DataEng S24: Data Storage In-class Assignment

A. Discussion Question

Do you have any experience with ingesting bulk data into a DBMS? If yes, then describe your experience, especially what method was used to input the data into the database. If no, then describe how you might ingest daily incremental breadcrumb data for your class project.

Response:

I have little experience with ingesting large volumes of data into a database.We used to ingest data every week while working for a project at work. We collect the data and store it in a file, and then use the copy statement to load it into the database from the file. This method significantly speeds up the data loading process compared to inserting individual records, especially when dealing with millions of records. For instance, we had to process around 8 million records, and using the file-based loading approach reduced the loading time to around 3 hours.

C. Prepare Data for Loading

```
(env) sridhamo@storage-vm:~$ python3 load_inserts.py -d ./AL2015_1.csv -c
readdata: reading from File: ./AL2015_1.csv
Created CensusData
Loading 1181 rows
Finished Loading. Elapsed Time: 1.08 seconds
(env) sridhamo@storage-vm:~$ python3 load_inserts.py -d ./OR2015_2.csv
readdata: reading from File: ./OR2015_2.csv
Loading 837 rows
Finished Loading. Elapsed Time: 0.7907 seconds
(env) sridhamo@storage-vm:~$
```

D. Baseline - Simple INSERT

```
sridhamo@storage-vm:~$ source env/bin/activate
(env) sridhamo@storage-vm:~$ python3 load_inserts.py -d ./acs2015_census_tract_data_part1.csv -c
readdata: reading from File: ./acs2015_census_tract_data_part1.csv
Created CensusData
Loading 36844 rows
Finished Loading. Elapsed Time: 27.63 seconds
(env) sridhamo@storage-vm:~$ python3 load_inserts.py -d ./acs2015_census_tract_data_part2.csv
readdata: reading from File: ./acs2015_census_tract_data_part2.csv
Loading 37157 rows
Finished Loading. Elapsed Time: 29.53 seconds
(env) sridhamo@storage-vm:~$
```

E. Disabling Indexes and Constraints

```
(env) sridhamo@storage-vm:~$ python3 load_inserts1.py -d ./acs2015_census_tract_data_part1.csv -c
readdata: reading from File: ./acs2015_census_tract_data_part1.csv
Created CensusData
Loading 36844 rows
Finished Loading. Elapsed Time: 27.32 seconds
Created indexes for CensusData
(env) sridhamo@storage-vm:~$ python3 load_inserts1.py -d ./acs2015_census_tract_data_part2.csv
readdata: reading from File: ./acs2015_census_tract_data_part2.csv
Loading 37157 rows
Finished Loading. Elapsed Time: 27.74 seconds
```

F. Disabling Autocommit

```
(env) sridhamo@storage-vm:~$ python3 load_inserts1.py -d ./acs2015_census_tract_data_part1.csv -c
readdata: reading from File: ./acs2015_census_tract_data_part1.csv
Created CensusData
Loading 36844 rows
Finished Loading. Elapsed Time: 5.886 seconds
Created indexes for CensusData
```

```
(env) sridhamo@storage-vm:~$ python3 load_inserts1.py -d ./acs2015_census_tract_data_part2.csv
readdata: reading from File: ./acs2015_census_tract_data_part2.csv
Created CensusData
Loading 37157 rows
Finished Loading. Elapsed Time: 6.621 seconds
Created indexes for CensusData
```

G. UNLOGGED table

```
(env) sridhamo@storage-vm:~$ python3 load_inserts2.py -d ./acs2015_census_tract_data_part1.csv
readdata: reading from File: ./acs2015_census_tract_data_part1.csv
Created CensusData_staging
Loading 36844 rows
Finished Loading. Elapsed Time: 5.479 seconds
```

```
(env) sridhamo@storage-vm:~$ python3 load_inserts2.py -d ./acs2015_census_tract_data_part2.csv
readdata: reading from File: ./acs2015_census_tract_data_part2.csv
Created CensusData_staging
Loading 37157 rows
Finished Loading. Elapsed Time: 5.748 seconds
```

H. Temp Tables and Memory Tuning

```
(env) sridhamo@storage-vm:~$ python3 load_inserts2.py -d ./acs2015_census_tract_data_part1.csv -c
readdata: reading from File: ./acs2015_census_tract_data_part1.csv
Created CensusData_staging
Loading 36844 rows
Finished Loading. Elapsed Time: 5.328 seconds
```

```
(env) sridhamo@storage-vm:~$ python3 load_inserts2.py -d ./acs2015_census_tract_data_part2.csv
readdata: reading from File: ./acs2015_census_tract_data_part2.csv
Created CensusData_staging
Loading 37157 rows
Finished Loading. Elapsed Time: 5.503 seconds
```

After memory tuning:

```
(env) sridhamo@storage-vm:~$ python3 load_inserts2.py -d ./acs2015_census_tract_data_part1.csv -c
readdata: reading from File: ./acs2015_census_tract_data_part1.csv
Created CensusData_staging
Loading 36844 rows
Finished Loading. Elapsed Time: 4.926 seconds
```

```
(env) sridhamo@storage-vm:~$ python3 load_inserts2.py -d ./acs2015_census_tract_data_part2.csv
readdata: reading from File: ./acs2015_census_tract_data_part2.csv
Created CensusData_staging
Loading 37157 rows
Finished Loading. Elapsed Time: 5.16 seconds
```

I. Built In Facility (copy_from)

```
(env) sridhamo@storage-vm:~$ python3 load_inserts3.py -d ./acs2015_census_tract_data_part1.csv -c
Created censusdata
Loading data from ./acs2015_census_tract_data_part1.csv
Finished Loading. Elapsed Time: 0.3265 seconds
```

```
(env) sridhamo@storage-vm:~$ python3 load_inserts3.py -d ./acs2015_census_tract_data_part2.csv
Created censusdata
Loading data from ./acs2015_census_tract_data_part2.csv
Finished Loading. Elapsed Time: 0.3317 seconds
(env) sridhamo@storage-vm:~$
```

J. Results

Method	Time to load part1	Time to load part2
D. Simple inserts	27.63 seconds	29.53 seconds
E. Drop Indexes and Constraints	27.32 seconds	27.74 seconds
F. Disable Autocommit	5.886 seconds	6.727 seconds
G. Use UNLOGGED table(run)	5.479 seconds	5.748 seconds
H. Temp Table with memory tuning	5.328 seconds 4.926 seconds	5.503 seconds 5.16 seconds
I. copy_from(run)	0.3265 seconds	0.3317 seconds

K. Observations

It can be observed that simple insert statements took a lot of time compared to other methods. Dropping indexes and constraints before loading and recreating them later did not significantly improve performance compared to simple inserts.But, Disabling autocommit improved performance as the database did not have to commit after each insert statement. Unlogged tables improved the performance.Using a temporary table with memory tuning settings further improved performance, indicating that optimizing memory usage can enhance loading speed.The copy_from method performed the best among all methods, with significantly shorter loading times.