Eesha Singh
John Gabriel Gutierrez

Overview:

For this project, we stress tested read and write speeds on a MySQL database while adjusting different parameters on both our GCP machine and insert methods in order to improve the performance time. Our first write-speed improvement came from upgrading our database server to a machine with 30 CPUs and 120GB of RAM, and our second improvement came from improving our server to a high-bandwidth server.

Baseline run (Milestone 1):

For the baseline run, we read in a CSV file with a million records to a Python pandas dataframe and attempted to insert these records into our MySQL database one record at a time, committing the changes every 5000 records. We timed this baseline insert (and every insert from this point on) and obtained an average speed of 5min 56s ± 24.3 s. Once we got past the technical challenge of initially writing the code for the insert, the rest of the code in this project was just minor adjustments to the code in this section. Our iterative inserts looked something like this:

```
for row in rows:
        counter += 1
        cursor.execute(sql, row)
        if counter >= 5000:
                counter = 0
                connection.commit()
                print(total_counter, "records inserted successfully into Person
table")
```

Batch runs (Milestone 2):

We then tried to speed up this process by performing batched runs with the `cursor.executemany()` method. We combined our inserts into batches of size 5000, 50,000 and 500,000 for three different batch runs in total. Batch size 5000 had a time of 30.8 s ± 156 ms, batch size 50,000 had a time of 29.5 s ± 80.9 ms, and batch size 500,000 had a time of 30.8 s ± 261 ms. This write speed is a significant improvement from our write speed when we inserted rows one-by-one. Technical challenges included effectively updating the “max_allowed_packet” variable to one that would be able to hold our data, but once getting past this step we encountered very few technical difficulties.

Snippet:

```
cursor = connection.cursor()
temp = []
for row in rows:
    temp.append(row)
    counter += 1
    total_counter += 1
    if counter >= 50000:
        cursor.executemany(sql, temp)
        temp = []
        counter = 0
        connection.commit()
        print(total_counter, "records inserted successfully into Person table")
connection.commit()
cursor.close()
```

Hardware upgrade runs (Milestone 2):

We then upgraded our GCP notebook server from n1-standard-8 to c2-standard-30, which took us from 8 CPUs and 32 GB of RAM to 30 CPUs and 120 GB of RAM. We reran the 500,000 size batch insert and got an average time of 26.8 s ± 144 ms. Next, we upgraded our database server to a high-network bandwidth. The resulting batch insert took 26.1 s ± 215 ms. This process was very labor intensive with lots of troubleshooting (especially configuring the SSH), but the resulting write speed was a slight improvement from before.

Potential future improvements:

Perhaps the write speed could be improved by partitioning the data and writing from multiple sources. Upgrading these machines doing the writing even further could definitely help speed up the process. There could also be some merit in setting up the database so that you're only reading and writing the data that is essential to your operations, so that the speeds for reading and writing are improved.
In addition, it would be interesting for us to complete the same project using Postgres, or even a NoSQL database system to compare results with MySQL and determine which database system has faster and slower write speeds.