# CS315 Assignment : Report

Rohit Ranjan
180629

1.) (i)The Mongodb queries are as follows:

a) db.A.find({ A1: {$lte : 50}})

b)db.B.aggregate([{$sort : {B3:1}}])

c)db.B.aggregate([{$group:{_id:"$B2", count: {$sum: 1}}}])

d)db.B.aggregate(

```
[
    { $lookup:
        { from: "A", localField: "B2", foreignField: "A1", as: "_A" }
    } ,
    { $project:
        { B1:1, B2:1,B3:1,"_A.A2":1 }
    }
] ,
{"allowDiskUse":true}
```

)

(ii)The SQL queries are as follows:

a)

```
SELECT *
FROM A
WHERE A1 <= 50;
```

b)

```
SELECT *
FROM B
ORDER BY B3;
```

c)

```
SELECT count(B1), B2
FROM B
GROUP BY B2;
```

d)

```
SELECT B1, B2, B3, A2
FROM A INNER JOIN B ON A1 = B2;
```

# CS315 Assignment : Report

Rohit Ranjan
180629

2.) The table for average time is as follows:

Note: each entry of table denotes a ± b.
where, a is the average time taken (in milliseconds)
b is the standard deviation (in milliseconds)
I've computed each query 7 times for each set of databases and computed average and standard deviation with discarding minimum and maximum time.

TABLE:

| | | B-100-3-1.csv | B-100-5-2.csv | B-100-10-4.csv | B-1000-5-2.csv | B-1000-10-4.csv | B-1000-50-3.csv | B-10000-5-4.csv | B-10000-50-3.csv | B-10000-500-1.csv |
|---|---|---|---|---|---|---|---|---|---|---|
| **sqlite** | **query1** | 0.0±0 | 0.2±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.2±0 | 0.4±0 | 0.2±0 | 0.0±0 |
| | **query2** | 0.0±0 | 0.6±0 | 1.0±0 | 4.0±0 | 8.8±1.0 | 45±6 | 58±7 | (4.5±0.4)e+02 | (5.4±0.4)e+03 |
| | **query3** | 0.6±0 | 0.6±0 | 0.6±0 | 1.8±0 | 2.6±0 | 11.2±2.0 | 24±4 | 131±16 | (1.60±0.07)e+03 |
| | **query4** | 0.8±0 | 0.6±0 | 0.8±0 | 3.0±0 | 5.4±0 | 26.0±3.0 | 38±7 | 248±17 | (2.70±0.24)e+03 |
| **mariadb_idx** | **query1** | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 |
| | **query2** | 0.0±0 | 0.0±0 | 0.0±0 | 2.0±0 | 6.0±0 | 26.8±2.0 | 27.2±1.0 | 232±8 | (4.3±0.5)e+03 |
| | **query3** | 0.0±0 | 0.0±0 | 0.0±0 | 1.0±0 | 2.4±0 | 7.6±0 | 12.2±0 | 69±4 | (7.1±0.5)e+02 |
| | **query4** | 0.0±0 | 1.0±0 | 1.0±0 | 6.0±0 | 16.2±2.0 | 102±5 | 76.0±3.0 | 581±20 | (2.37±0.11)e+04 |
| **mariadb** | **query1** | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 4.6±0 | 3.8±0 | 3.2±0 |
| | **query2** | 0.0±0 | 0.0±0 | 1.0±0 | 10.4±1.0 | 15.2±0 | 102±20 | 113±13 | (9.7±0.8)e+02 | (2.41±0.09)e+04 |
| | **query3** | 0.0±0 | 0.0±0 | 0.0±0 | 2.8±0 | 4.0±0 | 21.2±3.0 | 32.8±3.0 | 195±11 | (1.80±0.10)e+03 |
| | **query4** | 3.0±0 | 4.0±0 | 7.6±0 | (4.3±0.4)e+02 | 673±14 | (3.86±0.09)e+03 | (4.55±0.06)e+04 | (2.84±0.04)e+05 | (2.700±0.015)e+06 |
| **mongo** | **query1** | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 0.0±0 | 4.4±0 | 4.0±0 | 4.0±0 |
| | **query2** | 0.0±0 | 0.0±0 | 0.2±0 | 4.2±0 | 10.8±2.0 | 35.8±0 | 54±5 | 442±18 | (5.7±0.8)e+02 |
| | **query3** | 0.0±0 | 0.0±0 | 0.0±0 | 3.6±0 | 5.8±0 | 22.6±1.0 | 39±9 | 236±16 | (2.17±0.23)e+03 |
| | **query4** | 23±4 | 25.4±1.0 | 48±7 | (1.83±0.25)e+03 | (3.03±0.08)e+03 | (1.47±0.10)e+04 | (1.69±0.10)e+05 | (1.153±0.031)e+06 | (1.052±0.027)e+07 |

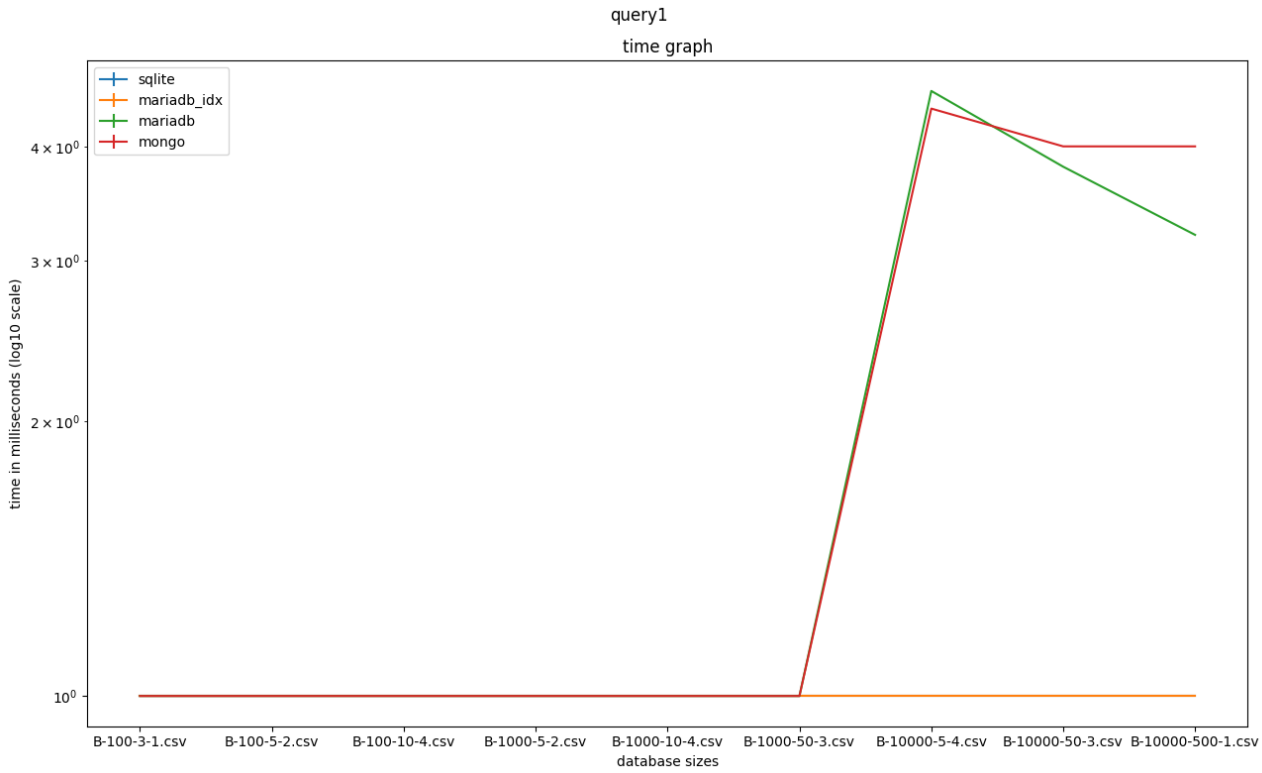
# CS315 Assignment : Report

Rohit Ranjan
180629

3.)
All graphs are on the logarithmic scale, and standard deviations are shown using vertical lines at appropriate places.
All graphs have 9 points(for each database file set) with name of only second file, the file can be inferred from the size of second file.
For eg.

If point on graph shows time for **B-1000-5-2.csv**. Means the set of files used are **A-1000.csv and B-1000-5-2.csv** (as both files have size 1000).

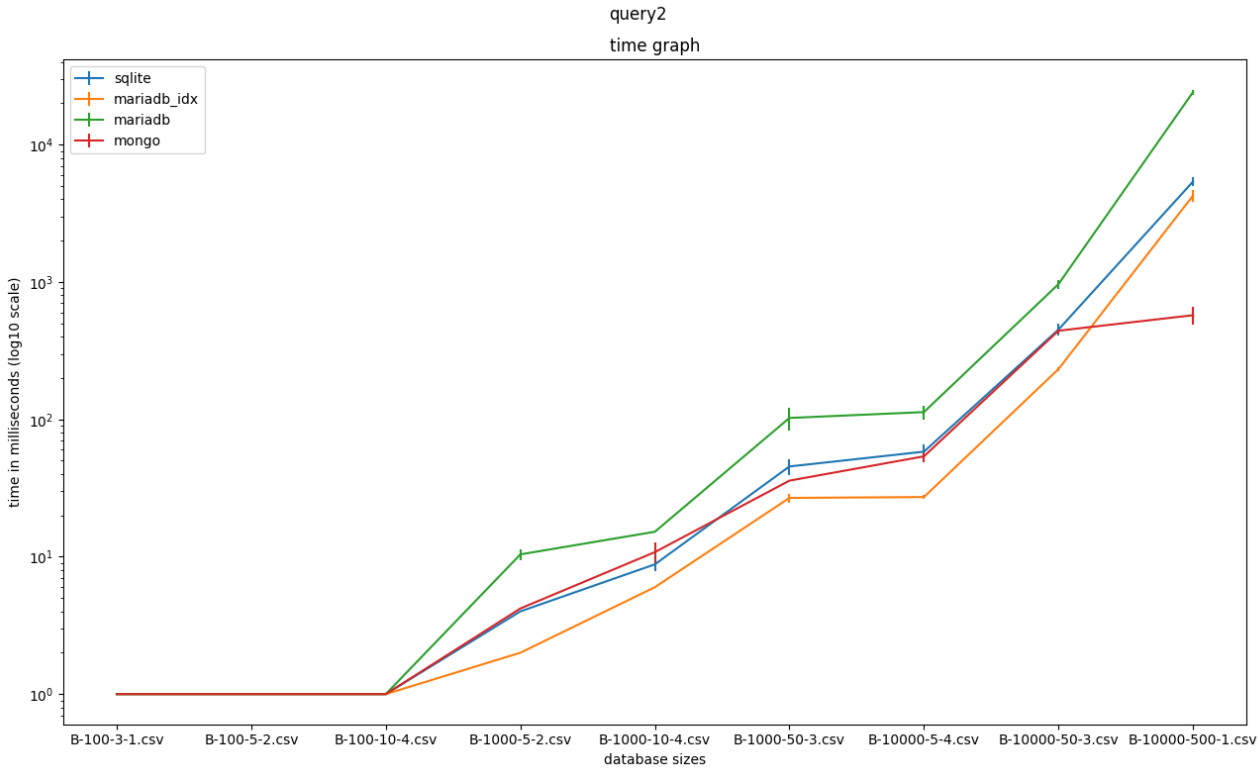
i)Graph for query1 is as follows:



# CS315 Assignment : Report

Rohit Ranjan
180629

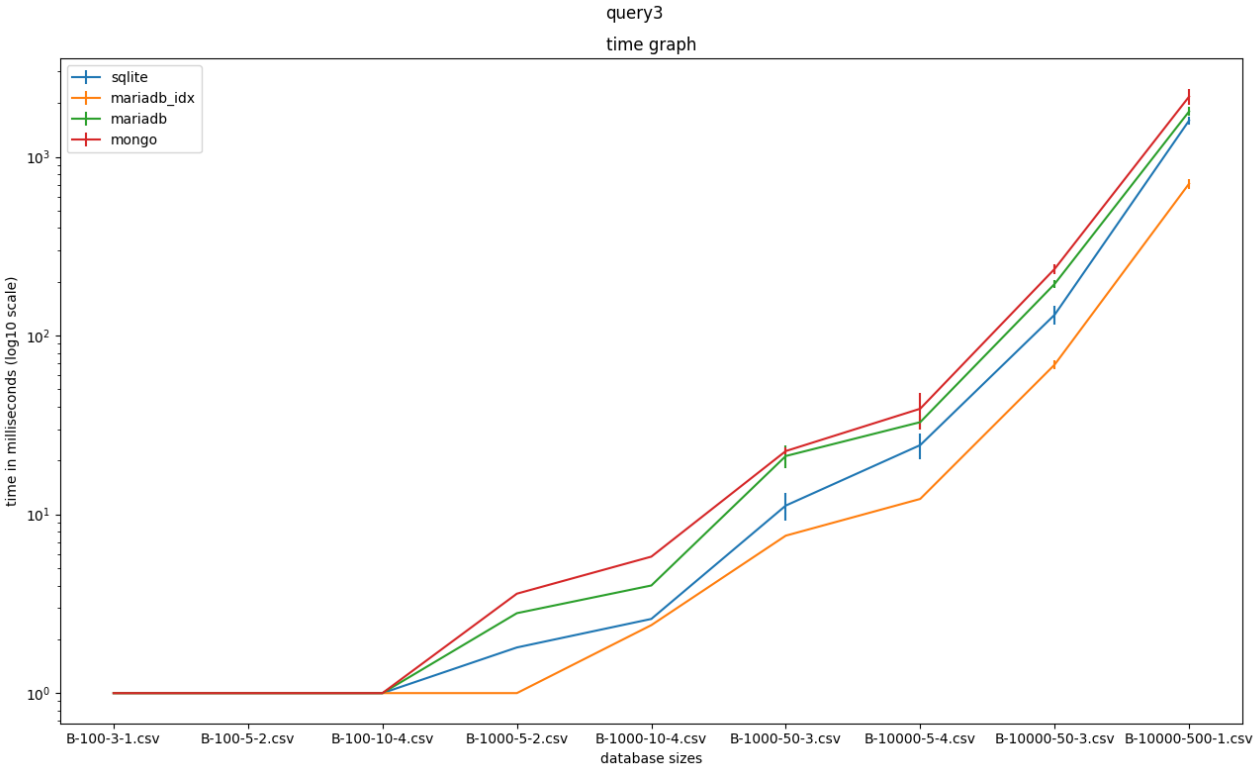
(ii) Graph for query2 is as follows:



# CS315 Assignment : Report

Rohit Ranjan
180629

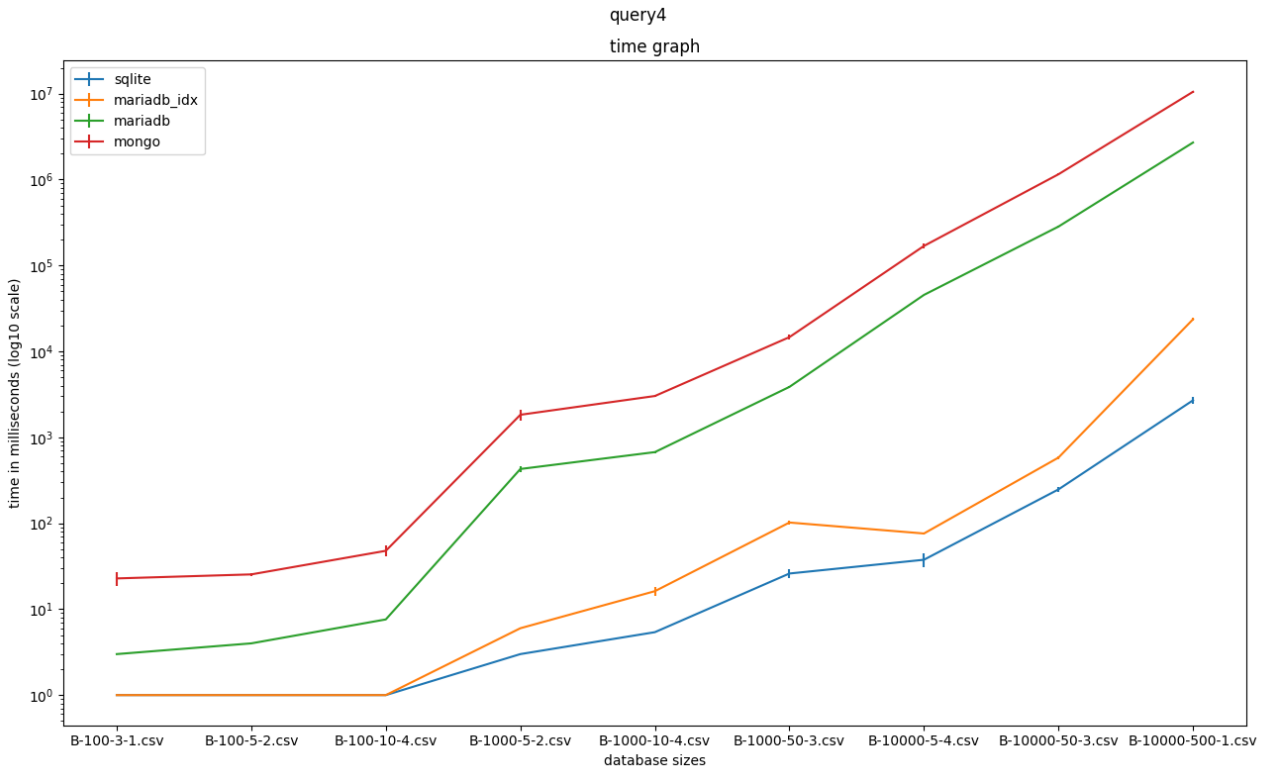
(iii) Graph for query3 is as follows:



# CS315 Assignment : Report

Rohit Ranjan
180629

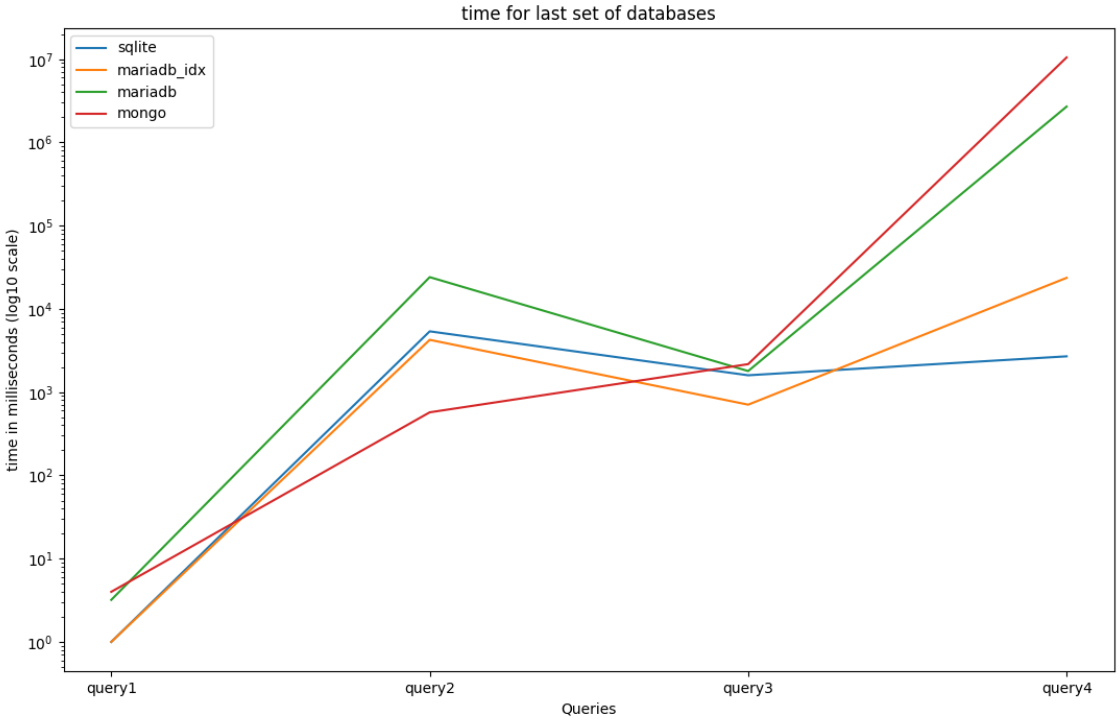
(iv) Graph for query4 is as follows:



# CS315 Assignment : Report

Rohit Ranjan
180629

(v) Query time comparison for last database file set. The graph is as follows:



# CS315 Assignment : Report

Rohit Ranjan
180629

4). Operating System: Ubuntu 20.04.2 LTS
OS type: 64 bit
Graphics: llvmpipe (LLVM 11.0.0, 256 bits)
Processor: Intel® Core™ i5-8265U CPU @ 1.60GHz × 8
Disk Capacity: 274.9 GB
Memory(RAM) : 7.8 GiB

**Conclusion:**

Overview:

- Mariadb with indexing is always faster than mariadb without indexing in all the cases.
  Reason:
  Indexing always saves time. Indexing is done usually by pre-calculation on some attributes and store information about it. Indexing makes record access more faster than without indexing, it helps very much in sorting of records considering some attributes.
  Which is evident from all graphs shown above. We're doing indexing on primary key(by default) on B(B1) and also creating other index named **sort** on B(B3 asc, B1,B2).
- MongoDb joining table is slower than every other engine
  Reason:
  MongoDB tables are not usually a table rather it is a collection of records. It is not an relational database like MariaDB and SQLite. Hence, it does not provide foreign key relation system.
  Which slows down join much further. So, It have to look up to every record by record to perform the operations. Which can be seen on the graph of **query4** above.
- Query1 takes lowest in comparison to other queries
  Reason:
  In this particular query, we can do linear search on every records (which is indeed by mariadb without indexing and mongoDB). Indexing also makes time consumption to decrease. We can infer this from last graph.
- Query4 takes highest in comparison to all other queries
  Reason:
  It involves join on table on two tables. We can infer this from last graph.
- On Query3 mongoDB takes more time than other engines.
  Reason:
  same thoughts as described in the 2nd point. It is an unconstructed table(collection). The group by operation is comparatively slower than other engines. We can observe this from graph for **query3**.
- selection of all entries in mongoDB is bit faster than other engines as data size grows(query2)
  Reason:
  It is a collection of data. Availability of data for MongoDB is great factor for this particular query performance. Which MongoDB supports while other engines do not. Graph for **query2** indicates this.
- In general SQLite takes lesser amount of time if data size grows in comparison to all other database engines
  Reason:
  It is because Sqlite designed for local user computation while other engines run on server service system which can handle other users computation too.

# CS315 Assignment : Report

Rohit Ranjan
180629

Scalabilty:

We can see that as the data file size (number of entries) grows the time for queries will increase significantly. This very much depict-able from the all first four graphs.
As for SQLite the query time increases from 0 milliseconds to upto tens of thousands of milliseconds. Same kind of pattern follows for every other database engine.

For MongoDB,

for query4 , time taken of size 100 is 23 milliseconds while for size 10000 it is around $10^7$ milliseconds.

Databases Engines:

SQLite is quite fast to handle big data sizes we can the it only consumes some tens of seconds on file size of 10000 entries while others engine takes significant amount of time.
Hence, SQLite is an good choice for personal computation(for testing).
While for production perspective Mariadb with indexing is a better choice. It can handle concurrent requests from multiple users.
For mongoDB. It has faster access of data and has deep query ability and less schema overhead.

System issues:

Challenge of our system to get the accurate time for query run. As our normal operating system has tons of processes running simultaneously, which is an overhead of O/S to schedule processes. Scheduling and rescheduling processes may incur significant time to execution of queries.
Another problem, when we produce the output the result of queries. It requires either redirection results to a file or on the console. In both of cases O/S has to create an Interrupt to write back on the disk/console. Which also may introduce some more time to query computation time.
All above the list interference are quite random and depends on system environment and performance.

# CS315 Assignment : Report

Rohit Ranjan
180629

5.) My roll number is 180629

Thus, a = 6, b = 2 and c = 9
The set of 9 integers are : [1,2,4,2,4,3,4,3,1]

The Databases sets I’ve used are:

1. A-100.csv, B-100-3-1.csv
2. A-100.csv, B-100-5-2.csv
3. A-100.csv, B-100-10-4.csv
4. A-1000.csv, B-1000-5-2.csv
5. A-1000.csv, B-1000-10-4.csv
6. A-1000.csv, B-1000-50-3.csv
7. A-10000.csv, B-10000-5-4.csv
8. A-10000.csv, B-10000-50-3.csv
9. A-10000.csv, B-10000-500-1.csv

# CS315 Assignment : Report

Rohit Ranjan
180629

# How to run the scripts?

To run all the scripts make sure you've all sets of data files in the directory named **dbs/**(without forward slash).

To run the scripts following things/packages are required(must be installed):

1.**matplotlib** (pip3 python package)
    To install if required: `pip3 install matplotlib`
2.**numpy**(pip3 python package)
    To install if required: `pip3 install numpy`
3.**pandas**(pip3 python package)
    To install if required: `pip3 install pandas`
4.**dataframe_image**(pip3 python package)
    To install if required: `pip3 install dataframe_image`
5.**uncertainties**(pip3 python package)
    To install if required: `pip3 uncertainties`

**To run the all queries for all databases for all engines:**

I) Visit to script directory
II) Add the following command

```
$sudo bash run.sh
```

- enter your root password to execute
- wait to finish the program, it usually takes very long time around (22-24 hr) depending the  enviornment and machine

To run only for **sqlite:**

You can enter following command:

```
$sudo bash run.sh sqlite
```

This will run on the 9 data files sets corresponding to my roll number. To run on specific files you may run following command:

```
$sudo bash sqlite/sqlite.sh file_name_for_table_A  file_name_for_table_B
```

To run only for **mariadb with indexing:**

You can enter following command:

```
$sudo bash run.sh mariadb_idx
```

This will run on the 9 data files sets corresponding to my roll number. To run on specific files you may run following command:

```
$sudo bash mariadb_idx/mariadb_idx.sh file_name_for_table_A  file_name_for_table_B
```

# CS315 Assignment : Report

Rohit Ranjan
180629

To run only for **maridb without indexing:**

You can enter following command:

```
$sudo bash run.sh mariadb
```

This will run on the 9 data files sets corresponding to my roll number. To run on specific files you may run following command:

```
$sudo bash mariadb/mariadb.sh file_name_for_table_A  file_name_for_table_B
```

To run only for **mongodb:**

You can enter following command:

```
$sudo bash run.sh mongo
```

This will run on the 9 data files sets corresponding to my roll number. To run on specific files you may run following command:

```
$sudo bash mongo/mongo.sh file_name_for_table_A  file_name_for_table_B
```

The time consumptions for each query will appear in form of text file in **${engine}/${engine}_time.txt**

Eg . For **sqlite** it'll appear in **sqlite/** directory in file **sqlite_time.txt**

**For Graph and table creation** (all graphs and tables will appear in **graph/** directory):
Enter the following command in same directory

Note: Make sure you've all packages installed as mentioned before

```
$python3 draw.py
```

It'll  draw graphs and table corresponding to all engines

For Specific engines you may also use:

```
$python3 draw.py [enigne names]
```

for eg: `$python3 draw.py sqlite mariadb_idx`

It'll only draw graphs and tables for sqlite and mariadb with indexing