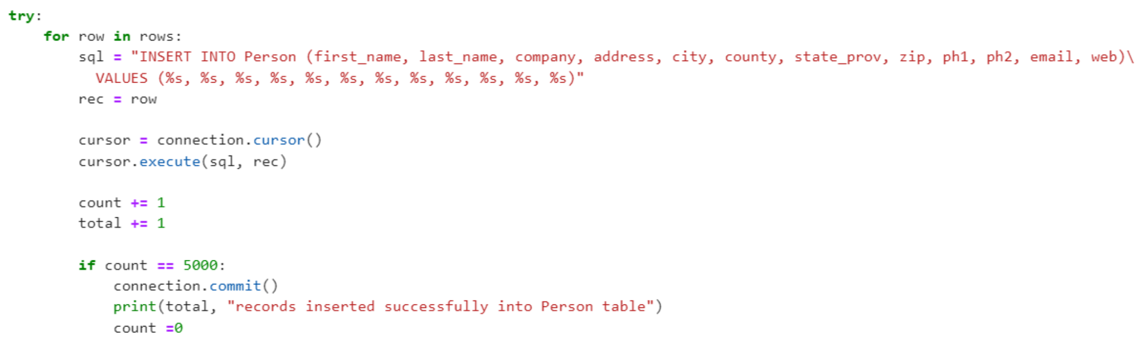
Final Project Report

Overview

The goal of this project is to measure the execution time of inserting records under different conditions and insert methods. We will be implementing methods like execute (), executemany() to either insert one statement at a time or batch insert records using different batch sizes. We will also increase the CPU quota and bandwidth to measure the performance under different hardware conditions.

Baseline run

The baseline run entailed method like execute () to execute one insert statement at a time. The count size per insert is 5000.



```
try:
    for row in rows:
        sql = "INSERT INTO Person (first_name, last_name, company, address, city, county, state_prov, zip, ph1, ph2, email, web)\
          VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
        rec = row

        cursor = connection.cursor()
        cursor.execute(sql, rec)

        count += 1
        total += 1

        if count == 5000:
            connection.commit()
            print(total, "records inserted successfully into Person table")
            count =0
```

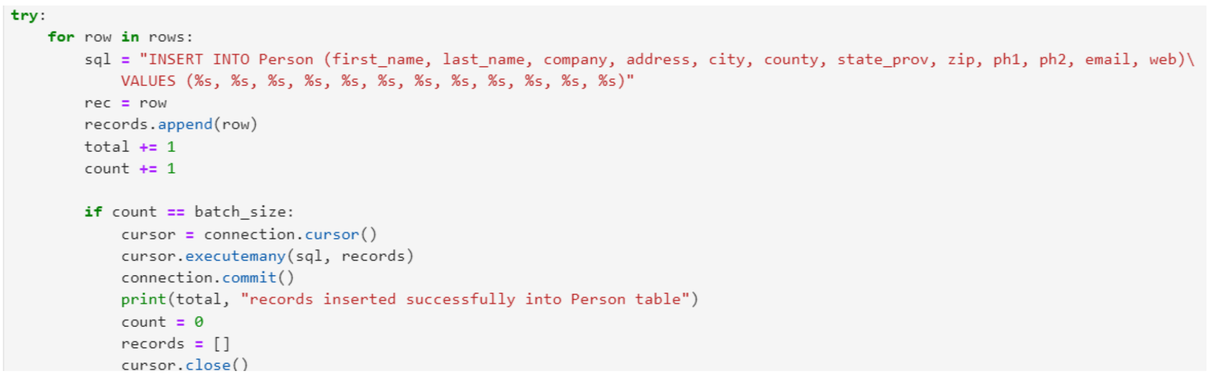
The run achieved a result of the following:

8min 45s ± 10.8 s per loop (mean ± std. dev. of 7 runs, 1 loop each)

The insert was fairly slow by using this method. We got errors for address and email fields being too long when loading the large file, so we added 10 to those fields instead of 5.

Batch runs

The three batch runs use three different batch sizes and the executemany() method to compare the impact batch size have on inserting records.



```
try:
    for row in rows:
        sql = "INSERT INTO Person (first_name, last_name, company, address, city, county, state_prov, zip, ph1, ph2, email, web)\
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"
        rec = row
        records.append(row)
        total += 1
        count += 1

        if count == batch_size:
            cursor = connection.cursor()
            cursor.executemany(sql, records)
            connection.commit()
            print(total, "records inserted successfully into Person table")
            count = 0
            records = []
            cursor.close()
```

The results are as follows:

Batch size=5000: 40.1 s ± 343 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Batch size=50,000: 38.1 s ± 171 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Batch size= 500,000: 40.9 s ± 426 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
Batch size 50,000 runs the fastest of the three.

Hardware upgrade runs

The two hardware upgrades are upgrading the CPU and the bandwidth. By running the batch 500,000 after the upgrades, we can determine whether hardware upgrades can have effects on insert time. The results are as follows:

CPU upgrade: 35.5 s ± 109 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
High bandwidth: 44.2 s ± 279 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
In [15]:
The CPU upgrade saw a 5.4-second upgrade over the original run, yet there is no significant difference under the high bandwidth mode. We re-upgraded the CPU on a second try and it saw a significant decrease in insert time.

Potential future improvements

We could include more batch sizes in the batch run, or we could compare results of CPU/batch size combos. If timeline allows for it, we will try out Postgres as our major database system.