        i) 0.911 secs for 11.7MB
        ii) 2 .195 sec for 23.4MB
        iii) 4.68 sec for 34.9MB
        iv) 5.385 sec for 46.7MB
        v) 6.86 sec for 58.8MB
    b) Not bulk loading (insert statements, .sql file)
        i) 0.501 secs for 598.3KB
        ii) 0.689 secs for 836KB
        iii) 0.919 secs for 1.1MB
        iv) 1.11 secs for 1.3MB
        v) 1.292 secs for 1.6MB
    c) Bulk loading using python drivers
        i) 0.6217 seconds for 11.7MB
        ii) 1.1741 secs for 23.4MB
        iii) 1.8025 secs for 34.9MB
        iv) 2.4874 secs for 46.7MB
        v) 5.4935 secs for 58.8MB
    d) Tuple insert one-by-one Python, reuse connection
        i) 11.8543 Secs for 11.7MB
        ii) 26.6472 Secs for 23.4 MB
        iii) 33.4170 Secs for 34.9 MB
        iv) 44.1062 Secs for 46.7 MB
        v) 55.9792 Secs for 58.8MB
    e) Tuple insert one-by-one Python, fresh connection each time
        i) 29.1483 secs for 598.3KB
        ii) 42.4239 secs for 836KB
        iii) 55.7311 secs for 1.1MB
        iv) 68.3306 secs for 1.3MB
        v) 79.3499 secs for 1.6MB

3)How the above loadings were done.

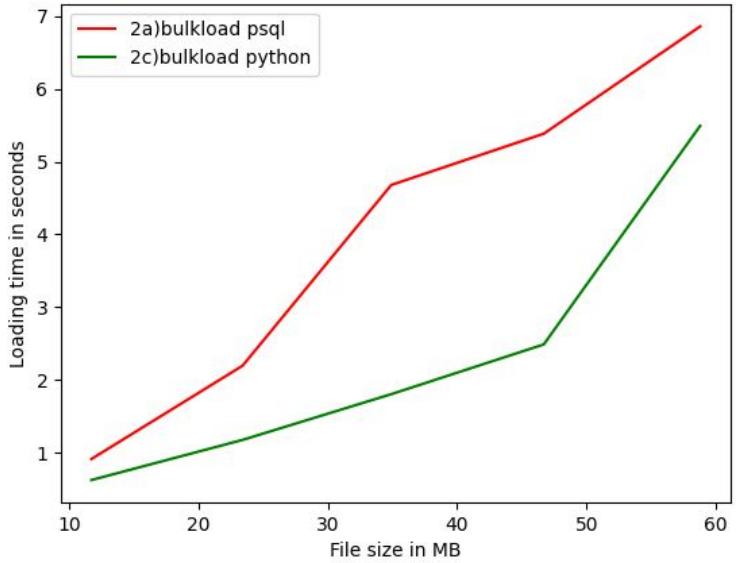
For 2a and 2c, bulk loading, it was achieved using the ‘COPY’ command in sql. ‘COPY’ loads all the rows in one command (we can specify that it is say, a comma-separated csv file from which we are loading the data). It is much faster than stacked ‘INSERT’ statements. Also, the commit is done directly at the end of all loading (as it is a single command), so we don’t have to manually disable autocommit. Either this command was written on the terminal for postgres (or even PGAdmin4, same result), or it was a string in a python script (2c). This string was passed as a query in the python script, to be executed after creating a connection using psycopg2.

For 2b, 2d and 2e, we used .sql files (autogenerated by python scripts from CSV data) which were executed. They contain a long list of INSERT statements. If to be done by python drivers, the file is read from line-by-line, the query loaded and executed. The difference between 2d and 2e is that the psycopg2 connection is remade for 2e, while in 2d it is opened and used until all INSERT statements have been executed.

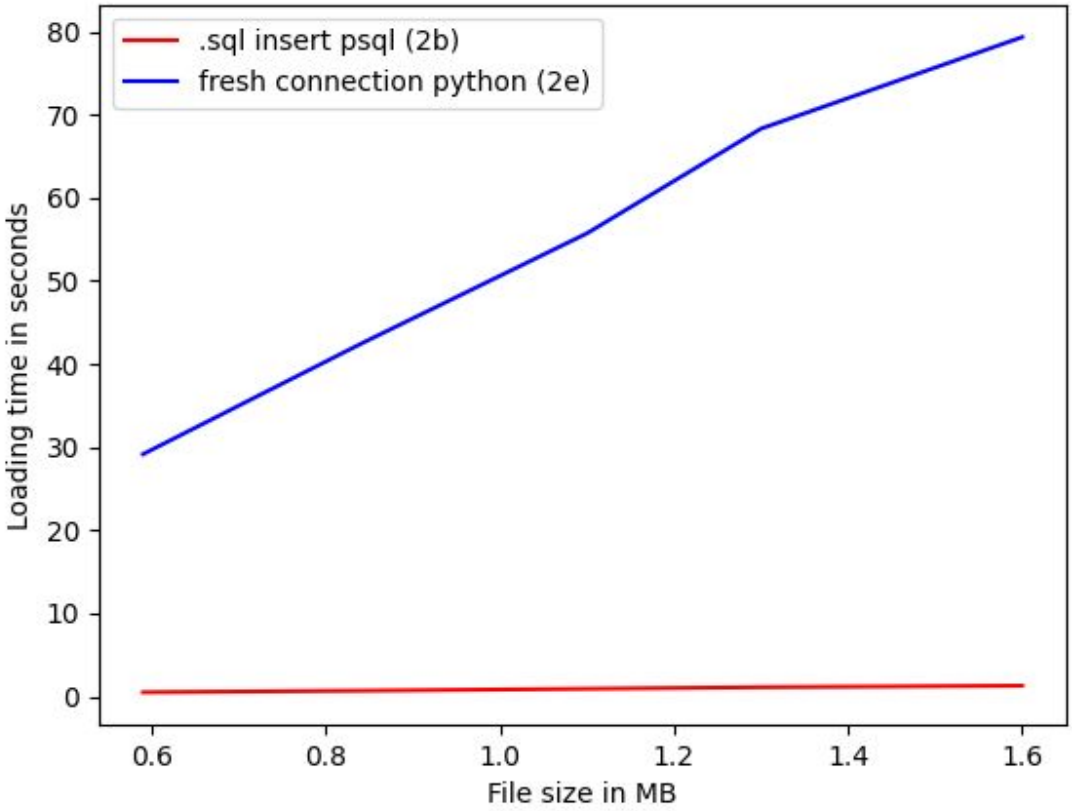
Combined plot for 2a,c,d:



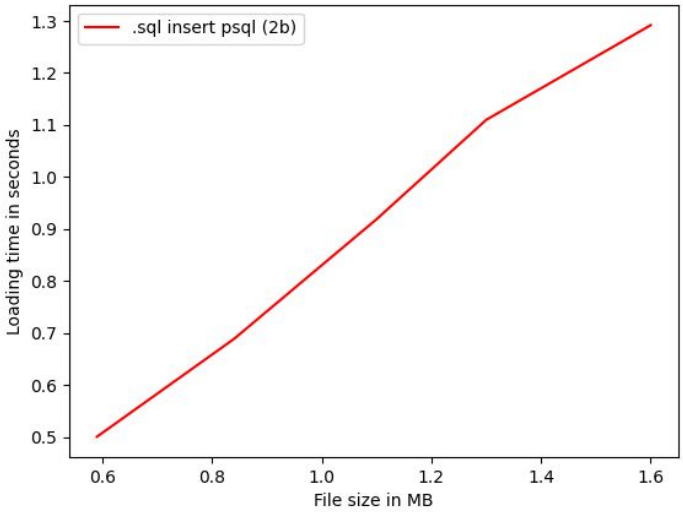
Just 2a,2c for reference:



Combined plot for 2b and 2e:



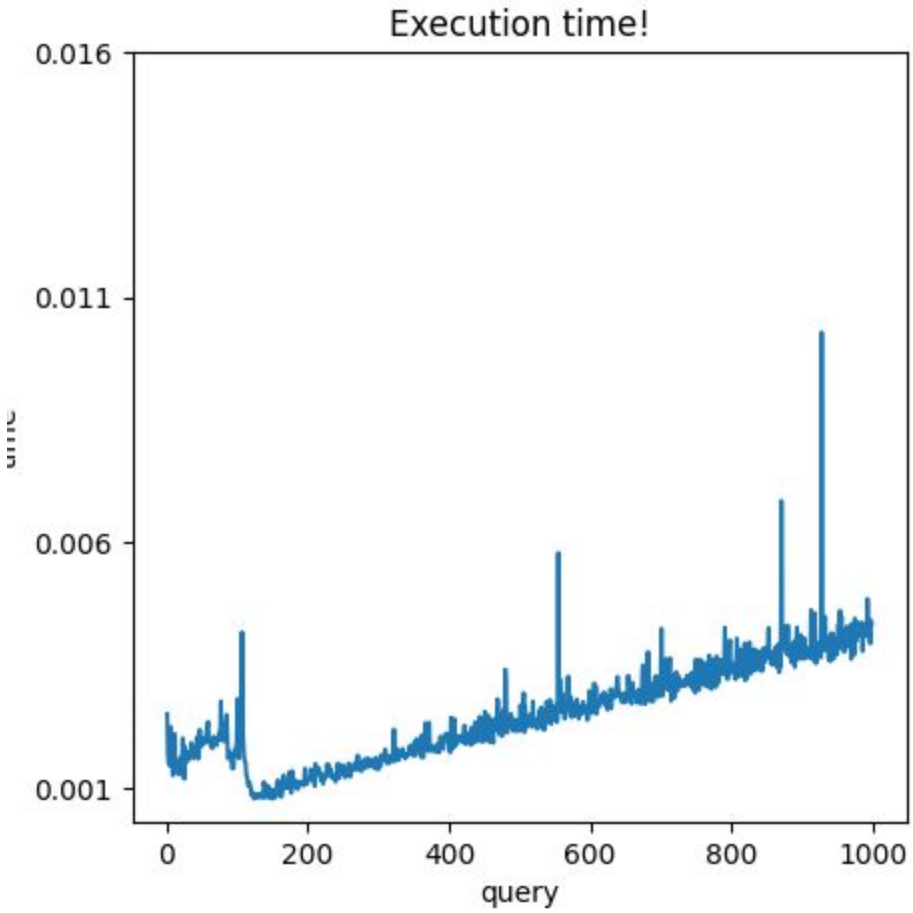
2b plot solo, for reference:



Exercise 2:

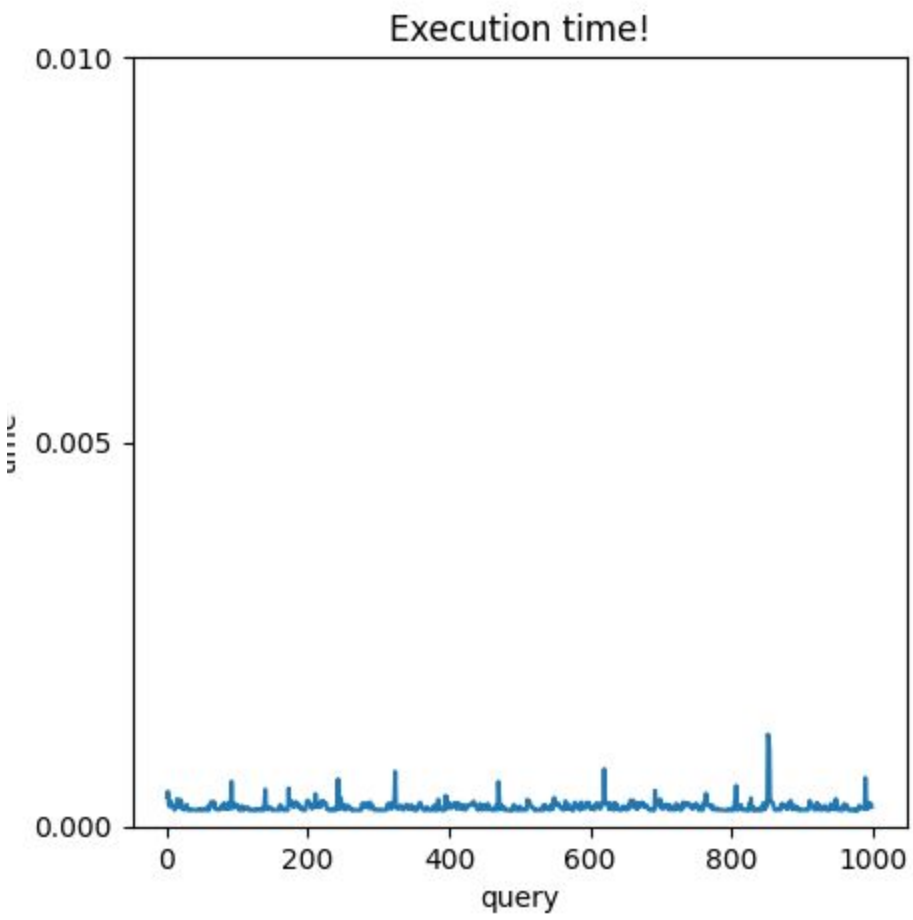
For 1,00,000 dataset size:

a) Takes 2.4890 secs to entire dataset



b) Takes 0.2634 secs to fetch entire dataset

On y axis is time:



Explanation:
If a normal cursor is used, psycopg2 is set to client by default and fetches the entire result at once. When we use a named cursor the result is maintained on the server side and we can fetch rows as necessary. If we had atleast one column in the table which had unique values, another method that could be used for doing this would be to order by that column, now once we fetch the first 100 rows, the next time we could use >=max value in the previous query and limit 100. In this way, offset would not have to be used. In the offset query in 2a, I didn't use order by clause because there is no unique constraint on any of the columns