Overview

Given so many companies and applications rely on robust and high-performing database systems, it's important to understand what impacts the performance. Testing, tuning, and understanding the vulnerabilities between database systems may help us in the selection of a system that best suits our needs in the future.

In this project, we improved the write performance of a Postgres database system through batch inserts, system variable tuning, and hardware resources (CPUs, RAM, Network). Load testing was conducted using two datasets from Google Cloud Storage. One file contained 500 rows for sampling and debugging; the other contained 1,000,000 for loading into the database. Each row in the CSV files contained seven cells describing a person (first_name, last_name, company_name, etc.)

Baseline run (Milestone 1)

The baseline run used the execute() method to commit the data from the us-1000000.csv file every 5000 rows. It also prints the number of records that were successfully inserted into the table. The %%timeit function was used to measure the average time it took for the code block to execute across eight trials. Here, we used a more efficient version of the execute method, explaining why load times are comparable to subsequent tests.

```
# 990000 records inserted successfully into Person table
# 995000 records inserted successfully into Person table
# 1000000 records inserted successfully into Person table
# Postgres connection is closed
# 47.2 s ± 353 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Batch runs (Milestone 2)

As an alternative to the execute method, this test uses the execute_batch() method. This test batch inserts data in sizes of 5000, 50,000, and 500,000. It loads the same dataset and performance is measured the same using the %%timeit function. Here, the load time stayed fairly consistent between batch sizes.

```
# 990000 records inserted successfully into Person table
# 995000 records inserted successfully into Person table
# 1000000 records inserted successfully into Person table
# Postgres connection is closed
# 1min 12s ± 379 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

# 900000 records inserted successfully into Person table
# 950000 records inserted successfully into Person table
# 1000000 records inserted successfully into Person table
```

```
# Postgres connection is closed
# 1min 10s ± 314 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

# 500000 records inserted successfully into Person table
# 1000000 records inserted successfully into Person table
# Postgres connection is closed
# 1min 10s ± 530 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Hardware upgrade runs (Milestone 2)

We create a new VM instance changing the machine type from n1-standard-8 to c2-standard-30. This adjusts our VM's computing capacity upgrading it to a machine with 30 CPUs and 120 GB of RAM. We then measure its change in time using the %%timeit function on a batch write of 500,000. The records were loaded significantly faster than the batch runs without the additional CPUs and RAM.

```
# 500000 records inserted successfully into Person table
# 1000000 records inserted successfully into Person table
# Postgres connection is closed
# 52.7 s ± 171 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

We then stopped our server instance and created a disk image and a database server with high-network bandwidth. After configuring the DBMS on the high-bandwidth server, we created the database/schema and table, copied the loader code, and ran the load test. Even with the extra bandwidth, the load speed did not increase drastically. This is likely because bandwidth is not a measure of the speed of data, but rather the amount. Since our load test is limited by the speed of data transfer, an increase in bandwidth will not improve the performance of our system.

```
# 500000 records inserted successfully into Person table
# 1000000 records inserted successfully into Person table
# Postgres connection is closed
# 52 s ± 222 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

Potential Future improvements

As for expanding on the study, we could increase the range of sizes of batch inserts to see how performance changes with larger or smaller batches. For a future experiment, we could test with different types of executing algorithms to see how performance varies between the types. Additionally, the creation of secondary indexes could be an interesting future experiment to see how performance varies.